

NODEWORTHY

Heroes, Things and Nodes

Every ready-made player, thing and node that comes with NodeWorthy, one at a time. What it is, how to add it, what every setting does, how to make it your own, and what to do when it does not work.

For NodeWorthy on Godot 4.7. 30 September 2026.

NodeWorthy © 2026 StateCraft Games. All rights reserved.

How to use this book

NodeWorthy comes with a good deal that is already made. There are whole players, enemies, chests, villagers and menus, and nodes that put a health bar or a map on the screen. This book gives each of them a few pages of its own. Look one up when you are about to add it, or when you have added it and want to know what a setting does or why it is not doing what you expected.

You do not need to read it from the front. Every entry stands by itself. It does help to read this page first, because every entry leans on the few ideas here.

Where everything is

Nearly everything in this book is added from one place. In Godot's Scene dock, the list of nodes at the top left, right-click a node and point at **NodeWorthy**. The menu offers only what suits the node you right-clicked.

To start a whole level with many of them at once, press **+** in the NodeWorthy panel and choose **New Level....** Tick a player, enemies, pickups, a chest, a villager, what the screen shows and the menus, and they are put in a new level laid out along its ground, the same as adding each one from the right-click menu.

RIGHT-CLICK	POINT AT	AND YOU ARE OFFERED
The top node of an empty scene	Start A 2D Level... or Start A 3D Level...	A Ground to stand on, a Camera, and a sky. It is the quickest way to a level you can add things to.
The top node of a level	Add Player	The heroes, and only those that suit the scene, 2D or 3D.
The top node of a level	Add Thing	Ground, Enemy, Level Exit, Item Pickup, Villager and Chest.
The top node of a level	Add Menu	Main Menu and Pause Menu.
The top node of a level	Add To Screen	The nodes that draw on the screen, such as Stat Panel, Bar and Minimap.
A character or a thing	Give It	Health, a Gun, and the nodes a character carries, such as Inventory and Perception.
A character or a thing	Looks	Your own picture, sprite sheet or 3D model in place of the plain shape.
A character or a thing	Open Its State Machine	Its template, opened in the NodeWorthy panel.

Whatever you add, one Undo takes all of it away again.

What every entry tells you

What it is good for	The kinds of game it suits, and which other entry to pick when it does not.
Add it	Every click, from a scene to the thing standing in it, and what to press to see it work.
What you get	Each node that appears in the Scene dock, what Godot calls it and what it is for.
The controls	What the player presses, on the keyboard and on a gamepad.
Its settings	Every setting by the name the Inspector gives it, what it changes in the game, and what it starts as.

How it works	For a hero or a thing, each state of its template in plain words.
Make it your own	The changes people most often want, step by step.
When it goes wrong	What you might see, why, and what to do about it.

Settings live in the Inspector

Click a hero or a thing in the Scene dock and look at the Inspector, the panel on the right. Its settings are at the top, above Godot's own, in sections that fold, such as Moving, Jumping and Dash. Hover a setting's name and it says what it does. A number has its unit beside it, such as px/s for pixels a second in 2D, or m/s for metres a second in 3D.

A change made there belongs to the one you clicked. Two enemies can walk at different speeds. An arrow appears beside a setting you have changed, and pressing it puts the template's value back. **Use These For Every Copy**, under the settings, makes your numbers the template's own, so every copy starts with them.

The nodes in the last two parts of the book are simpler. Click one and its settings are in the Inspector under its own name, like any Godot node's.

The Inspector is there on the **2D** and **3D** tabs. On the **NodeWorthy** tab it steps aside, so the graph has the room. Press **Show Inspector** at the top right of the editor to have it beside the graph as well, where it then stays. **Hide Inspector** in the same place puts it away again.

Behaviour lives in a template

What a hero or a thing does is drawn in its template, which is a set of states joined by arrows. A state is what it is doing now, such as On The Ground or Chasing. Each state holds actions, which run while it is in that state. An arrow is a transition, and it waits for an event, such as a key pressed or the player coming near, then moves on to the next state.

The node that runs a template is called a Machine. It sits under the hero or thing in the Scene dock, where it is named SCMachine. Press **Open Its State Machine...** under the settings and the NodeWorthy panel shows the template, with the states drawn as boxes and the actions of the state you click listed beside them.

The template a hero or a thing is added with is a copy of your own, saved in your project's **nodeworthy** folder. Change it as much as you like and nothing that came with NodeWorthy changes. A second Enemy or Chest added to the same scene runs the first one's template, so one change to the template changes them both. To make one behave differently from the rest, right-click its SCMachine and choose **NodeWorthy → New Template for This Machine...**

Using your own art

Nothing here has to look the way it comes. The pictures, models, colours and panels that come with NodeWorthy are there so that everything works and can be seen the moment it is added. Every one of them can be swapped for your own, and none of it needs code. NodeWorthy does not draw art for you. It shows the pictures and plays the animations your art already has.

TO CHANGE	DO THIS	WHAT IT TAKES
A 2D hero or thing, to one still picture	Right-click it in the Scene dock and choose NodeWorthy → Looks → Set Texture...	Any picture on your computer. It is copied into your project and sized to the thing.

TO CHANGE	DO THIS	WHAT IT TAKES
A 2D hero or thing, animated	Right-click it and choose NodeWorthy → Looks → Import Sprite Sheet...	A sprite sheet, which is one picture holding every pose, with a row for each animation. You name the rows, such as idle, run and jump, and the states play them.
A 3D hero or thing	Right-click it and choose NodeWorthy → Looks → Set Model...	A 3D model that comes with its own animations.
The colour of a picture	Click the thing and set Tint , under Look at the top of the Inspector	One set of art then makes a skeleton, a ghost and a boss.
The floor and walls of a 2D level	Right-click the top node and choose NodeWorthy → Paint With My Own Tiles → Create TileSet From Image...	A picture of tiles. You say how many across and down, and which ones block the way.
Every menu, panel and button at once	Open the Theme tab, the last tab on the right of the NodeWorthy panel	Pick a ready-made look, then change the font, the colours and the corners, or give it your own pictures for buttons and panels, and your own mouse pointer. Use For The Whole Game makes it the look of the whole game.
One thing on the screen	Click it and look at its own settings in the Inspector	A Bar has its colour, an Icon Meter its full, half and empty pictures, a Minimap and a Crosshair their colours. Each entry in this book lists them.

Each entry's **Make it your own** gives the steps for that hero or thing. One Undo puts the old look back. Free packs of sprites and models are easy to find, and many may be used in a game you sell as long as you credit the artist.

Godot's words, as this book uses them

Node	One piece of a game. A picture, a sound, a body that falls and a button are all nodes. Nodes sit under other nodes, and a node moves with the one it sits under.
Scene	A set of nodes saved as one file, such as a level or a menu. The node at the top is the scene's root.
Scene dock	The list of the open scene's nodes, at the top left of Godot.
Inspector	The panel on the right that shows the settings of whatever is clicked. It is there on the 2D and 3D tabs, and on the NodeWorthy tab once you press Show Inspector.
CharacterBody	Godot's node for a character that walks and is stopped by walls and floors. CharacterBody2D in a 2D game, CharacterBody3D in a 3D one.
StaticBody	Something solid that never moves, such as the ground or a wall.
Area	A region that notices what enters it and stops nothing. A pickup and a way out are areas.
CollisionShape	The shape that says how big a body or an area is. Without one, nothing can touch it.
CanvasLayer	A layer drawn over the game that does not move with the camera. What goes on the screen sits under one, called HUD.
Group	A name given to a set of nodes so they can be found together. The player is in the group called player, and enemies are in enemies.

Collision layer	Which kind of solid thing a body is, such as World, Player, Enemies or Pickups. It decides what bumps into what.
F6	Runs the scene that is open. F5 runs the whole game from its main scene.

Contents

Heroes

Platform Hero (2D)	A ready-made player for a 2D game seen from the side.	8
Side-On Platform Hero (3D)	The Platform Hero in a 3D world watched from the side.	17
Platform Hero (3D)	A ready-made player free to run anywhere in a 3D world, with the camera behind it turni... ..	27
Third Person Hero (3D)	A ready-made player for a 3D game seen over the shoulder.	37
Top-Down Hero (2D)	A ready-made player for a 2D game seen from above.	47
Top-Down Hero (3D)	A ready-made player for a 3D game seen from above.	54

Things

Ground	A wide, solid floor for everything to stand on.	62
Terrain	A landscape for a 3D level, with hills to walk over, solid ground and grass.	65
Model...	A model of your own, from anywhere on your computer, put into a 3D level.	71
Enemy	An enemy that thinks for itself.	74
Level Exit	A doorway out of the level.	86
Item Pickup	Something lying in the level for the player to collect.	91
Villager	Someone for the player to talk to.	97
Chest	A chest the player opens for loot.	104
Main Menu	A title screen.	111
Pause Menu	A pause screen for a level.	116

Nodes for the screen

Stat Panel	A panel that lists a character's numbers on the screen, such as Strength 15, Defence 8... ..	122
Bar	A long bar on the screen that follows one number of one character, with a name written... ..	127
Globe	A round glass globe that fills with liquid to show a number, with the number written acro... ..	132
Icon Meter	A row of icons that shows a number, such as hearts for health, skulls for lives or shells	137
Inventory Grid	A grid of slots that draws what an Inventory is carrying, with a count on any stack of mo... ..	141
Quest List	A panel that lists the quests under way and their steps, with a box against each step th... ..	146
Quest Log	The player's quest log, a screen of its own that opens over the game with the Q key.	150
Minimap	A small map in the bottom right corner of the screen.	155
Crosshair	The aiming mark in the middle of a shooter's screen.	162
Skill Bar	A row of skill slots along the bottom of the screen, each with its picture and its key.	166
Dialogue Box	The box a dialogue is shown in, along the bottom of the screen.	171
Item Collection	A grid of item pictures for a collection page.	175

Nodes a character carries

Hurt Box	A Hurt Box makes the thing it sits under able to be hurt.	180
Inventory	An Inventory is what something is carrying.	185
Perception	Perception is what a character can notice.	191
Experience	Experience gives the character it sits under levels.	195
Hit Box	A Hit Box is the part that does the hurting.	200
Data Node	A Data Node gives one thing values of its own, such as this goblin's Health and Damag... ..	205
Performance Overlay	A Performance Overlay is a small readout in the top left corner of the running game tha... ..	210

HEROES

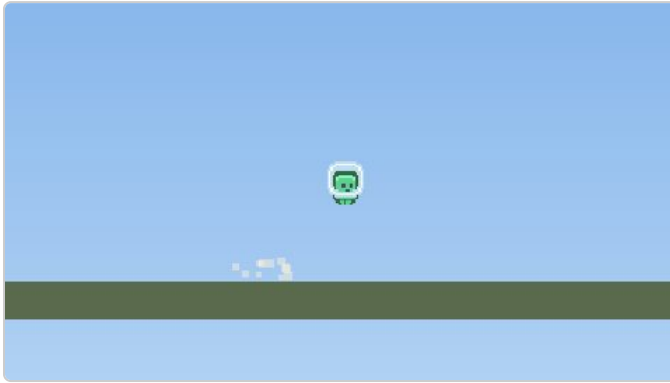
The ready-made players

A hero is a whole player. It comes as a body, its collision shape, something to see it by, a camera that follows it and a template that already makes it move. Add one to a level and the game can be played at once. There are six, two for 2D games and four for 3D, and each suits a different kind of game. They are under **NodeWorthy** → **Add Player** when you right-click the top node of a level.

Platform Hero (2D)	8
Side-On Platform Hero (3D)	17
Platform Hero (3D)	27
Third Person Hero (3D)	37
Top-Down Hero (2D)	47
Top-Down Hero (3D)	54

Platform Hero (2D) HERO, 2D

A ready-made player for a 2D game seen from the side. The moment it is added it runs, jumps, dashes eight ways, slides down walls, jumps off them, climbs, crouches and crawls. It comes as a small drawn character with a puff of dust at its feet and a camera that follows a little ahead of it.



Platform Hero (2D) as the game runs.

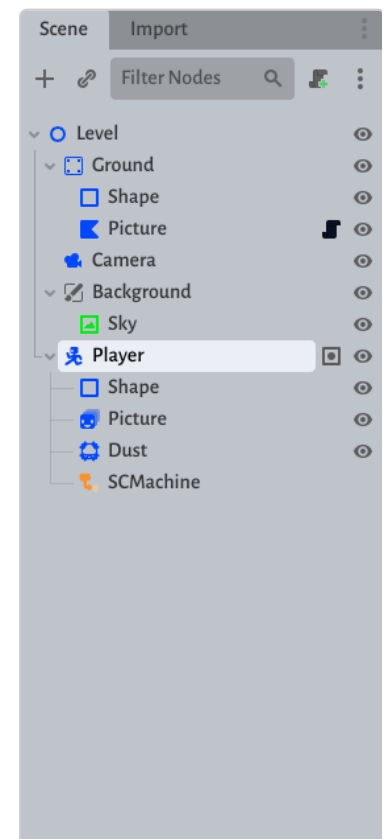
What it is good for

Platformers of every kind, from a gentle coin collector to a hard game of dashes and wall jumps, because each ability has a tick to switch it off. Pick it when the level is made of sprites or tiles. For the same hero in a world of 3D models pick the Side-On Platform Hero (3D), and for a 2D game seen from above pick the Top-Down Hero (2D).

Add it

1. Start with a 2D level that has something solid to stand on. A new 2D scene set up with **NodeWorthy** → **Start A 2D Level...** has a **Ground**, a **Camera** and a **Background**, which is all it needs.
2. In the Scene dock, Godot's list of the nodes in the scene, right-click the top node of the level.
3. Point at **NodeWorthy**, then **Add Player**, and choose **Platform Hero (2D)**.
4. A node called **Player** appears standing on the ground, already picked, and a note at the bottom right of the editor says what was added. The level's **Camera** is handed to it, or a new **Camera** is added when the scene has none.
5. Drag **Player** in the 2D view to the spot where the level should begin.
6. Press **F6**, or the **Run Current Scene** button at the top right of Godot, to play.

Try it. Hold **D** and the hero runs to the right, with the camera sliding a little ahead of it. Press the **Space** bar for a jump, hold it for a higher one, and press **Shift** in mid air to dash, after which the hero is tinted blue until it lands. Where the level has a wall, jump at it to slide down it, then hold **J** against it and press **W** to climb.



The Scene dock once it is added.

What you get

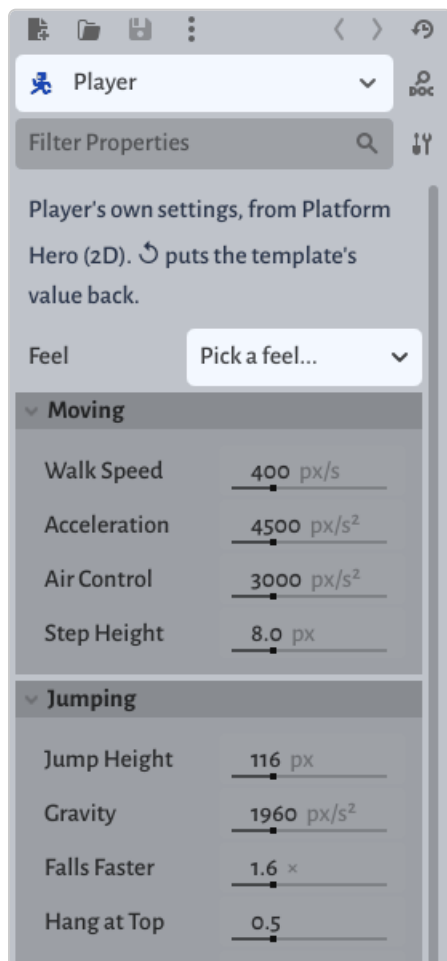
IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Player	CharacterBody2D	The hero's body. A CharacterBody2D is Godot's node for a character that moves and is stopped by floors and walls. It is on the Player collision layer and in the group called player, which is how enemies and the Minimap find it.
Shape	CollisionShape2D	The solid part of the hero, a rectangle 24 pixels wide and 40 tall. A CollisionShape2D is Godot's node that says what bumps into things. Crouching makes this rectangle shorter.
Picture	AnimatedSprite2D	What you see, a small drawn hero with an animation for each move. An AnimatedSprite2D is Godot's node that plays named sets of pictures. The template plays them by name, and also squashes and tints this node.
Dust	CPUParticles2D	A puff of pale dust at its feet, started by Jumping and Landing . A CPUParticles2D is Godot's node for a burst of small moving specks.
SCMachine	Machine	The Machine, NodeWorthy's node that runs a template. The template is the drawing of states, actions and transitions that makes the hero move, and this one is your own copy, saved in your project's nodeworthy folder, unless you have picked another folder for templates.
Camera	Camera2D	Added at the top of the scene, not under the player, and only when the scene has no Camera2D yet. A Camera2D is Godot's node for the part of a 2D level the screen shows. The template moves it to follow the hero.
HurtBox	Hurt Box	Added under Player only when the scene already holds an Enemy , or later when you choose Give It → Health . It is what lets the hero be hurt. With it come a health of 3 and a Dead state in the template that starts the level again.
HealthPanel	Stat Panel	Added with the HurtBox , on a CanvasLayer, which is Godot's node for things drawn over the game. It shows the hero's health on the screen. When the scene already has a Stat Panel , health is added to that one and no new panel is made.
GoldPanel	Stat Panel	Added only when something in the scene drops gold, as a Chest does. It shows the gold the hero has picked up. When the scene already has a Stat Panel , gold is added to that one and no new panel is made.

The controls

PRESS	WHAT IT DOES
A or the Left arrow, or the left stick pushed left	Runs left, and steers left in the air.
D or the Right arrow, or the left stick pushed right	Runs right, and steers right in the air.
The Space bar, or the bottom button on a gamepad	Jumps. A tap is a short hop and a held press goes the full Jump Height . Pressed while the hero holds a wall or climbs one, it jumps away from the wall.
Shift , or the right bumper	Dashes the way the keys or the stick point, in any of eight directions, or the way the hero faces when nothing is held.

PRESS	WHAT IT DOES
S or the Down arrow, or the left stick pulled down	On the ground it crouches for as long as it is held. In the air it lets the hero fall half as fast again. On a wall being climbed it climbs down.
C , or the right face button (B / Circle) on a gamepad	Crouches for as long as it is held, the same as holding down on the ground.
W or the Up arrow, or the left stick pushed up	While the hero slides down a wall, it pulls the hero up over a ledge within reach. On a wall being climbed it climbs up.
J or Z , or the left trigger	Held against a wall, it grabs the wall so the hero can climb until its grip runs out.

Its settings



Click **Player** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot's own settings, in sections that fold.

A change here changes this one only. The arrow that appears beside a changed setting puts the template's value back. **Use These For Every Copy**, under the settings, makes them the template's own, for every copy.

Its settings in the Inspector.

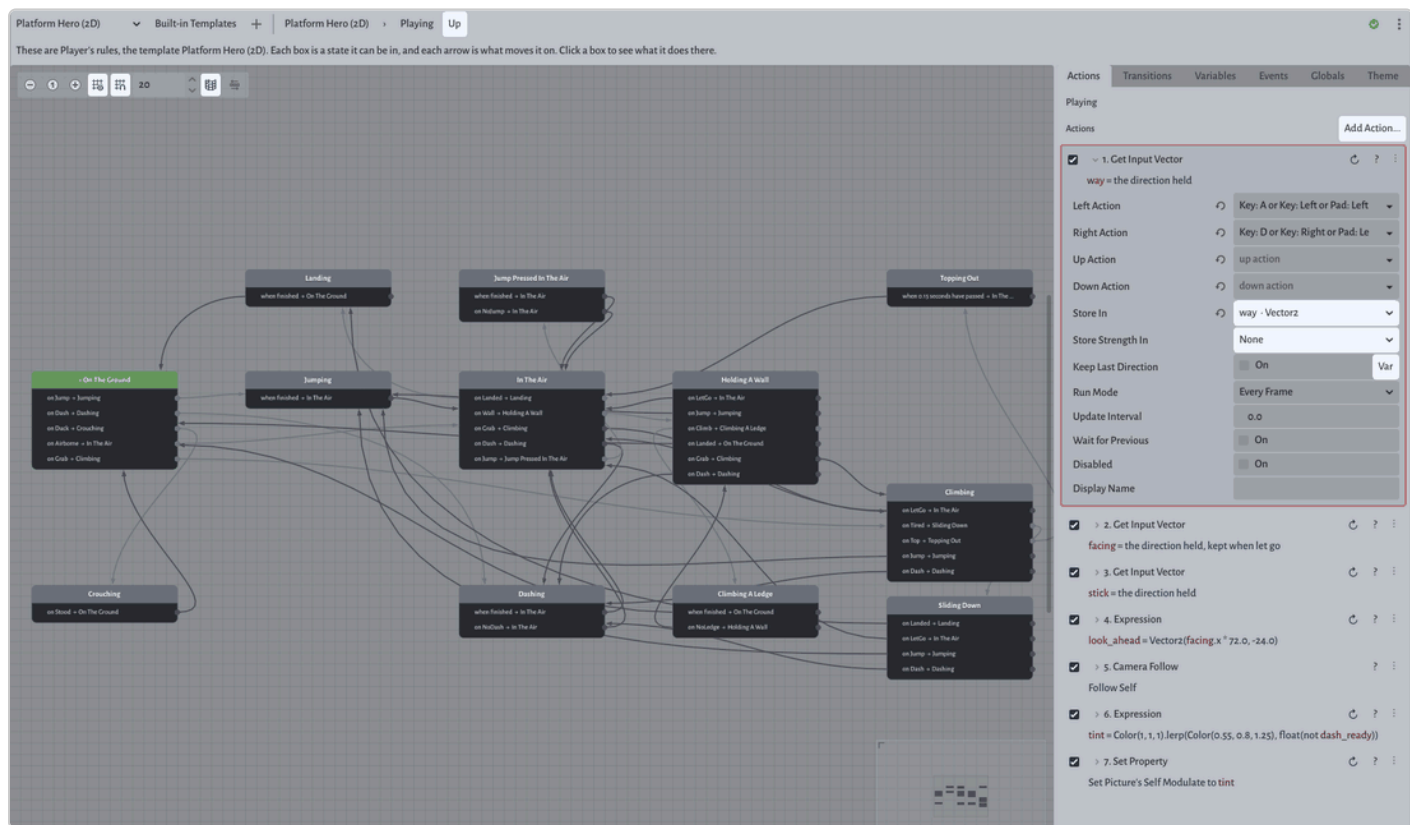
SETTING	WHAT IT DOES	STARTS AS
Feel	Pick how it feels to play and the numbers in Moving and Jumping are set for it, from its walk speed. Change any of them after. Tight Quick to get going, quick to stop, heavy on the way down with a sharp moment at the top. Precise and fair, for a game about landing on things. Floaty Slow to build up, a long hang at the top, gentle on the way down. Dreamy and drifting. Heavy Little steering once off the ground, a fast drop and a long moment at the top, so it commits to a jump. Arcade Full speed at once, the same arc up as down, no cap on falling. The old way, with no weight.	Pick a feel...

SETTING	WHAT IT DOES	STARTS AS
MOVING		
Walk Speed	How fast the hero runs on the ground and steers in the air. It starts at 400 pixels a second, which carries it ten times its own height every second. Raise it for a faster game and widen the gaps in your level to match, or lower it for slow and careful platforming.	400 px/s
Acceleration	How quickly it reaches Walk Speed , and how quickly it stops, on the ground. At 4500 it is at full speed in under a tenth of a second, which feels sharp and obedient. A low number makes it slide as if on ice.	4500 px/s ²
Air Control	The same thing while it is off the ground. At 3000 it is at full speed in a little over a tenth of a second, so it can still be steered in mid air. Lower it and a jump keeps going the way it started, which makes the player commit to it.	3000 px/s ²
Step Height	The tallest step the hero walks up without a jump. At 8 pixels it runs over small bumps and the edges of tiles. Anything taller stops it like a wall, so keep it well below Jump Height .	8 px
JUMPING		
Jump Height	How high a full jump lifts the hero when the jump key is held. It starts at 116 pixels, nearly three times its own height. Letting go early cuts the jump short, so this is the most it can reach, and every ledge it must jump onto has to be lower.	116 px
Gravity	How hard the hero is pulled down. It starts at 1960, twice Godot's usual 980. The jump is worked out from this number, so a change does not alter how high a jump goes, only how long it takes, with a higher number giving a quick snappy jump and a lower one a slow floaty one.	1960 px/s ²
Falls Faster	How much harder it is pulled on the way down than on the way up. At 1.6 a jump rises smoothly and comes down quickly, which reads as weight. At 1 it falls exactly as it rose.	1.6×
Hang at Top	How strongly it is pulled for the moment at the very top of a jump, as a share of the usual pull. At 0.5 the pull is halved there, so the hero hangs for an instant and the player has time to aim the landing. A lower number hangs longer, and 1 gives a plain arc with no hang.	0.5
Fastest Fall	The top speed of a fall, so a long drop never becomes too fast to follow. It starts at 720 pixels a second. Holding down in the air raises the limit by half, to 1080.	720 px/s
Coyote Time	How long after running off a ledge the jump key still works. A tenth of a second is too short to notice, but long enough that a press made a touch late still jumps. At 0 the hero can only jump while it is really standing on something.	0.1 s
Jump Buffer	How early a jump can be pressed before landing and still count. At 0.1 a press made a tenth of a second before the feet touch down jumps the moment they do. At 0 an early press is lost.	0.1 s
WALLS AND LEDGES		
Wall Slide Speed	How fast the hero slips down a wall it is holding. It starts at 120 pixels a second, a sixth of its fastest fall, which leaves time for a wall jump. At 0 it sticks to the wall without sliding.	120 px/s

SETTING	WHAT IT DOES	STARTS AS
Wall Jump Push	How hard a jump from a wall throws the hero sideways, away from the wall. It starts at 580 pixels a second, which is faster than it runs. A higher number carries it further from the wall before the player can steer back.	580 px/s
Ledge Reach	How high above the hero the top of a wall can be and still be pulled up onto. It starts at 60 pixels, one and a half times its own height. Raise it and the hero hauls itself onto taller ledges.	60 px
Ledge Climb Time	How long the pull up onto a ledge takes. At 0.3 of a second it is a quick vault. A longer time reads as an effort, and the hero cannot be steered until it is over.	0.3 s
Climb Speed	How fast the hero moves up and down a wall while grab is held. It starts at 120 pixels a second. Its grip lasts two and a half seconds from full, so this number also decides how high it can climb before it slips.	120 px/s
Can Wall Slide	Whether the hero takes hold of a wall it touches in the air. With it off it no longer slides down walls, jumps off them or pulls itself over a ledge, and a wall only stops it. Holding grab against a wall still climbs it while Can Climb is on, and a jump from the climb still pushes away from the wall.	on
Can Climb	Whether holding grab against a wall lets the hero climb it. Untick it and the grab key does nothing, while sliding down walls and wall jumps stay as they are.	on
DASH		
Dash Distance	How far one dash carries the hero. It starts at 162 pixels, about four times its own height. Raise it and the hero crosses wider gaps, so check that your level still holds it in.	162 px
Dash Time	How long the dash takes to cover Dash Distance . At 0.15 of a second it is a sharp burst. A longer time turns it into a glide, and a shorter one into a blink.	0.15 s
One Dash in the Air	With this on, the hero has one dash and must stand on the ground to get it back, and it is tinted blue until then. With it off, the hero can dash again and again without landing, which lets it fly across a level.	on
Can Dash	Whether the hero can dash at all. Untick it for a plain run and jump game, and the hero no longer dashes when the dash key is pressed.	on
CROUCH		
Crouch Speed	How fast it moves while crouched. It starts at 120 pixels a second, under a third of its running speed. At 0 it crouches on the spot and cannot move.	120 px/s
Crouch Height	How tall the hero's solid shape is while crouching. It stands 40 pixels tall and crouches to 24, so it fits under anything with more room than that. It will not stand up again until there is room overhead.	24 px
Can Crouch	Whether holding down on the ground crouches the hero. Untick it for a hero that always stands, and down then does nothing on the ground.	on
LOOK		
Tint	A colour laid over its picture, so one set of art makes a skeleton, a ghost and a boss. White shows the picture as drawn. It is kept on its Picture node.	white

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.



Inside **Playing**, which you open with a double-click on its box. The states that move it are in here.

STATE	WHAT HAPPENS IN IT
Playing	The state the hero is always in, with every other state nested inside it. It reads the keys and the stick each frame, keeps the camera a little ahead of the way the hero faces, and tints the picture blue while the dash is spent, over any Tint chosen for it under Look . It has no way out until the hero is given health, when DIED leads to Dead .
On The Ground	The hero runs at Walk Speed , walks up small steps, gets its dash back and recovers its grip for climbing. The jump key leads to Jumping , the dash key to Dashing , holding down to Crouching , holding grab against a wall to Climbing , and running off an edge to In The Air .
Jumping	Pushes the hero up to Jump Height , stretches the picture and puffs dust, then hands over to In The Air . The jump key leads here from the ground and from a wall.
In The Air	The hero rises and falls, steered at Air Control , showing its jump picture on the way up and its fall picture on the way down. Touching the floor leads to Landing , touching a wall leads to Holding A Wall , holding grab against a wall leads to Climbing , the dash key leads to Dashing , and the jump key leads to Jump Pressed In The Air .
Jump Pressed In The Air	Where a jump pressed in mid air is tried. A press made a moment after leaving a ledge still jumps, one made a moment before landing waits for the ground, and one made against a wall the hero may jump off pushes away from it. Only then does it stretch the picture, puff dust and show the jump picture. A press that comes to nothing shows nothing and goes back to In The Air .

STATE	WHAT HAPPENS IN IT
Landing	A squash of the picture and a puff of dust as the hero touches down, then straight on to On The Ground .
Holding A Wall	The hero slides down the wall at Wall Slide Speed . The jump key jumps away from the wall, pressing up leads to Climbing A Ledge , holding grab leads to Climbing , reaching the floor leads to On The Ground , and coming away from the wall leads back to In The Air .
Climbing A Ledge	Lifts the hero up and over the top of the wall in Ledge Climb Time , then goes to On The Ground . With no ledge within Ledge Reach it goes back to Holding A Wall .
Dashing	Checks that a dash is allowed, then sends the hero Dash Distance in Dash Time along one of eight directions, with nothing pulling it down on the way. It ends in In The Air . A dash that is not allowed goes to In The Air at once.
Crouching	Shortens the hero to Crouch Height and moves it at Crouch Speed , showing the crouch and crawl pictures. Letting go of down leads to On The Ground , but only once there is room overhead to stand.
Climbing	With grab held against a wall, up and down climb it at Climb Speed while its grip drains, which takes two and a half seconds from full. Letting go of grab leads to In The Air , passing the top of the wall leads to Topping Out , and running out of grip leads to Sliding Down . The jump and dash keys still work.
Sliding Down	The grip has run out, so the hero slips down the wall at 300 pixels a second until it lands, jumps away, dashes or comes off the wall.
Topping Out	A short hop up and forwards that carries the hero over the top of the wall it was climbing. After a moment In The Air takes over.

Make it your own

Make it faster, jump higher, or switch an ability off

1. Click **Player** in the Scene dock. Its settings show at the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, in sections that fold, called **Moving**, **Jumping**, **Walls And Ledges**, **Dash**, **Crouch** and **Look**.
2. Open **Moving** and raise **Walk Speed**, or open **Jumping** and raise **Jump Height**. Drag the number, or click it and type one.
3. To change the whole feel at once, pick **Tight**, **Floaty**, **Heavy** or **Arcade** beside **Feel**, above the sections. It sets **Acceleration**, **Air Control**, **Falls Faster**, **Hang At Top** and **Fastest Fall** from the **Walk Speed**, so set the speed first.
4. To switch an ability off, untick **Can Dash** under **Dash**, **Can Wall Slide** or **Can Climb** under **Walls And Ledges**, or **Can Crouch** under **Crouch**.
5. The ↺ arrow beside a changed setting puts the template's value back. Only this player is changed, until you press **Use These For Every Copy**. **Restore Defaults**, beside it, puts the whole template back as it came with NodeWorthy, keeping its name, and Undo brings your changes back.
6. Press **F6** to try it.

Give it your own sprite sheet

1. Make or find a sprite sheet, which is one picture holding every pose of a character in rows. Give it a row each for idle, run, jump, fall, wall, climb, dash, crouch and crawl, or as many of those as you have.
2. Right-click **Player** and choose **NodeWorthy** → **Looks** → **Import Sprite Sheet...**, then pick the sheet anywhere on your computer. It is copied into your project's pictures folder.
3. A window shows the sheet with a grid over it. Set **Across** and **Down** to the number of pictures each way, and type a name for each row. Use the names the hero's states play, which are idle, run, jump, fall, wall, climb, dash, crouch and crawl.
4. Leave **Keep the animations it has** ticked, so that a move your sheet has no row for keeps the drawn hero's pictures.
5. Pick **Drawn facing right** or **Drawn facing left** and press **Use It**. The hero plays your pictures, sized to its height. One undo puts the old look back.

Give it health and a gun

1. Right-click **Player** in the Scene dock and choose **NodeWorthy** → **Give It** → **Health**. It gets a **HurtBox**, a health of 3, and a **Dead** state in its template that waits a second and starts the level again. Its health goes on the screen in a **Stat Panel**.
2. Click **Player**. A **Health** section has joined its settings in the Inspector, with **Max Health**, and **Invincibility Time** for how long it is safe after a hit.
3. For a gun, right-click **Player** again and choose **NodeWorthy** → **Give It** → **Gun**. A second Machine called **Gun** and a marker called **Muzzle** are added under **Player**, and the bullet is a small scene saved in your project's bullets folder.
4. Run the game and press **F**, or the left face button on a gamepad. A bullet flies the way the hero faces and takes 1 health from an **Enemy** it hits.
5. Click **Player** to tune the gun. Its settings are under **Shooting**, below the heading **Gun**, and they are **Time Between Shots**, **Bullet Speed**, **Damage** and **Bullet Range Time**.

Change the key that dashes

1. Click **Player** and press **Open Its State Machine...** under its settings in the Inspector. The NodeWorthy panel, where templates are drawn, shows its template.
2. In the graph, double-click the **Playing** box to see the states inside it. A trail of names above the graph shows where you are, and **Up** takes you back out.
3. Click the **On The Ground** box. The **Actions** tab on the right lists what that state does, each action on a card of its own.
4. Open the card for the Input check whose line ends with → **Dash**. Beside **Input Action** press ▼ and pick the key you want. It takes the place of **Shift** and the right bumper.
5. To keep a gamepad button as well, press ▼ again, tick **Add the one picked to what is there (either works)** and pick the button.
6. Do the same in **In The Air**, **Holding A Wall**, **Climbing** and **Sliding Down**, which each ask for the dash key on a card of their own.

Use the same hero in the next level

1. In the first level, click **Player** and press **Use These For Every Copy** under its settings. The numbers you changed now belong to the template, not only to this one player.
2. Open the next level and add the hero again with **NodeWorthy** → **Add Player** → **Platform Hero (2D)**. It arrives with a fresh template of its own, named with a 2 at the end.
3. Right-click the new **Player** and choose **NodeWorthy** → **Behave Like**, then pick your first template, which is named with your first level, such as **Platform Hero (2D), in Level 1**. Both players now run the very same template, so a change made in one level is made in both.
4. If the first player was given health, choose **NodeWorthy** → **Give It** → **Health** on the new one too, so it has a **HurtBox** of its own. The **Dead** state is already in the template and is left as it is.
5. To carry health and gold from one level into the next, open the template with **Open Its State Machine...**, go to the **Variables** tab and tick **Carries Over Between Scenes**.

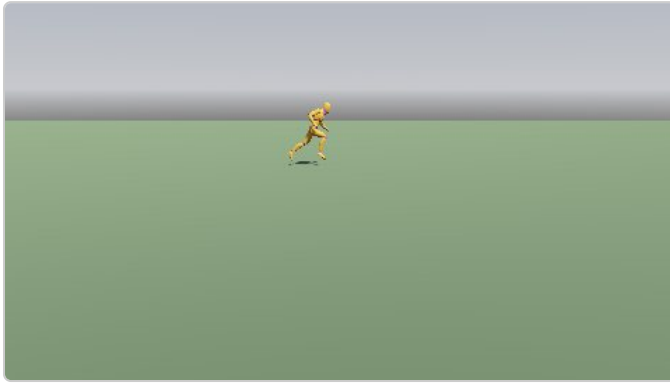
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
Set Texture... is greyed out under Looks.	This hero's picture is animated, and a still picture in its place would stop the animations its states play. Use Import Sprite Sheet... to give it new ones.
The hero drops out of sight as soon as the game starts.	Nothing solid is under it, so it falls for ever. Add a floor with NodeWorthy → Add Thing → Ground and move Player above it.
A number you change in the Inspector no longer changes the game.	An action in the template that used to read that setting has been given a number of its own, so it no longer follows the Inspector. A line at the top of the player's settings says so, naming the setting and the action. Press the button under that line, which begins with Follow , and the action reads the setting again.
Health and gold start again from nothing in the next level.	The player in the next level runs a different copy of the template, or the template has no tick in Carries Over Between Scenes . The Output panel at the bottom of Godot names which. Give the new player the first template with NodeWorthy → Behave Like , and tick the box at the top of the template's Variables tab.

See also **Side-On Platform Hero (3D)** 17, **Top-Down Hero (2D)** 47, **Ground** 62, **Enemy** 74, **Stat Panel** 122

Side-On Platform Hero (3D) HERO, 3D

The Platform Hero in a 3D world watched from the side. The moment it is added it runs, jumps, dashes, slides down walls, climbs and crouches along one line, while the camera keeps pace from the side. It comes as an animated person with a puff of dust at its feet.



Side-On Platform Hero (3D) as the game runs.

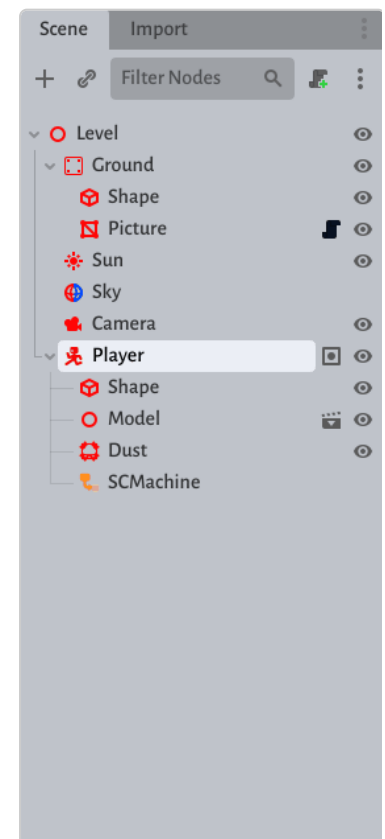
What it is good for

A platformer built from 3D models, with real lights and shadows, that is still played left and right like a 2D one. Pick it when the look is 3D and the play is side-on. For sprites or tiles pick the Platform Hero (2D), and for a hero free to run in every direction pick the Platform Hero (3D).

Add it

1. Start with a 3D level that has a floor. A new 3D scene set up with **NodeWorthy** → **Start A 3D Level...** has a **Ground**, a **Sun**, a **Sky** and a **Camera**, which is all it needs.
2. In the Scene dock, Godot's list of the nodes in the scene, right-click the top node of the level.
3. Point at **NodeWorthy**, then **Add Player**, and choose **Side-On Platform Hero (3D)**.
4. A node called **Player** appears standing on the **Ground**, already picked, and a note at the bottom right of the editor says what was added. The level's **Camera** is handed to it, or a new **Camera** is added when the scene has none.
5. Build the level along the line the hero stands on. It moves left and right along the X axis, the red arrow in Godot's 3D view, and never towards or away from the camera, so platforms and walls must cross that line.
6. Press **F6**, or the **Run Current Scene** button at the top right of Godot, to play.

Try it. Hold **D** and the hero sprints to the right while the camera keeps pace from the side, looking a little ahead and rising over any hill that comes between, so it never goes into the ground. Press the **Space** bar to jump, and hold it for a higher jump. Press **Shift** and it rolls forward in a dash, then glows blue until it is back on the ground.



The Scene dock once it is added.

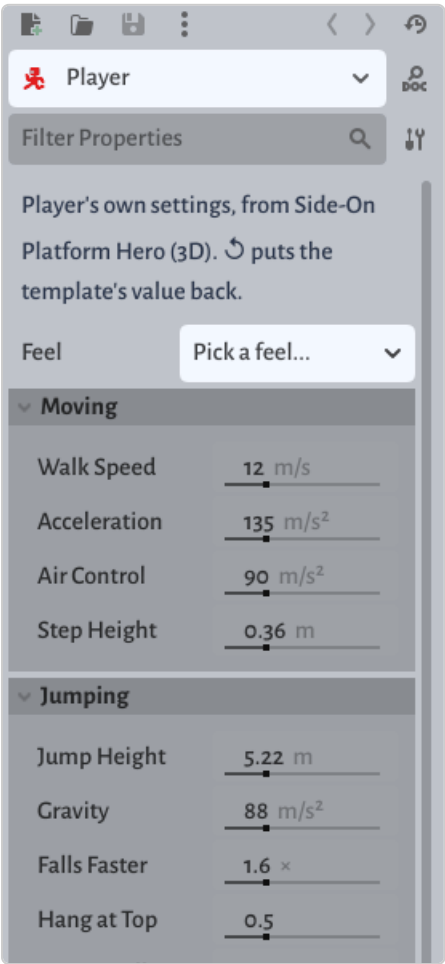
What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Player	CharacterBody3D	The hero's body. A CharacterBody3D is Godot's node for a character that moves and is stopped by floors and walls. It is held still in depth, so it only ever moves left, right, up and down. It is on the Player collision layer and in the group called player, which is how enemies and the Minimap find it.
Shape	CollisionShape3D	The solid part of the hero, a box 1.8 metres tall and 0.6 of a metre wide and deep. A CollisionShape3D is Godot's node that says what bumps into things. Crouching makes this box shorter.
Model	Node3D	What you see, an animated person with clips for standing, sprinting, jumping, rolling and crouching. It is a small scene placed under the player, and the AnimationPlayer inside it, Godot's node that plays a model's clips, is what the template's Play Animation actions play.
Dust	CPUParticles3D	A puff of pale dust at its feet, started by Jumping and Landing . A CPUParticles3D is Godot's node for a burst of small moving specks.
SCMachine	Machine	The Machine, NodeWorthy's node that runs a template. The template is the drawing of states, actions and transitions that makes the hero move, and this one is your own copy, saved in your project's nodeworthy folder, unless you have picked another folder for templates.
Camera	Camera3D	Added at the top of the scene, not under the player, and only when the scene has no Camera3D yet. A Camera3D is Godot's node for the view the screen shows. A new one starts 20 metres to the side and 2 metres up, looking level at the hero, and the template keeps it there as the hero moves.
Sun	DirectionalLight3D	Added at the top of the scene only when the scene has neither a DirectionalLight3D nor a WorldEnvironment of its own. A DirectionalLight3D is Godot's node for sunlight that falls the same way everywhere, and this one casts shadows. A level made with Start A 3D Level... already has one.
Sky	WorldEnvironment	Added with the Sun . A WorldEnvironment is Godot's node for the sky and the soft light that comes from it. Without the two, a 3D game runs almost black.
HurtBox	Hurt Box	Added under Player only when the scene already holds an Enemy , or later when you choose Give It → Health . It is what lets the hero be hurt. With it come a health of 3 and a Dead state in the template that starts the level again.
HealthPanel	Stat Panel	Added with the HurtBox , on a CanvasLayer, which is Godot's node for things drawn over the game. It shows the hero's health on the screen. When the scene already has a Stat Panel , health is added to that one and no new panel is made.
GoldPanel	Stat Panel	Added only when something in the scene drops gold, as a Chest does. It shows the gold the hero has picked up. When the scene already has a Stat Panel , gold is added to that one and no new panel is made.

The controls

PRESS	WHAT IT DOES
A or the Left arrow, or the left stick pushed left	Runs left, and steers left in the air.
D or the Right arrow, or the left stick pushed right	Runs right, and steers right in the air.
The Space bar, or the bottom button on a gamepad	Jumps. A tap is a short hop and a held press goes the full Jump Height . Pressed while the hero holds a wall or climbs one, it jumps away from the wall.
Shift , or the right bumper	Dashes the way the keys or the stick point, in any of eight directions, or the way the hero faces when nothing is held.
S or the Down arrow, or the left stick pulled down	On the ground it crouches for as long as it is held. In the air it lets the hero fall half as fast again. On a wall being climbed it climbs down.
C , or the right face button (B / Circle) on a gamepad	Crouches for as long as it is held, the same as holding down on the ground.
W or the Up arrow, or the left stick pushed up	While the hero slides down a wall, it pulls the hero up over a ledge within reach. On a wall being climbed it climbs up.
J or Z , or the left trigger	Held against a wall, it grabs the wall so the hero can climb until its grip runs out.

Its settings



Click **Player** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot’s own settings, in sections that fold.

A change here changes this one only. The arrow that appears beside a changed setting puts the template’s value back. **Use These For Every Copy**, under the settings, makes them the template’s own, for every copy.

Its settings in the Inspector.

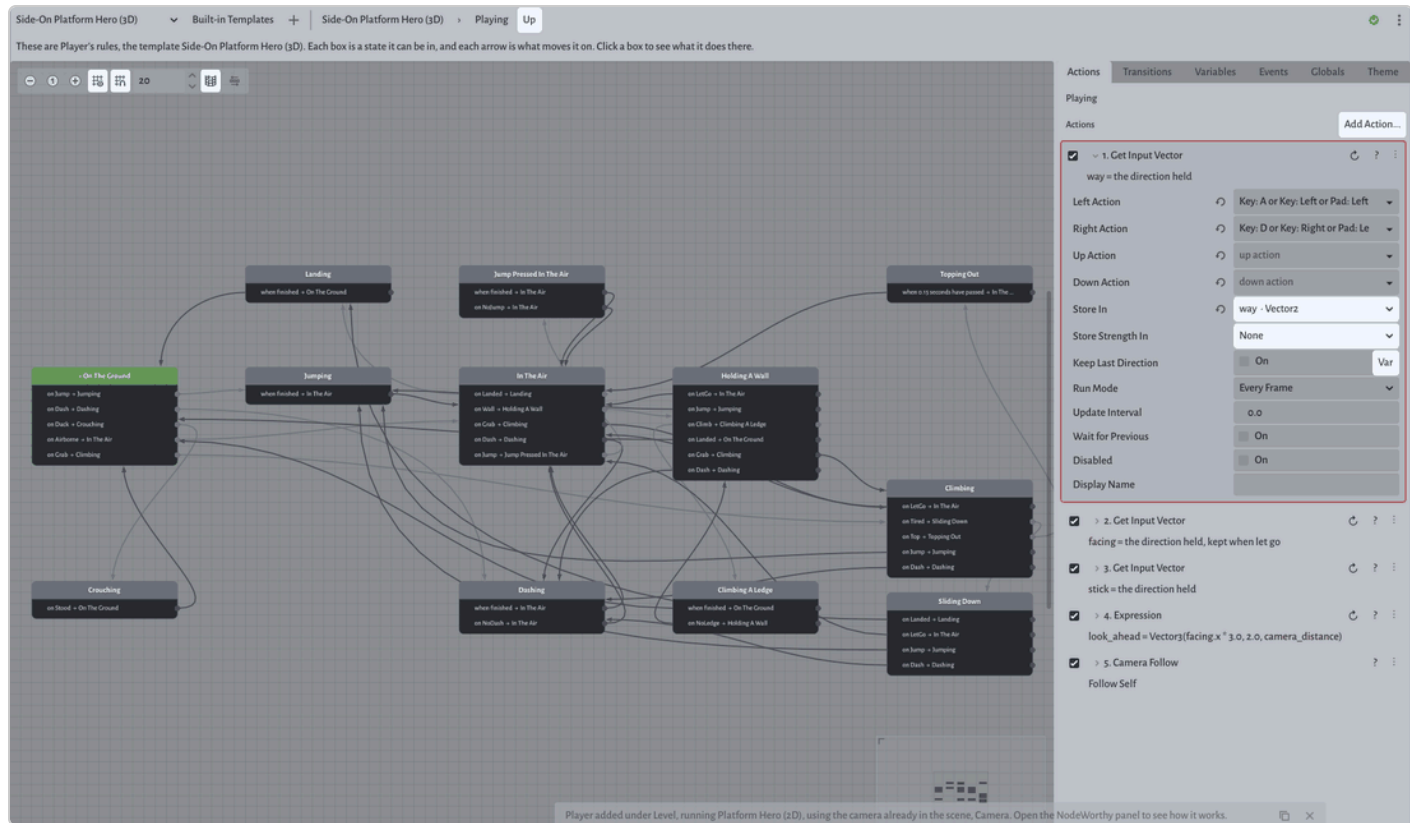
SETTING	WHAT IT DOES	STARTS AS
Feel	Pick how it feels to play and the numbers in Moving and Jumping are set for it, from its walk speed. Change any of them after. Tight Quick to get going, quick to stop, heavy on the way down with a sharp moment at the top. Precise and fair, for a game about landing on things. Floaty Slow to build up, a long hang at the top, gentle on the way down. Dreamy and drifting. Heavy Little steering once off the ground, a fast drop and a long moment at the top, so it commits to a jump. Arcade Full speed at once, the same arc up as down, no cap on falling. The old way, with no weight.	Pick a feel...
MOVING		
Walk Speed	How fast the hero runs on the ground and steers in the air. It starts at 12 metres a second, a flat out sprint. Raise it for a faster game and widen the gaps in your level to match, or lower it for slow and careful platforming.	12 m/s
Acceleration	How quickly it reaches Walk Speed , and how quickly it stops, on the ground. At 135 it is at full speed in under a tenth of a second, which feels sharp and obedient. A low number makes it slide as if on ice.	135 m/s²
Air Control	The same thing while it is off the ground. At 90 it is at full speed in a little over a tenth of a second, so it can still be steered in mid air. Lower it and a jump keeps going the way it started, which makes the player commit to it.	90 m/s²

SETTING	WHAT IT DOES	STARTS AS
Step Height	The tallest step the hero walks up without a jump. At 0.36 of a metre it runs over kerbs and low stairs. Anything taller stops it like a wall, so keep it well below Jump Height .	0.36 m
JUMPING		
Jump Height	How high a full jump lifts the hero when the jump key is held. It starts at 5.22 metres, nearly three times its own height of 1.8 metres. Letting go early cuts the jump short, so this is the most it can reach, and every ledge it must jump onto has to be lower.	5.22 m
Gravity	How hard the hero is pulled down. It starts at 88.2, nine times the real 9.8. The jump is worked out from this number, so a change does not alter how high a jump goes, only how long it takes, with a higher number giving a quick snappy jump and a lower one a slow floaty one.	88.2 m/s ²
Falls Faster	How much harder it is pulled on the way down than on the way up. At 1.6 a jump rises smoothly and comes down quickly, which reads as weight. At 1 it falls exactly as it rose.	1.6×
Hang at Top	How strongly it is pulled for the moment at the very top of a jump, as a share of the usual pull. At 0.5 the pull is halved there, so the hero hangs for an instant and the player has time to aim the landing. A lower number hangs longer, and 1 gives a plain arc with no hang.	0.5
Fastest Fall	The top speed of a fall, so a long drop never becomes too fast to follow. It starts at 32.4 metres a second. Holding down in the air raises the limit by half, to 48.6.	32.4 m/s
Coyote Time	How long after running off a ledge the jump key still works. A tenth of a second is too short to notice, but long enough that a press made a touch late still jumps. At 0 the hero can only jump while it is really standing on something.	0.1 s
Jump Buffer	How early a jump can be pressed before landing and still count. At 0.1 a press made a tenth of a second before the feet touch down jumps the moment they do. At 0 an early press is lost.	0.1 s
WALLS AND LEDGES		
Wall Slide Speed	How fast the hero slips down a wall it is holding. It starts at 5.4 metres a second, a sixth of its fastest fall, which leaves time for a wall jump. At 0 it sticks to the wall without sliding.	5.4 m/s
Wall Jump Push	How hard a jump from a wall throws the hero sideways, away from the wall. It starts at 17.4 metres a second, which is faster than it runs. A higher number carries it further from the wall before the player can steer back.	17.4 m/s
Ledge Reach	How high above the hero the top of a wall can be and still be pulled up onto. It starts at 2.7 metres, one and a half times its own height. Raise it and the hero hauls itself onto taller ledges.	2.7 m
Ledge Climb Time	How long the pull up onto a ledge takes. At 0.3 of a second it is a quick vault. A longer time reads as an effort, and the hero cannot be steered until it is over.	0.3 s
Climb Speed	How fast the hero moves up and down a wall while grab is held. It starts at 5.4 metres a second. Its grip lasts two and a half seconds from full, so this number also decides how high it can climb before it slips.	5.4 m/s

SETTING	WHAT IT DOES	STARTS AS
Can Wall Slide	Whether the hero takes hold of a wall it touches in the air. With it off it no longer slides down walls, jumps off them or pulls itself over a ledge, and a wall only stops it. Holding grab against a wall still climbs it while Can Climb is on, and a jump from the climb still pushes away from the wall.	on
Can Climb	Whether holding grab against a wall lets the hero climb it. Untick it and the grab key does nothing, while sliding down walls and wall jumps stay as they are.	on
DASH		
Dash Distance	How far one dash carries the hero. It starts at 7.29 metres, about four times its own height. Raise it and the hero crosses wider gaps, so check that your level still holds it in.	7.29 m
Dash Time	How long the dash takes to cover Dash Distance . At 0.15 of a second it is a sharp burst. A longer time turns it into a glide, and a shorter one into a blink.	0.15 s
One Dash in the Air	With this on, the hero has one dash and must stand on the ground to get it back, and it glows blue until then. With it off, the hero can dash again and again without landing, which lets it fly across a level.	on
Can Dash	Whether the hero can dash at all. Untick it for a plain run and jump game, and the hero no longer dashes when the dash key is pressed.	on
CROUCH		
Crouch Speed	How fast it moves while crouched. It starts at 3 metres a second, a quarter of its running speed. At 0 it crouches on the spot and cannot move.	3 m/s
Crouch Height	How tall the hero's solid shape is while crouching. It stands 1.8 metres tall and crouches to 1.08, so it fits under anything with more room than that. It will not stand up again until there is room overhead.	1.08 m
Can Crouch	Whether holding down on the ground crouches the hero. Untick it for a hero that always stands, and down then does nothing on the ground.	on
CAMERA		
Camera Distance	How far to the side the camera stands, watching the hero. At 20 metres the hero is small on the screen and a good stretch of the level shows. A lower number brings the camera nearer and makes the hero bigger.	20 m

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.



Inside **Playing**, which you open with a double-click on its box. The states that move it are in here.

STATE	WHAT HAPPENS IN IT
Playing	The state the hero is always in, with every other state nested inside it. It reads the keys and the stick each frame and keeps the camera to the side of the hero, a little above it and a little ahead of the way it faces. It has no way out until the hero is given health, when DIED leads to Dead .
On The Ground	The hero runs at Walk Speed , walks up small steps, gets its dash back, loses the blue glow and recovers its grip for climbing. The jump key leads to Jumping , the dash key to Dashing , holding down to Crouching , holding grab against a wall to Climbing , and running off an edge to In The Air .
Jumping	Pushes the hero up to Jump Height , stretches the model and puffs dust, then hands over to In The Air . The jump key leads here from the ground and from a wall.
In The Air	The hero rises and falls, steered at Air Control . Touching the floor leads to Landing , touching a wall leads to Holding A Wall , holding grab against a wall leads to Climbing , the dash key leads to Dashing , and the jump key leads to Jump Pressed In The Air .
Jump Pressed In The Air	Where a jump pressed in mid air is tried. A press made a moment after leaving a ledge still jumps, one made a moment before landing waits for the ground, and one made against a wall the hero may jump off pushes away from it. Only then does it stretch the model, puff dust and play its jump clip. A press that comes to nothing shows nothing and goes back to In The Air .
Landing	A squash of the model and a puff of dust as the hero touches down, then straight on to On The Ground .
Holding A Wall	The hero slides down the wall at Wall Slide Speed . The jump key jumps away from the wall, pressing up leads to Climbing A Ledge , holding grab leads to Climbing , reaching the floor leads to On The Ground , and coming away from the wall leads back to In The Air .

STATE	WHAT HAPPENS IN IT
Climbing A Ledge	Lifts the hero up and over the top of the wall in Ledge Climb Time , then goes to On The Ground . With no ledge within Ledge Reach it goes back to Holding A Wall .
Dashing	Checks that a dash is allowed, then turns the model's glow blue and sends the hero Dash Distance in Dash Time along one of eight directions, with nothing pulling it down on the way. It ends in In The Air . A dash that is not allowed goes to In The Air at once.
Crouching	Shortens the hero to Crouch Height and moves it at Crouch Speed , playing its crouching clips. Letting go of down leads to On The Ground , but only once there is room overhead to stand.
Climbing	With grab held against a wall, up and down climb it at Climb Speed while its grip drains, which takes two and a half seconds from full. Letting go of grab leads to In The Air , passing the top of the wall leads to Topping Out , and running out of grip leads to Sliding Down . The jump and dash keys still work.
Sliding Down	The grip has run out, so the hero slips down the wall at 13.5 metres a second until it lands, jumps away, dashes or comes off the wall.
Topping Out	A short hop up and forwards that carries the hero over the top of the wall it was climbing. After a moment In The Air takes over.

Make it your own

Make it faster, jump higher, or switch an ability off

1. Click **Player** in the Scene dock. Its settings show at the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, in sections that fold, called **Moving**, **Jumping**, **Walls And Ledges**, **Dash**, **Crouch** and **Camera**.
2. Open **Moving** and raise **Walk Speed**, or open **Jumping** and raise **Jump Height**. Drag the number, or click it and type one.
3. To change the whole feel at once, pick **Tight**, **Floaty**, **Heavy** or **Arcade** beside **Feel**, above the sections. It sets **Acceleration**, **Air Control**, **Falls Faster**, **Hang At Top** and **Fastest Fall** from the **Walk Speed**, so set the speed first.
4. To switch an ability off, untick **Can Dash** under **Dash**, **Can Wall Slide** or **Can Climb** under **Walls And Ledges**, or **Can Crouch** under **Crouch**.
5. The ↺ arrow beside a changed setting puts the template's value back. Only this player is changed, until you press **Use These For Every Copy**. **Restore Defaults**, beside it, puts the whole template back as it came with NodeWorthy, keeping its name, and Undo brings your changes back.
6. Press **F6** to try it.

Give it your own model

1. Right-click **Player** and choose **NodeWorthy** → **Looks** → **Set Model...**, then pick a model file anywhere on your computer. It can be a .glb, a .gltf or an .fbx file, and it is copied into a folder of its own in your project's models folder, with the textures its file names. A model with no animations of its own is given them afterwards with **NodeWorthy** → **Looks** → **Use Animations From... (Experimental)**.
2. A window asks which way the model faces. Pick the line that begins **Towards you** or the one that begins **Away from you**, and press **Use It**. When the file holds a whole cast, as a character pack's one file does, the same window asks which character, and the extras it wears, such as hair or a hat. Your model takes the place of the animated person, sized to the hero's height. One undo brings the person back.
3. Each state that played one of the person's clips now plays one of your model's, guessed from the names of its clips. **On The Ground** plays a standing clip and a walking or running one while it moves, **Jumping** a jump, **In The Air** a fall, **Landing** a landing, **Holding A Wall** and **Sliding Down** a wall clip, **Climbing** and **Climbing A Ledge** a climb, **Dashing** a dash or a roll and **Crouching** a crouch. Where the model has no clip for a move the nearest one stands in, such as its fall for holding a wall.
4. Click **Player**. A new **Animations** section in the Inspector lists each guess, as **Idle Clip**, **Walk Clip**, **Jump Clip**, **Fall Clip**, **Crouch Clip**, **Crouch Walk Clip**, **Land Clip**, **Wall Clip**, **Climb Clip** and **Dash Clip**. Where a guess is wrong, pick another clip from its list.
5. **Blend Time** in the same section is how long one clip takes to take over from the last. It starts at a tenth of a second, as quick as the hero came, and 0 snaps.

Give it health and a gun

1. Right-click **Player** in the Scene dock and choose **NodeWorthy** → **Give It** → **Health**. It gets a **HurtBox**, a health of 3, and a **Dead** state in its template that waits a second and starts the level again. Its health goes on the screen in a **Stat Panel**.
2. Click **Player**. A **Health** section has joined its settings in the Inspector, with **Max Health**, and **Invincibility Time** for how long it is safe after a hit.
3. For a gun, right-click **Player** again and choose **NodeWorthy** → **Give It** → **Gun**. A second Machine called **Gun** and a marker called **Muzzle** are added under **Player**, and the bullet is a small scene saved in your project's bullets folder.
4. Run the game and press **F**, or the left face button on a gamepad. A bullet flies the way the hero faces and takes 1 health from an **Enemy** it hits.
5. Click **Player** to tune the gun. Its settings are under **Shooting**, below the heading **Gun**, and they are **Time Between Shots**, **Bullet Speed**, **Damage** and **Bullet Range Time**.

Change the key that grabs a wall

1. Click **Player** and press **Open Its State Machine...** under its settings in the Inspector. The NodeWorthy panel, where templates are drawn, shows its template.
2. In the graph, double-click the **Playing** box to see the states inside it. A trail of names above the graph shows where you are, and **Up** takes you back out.
3. Click the **On The Ground** box. The **Actions** tab on the right lists what that state does, each action on a card of its own.
4. Open the card for the Input check whose line begins with grab_held. Beside **Input Action** press ▼ and pick the key you want. It takes the place of **J, Z** and the left trigger.
5. To keep a gamepad button as well, press ▼ again, tick **Add the one picked to what is there (either works)** and pick the button.
6. Do the same in **In The Air** and **Holding A Wall**, on the card whose line begins with grab_held, and in **Climbing**, on the card whose line ends with → **LetGo**.

Use the same hero in the next level

1. In the first level, click **Player** and press **Use These For Every Copy** under its settings. The numbers you changed now belong to the template, not only to this one player.
2. Open the next level and add the hero again with **NodeWorthy** → **Add Player** → **Side-On Platform Hero (3D)**. It arrives with a fresh template of its own, named with a 2 at the end.
3. Right-click the new **Player** and choose **NodeWorthy** → **Behave Like**, then pick your first template, which is named with your first level, such as **Side-On Platform Hero (3D), in Level 1**. Both players now run the very same template, so a change made in one level is made in both.
4. If the first player was given health, choose **NodeWorthy** → **Give It** → **Health** on the new one too, so it has a **HurtBox** of its own. The **Dead** state is already in the template and is left as it is.
5. To carry health and gold from one level into the next, open the template with **Open Its State Machine...**, go to the **Variables** tab and tick **Carries Over Between Scenes**.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
After Set Model... the model glides about without moving its limbs.	Click Player and look for an Animations section in the Inspector. If there is none, the model file holds no animations. Give it some with NodeWorthy → Looks → Use Animations From... (Experimental) , from any animation file. It is experimental, so watch each move before you rely on it. If there is one, a clip was guessed wrongly from its name, so pick the right clip on each line.
After Set Model... the model is plain white, or a part of it is.	The pictures its file names were not found beside it. Right-click Player and choose NodeWorthy → Looks → Set Model Textures... , and pick each part's picture from the pack's Textures folder.
The hero runs straight past a platform or a wall that looks as if it is in the way.	The hero never moves in depth, so it only touches things that cross its own line. Click the platform and Player in turn and compare the Z number of Position in the Inspector, then move the platform until it sits on the hero's line.
The hero drops out of sight as soon as the game starts.	Nothing solid is under it, so it falls for ever. Add a floor with NodeWorthy → Add Thing → Ground and move Player above it.

See also **Platform Hero (2D)** 8, **Platform Hero (3D)** 27, **Ground** 62, **Enemy** 74

Platform Hero (3D) HERO, 3D

A ready-made player free to run anywhere in a 3D world, with the camera behind it turning with the mouse or the right stick. The moment it is added it runs where the camera looks, jumps, dashes, slides down walls, jumps off them, pulls itself over ledges, climbs and crouches. It comes as an animated person with a puff of dust at its feet.



Platform Hero (3D) as the game runs.

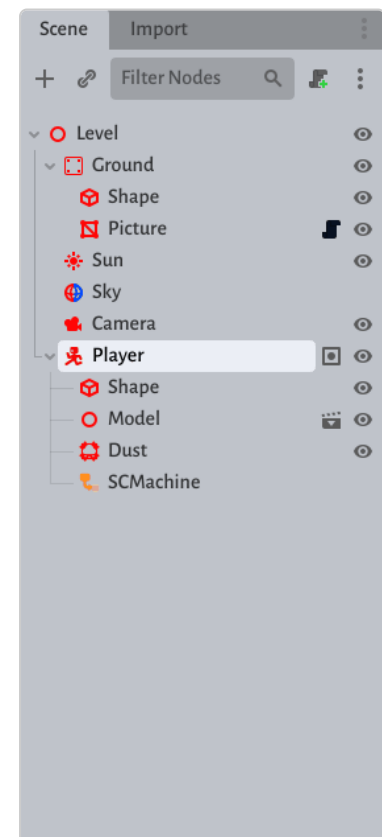
What it is good for

3D platformers and adventure games where the player explores in every direction and climbing is part of the fun. For a calmer game seen over the shoulder, with sprinting and no wall moves, pick the Third Person Hero (3D). For a 3D world played along one line pick the Side-On Platform Hero (3D), and for a camera that looks down from above pick the Top-Down Hero (3D).

Add it

1. Start with a 3D level that has a floor. A new 3D scene set up with **NodeWorthy** → **Start A 3D Level...** has a **Ground**, a **Sun**, a **Sky** and a **Camera**, which is all it needs.
2. In the Scene dock, Godot's list of the nodes in the scene, right-click the top node of the level.
3. Point at **NodeWorthy**, then **Add Player**, and choose **Platform Hero (3D)**.
4. A node called **Player** appears standing on the **Ground**, already picked, and a note at the bottom right of the editor says what was added. The level's **Camera** is handed to it, or a new **Camera** is added behind the hero when the scene has none.
5. Drag **Player** in the 3D view to the spot where the level should begin.
6. Press **F6**, or the **Run Current Scene** button at the top right of Godot, to play.

Try it. Move the mouse and the camera swings round the hero. Hold **W** and it runs away from the camera, turning to face the way it goes. Press the **Space** bar to jump, **Shift** to roll in a dash that leaves it glowing blue until it lands, and hold **C** to crouch. Press **Escape** to get the mouse pointer back.



The Scene dock once it is added.

What you get

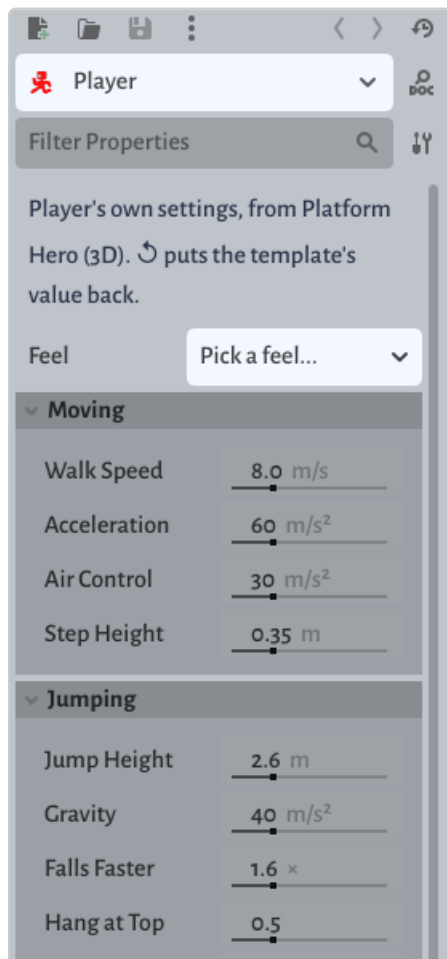
IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Player	CharacterBody3D	The hero's body. A CharacterBody3D is Godot's node for a character that moves and is stopped by floors and walls. It is on the Player collision layer and in the group called player, which is how enemies and the Minimap find it.
Shape	CollisionShape3D	The solid part of the hero, a capsule 1.8 metres tall. A CollisionShape3D is Godot's node that says what bumps into things. Crouching makes this capsule shorter.
Model	Node3D	What you see, an animated person with clips for standing, sprinting, jumping, rolling and crouching. It is a small scene placed under the player, and the AnimationPlayer inside it, Godot's node that plays a model's clips, is what the template's Play Animation actions play.
Dust	CPUParticles3D	A puff of pale dust at its feet, started by Jumping and Landing . A CPUParticles3D is Godot's node for a burst of small moving specks.
SCMachine	Machine	The Machine, NodeWorthy's node that runs a template. The template is the drawing of states, actions and transitions that makes the hero move, and this one is your own copy, saved in your project's nodeworthy folder, unless you have picked another folder for templates.
Camera	Camera3D	Added at the top of the scene, not under the player, and only when the scene has no Camera3D yet. A Camera3D is Godot's node for the view the screen shows. A new one starts behind and above the hero, looking down at it, and the template swings it round the hero as the mouse moves.
Sun	DirectionalLight3D	Added at the top of the scene only when the scene has neither a DirectionalLight3D nor a WorldEnvironment of its own. A DirectionalLight3D is Godot's node for sunlight that falls the same way everywhere, and this one casts shadows. A level made with Start A 3D Level... already has one.
Sky	WorldEnvironment	Added with the Sun . A WorldEnvironment is Godot's node for the sky and the soft light that comes from it. Without the two, a 3D game runs almost black.
HurtBox	Hurt Box	Added under Player only when the scene already holds an Enemy , or later when you choose Give It → Health . It is what lets the hero be hurt. With it come a health of 3 and a Dead state in the template that starts the level again.
HealthPanel	Stat Panel	Added with the HurtBox , on a CanvasLayer, which is Godot's node for things drawn over the game. It shows the hero's health on the screen. When the scene already has a Stat Panel , health is added to that one and no new panel is made.

The controls

PRESS	WHAT IT DOES
W or the Up arrow, or the left stick pushed up	Runs away from the camera. On a wall being climbed it climbs up.

PRESS	WHAT IT DOES
S or the Down arrow, or the left stick pulled down	Runs towards the camera. On a wall being climbed it climbs down.
A and D , the Left and Right arrows, or the left stick pushed sideways	Run to the left and the right of where the camera looks.
The Space bar, or the bottom button on a gamepad	Jumps. A tap is a short hop and a held press goes the full Jump Height . Pressed while the hero holds a wall or climbs one, it jumps away from the wall.
Shift , or the right bumper	Dashes the way the hero is running, or the way it faces when no direction is held.
C , or the right face button (B / Circle) on a gamepad	Crouches for as long as it is held.
J , or the left trigger	Held against a wall, it grabs the wall so the hero can climb until its grip runs out.
Moving the mouse, or pushing the right stick	Swings the camera round the hero.
The mouse wheel	Moves the camera nearer to the hero or further from it, between 2 and 12 metres.
Escape	Gives the mouse pointer back while the game runs. A click in the game hands it to the camera again.

Its settings



Its settings in the Inspector.

Click **Player** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot's own settings, in sections that fold.

A change here changes this one only. The arrow that appears beside a changed setting puts the template's value back. **Use These For Every Copy**, under the settings, makes them the template's own, for every copy.

SETTING	WHAT IT DOES	STARTS AS
Feel	Pick how it feels to play and the numbers in Moving and Jumping are set for it, from its walk speed. Change any of them after. Tight Quick to get going, quick to stop, heavy on the way down with a sharp moment at the top. Precise and fair, for a game about landing on things. Floaty Slow to build up, a long hang at the top, gentle on the way down. Dreamy and drifting. Heavy Little steering once off the ground, a fast drop and a long moment at the top, so it commits to a jump. Arcade Full speed at once, the same arc up as down, no cap on falling. The old way, with no weight.	Pick a feel...
MOVING		
Walk Speed	How fast the hero runs on the ground and steers in the air. It starts at 8 metres a second, a brisk run. Raise it for a faster game and widen the gaps in your level to match, or lower it for slow and careful platforming.	8 m/s
Acceleration	How quickly it reaches Walk Speed , and how quickly it stops, on the ground. At 60 it is at full speed in about an eighth of a second, which feels sharp and obedient. A low number makes it slide as if on ice.	60 m/s ²
Air Control	The same thing while it is off the ground. At 30 it is at full speed in about a quarter of a second, so it can still be steered in mid air. Lower it and a jump keeps going the way it started, which makes the player commit to it.	30 m/s ²
Step Height	The tallest step the hero walks up without a jump. At 0.35 of a metre it runs over kerbs and stairs. Anything taller stops it like a wall, so keep it well below Jump Height .	0.35 m
JUMPING		
Jump Height	How high a full jump lifts the hero when the jump key is held. It starts at 2.6 metres, about one and a half times its own height of 1.8 metres. Letting go early cuts the jump short, so this is the most it can reach, and every ledge it must jump onto has to be lower.	2.6 m
Gravity	How hard the hero is pulled down. It starts at 40, four times the real 9.8. The jump is worked out from this number, so a change does not alter how high a jump goes, only how long it takes, with a higher number giving a quick snappy jump and a lower one a slow floaty one.	40 m/s ²
Falls Faster	How much harder it is pulled on the way down than on the way up. At 1.6 a jump rises smoothly and comes down quickly, which reads as weight. At 1 it falls exactly as it rose.	1.6×
Hang at Top	How strongly it is pulled for the moment at the very top of a jump, as a share of the usual pull. At 0.5 the pull is halved there, so the hero hangs for an instant and the player has time to aim the landing. A lower number hangs longer, and 1 gives a plain arc with no hang.	0.5
Fastest Fall	The top speed of a fall, so a long drop never becomes too fast to follow. It starts at 30 metres a second. A lower number gives a gentler drop, and a higher one a harder landing to judge.	30 m/s
Coyote Time	How long after running off a ledge the jump key still works. A tenth of a second is too short to notice, but long enough that a press made a touch late still jumps. At 0 the hero can only jump while it is really standing on something.	0.1 s

SETTING	WHAT IT DOES	STARTS AS
Jump Buffer	How early a jump can be pressed before landing and still count. At 0.1 a press made a tenth of a second before the feet touch down jumps the moment they do. At 0 an early press is lost.	0.1 s
WALLS AND LEDGES		
Wall Slide Speed	How fast the hero slips down a wall it is holding. It starts at 3 metres a second, a tenth of its fastest fall, which leaves time for a wall jump. At 0 it sticks to the wall without sliding.	3 m/s
Wall Jump Push	How hard a jump from a wall throws the hero sideways, away from the wall. It starts at 9 metres a second, which is faster than it runs. A higher number carries it further from the wall before the player can steer back.	9 m/s
Ledge Reach	How high above the hero the top of a wall can be and still be pulled up onto. It starts at 1.5 metres, a little less than its own height. Raise it and the hero hauls itself onto taller ledges.	1.5 m
Ledge Climb Time	How long the pull up onto a ledge takes. At 0.3 of a second it is a quick vault. A longer time reads as an effort, and the hero cannot be steered until it is over.	0.3 s
Climb Speed	How fast the hero moves up and down a wall while grab is held. It starts at 3 metres a second. Its grip lasts two and a half seconds from full, so this number also decides how high it can climb before it slips.	3 m/s
Can Wall Slide	Whether the hero takes hold of a wall it touches in the air. With it off it no longer slides down walls, jumps off them or pulls itself over a ledge, and a wall only stops it. Holding grab against a wall still climbs it while Can Climb is on, and a jump from the climb still pushes away from the wall.	on
Can Climb	Whether holding grab against a wall lets the hero climb it. Untick it and the grab key does nothing, while sliding down walls and wall jumps stay as they are.	on
DASH		
Dash Distance	How far one dash carries the hero. It starts at 5 metres, nearly three times its own height. Raise it and the hero crosses wider gaps, so check that your level still holds it in.	5 m
Dash Time	How long the dash takes to cover Dash Distance . At 0.15 of a second it is a sharp burst. A longer time turns it into a glide, and a shorter one into a blink.	0.15 s
One Dash in the Air	With this on, the hero has one dash and must stand on the ground to get it back, and it glows blue until then. With it off, the hero can dash again and again without landing, which lets it fly across a level.	on
Can Dash	Whether the hero can dash at all. Untick it for a plain run and jump game, and the hero no longer dashes when the dash key is pressed.	on
CROUCH		
Crouch Speed	How fast it moves while crouched. It starts at 2.5 metres a second, under a third of its running speed. At 0 it crouches on the spot and cannot move.	2.5 m/s
Crouch Height	How tall the hero's solid shape is while crouching. It stands 1.8 metres tall and crouches to 1, so it fits under anything with more room than that. It will not stand up again until there is room overhead.	1 m

STATE	WHAT HAPPENS IN IT
In The Air	The hero rises and falls, steered at Air Control . Touching the floor leads to Landing , touching a wall leads to Reaching For A Ledge , holding grab against a wall leads to Climbing , the dash key leads to Dashing , and the jump key leads to Jump Pressed In The Air .
Jump Pressed In The Air	Where a jump pressed in mid air is tried. A press made a moment after leaving a ledge still jumps, one made a moment before landing waits for the ground, and one made against a wall the hero may jump off pushes away from it. Only then does it stretch the model, puff dust and play its jump clip. A press that comes to nothing shows nothing and goes back to In The Air .
Landing	A squash of the model and a puff of dust as the hero touches down, then straight on to On The Ground .
Reaching For A Ledge	The moment the hero takes hold of a wall it reaches up for the top. With a ledge within Ledge Reach it pulls itself over in Ledge Climb Time and goes to On The Ground . With none it goes to Holding A Wall .
Holding A Wall	The hero slides down the wall at Wall Slide Speed . The jump key jumps away from the wall, holding grab leads to Climbing , reaching the floor leads to On The Ground , and coming away from the wall leads back to In The Air .
Dashing	Checks that a dash is allowed, then turns the model's glow blue and sends the hero Dash Distance in Dash Time the way it is running. With no direction held it hands over to Dashing Ahead . It ends in In The Air , and a dash that is not allowed goes there at once.
Dashing Ahead	The dash for a hero that is standing still. It goes Dash Distance the way the hero faces, then ends in In The Air .
Crouching	Shortens the hero to Crouch Height and moves it at Crouch Speed , playing its crouching clips. Letting go of the crouch key leads to On The Ground , but only once there is room overhead to stand.
Climbing	With grab held against a wall, forward and back on the keys or the stick climb up and down it at Climb Speed while its grip drains, which takes two and a half seconds from full. Letting go of grab leads to In The Air , passing the top of the wall leads to Topping Out , and running out of grip leads to Sliding Down . The jump and dash keys still work.
Sliding Down	The grip has run out, so the hero slips down the wall at 7 metres a second until it lands, jumps away, dashes or comes off the wall.
Topping Out	A short hop up and forwards that carries the hero over the top of the wall it was climbing. After a moment In The Air takes over.

Make it your own

Make it faster, jump higher, or switch an ability off

1. Click **Player** in the Scene dock. Its settings show at the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, in sections that fold, called **Moving**, **Jumping**, **Walls And Ledges**, **Dash**, **Crouch** and **Camera**.
2. Open **Moving** and raise **Walk Speed**, or open **Jumping** and raise **Jump Height**. Drag the number, or click it and type one.
3. To change the whole feel at once, pick **Tight**, **Floaty**, **Heavy** or **Arcade** beside **Feel**, above the sections. It sets **Acceleration**, **Air Control**, **Falls Faster**, **Hang At Top** and **Fastest Fall** from the **Walk Speed**, so set the speed first.
4. To switch an ability off, untick **Can Dash** under **Dash**, **Can Wall Slide** or **Can Climb** under **Walls And Ledges**, or **Can Crouch** under **Crouch**.
5. The ↺ arrow beside a changed setting puts the template's value back. Only this player is changed, until you press **Use These For Every Copy**. **Restore Defaults**, beside it, puts the whole template back as it came with NodeWorthy, keeping its name, and Undo brings your changes back.
6. Press **F6** to try it.

Give it your own model

1. Right-click **Player** and choose **NodeWorthy** → **Looks** → **Set Model...**, then pick a model file anywhere on your computer. It can be a .glb, a .gltf or an .fbx file, and it is copied into a folder of its own in your project's models folder, with the textures its file names. A model with no animations of its own is given them afterwards with **NodeWorthy** → **Looks** → **Use Animations From... (Experimental)**.
2. A window asks which way the model faces. Pick the line that begins **Towards you** or the one that begins **Away from you**, and press **Use It**. When the file holds a whole cast, as a character pack's one file does, the same window asks which character, and the extras it wears, such as hair or a hat. Your model takes the place of the animated person, sized to the hero's height. One undo brings the person back.
3. Each state that played one of the person's clips now plays one of your model's, guessed from the names of its clips. **On The Ground** plays a standing clip and a walking or running one while it moves, **Jumping** a jump, **In The Air** a fall, **Landing** a landing, **Holding A Wall** and **Sliding Down** a wall clip, **Climbing** and **Reaching For A Ledge** a climb, **Dashing** and **Dashing Ahead** a dash or a roll and **Crouching** a crouch. Where the model has no clip for a move the nearest one stands in, such as its fall for holding a wall.
4. Click **Player**. A new **Animations** section in the Inspector lists each guess, as **Idle Clip**, **Walk Clip**, **Jump Clip**, **Fall Clip**, **Crouch Clip**, **Crouch Walk Clip**, **Land Clip**, **Wall Clip**, **Climb Clip** and **Dash Clip**. Where a guess is wrong, pick another clip from its list.
5. **Blend Time** in the same section is how long one clip takes to take over from the last. It starts at a tenth of a second, as quick as the hero came, and 0 snaps.

Give it health and a gun

1. Right-click **Player** in the Scene dock and choose **NodeWorthy** → **Give It** → **Health**. It gets a **HurtBox**, a health of 3, and a **Dead** state in its template that waits a second and starts the level again. Its health goes on the screen in a **Stat Panel**.
2. Click **Player**. A **Health** section has joined its settings in the Inspector, with **Max Health**, and **Invincibility Time** for how long it is safe after a hit.
3. For a gun, right-click **Player** again and choose **NodeWorthy** → **Give It** → **Gun**. A second Machine called **Gun** and a marker called **Muzzle** are added under **Player**, and the bullet is a small scene saved in your project's bullets folder.
4. Run the game and press **F**, or the left face button on a gamepad. A bullet flies the way the hero faces and takes 1 health from an **Enemy** it hits.
5. Click **Player** to tune the gun. Its settings are under **Shooting**, below the heading **Gun**, and they are **Time Between Shots**, **Bullet Speed**, **Damage** and **Bullet Range Time**.

Change the key that crouches

1. Click **Player** and press **Open Its State Machine...** under its settings in the Inspector. The NodeWorthy panel, where templates are drawn, shows its template.
2. In the graph, double-click the **Playing** box to see the states inside it. A trail of names above the graph shows where you are, and **Up** takes you back out.
3. Click the **On The Ground** box. The **Actions** tab on the right lists what that state does, each action on a card of its own.
4. Open the card for the Input check whose line begins with crouch_held. Beside **Input Action** press ▼ and pick the key you want. It takes the place of **C** and the right face button.
5. To keep a gamepad button as well, press ▼ again, tick **Add the one picked to what is there (either works)** and pick the button.
6. Click the **Crouching** box, open its **Crouch** card and set **Stays While Held** to the same key in the same way. This is the key that keeps the hero down, so the two must match.

Use the same hero in the next level

1. In the first level, click **Player** and press **Use These For Every Copy** under its settings. The numbers you changed now belong to the template, not only to this one player.
2. Open the next level and add the hero again with **NodeWorthy** → **Add Player** → **Platform Hero (3D)**. It arrives with a fresh template of its own, named with a 2 at the end.
3. Right-click the new **Player** and choose **NodeWorthy** → **Behave Like**, then pick your first template, which is named with your first level, such as **Platform Hero (3D), in Level 1**. Both players now run the very same template, so a change made in one level is made in both.
4. If the first player was given health, choose **NodeWorthy** → **Give It** → **Health** on the new one too, so it has a **HurtBox** of its own. The **Dead** state is already in the template and is left as it is.
5. To carry health from one level into the next, open the template with **Open Its State Machine...**, go to the **Variables** tab and tick **Carries Over Between Scenes**.

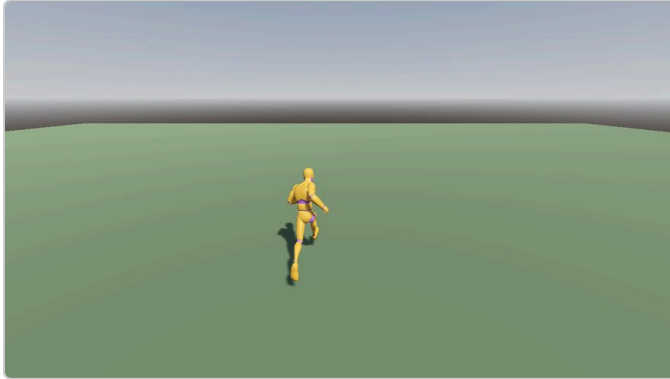
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The mouse pointer vanishes when the game runs.	The camera takes hold of the mouse so that moving it turns the view. Press Escape to get the pointer back, and click in the game to hand it over again.
After Set Model... the model glides about without moving its limbs.	Click Player and look for an Animations section in the Inspector. If there is none, the model file holds no animations. Give it some with NodeWorthy → Looks → Use Animations From... (Experimental) , from any animation file. It is experimental, so watch each move before you rely on it. If there is one, a clip was guessed wrongly from its name, so pick the right clip on each line.
After Set Model... the model is plain white, or a part of it is.	The pictures its file names were not found beside it. Right-click Player and choose NodeWorthy → Looks → Set Model Textures... , and pick each part's picture from the pack's Textures folder.

See also **Third Person Hero (3D)** 37, **Side-On Platform Hero (3D)** 17, **Top-Down Hero (3D)** 54, **Ground Enemy** 74

Third Person Hero (3D) HERO, 3D

A ready-made player for a 3D game seen over the shoulder. The moment it is added it jogs where the camera looks, sprints until it runs out of breath, jumps, crouches and walks up stairs, with a camera that turns with the mouse or the right stick. It comes as an animated person, its clips played as it moves, until you give it a model of your own.



Third Person Hero (3D) as the game runs.

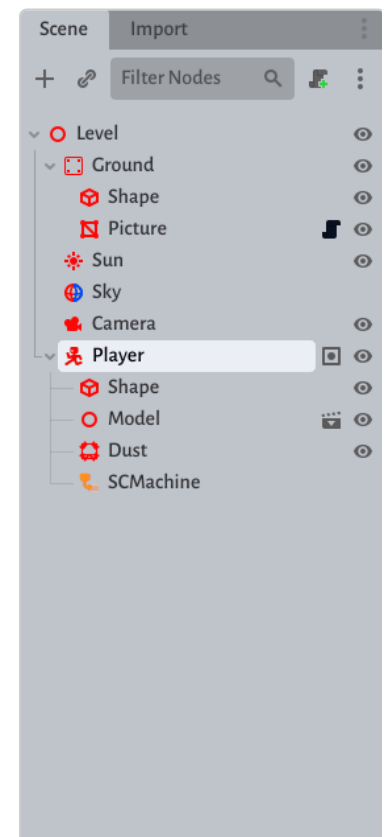
What it is good for

Adventure, exploring and shooting games where the camera sits behind the character and a little to one side. Its movement is weighty, like a real body, not springy like a platformer's. For wall slides, climbing and a dash pick the Platform Hero (3D), and for a camera that looks down from above pick the Top-Down Hero (3D).

Add it

1. Start with a 3D level that has a floor. A new 3D scene set up with **NodeWorthy** → **Start A 3D Level...** has a **Ground**, a **Sun**, a **Sky** and a **Camera**, which is all it needs.
2. In the Scene dock, Godot's list of the nodes in the scene, right-click the top node of the level.
3. Point at **NodeWorthy**, then **Add Player**, and choose **Third Person Hero (3D)**.
4. A node called **Player** appears standing on the **Ground** as an animated person, already picked, and a note at the bottom right of the editor says what was added. The level's **Camera** is handed to it, or a new **Camera** is added behind the hero when the scene has none.
5. Drag **Player** in the 3D view to the spot where the level should begin.
6. Press **F6**, or the **Run Current Scene** button at the top right of Godot, to play.

Try it. Move the mouse and the camera swings round the capsule, looking over its shoulder. Hold **W** and it jogs away from the camera, and hold **Shift** as well and it sprints for about eight seconds before it has to slow down. Press the **Space** bar to jump and hold **C** to crouch. Press **Escape** to get the mouse pointer back.



The Scene dock once it is added.

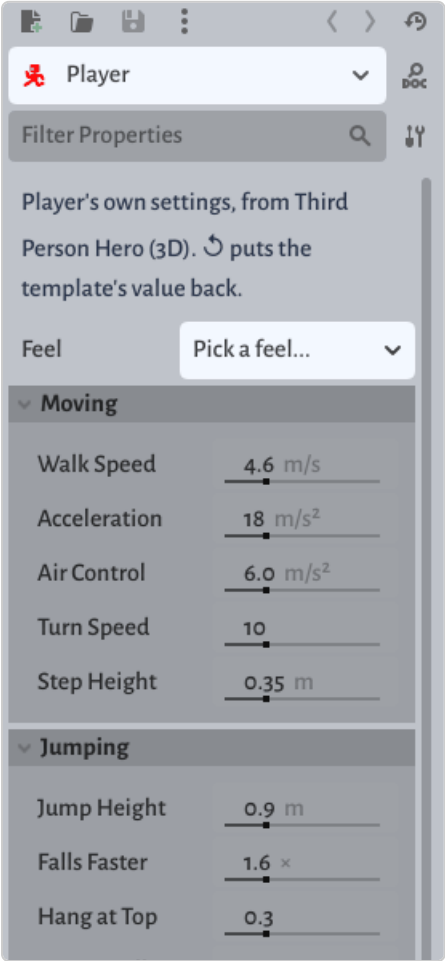
What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Player	CharacterBody3D	The hero's body. A CharacterBody3D is Godot's node for a character that moves and is stopped by floors and walls. It is on the Player collision layer and in the group called player, which is how enemies and the Minimap find it.
Shape	CollisionShape3D	The solid part of the hero, a capsule 1.8 metres tall. A CollisionShape3D is Godot's node that says what bumps into things. Crouching makes this capsule shorter.
Model	Node3D	What you see, an animated person with clips for standing, jogging, sprinting, jumping and crouching. It is a small scene placed under the player, and the AnimationPlayer inside it, Godot's node that plays a model's clips, is what the template's Play Animation actions play.
Dust	CPUParticles3D	A puff of pale dust at its feet, ready for an Emit Particles action of your own, such as on landing. A CPUParticles3D is Godot's node for a burst of small moving specks.
SCMachine	Machine	The Machine, NodeWorthy's node that runs a template. The template is the drawing of states, actions and transitions that makes the hero move, and this one is your own copy, saved in your project's nodeworthy folder, unless you have picked another folder for templates.
Camera	Camera3D	Added at the top of the scene, not under the player, and only when the scene has no Camera3D yet. A Camera3D is Godot's node for the view the screen shows. A new one starts behind and above the hero, looking down at it, and the template swings it round the hero as the mouse moves.
Sun	DirectionalLight3D	Added at the top of the scene only when the scene has neither a DirectionalLight3D nor a WorldEnvironment of its own. A DirectionalLight3D is Godot's node for sunlight that falls the same way everywhere, and this one casts shadows. A level made with Start A 3D Level... already has one.
Sky	WorldEnvironment	Added with the Sun . A WorldEnvironment is Godot's node for the sky and the soft light that comes from it. Without the two, a 3D game runs almost black.
HurtBox	Hurt Box	Added under Player only when the scene already holds an Enemy , or later when you choose Give It → Health . It is what lets the hero be hurt. With it come a health of 3 and a Dead state in the template that starts the level again.
HealthPanel	Stat Panel	Added with the HurtBox , on a CanvasLayer, which is Godot's node for things drawn over the game. It shows the hero's health on the screen. When the scene already has a Stat Panel , health is added to that one and no new panel is made.
GoldPanel	Stat Panel	Added only when something in the scene drops gold, as a Chest does. It shows the gold the hero has picked up. When the scene already has a Stat Panel , gold is added to that one and no new panel is made.

The controls

PRESS	WHAT IT DOES
W or the Up arrow, or the left stick pushed up	Jogs away from the camera.
S or the Down arrow, or the left stick pulled down	Jogs towards the camera.
A and D , the Left and Right arrows, or the left stick pushed sideways	Jog to the left and the right of where the camera looks.
The Space bar, or the bottom button on a gamepad	Jumps, from a jog or a sprint. A tap is a short hop and a held press goes the full Jump Height .
Shift , or pressing the left stick in	Sprints for as long as it is held and there is breath left.
C , or the right face button (B / Circle) on a gamepad	Crouches for as long as it is held.
Moving the mouse, or pushing the right stick	Swings the camera round the hero.
The mouse wheel	Moves the camera nearer to the hero or further from it, between 2 and 12 metres.
Escape	Gives the mouse pointer back while the game runs. A click in the game hands it to the camera again.

Its settings



Click **Player** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot's own settings, in sections that fold.

A change here changes this one only. The arrow that appears beside a changed setting puts the template's value back. **Use These For Every Copy**, under the settings, makes them the template's own, for every copy.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Feel	Pick how it feels to play and the numbers in Moving and Jumping are set for it, from its walk speed. Change any of them after. Tight Quick to get going, quick to stop, heavy on the way down with a sharp moment at the top. Precise and fair, for a game about landing on things. Floaty Slow to build up, a long hang at the top, gentle on the way down. Dreamy and drifting. Heavy Little steering once off the ground, a fast drop and a long moment at the top, so it commits to a jump. Arcade Full speed at once, the same arc up as down, no cap on falling. The old way, with no weight.	Pick a feel...
MOVING		
Walk Speed	How fast the hero jogs, on the ground and while steering in the air. It starts at 4.6 metres a second, an easy run. Raise it for a livelier game, and pick a Feel again afterwards if you use one, because a feel works its numbers out from this speed.	4.6 m/s
Acceleration	How quickly it reaches Walk Speed , and how quickly it stops. At 18 it takes about a quarter of a second, which gives it the weight of a real body. A higher number makes it start and stop sharply.	18 m/s²

SETTING	WHAT IT DOES	STARTS AS
Air Control	How quickly it can change direction while off the ground. At 6 it takes most of a second to get up to speed in the air, so a jump mostly goes where it was aimed. Raise it for a hero that steers freely in mid air.	6 m/s ²
Turn Speed	How quickly the hero swings round to face the way it is going. At 10 it is round in about a fifth of a second. A low number such as 2 gives a slow, heavy turn of about a second.	10
Step Height	The tallest step or kerb the hero walks up without a jump. At 0.35 of a metre it climbs ordinary stairs. Anything taller stops it like a wall, so keep it below Jump Height .	0.35 m
JUMPING		
Jump Height	How high a full jump lifts the hero when the jump key is held. It starts at 0.9 of a metre, half its own height, which clears a low wall or a crate. Letting go early cuts the jump short.	0.9 m
Falls Faster	How much harder it is pulled on the way down than on the way up. At 1.6 a jump rises smoothly and comes down quickly, which reads as weight. At 1 it falls exactly as it rose.	1.6×
Hang at Top	How strongly it is pulled for the moment at the very top of a jump, as a share of the usual pull. At 0.3 the pull is less than a third there, so the hero hangs for an instant. A lower number hangs longer, and 1 gives a plain arc with no hang.	0.3
Fastest Fall	The top speed of a fall, so a long drop never becomes too fast to follow. It starts at 25 metres a second. A lower number gives a gentler drop.	25 m/s
Coyote Time	How long after stepping off an edge the jump key still works. A tenth of a second is too short to notice, but long enough that a press made a touch late still jumps. At 0 the hero can only jump while it is really standing on something.	0.1 s
Jump Buffer	How early a jump can be pressed before landing and still count. At 0.1 a press made a tenth of a second before the feet touch down jumps the moment they do. At 0 an early press is lost.	0.1 s
CROUCH		
Crouch Speed	How fast it moves while crouched. It starts at 1.8 metres a second, well under half its jog. At 0 it crouches on the spot and cannot move.	1.8 m/s
Crouch Height	How tall the hero's solid shape is while crouching. It stands 1.8 metres tall and crouches to 1, so it fits under anything with more room than that. It will not stand up again until there is room overhead.	1 m
Can Crouch	Whether holding the crouch key on the ground crouches the hero. Untick it for a hero that always stands, and the crouch key then does nothing.	on
SPRINT		
Sprint Speed	How fast the hero runs while the sprint key is held. It starts at 7.2 metres a second, a little over one and a half times its jog. Keep it above Walk Speed , or sprinting is no quicker than jogging.	7.2 m/s
Stamina Drain	How fast sprinting uses up breath, out of 100. At 12 a full breath lasts about eight seconds. A higher number shortens the sprint, and 0 lets the hero sprint for ever.	12 /s

SETTING	WHAT IT DOES	STARTS AS
Stamina Recovery	How fast breath comes back once the hero stops sprinting. At 18 it is full again in about six seconds. After running right out, the hero cannot sprint again until 20 is back.	18 /s
CAMERA		
Camera Distance	How far behind the hero the camera starts. At 4 metres the hero fills a good part of the screen. While the game runs the mouse wheel moves the camera nearer or further, between 2 and 12 metres.	4 m
Camera Height	How far above the hero's feet the camera looks. At 1.5 metres it looks at about head height on a hero 1.8 metres tall. Lower it for a small character, so the camera does not look over its head.	1.5 m
Over the Shoulder	How far to one side the view slides, so the hero stands to the side of the picture and the player can see past it. At 0.6 of a metre it looks over the right shoulder. At 0 the camera sits straight behind, and a number below 0 puts it over the left shoulder.	0.6 m
Mouse Sensitivity	How far the view turns as the mouse moves. It starts at 0.25, a quarter of a degree for each pixel the mouse travels. A higher number turns faster, which suits a fast game, and a lower one gives a steadier aim.	0.25

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.

Make it your own

Make it faster or move the camera

1. Click **Player** in the Scene dock. Its settings show at the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, in sections that fold, called **Moving**, **Jumping**, **Crouch**, **Sprint** and **Camera**.
2. Open **Moving** and raise **Walk Speed**, or open **Sprint** and raise **Sprint Speed**. Drag the number, or click it and type one.
3. Open **Camera** and change **Camera Distance**, **Camera Height** and **Over The Shoulder** to frame the hero as you like.
4. To change the whole feel at once, pick **Tight**, **Floaty**, **Heavy** or **Arcade** beside **Feel**, above the sections. It sets **Acceleration**, **Air Control**, **Falls Faster**, **Hang At Top** and **Fastest Fall** from the **Walk Speed**.
5. The ↺ arrow beside a changed setting puts the template's value back. Press **F6** to try it.

Give it your own model

1. Right-click **Player** and choose **NodeWorthy** → **Looks** → **Set Model...**, then pick a model file anywhere on your computer. It can be a .glb, a .gltf or an .fbx file, and it is copied into a folder of its own in your project's models folder, with the textures its file names. A model with no animations of its own is given them afterwards with **NodeWorthy** → **Looks** → **Use Animations From... (Experimental)**.
2. A window asks which way the model faces. Pick the line that begins **Towards you** or the one that begins **Away from you**, and press **Use It**. When the file holds a whole cast, as a character pack's one file does, the same window asks which character, and the extras it wears, such as hair or a hat. The model takes the place of the capsule, sized to the hero's height. One undo brings the capsule back.
3. Each state of the template is given a **Play Animation** action with the clip that suits it, guessed from the names of the model's clips. **On The Ground** plays a standing clip and a walking or jogging one while it moves, **Sprinting** a run, **Jumping** a jump, **In The Air** a fall and **Crouching** a crouch.
4. Click **Player**. A new **Animations** section in the Inspector lists each guess, as **Idle Clip**, **Walk Clip**, **Run Clip**, **Jump Clip**, **Fall Clip**, **Crouch Clip** and **Crouch Walk Clip**. Where a guess is wrong, pick another clip from its list.
5. **Blend Time** in the same section is how long one clip takes to take over from the last. A fifth of a second looks natural, and 0 snaps.

Give it health and a gun

1. Right-click **Player** in the Scene dock and choose **NodeWorthy** → **Give It** → **Health**. It gets a **HurtBox**, a health of 3, and a **Dead** state in its template that waits a second and starts the level again. Its health goes on the screen in a **Stat Panel**.
2. Click **Player**. A **Health** section has joined its settings in the Inspector, with **Max Health**, and **Invincibility Time** for how long it is safe after a hit.
3. For a gun, right-click **Player** again and choose **NodeWorthy** → **Give It** → **Gun**. A second Machine called **Gun** and a marker called **Muzzle** are added under **Player**, and the bullet is a small scene saved in your project's bullets folder.
4. Run the game and press **F**, or the left face button on a gamepad. A bullet flies the way the hero faces and takes 1 health from an **Enemy** it hits.
5. Click **Player** to tune the gun. Its settings are under **Shooting**, below the heading **Gun**, and they are **Time Between Shots**, **Bullet Speed**, **Damage** and **Bullet Range Time**.

Change the key that sprints

1. Click **Player** and press **Open Its State Machine...** under its settings in the Inspector. The NodeWorthy panel, where templates are drawn, shows its template.
2. In the graph, double-click the **Playing** box to see the states inside it. A trail of names above the graph shows where you are, and **Up** takes you back out.
3. Click the **On The Ground** box. The **Actions** tab on the right lists what that state does, each action on a card of its own.
4. Open the card for the Input check whose line begins with `sprint_held`. Beside **Input Action** press ▼ and pick the key you want. It takes the place of **Shift** and the press of the left stick.
5. To keep a gamepad button as well, press ▼ again, tick **Add the one picked to what is there (either works)** and pick the button.
6. Click the **Sprinting** box and set the same key on the card for the Input check whose line ends with **Eased**. That check ends the sprint when the key is let go, so the two must match.

Use the same hero in the next level

1. In the first level, click **Player** and press **Use These For Every Copy** under its settings. The numbers you changed now belong to the template, not only to this one player.
2. Open the next level and add the hero again with **NodeWorthy** → **Add Player** → **Third Person Hero (3D)**. It arrives with a fresh template of its own, named with a 2 at the end.
3. Right-click the new **Player** and choose **NodeWorthy** → **Behave Like**, then pick your first template, which is named with your first level, such as **Third Person Hero (3D), in Level 1**. Both players now run the very same template, so a change made in one level is made in both.
4. If the first player was given health, choose **NodeWorthy** → **Give It** → **Health** on the new one too, so it has a **HurtBox** of its own. The **Dead** state is already in the template and is left as it is.
5. To carry health and gold from one level into the next, open the template with **Open Its State Machine...**, go to the **Variables** tab and tick **Carries Over Between Scenes**.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The mouse pointer vanishes when the game runs.	The camera takes hold of the mouse so that moving it turns the view. Press Escape to get the pointer back, and click in the game to hand it over again.
After Set Model... the model glides about without moving its limbs.	Click Player and look for an Animations section in the Inspector. If there is none, the model file holds no animations. Give it some with NodeWorthy → Looks → Use Animations From... (Experimental) , from any animation file. It is experimental, so watch each move before you rely on it. If there is one, a clip was guessed wrongly from its name, so pick the right clip on each line.
After Set Model... the model is plain white, or a part of it is.	The pictures its file names were not found beside it. Right-click Player and choose NodeWorthy → Looks → Set Model Textures... , and pick each part's picture from the pack's Textures folder.
The hero drops out of sight as soon as the game starts.	Nothing solid is under it, so it falls for ever. Add a floor with NodeWorthy → Add Thing → Ground and move Player above it.

See also [Platform Hero \(3D\)](#) 27, [Top-Down Hero \(3D\)](#) 54, [Ground](#) 62, [Enemy](#) 74, [Crosshair](#) 162

Top-Down Hero (2D) HERO, 2D

A ready-made player for a 2D game seen from above. The moment it is added it walks all eight ways, turns to face the way it goes and dashes, with a camera that looks a little ahead of it. Nothing pulls it down, so it needs no floor, and it comes as a little character from the free art that comes with NodeWorthy, who hops a little with each step and breathes while it stands.



Top-Down Hero (2D) as the game runs.

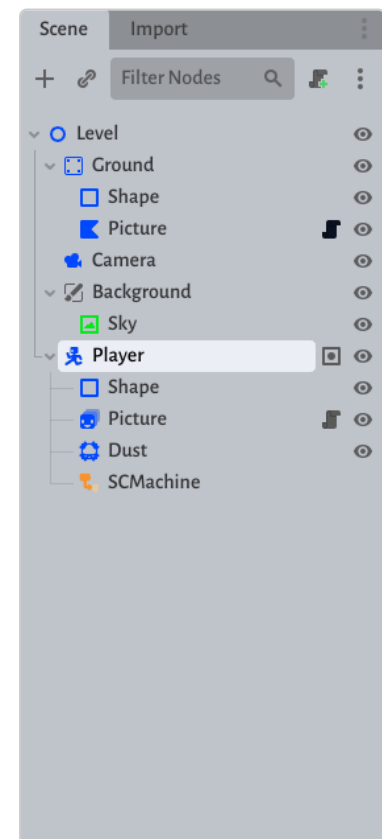
What it is good for

Adventure games, dungeon crawlers, arena games and any 2D game where the player looks down on the world and walls, not gravity, shape the level. Put your own states beside **Walking** for whatever else it does. For a 2D game seen from the side pick the Platform Hero (2D), and for the same kind of game in 3D pick the Top-Down Hero (3D).

Add it

1. Start with a 2D scene. A new scene made with **Scene** → **New Scene** and **2D Scene** is enough, because seen from above nothing falls and the hero needs no floor.
2. In the Scene dock, Godot's list of the nodes in the scene, right-click the top node of the level.
3. Point at **NodeWorthy**, then **Add Player**, and choose **Top-Down Hero (2D)**.
4. A node called **Player** appears at the spot of the node you right-clicked, already picked, and a **Camera** is added at the top of the scene when the scene has none.
5. Add this hero before any **Enemy**. Things are built for a game seen from above only once this hero is in the scene.
6. Press **F6**, or the **Run Current Scene** button at the top right of Godot, to play.

Try it. Hold **W**, **A**, **S** or **D** and the hero walks that way, facing where it goes and hopping a little with each step, with the camera looking a little ahead. Press the **Space** bar and it dashes with a squash and a puff of dust. It stays tinted blue for a moment, until the dash has come back.



The Scene dock once it is added.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Player	CharacterBody2D	The hero's body. A CharacterBody2D is Godot's node for a character that moves and is stopped by walls. This one is set to float, so nothing pulls it down and it slides along walls in any direction. It is on the Player collision layer and in the group called player, which is how enemies and the Minimap find it.
Shape	CollisionShape2D	The solid part of the hero, a square 28 pixels across. A CollisionShape2D is Godot's node that says what bumps into things. It is square so that turning never swings a corner into a wall.
Picture	AnimatedSprite2D	What you see, a hero in a tunic, from the free art that comes with NodeWorthy. It stays upright while the body turns to face the way it walks, and is mirrored when the hero walks left. An AnimatedSprite2D is Godot's node for a picture that steps through frames. The template tints and squashes this node.
Dust	CPUParticles2D	A puff of pale dust all round its feet, started by Dashing . A CPUParticles2D is Godot's node for a burst of small moving specks.
SCMachine	Machine	The Machine, NodeWorthy's node that runs a template. The template is the drawing of states, actions and transitions that makes the hero move, and this one is your own copy, saved in your project's nodeworthy folder, unless you have picked another folder for templates.
Camera	Camera2D	Added at the top of the scene, not under the player, and only when the scene has no Camera2D yet. A Camera2D is Godot's node for the part of a 2D level the screen shows. The template moves it to follow the hero.
HurtBox	Hurt Box	Added under Player only when the scene already holds an Enemy , or later when you choose Give It → Health . It is what lets the hero be hurt. With it come a health of 3 and a Dead state in the template that starts the level again.
HealthPanel	Stat Panel	Added with the HurtBox , on a CanvasLayer, which is Godot's node for things drawn over the game. It shows the hero's health on the screen. When the scene already has a Stat Panel , health is added to that one and no new panel is made.
GoldPanel	Stat Panel	Added only when something in the scene drops gold, as a Chest does. It shows the gold the hero has picked up. When the scene already has a Stat Panel , gold is added to that one and no new panel is made.

The controls

PRESS	WHAT IT DOES
W or the Up arrow, or the left stick pushed up	Walks up the screen.
S or the Down arrow, or the left stick pulled down	Walks down the screen.

PRESS	WHAT IT DOES
A or the Left arrow, or the left stick pushed left	Walks left.
D or the Right arrow, or the left stick pushed right	Walks right. Two keys held together walk on the diagonal.
Shift , or the right bumper	Dashes the way the keys or the stick point, or the way the hero faces when nothing is held.

Its settings



Click **Player** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot's own settings, in sections that fold.

A change here changes this one only. The arrow that appears beside a changed setting puts the template's value back. **Use These For Every Copy**, under the settings, makes them the template's own, for every copy.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
MOVING		
Walk Speed	How fast the hero walks. It starts at 220 pixels a second, which crosses about eight times its own width every second. Raise it for a faster game, or lower it for slow and careful exploring.	220 px/s
Acceleration	How quickly it reaches Walk Speed , and how quickly it stops. At 2400 it is at full speed in under a tenth of a second, which feels sharp. A low number makes it slide to a stop as if on ice.	2400 px/s²
DASH		

SETTING	WHAT IT DOES	STARTS AS
Dash Distance	How far one dash carries the hero. It starts at 110 pixels, about four times its own width. Raise it and the hero leaps further, so check that walls still hold it in.	110 px
Dash Time	How long the dash takes to cover Dash Distance . At 0.15 of a second it is a sharp burst. A longer time turns it into a glide, and a shorter one into a blink.	0.15 s
Dash Cooldown	How long the hero must wait after a dash before it can dash again. At 0.4 of a second the wait is short, and the hero is tinted blue while it lasts. Raise it to make each dash a choice the player has to think about.	0.4 s
Can Dash	Whether the hero can dash at all. Untick it for a hero that only walks, and the hero no longer dashes when the dash key is pressed.	on
LOOK		
Tint	A colour laid over its picture, so one set of art makes a skeleton, a ghost and a boss. White shows the picture as drawn. It is kept on its Picture node.	white

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.

Top-Down Hero (zD) Built-in Templates Top-Down Hero (zD) Playing Up

These are Player's rules, the template Top-Down Hero (zD). Each box is a state it can be in, and each arrow is what moves it on. Click a box to see what it does there.

Actions Transitions Variables Events Globals Theme

Playing

Actions Add Action...

- 1. Get Input Vector
way = the direction held
- Left Action Key: A or Key: Left or Pad: Left
- Right Action Key: D or Key: Right or Pad: Le
- Up Action Key: W or Key: Up or Pad: Left
- Down Action Key: S or Key: Down or Pad: Le
- Store In way - Vector2
- Store Strength In None
- Keep Last Direction On Var
- Run Mode Every Frame
- Update Interval 0.0
- Wait for Previous On
- Disabled On
- Display Name

- 2. Get Input Vector
facing = the direction held, kept when let go
- 3. Expression
dash_charge = minf(dash_charge + delta / maxf(dash_cooldown, 0.0), 1.0)
- 4. Expression
tint = Color(1, 1, 1).lerp(Color(0.55, 0.8, 1.25), float(dash_charge < 1.0))
- 5. Set Property
Set Picture's Self Module to tint
- 6. Expression
look_ahead = facing.normalized() * 48.0
- 7. Camera Follow
Follow Self

Player added under Level, running Third Person Hero (3D), using the camera already in the scene, Camera. Open the NodeWorthy panel to see how it works.

Inside **Playing**, which you open with a double-click on its box. The states that move it are in here.

STATE	WHAT HAPPENS IN IT
Playing	The state the hero is always in, with the other two nested inside it. It reads the keys and the stick each frame, fills the dash back up over Dash Cooldown , tints the picture blue until the dash is back, over any Tint chosen for it under Look , and keeps the camera a little ahead of the way the hero faces. It has no way out until the hero is given health, when DIED leads to Dead .
Walking	The hero walks at Walk Speed whichever way is held and turns to face that way. The dash key leads to Dashing . This is the state to put your own states beside, such as one for swinging a sword.
Dashing	When the dash has come back, it sends the hero Dash Distance in Dash Time the way that is held, or the way it faces, with a squash of the picture and a puff of dust, then goes back to Walking . When the dash is not ready, or Can Dash is off, it goes straight back to Walking .

Make it your own

Make it faster or switch the dash off

1. Click **Player** in the Scene dock. Its settings show at the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, in sections that fold, called **Moving**, **Dash** and **Look**.
2. Open **Moving** and raise **Walk Speed**. Drag the number, or click it and type one.
3. Open **Dash** and change **Dash Distance** or **Dash Cooldown**, or untick **Can Dash** to switch the dash off.
4. The ↺ arrow beside a changed setting puts the template's value back. Only this player is changed, until you press **Use These For Every Copy**. **Restore Defaults**, beside it, puts the whole template back as it came with NodeWorthy, keeping its name, and Undo brings your changes back.
5. Press **F6** to try it.

Give it your own picture or sprite sheet

1. For one still picture, right-click **Player** and choose **NodeWorthy** → **Looks** → **Set Texture...**, then pick a picture anywhere on your computer. It is copied into your project's pictures folder and sized to the height of the hero's **Shape**.
2. Say which way the picture is drawn facing with **Drawn facing**, at the bottom of the window. The whole body turns to face the way it walks and the picture turns with it, so a picture drawn from straight above suits it best.
3. For animations, choose **NodeWorthy** → **Looks** → **Import Sprite Sheet...** instead and pick a sheet, which is one picture holding the poses in rows. Set **Across** and **Down**, name the rows idle and walk, and add a row called dash if you have one, then press **Use It**. **Walking** then plays idle while the hero stands and walk while it moves.
4. For art drawn from several sides, pick one sheet for each animation, all at once. In the window choose **Each row is one animation, facing another way**, give the **Animation** its name and say which way each row faces. The hero then shows the side that faces the way it goes.
5. One undo puts the old look back.

Give it health and a gun

1. Right-click **Player** in the Scene dock and choose **NodeWorthy** → **Give It** → **Health**. It gets a **HurtBox**, a health of 3, and a **Dead** state in its template that waits a second and starts the level again. Its health goes on the screen in a **Stat Panel**.
2. Click **Player**. A **Health** section has joined its settings in the Inspector, with **Max Health**, and **Invincibility Time** for how long it is safe after a hit.
3. For a gun, right-click **Player** again and choose **NodeWorthy** → **Give It** → **Gun**. A second Machine called **Gun** and a marker called **Muzzle** are added under **Player**, and the bullet is a small scene saved in your project's bullets folder.
4. Run the game and press **F**, or the left face button on a gamepad. A bullet flies the way the hero faces and takes 1 health from an **Enemy** it hits.
5. Click **Player** to tune the gun. Its settings are under **Shooting**, below the heading **Gun**, and they are **Time Between Shots**, **Bullet Speed**, **Damage** and **Bullet Range Time**.

Change the key that dashes

1. Click **Player** and press **Open Its State Machine...** under its settings in the Inspector. The NodeWorthy panel, where templates are drawn, shows its template.
2. In the graph, double-click the **Playing** box to see the states inside it. A trail of names above the graph shows where you are, and **Up** takes you back out.
3. Click the **Walking** box. The **Actions** tab on the right lists what that state does, each action on a card of its own.
4. Open the card for the Input check whose line ends with → **DashPressed**. Beside **Input Action** press ▼ and pick the key you want. It takes the place of the **Space** bar, **Shift** and the bottom button.
5. To keep a gamepad button as well, press ▼ again, tick **Add the one picked to what is there (either works)** and pick the button. This is the only card that asks for the dash key.

Use the same hero in the next level

1. In the first level, click **Player** and press **Use These For Every Copy** under its settings. The numbers you changed now belong to the template, not only to this one player.
2. Open the next level and add the hero again with **NodeWorthy** → **Add Player** → **Top-Down Hero (2D)**. It arrives with a fresh template of its own, named with a 2 at the end.
3. Right-click the new **Player** and choose **NodeWorthy** → **Behave Like**, then pick your first template, which is named with your first level, such as **Top-Down Hero (2D), in Level 1**. Both players now run the very same template, so a change made in one level is made in both.
4. If the first player was given health, choose **NodeWorthy** → **Give It** → **Health** on the new one too, so it has a **HurtBox** of its own. The **Dead** state is already in the template and is left as it is.
5. To carry health and gold from one level into the next, open the template with **Open Its State Machine...**, go to the **Variables** tab and tick **Carries Over Between Scenes**.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
An Enemy added before the hero drops off the bottom of the screen when the game runs.	It was built for a game seen from the side, where things fall. Things are built for a game seen from above only once the Top-Down Hero is in the scene, and NodeWorthy says which ones were built before it as the hero is added. Delete that Enemy and add it again.
Your own picture comes out very small.	A picture is sized to the height of the hero's Shape , which is 28 pixels. Click Picture in the Scene dock and make it bigger with Scale in the Inspector, and make Shape bigger to match so that what is solid is what you see. For a sprite sheet, choose As big as it is drawn in the window that cuts it.
Health and gold start again from nothing in the next level.	The player in the next level runs a different copy of the template, or the template has no tick in Carries Over Between Scenes . The Output panel at the bottom of Godot names which. Give the new player the first template with NodeWorthy → Behave Like , and tick the box at the top of the template's Variables tab.

See also [Top-Down Hero \(3D\)](#) 54, [Platform Hero \(2D\)](#) 8, [Enemy](#) 74, [Chest](#) 104, [Minimap](#) 155

Top-Down Hero (3D) HERO, 3D

A ready-made player for a 3D game seen from above. The moment it is added it walks all eight ways, snaps round to face the way it goes, walks up steps, dashes, and falls off edges to land with a puff of dust, while the camera looks down on it from above and behind. It comes as an animated person.



Top-Down Hero (3D) as the game runs.

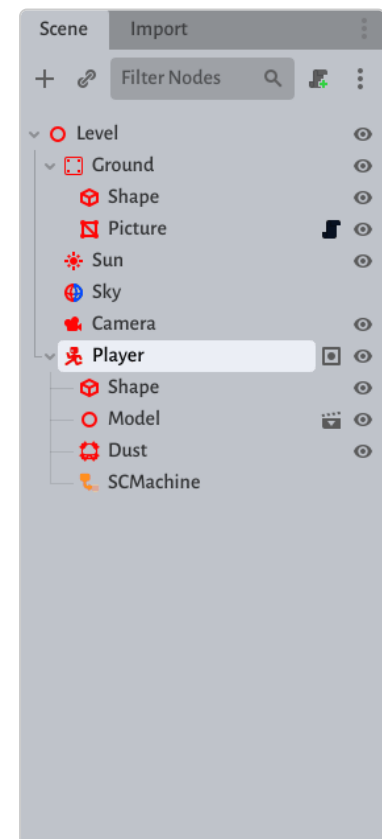
What it is good for

Action games and dungeon crawlers in 3D where the camera looks down on the player from a fixed angle. It has no jump, so levels are made of floors, steps and drops. For a camera the player turns with the mouse pick the Third Person Hero (3D) or the Platform Hero (3D), and for the same kind of game in 2D pick the Top-Down Hero (2D).

Add it

1. Start with a 3D level that has a floor. A new 3D scene set up with **NodeWorthy** → **Start A 3D Level...** has a **Ground**, a **Sun**, a **Sky** and a **Camera**, which is all it needs.
2. In the Scene dock, Godot's list of the nodes in the scene, right-click the top node of the level.
3. Point at **NodeWorthy**, then **Add Player**, and choose **Top-Down Hero (3D)**.
4. A node called **Player** appears standing on the **Ground**, already picked, and a note at the bottom right of the editor says what was added. The level's **Camera** is handed to it, or a new **Camera** is added above and behind the hero when the scene has none.
5. Drag **Player** in the 3D view to the spot where the level should begin.
6. Press **F6**, or the **Run Current Scene** button at the top right of Godot, to play.

Try it. Hold **W**, **A**, **S** or **D** and the hero runs that way across the floor, turning at once to face where it goes, while the camera looks down on it. Press the **Space** bar and it rolls forward in a dash with a puff of dust, then glows blue for a moment until the dash has come back.



The Scene dock once it is added.

What you get

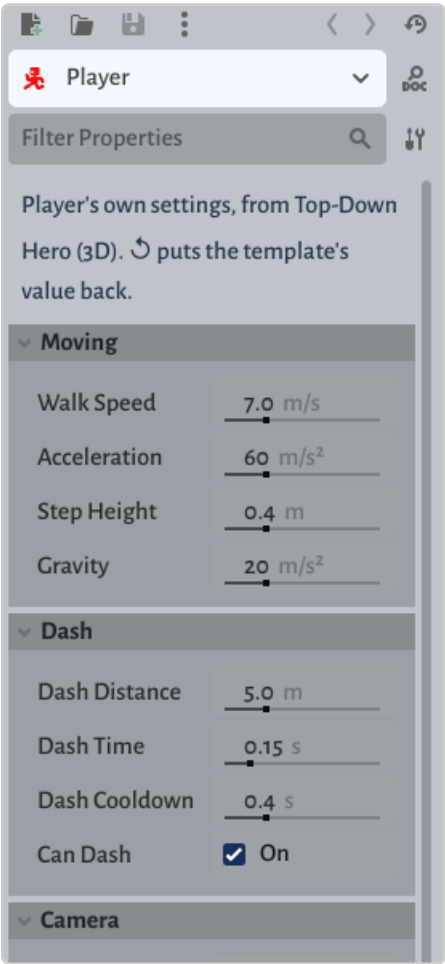
IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Player	CharacterBody3D	The hero's body. A CharacterBody3D is Godot's node for a character that moves and is stopped by floors and walls. It is on the Player collision layer and in the group called player, which is how enemies and the Minimap find it.
Shape	CollisionShape3D	The solid part of the hero, a capsule 1.8 metres tall. A CollisionShape3D is Godot's node that says what bumps into things.
Model	Node3D	What you see, an animated person with clips for standing, sprinting, rolling, falling and landing. It is a small scene placed under the player, and the AnimationPlayer inside it, Godot's node that plays a model's clips, is what the template's Play Animation actions play.
Dust	CPUParticles3D	A puff of pale dust at its feet, started by Dashing and Landing . A CPUParticles3D is Godot's node for a burst of small moving specks.
SCMachine	Machine	The Machine, NodeWorthy's node that runs a template. The template is the drawing of states, actions and transitions that makes the hero move, and this one is your own copy, saved in your project's nodeworthy folder, unless you have picked another folder for templates.
Camera	Camera3D	Added at the top of the scene, not under the player, and only when the scene has no Camera3D yet. A Camera3D is Godot's node for the view the screen shows. A new one starts 12 metres above the hero and 9 metres behind it, looking down, and the template keeps it there as the hero moves.
Sun	DirectionalLight3D	Added at the top of the scene only when the scene has neither a DirectionalLight3D nor a WorldEnvironment of its own. A DirectionalLight3D is Godot's node for sunlight that falls the same way everywhere, and this one casts shadows. A level made with Start A 3D Level... already has one.
Sky	WorldEnvironment	Added with the Sun . A WorldEnvironment is Godot's node for the sky and the soft light that comes from it. Without the two, a 3D game runs almost black.
HurtBox	Hurt Box	Added under Player only when the scene already holds an Enemy , or later when you choose Give It → Health . It is what lets the hero be hurt. With it come a health of 3 and a Dead state in the template that starts the level again.
HealthPanel	Stat Panel	Added with the HurtBox , on a CanvasLayer, which is Godot's node for things drawn over the game. It shows the hero's health on the screen. When the scene already has a Stat Panel , health is added to that one and no new panel is made.

The controls

PRESS	WHAT IT DOES
W or the Up arrow, or the left stick pushed up	Walks up the screen, away from the camera.

PRESS	WHAT IT DOES
S or the Down arrow, or the left stick pulled down	Walks down the screen, towards the camera.
A or the Left arrow, or the left stick pushed left	Walks left.
D or the Right arrow, or the left stick pushed right	Walks right. Two keys held together walk on the diagonal.
The Space bar, or the bottom button on a gamepad	Jumps, playing its jump clip. It comes down through Falling and Landing , as from an edge.
Shift , or the right bumper	Dashes the way the keys or the stick point, or the way the hero faces when nothing is held.

Its settings



Click **Player** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot’s own settings, in sections that fold.

A change here changes this one only. The arrow that appears beside a changed setting puts the template’s value back. **Use These For Every Copy**, under the settings, makes them the template’s own, for every copy.

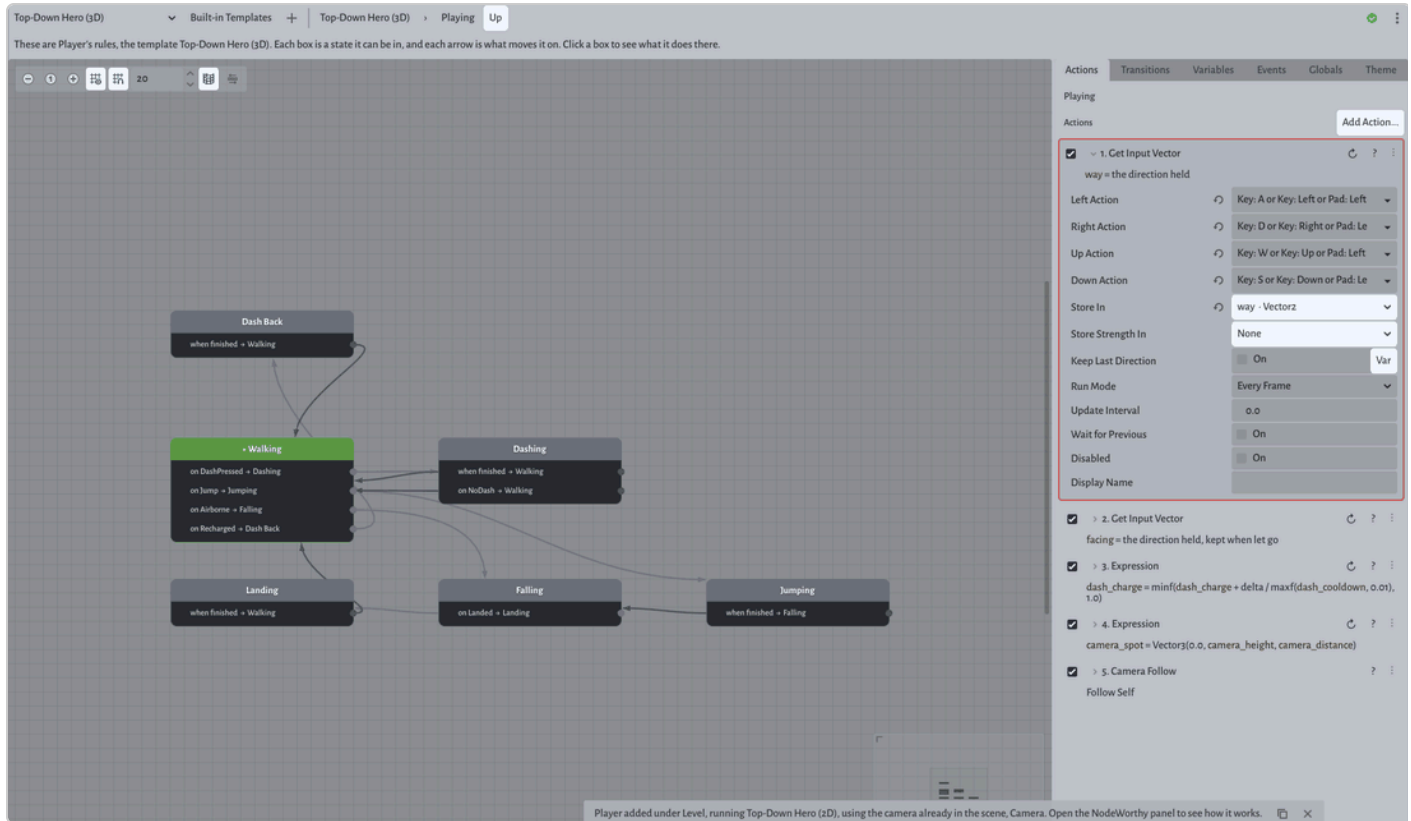
Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
MOVING		
Walk Speed	How fast the hero walks. It starts at 7 metres a second, a brisk run that suits a game of quick fights. Raise it for a faster game, or lower it for slow and careful exploring.	7 m/s

SETTING	WHAT IT DOES	STARTS AS
Acceleration	How quickly it reaches Walk Speed , and how quickly it stops. At 60 it is at full speed in about an eighth of a second, which feels sharp. A low number makes it slide to a stop as if on ice.	60 m/s ²
Step Height	The tallest step the hero walks up. At 0.4 of a metre it climbs stairs and kerbs. It has no jump, so anything taller is a wall to it.	0.4 m
Gravity	How hard the hero is pulled down when it walks off an edge. It starts at 19.6, twice the real 9.8, so a drop is over quickly. A lower number gives a slow, drifting fall.	19.6 m/s ²
DASH		
Dash Distance	How far one dash carries the hero. It starts at 5 metres, nearly three times its own height. Raise it and the hero leaps further, so check that walls still hold it in.	5 m
Dash Time	How long the dash takes to cover Dash Distance . At 0.15 of a second it is a sharp burst. A longer time turns it into a glide, and a shorter one into a blink.	0.15 s
Dash Cooldown	How long the hero must wait after a dash before it can dash again. At 0.4 of a second the wait is short, and the hero glows blue while it lasts. Raise it to make each dash a choice the player has to think about.	0.4 s
Can Dash	Whether the hero can dash at all. Untick it for a hero that only walks, and the hero no longer dashes when the dash key is pressed.	on
CAMERA		
Camera Height	How high above the hero the camera sits. It starts at 12 metres. Together with Camera Distance it sets the angle, so raising this alone looks down more steeply and shows less of what lies ahead.	12 m
Camera Distance	How far behind the hero the camera sits. It starts at 9 metres, which with a height of 12 looks down at a little over 50 degrees. A higher number gives a flatter view that shows more of the level ahead, and a lower one looks down more steeply.	9 m

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.



Inside **Playing**, which you open with a double-click on its box. The states that move it are in here.

STATE	WHAT HAPPENS IN IT
Playing	The state the hero is always in, with every other state nested inside it. It reads the keys and the stick each frame, fills the dash back up over Dash Cooldown , and keeps the camera above and behind the hero, looking down at it. It has no way out until the hero is given health, when DIED leads to Dead .
Walking	The hero walks at Walk Speed whichever way is held, snaps round to face that way and walks up steps. The jump key leads to Jumping , the dash key to Dashing , walking off an edge leads to Falling , and the dash coming back while the hero still glows leads to Dash Back .
Dash Back	Turns the blue glow off once the dash has come back, then goes straight back to Walking .
Dashing	When the dash has come back, it turns the model's glow blue and sends the hero Dash Distance in Dash Time the way that is held, or the way it faces, with a roll and a puff of dust, then goes back to Walking . When the dash is not ready, or Can Dash is off, it goes straight back to Walking .
Jumping	Leaps 1.6 metres up as Space is pressed, a tap going less high than a held press, and plays its jump clip, then hands over to Falling .
Falling	The hero has walked off an edge or jumped. It falls under Gravity , still steered by the keys, until it touches a floor, which leads to Landing .
Landing	A squash of the model and a puff of dust as the hero touches down, then straight on to Walking .

Make it your own

Make it faster, move the camera, or switch the dash off

1. Click **Player** in the Scene dock. Its settings show at the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, in sections that fold, called **Moving**, **Dash** and **Camera**.
2. Open **Moving** and raise **Walk Speed**. Drag the number, or click it and type one.
3. Open **Camera** and change **Camera Height** and **Camera Distance**. Raise both by the same share to pull the camera back without changing its angle.
4. Open **Dash** and change **Dash Distance** or **Dash Cooldown**, or untick **Can Dash** to switch the dash off.
5. The ↺ arrow beside a changed setting puts the template's value back. Press **F6** to try it.

Give it your own model

1. Right-click **Player** and choose **NodeWorthy** → **Looks** → **Set Model...**, then pick a model file anywhere on your computer. It can be a .glb, a .glTF or an .fbx file, and it is copied into a folder of its own in your project's models folder, with the textures its file names. A model with no animations of its own is given them afterwards with **NodeWorthy** → **Looks** → **Use Animations From... (Experimental)**.
2. A window asks which way the model faces. Pick the line that begins **Towards you** or the one that begins **Away from you**, and press **Use It**. When the file holds a whole cast, as a character pack's one file does, the same window asks which character, and the extras it wears, such as hair or a hat. Your model takes the place of the animated person, sized to the hero's height. One undo brings the person back.
3. Each state that played one of the person's clips now plays one of your model's, guessed from the names of its clips. **Walking** plays a standing clip and a walking or running one while it moves, **Dashing** a dash or a roll, **Falling** a fall and **Landing** a landing. Where the model has no clip for a move the nearest one stands in, such as its run for a dash.
4. Click **Player**. A new **Animations** section in the Inspector lists each guess, as **Idle Clip**, **Walk Clip**, **Fall Clip**, **Land Clip** and **Dash Clip**. Where a guess is wrong, pick another clip from its list.
5. **Blend Time** in the same section is how long one clip takes to take over from the last. It starts at a tenth of a second, as quick as the hero came, and 0 snaps.

Give it health and a gun

1. Right-click **Player** in the Scene dock and choose **NodeWorthy** → **Give It** → **Health**. It gets a **HurtBox**, a health of 3, and a **Dead** state in its template that waits a second and starts the level again. Its health goes on the screen in a **Stat Panel**.
2. Click **Player**. A **Health** section has joined its settings in the Inspector, with **Max Health**, and **Invincibility Time** for how long it is safe after a hit.
3. For a gun, right-click **Player** again and choose **NodeWorthy** → **Give It** → **Gun**. A second Machine called **Gun** and a marker called **Muzzle** are added under **Player**, and the bullet is a small scene saved in your project's bullets folder.
4. Run the game and press **F**, or the left face button on a gamepad. A bullet flies the way the hero faces and takes 1 health from an **Enemy** it hits.
5. Click **Player** to tune the gun. Its settings are under **Shooting**, below the heading **Gun**, and they are **Time Between Shots**, **Bullet Speed**, **Damage** and **Bullet Range Time**.

Change the key that dashes

1. Click **Player** and press **Open Its State Machine...** under its settings in the Inspector. The NodeWorthy panel, where templates are drawn, shows its template.
2. In the graph, double-click the **Playing** box to see the states inside it. A trail of names above the graph shows where you are, and **Up** takes you back out.
3. Click the **Walking** box. The **Actions** tab on the right lists what that state does, each action on a card of its own.
4. Open the card for the Input check whose line ends with → **DashPressed**. Beside **Input Action** press ▼ and pick the key you want. It takes the place of the **Space** bar, **Shift** and the bottom button.
5. To keep a gamepad button as well, press ▼ again, tick **Add the one picked to what is there (either works)** and pick the button. This is the only card that asks for the dash key.

Use the same hero in the next level

1. In the first level, click **Player** and press **Use These For Every Copy** under its settings. The numbers you changed now belong to the template, not only to this one player.
2. Open the next level and add the hero again with **NodeWorthy** → **Add Player** → **Top-Down Hero (3D)**. It arrives with a fresh template of its own, named with a 2 at the end.
3. Right-click the new **Player** and choose **NodeWorthy** → **Behave Like**, then pick your first template, which is named with your first level, such as **Top-Down Hero (3D), in Level 1**. Both players now run the very same template, so a change made in one level is made in both.
4. If the first player was given health, choose **NodeWorthy** → **Give It** → **Health** on the new one too, so it has a **HurtBox** of its own. The **Dead** state is already in the template and is left as it is.
5. To carry health from one level into the next, open the template with **Open Its State Machine...**, go to the **Variables** tab and tick **Carries Over Between Scenes**.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
After Set Model... the model glides about without moving its limbs.	Click Player and look for an Animations section in the Inspector. If there is none, the model file holds no animations. Give it some with NodeWorthy → Looks → Use Animations From... (Experimental) , from any animation file. It is experimental, so watch each move before you rely on it. If there is one, a clip was guessed wrongly from its name, so pick the right clip on each line.
After Set Model... the model is plain white, or a part of it is.	The pictures its file names were not found beside it. Right-click Player and choose NodeWorthy → Looks → Set Model Textures... , and pick each part's picture from the pack's Textures folder.
The hero drops out of sight as soon as the game starts.	Nothing solid is under it, so it falls for ever. Add a floor with NodeWorthy → Add Thing → Ground and move Player above it.

See also **Top-Down Hero (2D)** 47, **Third Person Hero (3D)** 37, **Ground** 62, **Enemy** 74, **Minimap** 155

THINGS

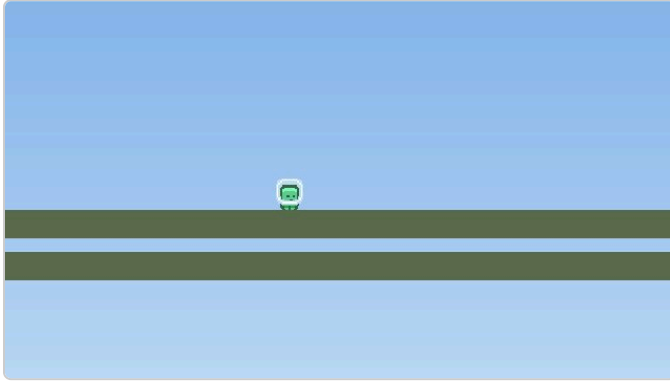
The ready-made things

A thing is something for the player to meet, such as a floor, an enemy, a chest, someone to talk to, a way out, or a menu. Each works the moment it is added, in 2D or 3D to match the scene. They are under **NodeWorthy** → **Add Thing** and **NodeWorthy** → **Add Menu** when you right-click the top node of a level.

Ground	62
Terrain	65
Model...	71
Enemy	74
Level Exit	86
Item Pickup	91
Villager	97
Chest	104
Main Menu	111
Pause Menu	116

Ground THING

A wide, solid floor for everything to stand on. It is there and solid the moment it is added. It does nothing else, so it has no template and no settings of NodeWorthy's own.



Ground as the game runs.

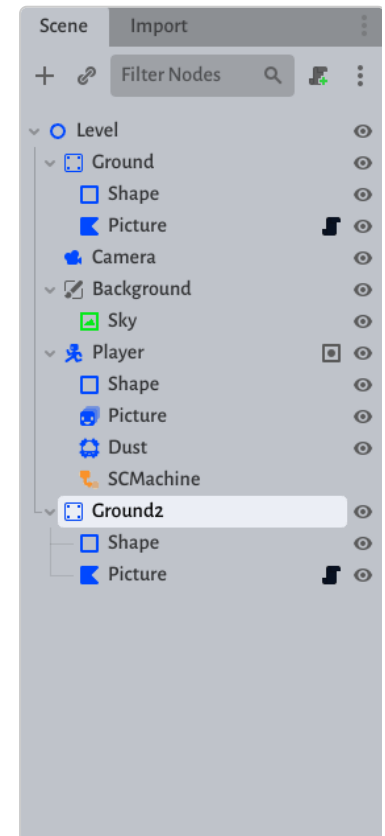
What it is good for

Use it as the floor of a side-on 2D level or of a 3D level, and add more of them as ledges and platforms. A level made with **Start A 2D Level...** or **Start A 3D Level...** already has one. In a 2D game seen from above nothing falls, so a Ground is only a long solid bar there, and a level painted with tiles suits that kind of game better. The side-on and 3D heroes in this book land standing on it, and so do the **Enemy**, the **Chest** and the **Villager**.

Add it

1. Open the scene the level is in. It needs at least one 2D or 3D node, and the top node of a new 2D Scene or 3D Scene counts as one.
2. In the Scene dock, the list of nodes at the top left, right-click the top node and choose **NodeWorthy** → **Add Thing** → **Ground**.
3. In 2D a dull green strip appears, 2400 pixels long and 40 thick, at the node you clicked or a little below it. In 3D it is a slab 60 metres square and 1 metre thick.
4. Drag it where the floor should be. Its top edge is where the node is, so the height you drag it to is the height things stand at.
5. Add the player and the other things after it. Each one looks for something solid under the spot it lands on and stands on top of it.

Try it. Press F6, or the **Run Current Scene** button at the top right of Godot. There is nothing to press for the Ground itself. With a player in the scene, the player lands on it and walks along it, and at either end walks off and falls.



The Scene dock once it is added.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Ground	StaticBody2D	The floor itself. A StaticBody2D is Godot's node for something solid that never moves. It stands on the collision layer called World, where walls and floors go. In a 3D scene it is a StaticBody3D.
Shape	CollisionShape2D	How big the floor is to Godot's physics, which is what makes things solid. It holds a rectangle, and dragging its handles is how the Ground is made longer or thicker. In 3D it is a CollisionShape3D holding a box.
Picture	Polygon2D	The green shape you see. A Polygon2D is Godot's node for a flat coloured shape. In the editor it follows the Shape, so what you see is always what is solid. In 3D it is a MeshInstance3D, Godot's node that draws a 3D shape.
Sun	DirectionalLight3D	Only in 3D, and only when the scene had no light and no sky. A DirectionalLight3D is Godot's node for light that falls on everything from one direction, as sunlight does. Move it to change where the light comes from.
Sky	WorldEnvironment	Only in 3D, and added with the Sun. A WorldEnvironment is Godot's node for the sky and the light that comes from it.

Its settings

It has no settings of NodeWorthy's own. Its size and place are Godot's, set in the view and in the Inspector.

Make it your own

Make it longer, wider or thicker

1. In the Scene dock, click the small arrow beside **Ground** to show what is under it, then click **Shape**.
2. In the 2D or 3D view, drag the orange handles on the edges of the shape. The green picture follows as you drag.
3. To move the whole floor, click **Ground** again and drag that. Moving the Ground keeps its top edge where the node is.

Add a second one as a platform

1. In the Scene dock, the list of nodes at the top left, right-click the top node and choose **NodeWorthy** → **Add Thing** → **Ground** once more. Each Ground has a shape of its own, so changing one leaves the others as they are.
2. Click the new Ground's **Shape** and drag its handles in until it is the length of a ledge.
3. Click the new **Ground** and drag it up to a height the player can jump to.
4. It is solid from every side, so the player lands on top of it and bumps into it from below.

Change its colour

1. In the Scene dock, click **Picture** under **Ground**.
2. In the Inspector, the panel on the right that shows the settings of whatever is clicked, press **Color** and pick a colour.
3. In 3D the colour belongs to a material. Click **Picture**, open **Geometry** in the Inspector, click the material beside **Material Override**, and change the **Color** under **Albedo**.

Paint a level with tiles instead

1. This suits a 2D level seen from above or at an angle, where one long strip of floor is no use.
2. In the Scene dock, the list of nodes at the top left, right-click the top node and choose **NodeWorthy** → **Paint With My Own Tiles** → **Create TileSet From Image...** and pick a picture of tiles from anywhere on your computer.
3. Say how many tiles the picture holds **Across** and **Down**, then click each tile nothing can walk through, such as a wall, a rock or water.
4. Press **Use It**. The level gets two layers, **Floor** and **Walls And Things**, to paint on with Godot's TileMap panel at the bottom of the editor.
5. Once the painted level does the Ground's job, click **Ground** in the Scene dock and press Delete.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
Ground is greyed out in the Add Thing menu	Nothing in the scene says whether the game is 2D or 3D, which happens when the top node is a plain Node. Hover over the greyed line and it says to add a 2D or a 3D node first. Add a Node2D or a Node3D, or make a new scene with 2D Scene or 3D Scene.
Start A 2D Level... is not in the menu	It is offered only while the scene is empty, because it sets up a whole level with a Ground, a Camera and a Background. Once the scene has things in it, add what is missing with Add Thing → Ground .
Clicking the Ground shows no NodeWorthy settings and no Open Its State Machine... button	That is how it is meant to be. The Ground does nothing, so it has no Machine, which is the node that runs a template, and no template. To make a floor do something, such as crumble or move, right-click it and choose NodeWorthy → Add State Machine... , which gives it a Machine and a template to build.
Right-clicking Shape or Picture offers one greyed line	They are parts of the Ground. The line says so and tells you to right-click the Ground. NodeWorthy's menu for the floor is on the Ground node itself.
A Sun and a Sky appear when the first Ground goes into a 3D scene	The scene had no light, and without one a 3D game runs almost black. NodeWorthy adds both once and says so. Move the Sun to change where the light comes from, or delete both to light the scene your own way.

See also [Platform Hero \(2D\)](#) 8, [Platform Hero \(3D\)](#) 27, [Enemy](#) 74, [Level Exit](#) 86

Terrain THING

A landscape for a 3D level, with hills to walk over, solid ground and grass. It is rolling hills the moment it is added, and **Try Another** at the top of its Inspector makes different ones. It can also be a valley, an island with sea round it, or mountains round a meadow.



Terrain as the game runs.

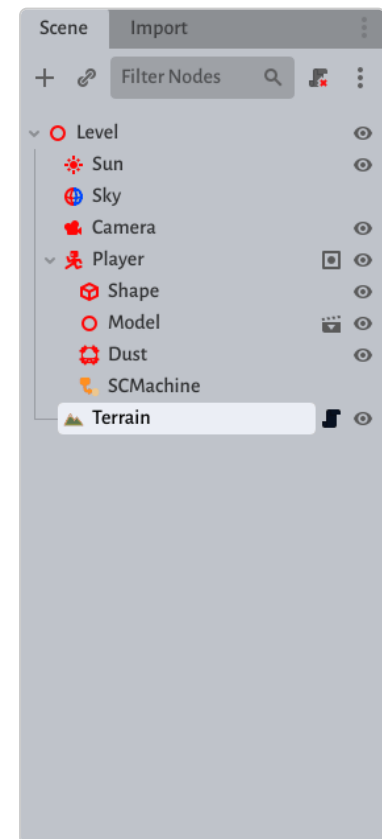
What it is good for

The open ground of a 3D adventure, where the player walks over hills towards something on the far side, such as a village, a camp of enemies or a chest on a rise. It is for one level, not a world without end, so it comes in three sizes and no bigger. For a bigger game, give each part of the world a scene and a Terrain of its own and join them with a **Level Exit**. In a 2D game the ground is a **Ground**, or tiles painted with **Paint With My Own Tiles**.

Add it

1. Open a 3D level. A new scene set up with **NodeWorthy** → **Start A 3D Level...** has a flat **Ground**, a **Camera**, a **Sun** and a **Sky**, which suits it well.
2. In the Scene dock, the list of nodes at the top left, right-click the top node and choose **NodeWorthy** → **Add Thing** → **Terrain**.
3. Rolling hills 256 metres across appear. Where the level had the flat **Ground**, the Terrain takes its place with its level patch in the middle where the Ground's top was, and the player, the enemies and their patrol points each stand on the new ground where they were. A note at the bottom right of the editor says so.
4. Click **Terrain** in the Scene dock and look at the Inspector, the panel on the right that shows the settings of whatever is clicked, on the **3D** tab. At the top are **Kind**, **Try Another**, **Use My Own Height Map...** and **Save Height Map As A Picture...**, and below them its settings. When the level has a **Minimap**, **Take Map Picture** is there too, so the map can be pictured again straight after the ground changes.
5. Pick a **Kind**, and press **Try Another** until you like the ground. Everything standing on it stays standing on it, and **Undo** goes back one at a time.

Try it. Press F6, or the **Run Current Scene** button at the top right of Godot, with a 3D player on the Terrain. Walk with W. The player walks up and down the hills and stops at ground steeper than 45 degrees as it would at a wall. Grass stands round the player and sways, and further off the ground's own colour takes over. Walk to an edge and a wall nobody can see stops you.

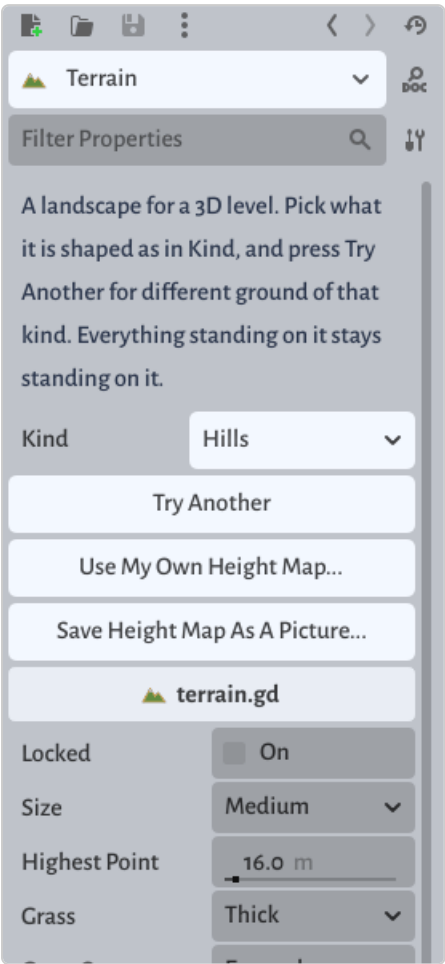


The Scene dock once it is added.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Terrain	StaticBody3D	The landscape. A StaticBody3D is Godot's node for something solid that never moves, and it is on the collision layer called World, where floors and walls go. It makes its ground, what is solid about it and its grass from its heights each time the scene opens, so none of that is listed under it and the scene file stays small. Its heights are kept in a file of their own in the terrain folder inside your project's nodeworthy folder.
Sun	DirectionalLight3D	Only when the scene had no light and no sky. A DirectionalLight3D is Godot's node for light that falls on everything from one direction, as sunlight does.
Sky	WorldEnvironment	Added with the Sun. A WorldEnvironment is Godot's node for the sky and the light that comes from it.

Its settings



Click **Terrain** in the Scene dock and these are in the Inspector, the panel on the right, under its own name, below the row that changes its ground. A change here is this one's own.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Locked	Tick it once the ground is the way you want it, before placing things on it. While it is locked, Kind, Try Another, Use My Own Height Map..., Size, Highest Point and the lake are greyed out, so nothing you have placed is moved by a slip of the mouse. Colours, grass, waves, trees, boulders and flowers can still be changed.	off
Size	How far the ground runs from side to side. Small is 128 metres, Medium 256 and Large 512. The ground keeps its shape and is stretched, and Highest Point changes with it so the slopes stay as steep as they were. The middle stays where it is, and everything standing on the ground stays standing on it. Choices: Small, Medium, Large.	Medium
Highest Point	How far above the lowest ground the highest ground is, in metres. Higher makes every slope steeper, and ground steeper than 45 degrees is a wall a player cannot walk up. Each kind starts at a height that suits it.	16 m
Grass	How much grass stands near the camera, which is None, Thin or Thick . Further off, the ground's own colour takes over. Pick Thin or None when the game runs slowly. Choices: None, Thin, Thick.	Thick
Grass Cover	How much of the gentle ground the grass covers. Everywhere is a meadow. Mostly, In Patches and Here And There leave more and more bare ground, in Bare Ground Colour , with the grass growing in patches across it. The blades of grass grow only on the patches. Choices: Everywhere, Mostly, In Patches, Here And There.	Everywhere
Dry Patches	Turn it on and some patches of grass are dry, in Dry Grass Colour , and the rest are green. With Grass Cover at In Patches it makes sandy ground with patches of yellow and green grass.	off
Trees	Trees on the ground, pines and leafy ones, which is None, Mild, Medium or Heavy . They never stand in water, on rock or on the level middle where the player starts, and they are solid, so characters bump into them. A Terrain from Add Thing starts with Mild . Choices: None, Mild, Medium, Heavy.	Mild
Boulders	Boulders on the ground, None, Mild, Medium or Heavy . They sit on slopes and by the shore, never on the level middle, and they are solid. Choices: None, Mild, Medium, Heavy.	Mild
Flowers	Flowers in clusters on gentle ground, in many colours, None, Mild, Medium or Heavy . They sway with the grass, and nothing bumps into them. Like the grass, only those near the camera are drawn. Choices: None, Mild, Medium, Heavy.	Mild
Waves	How the water moves, where the ground has water, as an Island does. Still water has only small ripples, Gentle rolls in long waves and Rough in tall ones. The sea goes on past the edges of the Terrain to the horizon. Choices: Still, Gentle, Rough.	Gentle
Lake	Turn it on for a lake where the open ground lies lowest, away from the level middle, with a shore that wanders and banks sloping down to it. Nothing else floods, and turning it off gives the ground back exactly as it was. An Island has the sea instead, so it has no lake, and Lake is greyed out there. Under the water the camera sees through a blue green haze, with the surface shining overhead.	off

SETTING	WHAT IT DOES	STARTS AS
Lake Size	How far across the lake is. Small is about 24 metres, Medium 40 and Large 64. Choices: Small, Medium, Large.	Medium
Lake Depth	How deep the lake is in its middle, in metres. Its edge is shallow whatever the depth, so a character can walk out of it.	4 m
Walls at the Edge	Walls nobody can see along the four edges, so nothing walks off the side of the world. Turn it off when the level goes on past the Terrain with ground of its own.	on
Grass Color	The colour of the grass, and of the gentle ground it grows on. Bare ground keeps its own colour, Bare Ground Colour .	green
Leaf Color	The colour of the trees' leaves. Each tree is a little lighter or darker, and the pines a little darker than the leafy trees.	green
Boulder Color	The colour of the boulders. Each one is a little lighter or darker.	grey
Dry Grass Color	The colour of the dry patches of grass, a straw yellow at first. It shows only when Dry Patches is on.	yellow
Bare Ground Color	The colour of gentle ground where no grass grows, a sandy brown at first. It shows only when Grass Cover leaves some ground bare.	orange
Rock Color	The colour of ground too steep for grass, which is about where a player can no longer walk up it.	grey
Earth Color	The colour of the bare ground between the grass and the rock.	brown
Sand Color	The colour of the ground beside the water and under it. It shows only where the ground has water, as an Island does.	yellow
Water Color	The colour of the water out where it is deep, and how much of the sea floor shows through it. In the shallows it is paler and clearer, and foam laps where it meets the shore. It shows only where the ground has water, as an Island does.	blue

Make it your own

Lock it before you build on it

1. Make the ground the way you want it with **Kind**, **Try Another** and the settings.
2. Tick **Locked**. The settings that change the ground's shape are greyed out, and so are the buttons at the top of the Inspector.
3. Place your houses, enemies and chests. Nothing moves the ground from under them.
4. Untick **Locked** to change the ground again.

Make different ground

1. Click **Terrain** in the Scene dock, on the **3D** tab.
2. Pick a **Kind** at the top of the Inspector. **Hills** roll gently. A **Valley** has a floor between two sides. An **Island** has sea all round it. **Mountains** ring a meadow with peaks too steep to walk.
3. Press **Try Another** for different ground of the same kind, as often as you like. The level patch in the middle stays where it is, and everything standing on the ground stays standing on it.
4. **Undo** goes back to the ground before, one press at a time.

Make it gentler or steeper

1. Lower **Highest Point** for gentler slopes, or raise it for steeper ones.
2. Ground steeper than 45 degrees is a wall to a player. It shows as rock.
3. The **Platform Hero (3D)** climbs such ground by holding J against it, as it climbs any wall, until its grip runs out. It goes up steep rock quickly, and ground only a little too steep to walk more slowly.

Add a lake

1. Click **Terrain** in the Scene dock, on the **3D** tab.
2. Tick **Lake**. A lake is carved where the open ground lies lowest, away from the level middle.
3. Pick **Lake Size** and set **Lake Depth** in metres.
4. For a different lake, press **Try Another**, which makes new ground with a lake of its own.

Trees, boulders and flowers

1. Set **Trees**, **Boulders** and **Flowers** to **Mild**, **Medium** or **Heavy**, or **None** for bare ground.
2. Open **Colours** and pick **Leaf Colour** and **Boulder Colour**.
3. They stay in the same places every time for the same ground. **Try Another** puts them on the new ground.

Grass in patches, with bare ground between

1. Click **Terrain** in the Scene dock, on the **3D** tab.
2. Set **Grass Cover** to **In Patches**. The gentle ground turns bare between patches of grass, and the blades grow only on the patches.
3. Open **Colours** and pick **Bare Ground Colour** for the bare ground, such as sand or dry earth.
4. Turn on **Dry Patches** for some patches of yellow grass among the green, and pick their colour in **Dry Grass Colour**.

Shape it from a picture of your own

1. Press **Use My Own Height Map...** at the top of the Inspector and pick a picture from anywhere on your computer. Lighter is higher and black is the lowest ground.
2. Any size of picture will do. It is made square and smoothed.
3. To paint on the ground you have, press **Save Height Map As A Picture...** first, change the picture in any picture editor, and bring it back with **Use My Own Height Map....** The height map is the shape of the ground, a grey picture, and not what the Minimap shows.

Let enemies find their way over it

1. An **Enemy** that sees the player comes straight over the hills with no more set up.
2. A machine that uses **Navigate To** walks round what stands in its way. When one is added, NodeWorthy bakes walkable ground over the level, which is the navigation mesh.
3. After changing the ground with **Try Another**, a new **Kind**, **Size** or **Highest Point**, right-click the level and choose **NodeWorthy** → **Bake Navigation Mesh**, so the walkable ground follows the new hills.

Make a bigger world

1. Make a scene for each part of the world, each with a Terrain of its own.
2. Put a **Level Exit** where one part should lead into the next, and pick the next scene in its **Goes To**.
3. The player's health, items and quests go with them from one scene to the next.

Change its colours

1. Open **Colours** in the Inspector and pick a colour for the grass, the rock, the bare earth, the sand and the water.
2. The blades of grass take the grass colour too, so a dry yellow field is one colour away.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
Terrain is greyed out, or missing, in the Add Thing menu	One is all a level needs, so it is greyed out once the level has one, and hovering over it names the one there. It is offered only in a 3D level, so a 2D level does not list it at all.
Kind and Try Another are nowhere to be seen	Terrain is not the node clicked, or the Inspector was put away with Hide Inspector on the NodeWorthy tab. Click Terrain in the Scene dock, and press Show Inspector at the top right of the editor if the Inspector is not there.
The player cannot get up a hill	That slope is steeper than 45 degrees and counts as a wall. Walk round it, lower Highest Point , or press Try Another . The Platform Hero (3D) can climb it by holding J.
Something dragged across the ground floats in the air or sinks into it	Things are stood on the ground when they are added and when the ground changes, but one dragged by hand keeps the height you left it at. Drag it up or down with the green arrow until its feet touch the ground.
Navigate To says there is no way, or a character walks into a hill	The walkable ground was baked from the ground before it changed. Right-click the level and choose NodeWorthy → Bake Navigation Mesh again.
The game runs slowly	Set Grass to Thin , or to None . The ground itself costs very little to draw.
The player walks into the sea	The water is there to look at. A player walks out along the sea floor until the wall at the edge stops them. Leave Walls at the Edge on so nobody goes past it.
The ground will not change, and Kind and Try Another are greyed out	Locked is ticked. Untick it to change the ground, and tick it again before placing more things.
There is no lake after ticking Lake	An Island has the sea round it instead. Pick another Kind , or untick Lake .

See also **Ground** 62, **Third Person Hero (3D)** 37, **Platform Hero (3D)** 27, **Enemy** 74, **Level Exit** 86

Model... THING

A model of your own, from anywhere on your computer, put into a 3D level. A table, a crate, a wall, a tree or a whole building from a pack you bought goes in at the size its file made it, wearing the textures its pack gives each part, and solid exactly where it is seen. A window asks what it is first, so the same model can come in as plain scenery, as one of the ready-made things wearing it, or doing what a template of your own does.

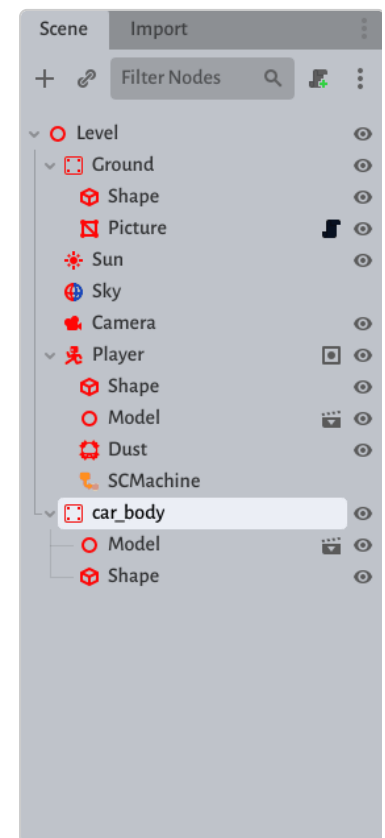
What it is good for

Use it to fill a 3D level with your own art. It works with .glb, .gltf and .fbx files, including those from packs made for other game engines, whose textures it finds in the pack's own folders. Because it is solid where it is seen, a hero walks round the legs of a table and through an open doorway, and bumps into the walls either side. It is for 3D levels only. In a 2D game, **Looks** → **Set Texture...** gives a thing a picture of your own.

Add it

1. Open the 3D scene the level is in.
2. In the Scene dock, the list of nodes at the top left, right-click the node it should stand beside, such as the player, or the top node, and choose **NodeWorthy** → **Add Thing** → **Model...**
3. A window opens on your computer's folders. Find the model's file, click it and press **Open**. It is copied into a folder of its own in the project's models folder, with the pictures it wears.
4. A window asks **What is it?**. Leave it at **Scenery** for something solid and still, such as a table or a wall. The line under it says what each choice does. Press **Put It In**.
5. It stands on the ground beside the node you right-clicked, and further along when something already stands there, so it is never put on top of anything. A line at the bottom of the editor says how tall it is.

Try it. Press F6, or the **Run Current Scene** button at the top right of Godot, and walk the player into it. The player is stopped where the model is seen, and walks through any gap in it, such as the space under a table or an open doorway.



The Scene dock once it is added.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
car_body	StaticBody3D	The model as a thing of the level, named after its file. A StaticBody3D is Godot's node for something solid that never moves. It stands on the collision layer called World, where walls and floors go.
Model	Node3D	The model itself, made from its file in the project's models folder. Its parts stay in that file, so the scene stays small and a change to the file shows in every scene that uses it.
Shape	CollisionShape3D	What makes it solid. A CollisionShape3D is Godot's node that gives a body its shape for physics. This one holds a shape made from every surface of the model, kept as a file beside the model, so every copy of it shares one.
Sun	DirectionalLight3D	Only when the scene had no light and no sky. A DirectionalLight3D is Godot's node for light that falls on everything from one direction, as sunlight does.
Sky	WorldEnvironment	Only when the scene had no light and no sky, added with the Sun. A WorldEnvironment is Godot's node for the sky and the light that comes from it.

Its settings

It has no settings of NodeWorthy's own. Its size and place are Godot's, set in the view and in the Inspector.

Make it your own

Make it a chest, a pickup, an enemy or a villager

1. Right-click the node it should stand beside and choose **NodeWorthy** → **Add Thing** → **Model...**, and pick the model's file.
2. In the **What is it?** window, pick **Chest**, **Item Pickup**, **Level Exit**, **Villager** or **Enemy** in place of **Scenery**. A file with chest in its name is taken for a Chest to begin with.
3. A Chest, an Item Pickup or a Level Exit keeps the size its file made it, and its shape is made to fit the model, so it opens, is picked up or is walked into where it is seen.
4. A Villager or an Enemy is a character. A second window asks which way the model faces and, when the file holds a whole cast of characters, which one it is. It is made as tall as a hero.
5. A character that came with no animations of its own is given those that came with NodeWorthy, made to fit its skeleton, so it stands, walks, fights and falls at once. This fitting is experimental, so watch it move before you rely on it. To give it a pack of your own, such as a zombie's, pick **Animations from files of my own, such as a zombie pack... (Experimental)** in the **What is it?** window, and it plays that pack's moves from the start. Later, right-click it and choose **Looks** → **Use Animations From... (Experimental)** to add a pack beside the moves it has. Fitting another pack's animations is experimental, so watch each move before you rely on it. None is swapped by itself, so pick each one you want under **Animations** in the Inspector.

Make it do what a template of yours does

1. In the **What is it?** window, the templates you have made yourself are listed after the ready-made things, each as Like and its name.
2. Pick one and press **Put It In**. The model goes in solid, as scenery does, with a Machine running that template, the node that runs a template. A change to the template changes everything that runs it.

Put another model in its place

1. Right-click it in the Scene dock and choose **NodeWorthy → Looks → Set Model....**
2. Pick the other file. It stands where the first one stood, at the size its own file made it, with a shape of its own. One undo, Ctrl+Z or Cmd+Z on a Mac, brings the first one back.

Change the textures it wears

1. Right-click it and choose **NodeWorthy → Looks → Set Model Textures....**
2. Each part of the model is listed with the colour picture and normal map it wears. Press the button beside one to pick another picture from anywhere on your computer.

Move it, turn it or make it bigger

1. Click it in the Scene dock and drag it in the 3D view, or use Godot's move, rotate and scale tools at the top of the 3D view. Its shape moves, turns and grows with it.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
Model... is not in the Add Thing menu	The scene is a 2D one. A model is for a 3D level. In a 2D game, Looks → Set Texture... gives a thing a picture of your own.
It comes in white	Its file names pictures that its pack does not hold under those names, and its pack says nothing about them. The Set Model Textures... window opens by itself to pick each part's picture.
Godot says it could not open a picture	The model's file names a picture as the artist's own computer kept it, often as a Photoshop file or in a folder that is not there. NodeWorthy finds the picture its pack holds and the next line says what the model wears.
It is far too big or far too small	It comes in at the size its file made it. Some files are made in other units. Click it and use Godot's scale tool at the top of the 3D view, and its shape grows or shrinks with it.
An Enemy or a Villager stands like a T	Its file held no animations a character can use and its skeleton is not a person's, two arms and two legs. Give it animations made for its own skeleton with Looks → Use Animations From... (Experimental) .

See also **Ground** 62, **Terrain** 65, **Chest** 104, **Enemy** 74

Enemy THING

An enemy that thinks for itself. The moment it is added it stands guard, and when the player comes into sight it pauses, calls any other enemies nearby, chases the player and hurts them by touching. It has 3 health, and at 0 it drops what it carries, fades and is gone.



Enemy as the game runs.

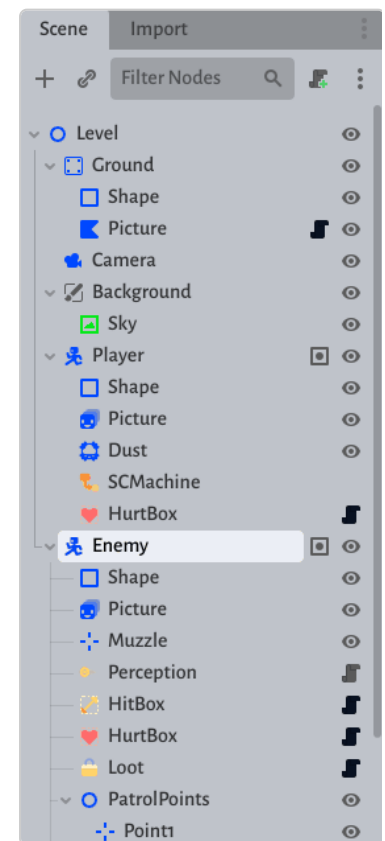
What it is good for

Use it for anything that comes after the player, such as a guard on a platform, a pack of monsters in a dungeon seen from above, or an archer on a wall in 3D. One template covers them all, because its settings choose whether it stands, patrols or wanders, and whether it touches, swings or shoots. For someone who only talks, the **Villager** is the better pick, and for loot that sits still, the **Chest**.

Add it

1. Start with a level that has a player, and a **Ground** under them unless the game is seen from above. In a 2D game seen from above, add the **Top-Down Hero (2D)** before the first Enemy, because the Enemy is built to match the player it finds in the scene. Added the other way round, NodeWorthy says so when the hero goes in.
2. In the Scene dock, the list of nodes at the top left, right-click the top node and choose **NodeWorthy** → **Add Thing** → **Enemy**.
3. An enemy lands well to the right of the node you clicked, 800 pixels away in 2D and 16 metres away in 3D, standing on the ground. Drag it to where it should start, and its two patrol points come with it.
4. If the player had no health, NodeWorthy gives it now and says so. The player gets a Hurt Box, a health of 3 shown on the screen, and a Dead state that starts the level again.
5. Click **Enemy** in the Scene dock. In the Inspector, the panel on the right that shows the settings of whatever is clicked, its settings are at the top in sections that fold, called **Who**, **Seeing**, **When Idle**, **Chasing**, **Fighting**, **Tactics**, **Health**, **Loot** and **Worth**, with **Look** as well in 2D.
6. Press F6, or the **Run Current Scene** button at the top right of Godot.

Try it. Walk towards the enemy. When you come within 400 pixels, or 12 metres in 3D, with no wall between you, it pauses for about a third of a second and comes after you. Each time it touches you, your health on the screen drops by 1 and you are knocked back, and at 0 the level starts again.



The Scene dock once it is added.

Walk out of its sight and after 2 seconds it goes to where it last saw you, looks round for 3 seconds, gives up and walks back to where it stood.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Enemy	CharacterBody2D	The enemy's body. A CharacterBody2D is Godot's node for a character that walks and is stopped by walls. It stands on the collision layer called Enemies and is in the group called enemies, which is how the player's Gun finds it. In 3D it is a CharacterBody3D.
Shape	CollisionShape2D	How big it is to Godot's physics, a rectangle 24 pixels wide and 40 tall. In 3D it is a box 0.8 metres wide and 1.8 tall.
Picture	AnimatedSprite2D	What you see. Seen from the side it is a little blue blob that squishes as it goes, and in a game seen from above a green slime that hops a little with each step and breathes while it waits, both from free art that comes with NodeWorthy. An AnimatedSprite2D is Godot's node for a picture that steps through frames. In 3D it is an animated person tinted red, a Model node, whose clips the template plays, walking on patrol, jogging when it chases, punching, shooting, flinching when hit and falling when beaten. Right-click the Enemy and choose NodeWorthy → Looks to put a picture, a sprite sheet or, in 3D, a model of your own in its place. A sprite sheet with rows named idle, walk, run, attack, shoot, hurt and die gives each state the row that suits it.
Muzzle	Marker2D	The spot its bullets leave from, at the middle of its body. A Marker2D is Godot's node for a spot with nothing drawn in the game. The bullet itself is a small scene saved in your project's bullets folder when the Enemy is added.
Perception	Perception	What it can notice. It holds how far it sees, how wide it looks, how long it remembers and how far it hears. It starts out seeing 400 pixels all round, or 12 metres in 3D, and a wall on the World layer hides the player from it.
HitBox	Hit Box	A Hit Box, the part that hurts by touch. It is a circle reaching 30 pixels from the middle of its body, or 1 metre in 3D, and whoever it touches that has a Hurt Box loses as much health as Damage says. Its touch hurts only while How It Fights is Touches . Each touch that lands sends its Machine the event InReach , which is what makes the enemy stop and wait.
HurtBox	Hurt Box	A Hurt Box, which is what lets it be hurt. Each hit comes off its health and knocks it back, and at 0 health it sends its Machine the event DIED . It also keeps who hurt it, so an enemy hurt by someone it has not seen can go and look for them.
Loot	Inventory	An Inventory of its own, holding what it drops when it is beaten. As the level starts it takes items from its Can Drop list, which starts empty.
PatrolPoints	Node2D	Holds the two spots it walks between when When Idle is Patrols . In the editor it moves with the enemy. Once the game runs it stays where it is in the level, so the enemy walks away from its points, and it marks the place a guard walks back to and a wanderer stays near. Keep it even for an enemy that never patrols, as it says when the pointer rests on it in the Scene dock. In 3D it is a Node3D.

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Point1	Marker2D	One end of its patrol, 200 pixels to the left of the enemy, or 5 metres in 3D. It comes along when the enemy is dragged. Drag the point itself to move that end of the patrol.
Point2	Marker2D	The other end of its patrol, the same distance to the right.
SCMachine	Machine	The Machine, which is the node that runs the enemy's template. The template is saved as Enemy in your project's nodeworthy folder, unless you have picked another folder for templates, and every Enemy in the scene runs the same one.
HurtBox	Hurt Box	A Hurt Box under the player, added when the player had no health, along with a health of 3 and a Dead state in the player's own template.
HealthPanel	Stat Panel	A Stat Panel that shows the player's health on the screen, added with it. When the scene already has a Stat Panel, the health is shown on that one and no new panel is made. It sits on the screen layer the scene has, or on a new CanvasLayer called HUD, which is Godot's node for things drawn over the game.
Sun	DirectionalLight3D	Only in 3D, and only when the scene had no light and no sky. It lights the level, and it comes with a Sky , a WorldEnvironment, which is Godot's node for the sky.

Its settings



Click **Enemy** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot's own settings, in sections that fold.

A change here changes this one only. The arrow that appears beside a changed setting puts the template's value back. **Use These For Every Copy**, under the settings, makes them the template's own, for every copy.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
WHO		
Attacks	Who it looks for, chases and hurts, which can be The player , Other enemies or Villagers . Its touch, its swing and its bullets all follow this one setting. Whoever it attacks needs a Hurt Box to lose any health, and Friends should be set to somebody else. Choices: The player, Other enemies, Villagers.	The player
Friends	Who hears its call when it spots someone, which can be Other enemies , Villagers or Nobody . It matters only while Alerts Friends is on, and the call reaches as far as Alert Distance . It only lists who to call, because who gets hurt is decided by Attacks . Choices: Other enemies, Villagers, Nobody.	Other enemies
SEEING		
Reaction Time	The pause between spotting someone and doing anything about it, spent in the state Noticed . It starts at 0.3 seconds, which feels alert, and a longer pause gives the player a head start. An enemy called by a friend skips the pause, and so does one that finds the player again while searching.	0.3 s
Sight Range	How far it can see. It is kept on its Perception node.	400 px
FOV	How wide it looks. 360 sees all round, 90 only ahead. It is kept on its Perception node.	360°

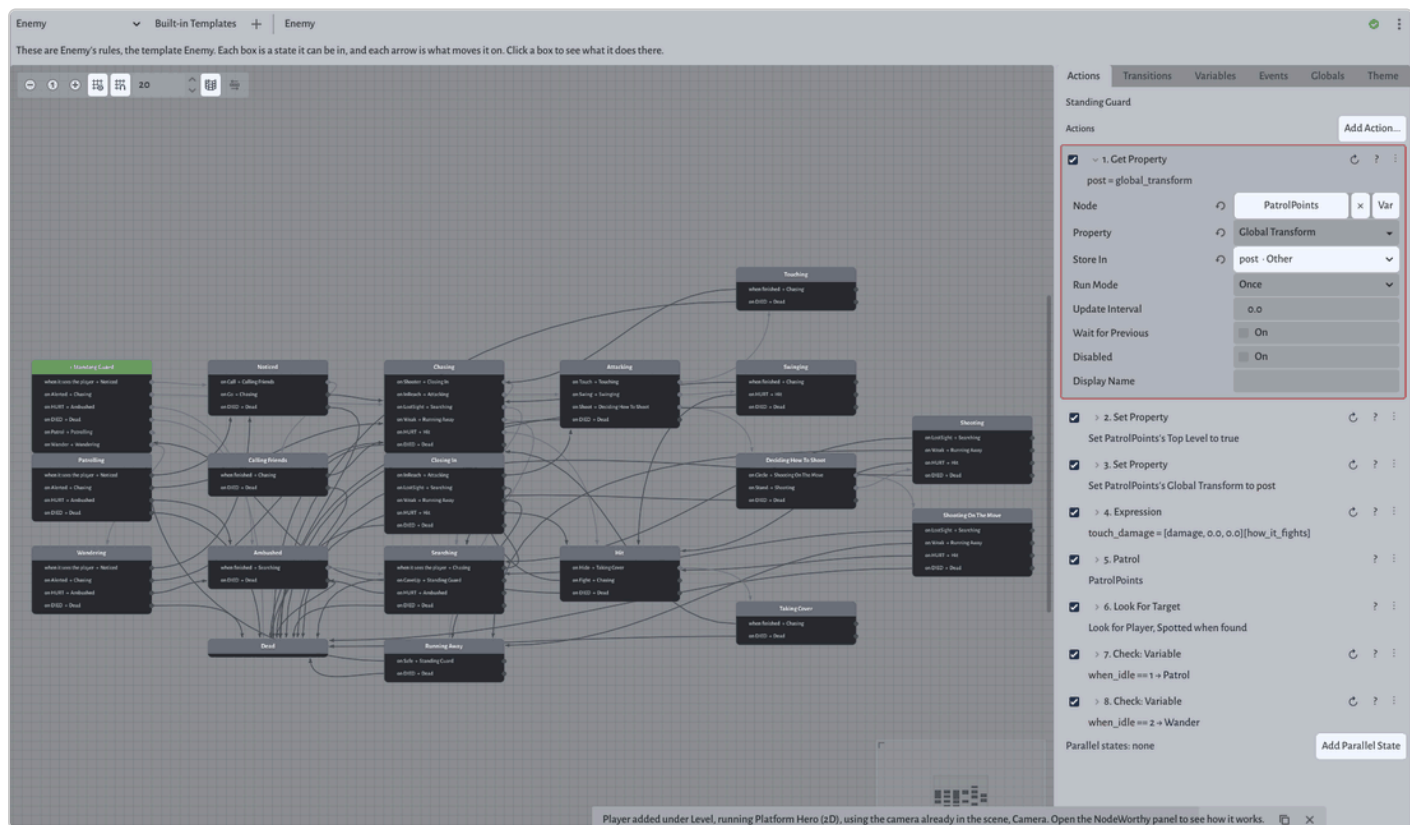
SETTING	WHAT IT DOES	STARTS AS
Remembers For	How long it keeps after something once it loses sight of it. It is kept on its Perception node.	60 s
Hearing Range	How far away it hears things, even out of sight. 0 hears nothing. It is kept on its Perception node.	0 px
WHEN IDLE		
When Idle	What it does while it has nobody to go after. Stands Guard stays where it was put, and walks back there after a chase. Patrols walks between Point1 and Point2 , and Wanders strolls about near where it was put. PatrolPoints marks that place, so keep it whichever you pick. Idle Speed goes with all three, Wait At Each Point with Patrols, and Wander Distance with Wanders. Choices: Stands Guard, Patrols, Wanders.	Stands Guard
Idle Speed	How fast it walks while it patrols or wanders. It starts at 60 pixels a second, or 1.5 metres a second in 3D, which is half its Chase Speed , so it strolls. For an enemy that stands guard it is how fast it walks back to its place after a chase.	60 px/s
Wander Distance	How far it strays when When Idle is Wanders . It starts at 200 pixels, or 5 metres in 3D, measured from where it was put in the level. After a chase it strolls back to within that distance of the same place.	200 px
Wait at Each Point	How long it stands at each patrol point before it walks to the other, when When Idle is Patrols . It starts at 1 second. Raise it for a sentry that lingers at each end of its beat.	1 s
CHASING		
Chase Speed	How fast it comes after the one it attacks. It starts at 120 pixels a second, or 3 metres a second in 3D, and an enemy that shoots also backs away and closes in at this speed. Set it below the player's own speed for an enemy that can be outrun, and above it for one that cannot.	120 px/s
Gives Up After	How long the one it chases can stay out of sight before it stops chasing and goes to look where it last saw them. It starts at 2 seconds. A low number makes it easy to shake off round a corner, a high one makes it dogged, and Searches For then sets how long the looking lasts.	2 s
Searches For	How long it stands and looks round at the place it last saw them before it gives up and goes back to what When Idle says. It starts at 3 seconds. An enemy that stands guard then walks back to where it was put, one that patrols walks back to its points, and one that wanders strolls back.	3 s
FIGHTING		
How It Fights	How it hurts. Touches comes right up and leaves the hurting to its Hit Box, Swings throws a punch once it is within Attack Distance , and Shoots stops at Shooting Distance and fires bullets. Its touch hurts only with Touches. Damage and Time Between Attacks go with all three, Attack Distance with Swings, and Shooting Distance , Bullet Speed , Inaccuracy and Circles While Shooting only with Shoots. Choices: Touches, Swings, Shoots.	Touches

SETTING	WHAT IT DOES	STARTS AS
Attack Distance	For Swings , how close it comes before it swings, and how far the swing reaches. It starts at 28 pixels, or 1.2 metres in 3D. Raise it for an enemy with a long reach. An enemy that touches always comes right up, whatever this says, and the setting is not used when it shoots.	28 px
Damage	How much health one touch, one swing or one bullet takes. It starts at 1, against a player who starts with 3 health. One number serves all three ways of fighting, and only the way picked in How It Fights hurts.	1
Time Between Attacks	The gap between one attack and the next, which starts at 1 second. With Touches it stands still this long after its touch lands, and its touch does not hurt again until the wait is over. With Swings it waits this long after each swing, and with Shoots it is the time between bullets. A player who is still safe from the last hit when the wait ends is touched as soon as that safety runs out. Lower it for a frantic enemy and raise it for a slow, heavy one.	1 s
Shooting Distance	For Shoots , how far away it stops to fire. It starts at 280 pixels, or 7 metres in 3D. While it shoots it backs off if the player comes nearer than 160 pixels, or 4 metres, and closes in again if the player gets further away than this setting. With a Shooting Distance under 240 pixels, or 6 metres, it backs off at two thirds of it instead.	280 px
Bullet Speed	For Shoots , how fast its bullets fly. It starts at 320 pixels a second, or 8 metres a second in 3D, which is slow enough to step out of the way of. Every bullet lasts two and a half seconds, so a slower bullet also reaches less far.	320 px/s
Inaccuracy	For Shoots , how many degrees off its aim may be. At 0 every bullet is aimed straight at the one it attacks. Raise it and its shots scatter by up to that many degrees.	0°
Touch Knockback	How hard a touch knocks what it hurts away, where 1 is a good push, 0 none and 2 twice as hard. It is kept on its HitBox node.	1×
TACTICS		
Circles While Shooting	For Shoots , whether it sidesteps round the one it is shooting. It starts on. Turn it off and it moves only to keep its distance, which suits an archer on a wall.	on
Takes Cover	Whether a hit sends it running behind cover before it comes back to the fight. It starts off, and it needs places marked as cover, which are picked in the Take Cover action of its Taking Cover state. With nothing marked it only stands still for as long as Hides For says.	off
Hides For	How long it stays behind cover before it comes out and chases again, when Takes Cover is on. It starts at 2 seconds.	2 s
Alerts Friends	Whether it calls for help when it spots someone. It starts on, so one enemy that notices the player brings every friend within Alert Distance straight into the chase. Turn it off for enemies that should be met one at a time, and set who is called with Friends .	on
Alert Distance	How far its call carries when Alerts Friends is on. It starts at 400 pixels, or 10 metres in 3D. At 0 the call reaches every friend in the level, however far away.	400 px
HEALTH		

SETTING	WHAT IT DOES	STARTS AS
Max Health	How much health it has, and it starts the level full. It starts at 3, so three hits of 1 beat it. Raise it for a boss, and remember that Runs Away At is measured against it.	3
Runs Away At	The health at which it turns and runs for its life. At 0 it never runs. With Max Health at 3, a 1 here makes it run on its last point of health, and Runs Until sets how far it goes.	0
Runs Until	How far from the one it fears it must get before it feels safe, which starts at 400 pixels, or 10 metres in 3D. Once safe it goes back to what it does when idle. Its health is still low, so it runs again the next time a chase begins.	400 px
Invincibility Time	How long it cannot be hurt again after a hit, blinking. It is kept on its HurtBox node.	0.15 s
Knocked Back By	How hard a hit knocks it back. 1 is a good push, 0 none, 2 twice as hard. It is kept on its HurtBox node.	1×
Shows Damage	Show how much each hit takes, rising and fading over it. It is kept on its HurtBox node.	off
Hurt Sound	A sound played when it is hurt, such as a grunt or bones knocking. Empty plays none. It is kept on its HurtBox node.	nothing picked
Death Sound	A sound played when it dies, in place of the hurt sound. Empty plays the hurt sound. It is kept on its HurtBox node.	nothing picked
LOOT		
Gold Least	The least gold it drops when it is beaten. The amount is picked afresh each time, somewhere between this and Gold Most . It starts at 0.	0
Gold Most	The most gold it drops when it is beaten, and at 0 it drops none. The gold lies on the floor for the player to walk over, and it is added to the player's gold variable when they have one. Raising it above 0 also puts that gold on the screen when nothing shows it yet.	0
Can Drop	The items it can hold. As the level starts it takes How Many Items of them at random, rarer ones less often (set each item's Rarity), and drops them when it is beaten. Empty gives gold only. It is kept on its Loot node.	empty
How Many Items	How many items it takes from Can Drop. 0 holds none. It is kept on its Loot node.	1
Item Chance	How likely each of those items is really there, from 0 to 100, where 35 drops one about one time in three. It is kept on its Loot node.	35%
WORTH		
XP Worth	The experience whoever beats it is given, when they have Experience. A skeleton might be worth 20. 0 gives none. It is kept on its HurtBox node.	0
LOOK		
Tint	A colour laid over its picture, so one set of art makes a skeleton, a ghost and a boss. White shows the picture as drawn. It is kept on its Picture node.	white

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.



Its template in the NodeWorthy panel.

STATE	WHAT HAPPENS IN IT
Standing Guard	Stands at the place it was put, walking back there first at Idle Speed when a chase has taken it away, and looks for whoever Attacks names. Each time it begins it also leaves PatrolPoints where it is in the level and works out what its touch takes. If When Idle is Patrols or Wanders it moves on at once to Patrolling or Wandering . Seeing someone sends Spotted and leads to Noticed , a friend's call, Alerted , leads straight to Chasing , and being hurt by someone it has not seen leads to Ambushed .
Patrolling	Walks between Point1 and Point2 at Idle Speed , standing at each for Wait At Each Point , and keeps looking as it goes. Spotted leads to Noticed , Alerted leads to Chasing , and being hurt leads to Ambushed .
Wandering	Strolls to places within Wander Distance of where it was put, at Idle Speed , pausing for a second between strolls, and keeps looking. Spotted leads to Noticed , Alerted leads to Chasing , and being hurt leads to Ambushed .
Noticed	Waits for Reaction Time . Then, if Alerts Friends is on, the event Call leads to Calling Friends , and if it is off, Go leads to Chasing .
Calling Friends	Tells every friend within Alert Distance who it has seen and sends each of them Alerted . That takes no time, and FINISHED leads to Chasing .

STATE	WHAT HAPPENS IN IT
Chasing	Runs after the one it spotted at Chase Speed . One that touches comes on until its touch lands, from whichever side, and one that swings stops Attack Distance short. Either sends InReach and leads to Attacking . An enemy that shoots leaves at once for Closing In . Losing sight for Gives Up After leads to Searching , health at or below Runs Away At leads to Running Away , and being hurt leads to Hit .
Closing In	The chase of an enemy that shoots. It stops Shooting Distance short, which leads to Attacking , and it has the same other ways out as Chasing .
Attacking	A crossroads that lasts no time. It reads How It Fights and leads to Touching , Swinging or Deciding How To Shoot .
Touching	Switches its Hit Box off, sends the event CaughtYou to every Machine in the scene, for anything of your own that should react, waits for Time Between Attacks and switches the Hit Box on again. So its touch hurts once and then not again until the wait is over. Then FINISHED leads back to Chasing .
Swinging	Throws a punch at the one it is after, which reaches as far as Attack Distance and takes Damage , then waits for Time Between Attacks . FINISHED leads back to Chasing , and being hurt leads to Hit .
Deciding How To Shoot	A crossroads that lasts no time. It works out how near is too near while it shoots, then with Circles While Shooting on it leads to Shooting On The Move , and with it off it leads to Shooting .
Shooting	Stays within Shooting Distance of its target, backs off when they come nearer than 160 pixels, or 4 metres in 3D, and fires a bullet every Time Between Attacks . With a short Shooting Distance it backs off at two thirds of it instead. Losing sight leads to Searching , low health leads to Running Away , and being hurt leads to Hit .
Shooting On The Move	The same as Shooting , and it also sidesteps round its target at 80 pixels a second, or 2 metres a second in 3D, changing direction every 2 seconds. It has the same ways out.
Hit	A crossroads entered when it is hurt while it chases, swings or shoots. With Takes Cover on it leads to Taking Cover , and with it off it leads straight back to Chasing .
Taking Cover	Runs to the nearest place marked as cover that hides it, then waits for Hides For . FINISHED leads back to Chasing .
Searching	Walks to where it last saw the one it was chasing and looks round for Searches For . Seeing them again leads straight to Chasing . Finding nobody sends GaveUp and leads to Standing Guard , and being hurt leads to Ambushed .
Ambushed	Entered when it is hurt while it has nobody to go after, or while it searches. It turns its attention to whoever hurt it, which takes no time, and FINISHED leads to Searching , so it walks to where they were when they hurt it. If it sees them on the way it chases them.
Running Away	Runs from the one it was fighting until it is Runs Until away, which sends Safe and leads to Standing Guard .
Dead	Reached from every other state when its health runs out and its Hurt Box sends DIED . It drops its gold and whatever its Loot holds, fades out over 0.6 seconds and is removed from the level.

Make it your own

Give it a health bar over its head

1. In the Scene dock, the list of nodes at the top left of Godot, right-click the **Enemy** and choose **NodeWorthy** → **Give It** → **Health Bar**. A small bar called **HealthBar** appears under the Enemy, with the Enemy picked as its **Character**, **Shows** set to health and **Over Its Head** ticked.
2. Press **F6**. The bar rides just above the Enemy's head, drains as the Enemy is hurt, hides while the Enemy is off the screen, and goes when the Enemy does. It works the same in a 2D game and a 3D one.
3. To change how it looks, click **HealthBar** and set **Color**, **Width** and **Thickness** in the Inspector, the panel on the right. Tick **Shows Number** to write the health on it.
4. Right-click the Enemy and choose **Duplicate** for a second enemy, and the copy has a bar of its own, because the bar sits under the Enemy.
5. A character with no health yet is given health too, the same as **NodeWorthy** → **Give It** → **Health** would.

A guard that patrols between two points

1. Click the **Enemy**. At the top of the Inspector, open **When Idle** and set **When Idle** to **Patrols**.
2. Drag the Enemy to where its beat should be, and its two points come with it. Then in the Scene dock click the arrow beside **Enemy**, then the arrow beside **PatrolPoints**. Click **Point1** and drag it in the view to one end of its beat, then drag **Point2** to the other end.
3. In a side-on level keep both points on the same floor, because a patroller that reaches the edge of its floor turns back.
4. Set **Idle Speed** for how fast it walks and **Wait At Each Point** for how long it stands at each end.
5. For a guard that can be crept up on, open **Seeing** and lower **FOV** from 360 to about 120. It then sees only the way it faces.
6. Press F6 and watch it walk its beat.

An enemy that shoots from a distance

1. Click the **Enemy**. Open **Fighting** and set **How It Fights** to **Shoots**.
2. Set **Shooting Distance** to how far away it should stand. It walks up to that distance and no closer, and it backs off if you get within 160 pixels, or 4 metres in 3D, or within two thirds of a shorter Shooting Distance.
3. Set **Time Between Attacks** for the gap between bullets, **Bullet Speed** for how fast they fly and **Damage** for what each one takes.
4. Leave **Inaccuracy** at 0 for shots that are always aimed true, or raise it to let some go wide.
5. Open **Tactics**. Leave **Circles While Shooting** on for an enemy that sidesteps round you, or turn it off for one that moves only to keep its distance.
6. Its touch does not hurt while **How It Fights** is **Shoots**, so its bullets are the only danger.
7. To change how the bullet looks, open the bullet scene in your project's bullets folder, which is called `bullet_2d`, or `bullet_3d` in a 3D game.

A tougher enemy with more health

1. Click the enemy that should be tough. Open **Health** and raise **Max Health**, to 10 for a small boss. It starts the level with that much.
2. Open **Fighting** and raise **Damage**, which is what its touch, its swing or its bullet takes.
3. Under **Health**, set **Knocked Back By** to 0 so that hits no longer push it, and tick **Shows Damage** to see a number rise from it with every hit.
4. In 2D, open **Look** and pick a **Tint**, so the player can tell it from the others. Set **XP Worth** under **Worth**, and **Gold Least** and **Gold Most** under **Loot**, for a better reward.
5. These numbers belong to this one enemy. For a setting that comes from the template, the arrow beside it puts the template's value back.
6. To give every Enemy the same numbers, press **Use These For Every Copy** under the settings. It copies the settings that come from the template. The ones that belong to the nodes under the enemy, such as **Touch Knockback**, **Knocked Back By**, **Tint** and **XP Worth**, stay with this enemy alone.

One enemy that behaves differently from the rest

1. Every Enemy in a scene runs one template, so a change to the template changes them all. To change how one of them behaves, give it a template of its own first.
2. In the Scene dock click the arrow beside that enemy and right-click the node called **SCMachine** under it, which is its Machine. Choose **NodeWorthy** → **New Template for This Machine...** from the menu.
3. Type a name, such as Boss. Under **Start from**, pick the row that reads **A copy of Enemy · enemy**. The last word is the file name of your own template, and rows that end with anything else are templates that came with NodeWorthy. Press **Create**.
4. That enemy now runs Boss and keeps the settings you gave it. The others still run Enemy, and so does the next Enemy you add.
5. Click the boss and press **Open Its State Machine...** under its settings. The NodeWorthy panel shows Boss as a drawing of its states, with an arrow for every way from one to the next.
6. Click the **Dead** box. The **Actions** tab on the right lists what happens there, which is **Drop Loot**, **Tween Property** and **Queue Free**. Click the name **Tween Property** to open it and change **Duration** from 0.6 to 2, so the boss takes two seconds to fade.
7. Add states and actions of your own in the same way. Nothing you change in Boss touches the other enemies.

Give it cover to hide behind

1. Right-click the top node, choose **Add Child Node...** and pick Node2D, or Node3D in a 3D scene. Rename it CoverPoints.
2. Right-click **CoverPoints** and add two or more Marker2D nodes under it the same way, or Marker3D in 3D. Drag each one behind a crate or a wall end the enemy could hide behind.
3. Click the **Enemy**, open **Tactics** and turn on **Takes Cover**. Then press **Open Its State Machine...** under its settings.
4. In the NodeWorthy panel, click the **Taking Cover** box, and in the **Actions** tab click the name **Take Cover** to open it.
5. Press the button beside **Cover Group** and pick the line that begins **Everything under CoverPoints**.
6. Give the player a way to hurt it, such as **NodeWorthy** → **Give It** → **Gun** on the player, and run the game. When it is hit it runs to the nearest marker within 320 pixels, or 8 metres in 3D, that a wall hides from you, waits there for **Hides For** and comes back.
7. Every Enemy in the scene runs the same template, so each one with **Takes Cover** on now uses these markers.

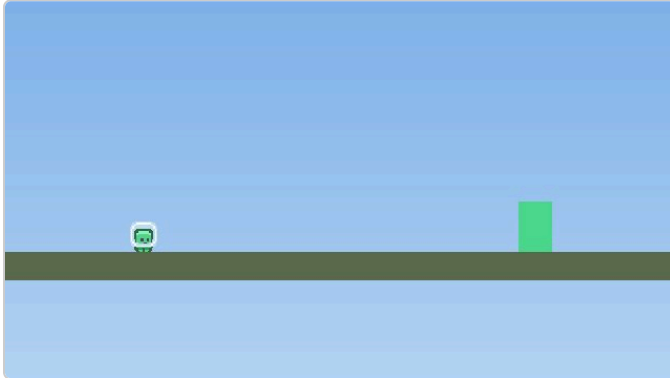
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The enemy reaches the player, or a Villager, and nothing happens	A hit lands only on something that has a Hurt Box. A yellow warning sign beside HitBox in the Scene dock says who has none. Right-click that character and choose NodeWorthy → Give It → Health . A ready-made player is given health when the Enemy is added, but a Villager is not.
Takes Cover is on and it only freezes when it is hit	Nothing in the scene is marked as cover yet. The problems button in the NodeWorthy panel's toolbar, which shows a tick or a count, says that Cover Group finds nothing. Follow the steps under Give it cover to hide behind.
A guard no longer walks back to its place, or one set to Wanders stands still	PatrolPoints was deleted from under the enemy. The problems button in the NodeWorthy panel's toolbar, which shows a tick or a count, says that the Patrol in Standing Guard walks to PatrolPoints, which is not in this scene, and to put it back with Undo. For one that patrols it names Point1 and Point2 , and for one that wanders its home. Press Ctrl+Z if it has only just gone. Otherwise right-click the Enemy , choose Add Child Node... , pick Node2D, or Node3D in a 3D scene, and name it PatrolPoints. For one that patrols, add two Marker2D nodes under it named Point1 and Point2, or Marker3D in 3D, and drag them to the ends of its beat.
In a 2D game seen from above it falls down the screen	It was added before the player, so it was built for a side-on level, where things fall. NodeWorthy says so when the hero is added after it. Delete it, and with the Top-Down Hero (2D) in the scene add the Enemy again.

See also **Bar** 127, **Hurt Box** 180, **Hit Box** 200, **Perception** 191, **Inventory** 185, **Villager** 97

Level Exit THING

A doorway out of the level. The player walks into it, the screen fades, and the game goes to the scene picked for it under **Goes To**. Until a scene is picked it only waits.



Level Exit as the game runs.

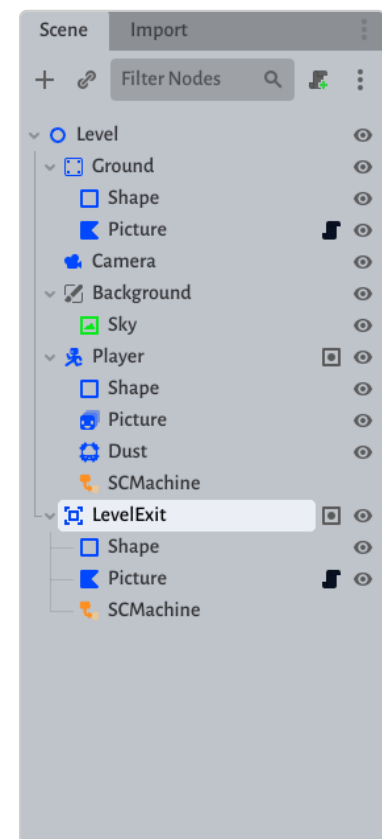
What it is good for

Use it for the end of a platformer level, a cave mouth, a staircase to the next floor or a door between two rooms kept in separate scenes. Two of them, one in each scene, join the scenes both ways. To start the game from a title screen, the **Main Menu** is the better pick, because its Start button already goes to the first level. For something the player collects, use the **Item Pickup**.

Add it

1. Start with a level that has a player, and a second scene saved in the project for the exit to lead to.
2. In the Scene dock, the list of nodes at the top left, right-click the top node and choose **NodeWorthy** → **Add Thing** → **Level Exit**.
3. A green block the height of a person lands far to the right of the node you clicked, 960 pixels away in 2D and about 19 metres away in 3D. Drag it to where the level should end.
4. Click **LevelExit** in the Scene dock. At the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, open the section **Going**.
5. Press the list beside **Goes To**, which reads **Pick a scene...**, and pick the scene to go to. Scenes are listed by file name, so level_2 reads Level 2. Two scenes of one name in two folders have their folder after the name in brackets.
6. Leave **Arrive At** empty for now. The player then starts the next scene wherever that scene has them.
7. Press F6, or the **Run Current Scene** button at the top right of Godot.

Try it. Walk the player into the green block. The screen fades to black over half a second, the other scene takes its place, and the picture fades back in. Nothing has to be pressed.

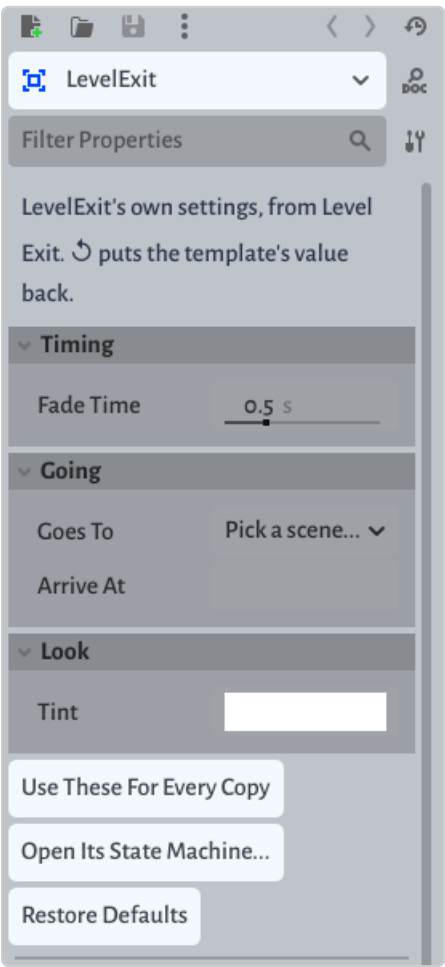


The Scene dock once it is added.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
LevelExit	Area2D	The doorway. An Area2D is Godot's node for a space that is not solid and notices what walks into it. This one watches for the player. In 3D it is an Area3D.
Shape	CollisionShape2D	The space that counts as the doorway, a rectangle 48 pixels wide and 72 tall standing on the floor. Drag its handles to make the doorway bigger. In 3D it is a box 1.2 metres wide and 2.2 tall.
Picture	Polygon2D	The green shape you see, which follows the Shape in the editor. A Polygon2D is Godot's node for a flat coloured shape. In 3D it is a MeshInstance3D, Godot's node that draws a 3D shape.
SCMachine	Machine	The Machine, which is the node that runs the exit's template. The template is saved as Level Exit in your project's nodeworthy folder, unless you have picked another folder for templates, and every Level Exit in the scene runs the same one.

Its settings



Its settings in the Inspector.

Click **LevelExit** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot’s own settings, in sections that fold.

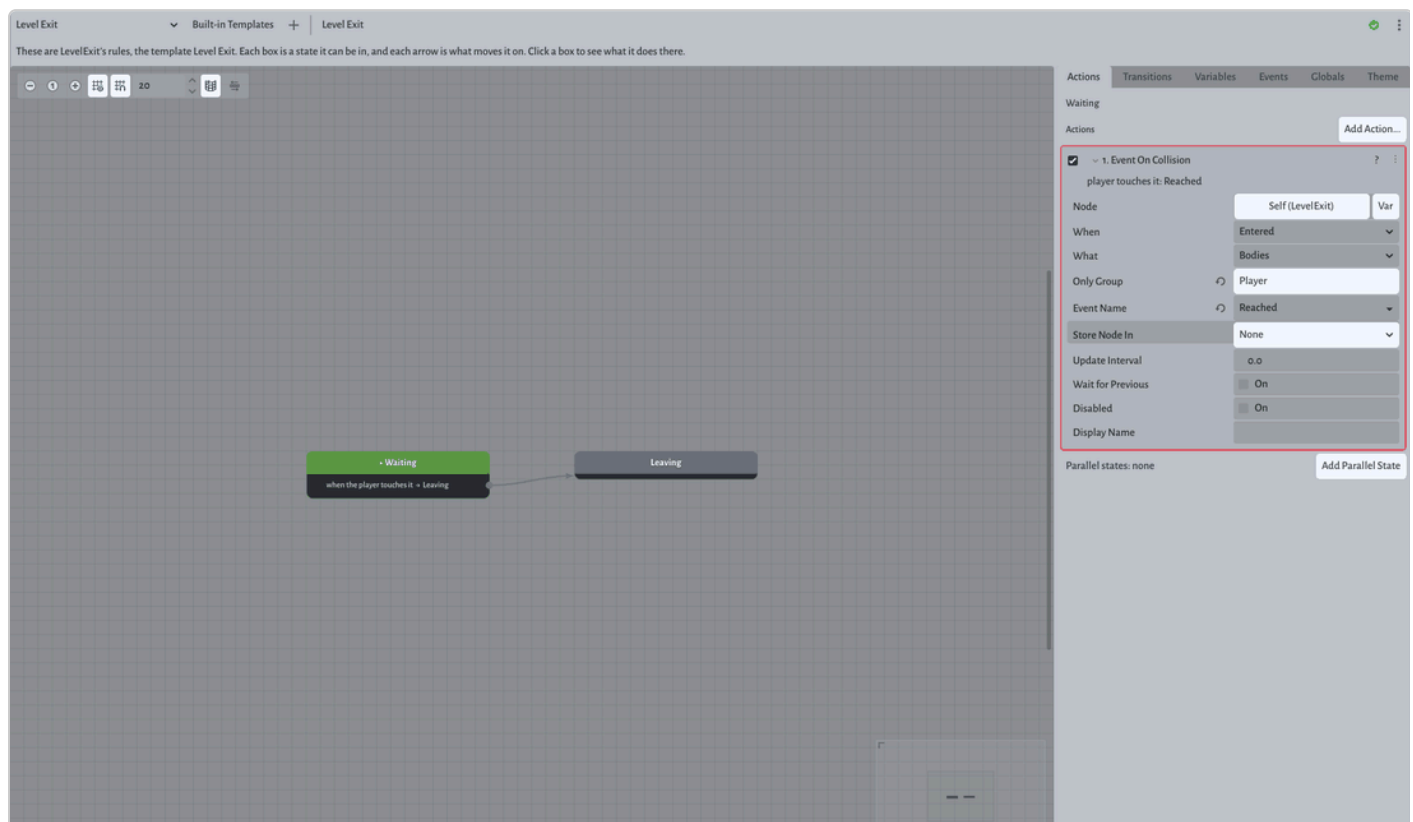
A change here changes this one only. The arrow that appears beside a changed setting puts the template’s value back. **Use These For Every Copy**, under the settings, makes them the template’s own, for every copy.

SETTING	WHAT IT DOES	STARTS AS
TIMING		

SETTING	WHAT IT DOES	STARTS AS
Fade Time	How long the screen takes to fade out, and the same again to fade back in once the new scene is there. It starts at half a second. At 0 the scene changes with a hard cut, and a larger number makes the change feel slower.	0.5 s
GOING		
Goes To	The scene this exit leads to, picked from a list of the scenes saved in your project. It starts empty, and each exit keeps its own, so two exits that share the template can lead to different places. The Change Scene action in Leaving reads the scene from here, so there is nothing to pick in the state itself. Choices: Pick a scene..., Enemy, Exit, Ground, Level, Platform Hero, Platform Hero 3d, Side On Hero, Terrain, Third Person Hero, Top Down Hero, Top Down Hero 3d.	Pick a scene...
Arrive At	The name of a node in the scene gone to, where the player is put as they arrive, typed exactly as the node is named. Use a Marker2D for a 2D player and a Marker3D for a 3D one, and in 3D the player also turns to face the way the marker faces. Left empty, the player starts where that scene has them.	empty
LOOK		
Tint	A colour laid over its picture, so one set of art makes a skeleton, a ghost and a boss. White shows the picture as drawn. It is kept on its Picture node.	white

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.



Its template in the NodeWorthy panel.

STATE	WHAT HAPPENS IN IT
Waiting	Watches for the player. When the player touches the exit it sends the event Reached , and the transition on Reached leads to Leaving .
Leaving	Runs one action, Change Scene , which goes to the scene in Goes To , puts the player at the spot named in Arrive At , and fades over Fade Time . It has no way out, because the scene it is in is replaced.

Make it your own

Join two levels both ways

1. Have two scenes saved, such as level_1 and level_2, each with a player and a Level Exit. Put the first level's exit at its right-hand end and the second level's exit at its left-hand end.
2. Open level_2. Right-click the top node, choose **Add Child Node...** and pick Marker2D, which is Godot's node for a spot with nothing drawn in the game. In a 3D level pick Marker3D. Rename it FromLevelOne and drag it to where the player should appear, a few steps clear of that level's exit so they do not arrive standing in the doorway.
3. Open level_1 and add a marker there the same way, called FromLevelTwo, a few steps clear of its exit.
4. Still in level_1, click **LevelExit**. Under **Going**, set **Goes To** to Level 2 and type FromLevelOne into **Arrive At**.
5. Open level_2 and click its **LevelExit**. Set **Goes To** to Level 1 and type FromLevelTwo into **Arrive At**.
6. Open level_1 and press F6. Walk into the exit and you come out at FromLevelOne in the second level. Walk into that level's exit and you come out at FromLevelTwo, beside the door you first went through.

Keep the player's health and items from level to level

1. A score or anything else kept in a global variable carries on by itself. The player's own health, variables and Inventory need one tick, and the same template in both levels.
2. Click the player in the first level and press **Open Its State Machine...** under its settings. In the NodeWorthy panel open the **Variables** tab and tick **Carries Over Between Scenes** at the top.
3. Open the second level. Right-click its player and choose **NodeWorthy** → **Behave Like**, then the template the first level's player runs, which is named with that level. The second player takes what the first carried only when both run the same template.
4. Run the first level, lose a point of health and walk through the exit. The player arrives with the health they left with.

Give it your own size and picture

1. Set the size first. Click **Shape** under **LevelExit** and drag its handles, and the green picture follows.
2. For your own picture in 2D, right-click **LevelExit** and choose **NodeWorthy** → **Looks** → **Set Texture...** and pick a picture from anywhere on your computer. It is copied into your project's pictures folder and sized to the exit's height. A picture of your own no longer follows the Shape.
3. In 3D the same menu offers **Looks** → **Set Model...** for a model of your own.
4. For a doorway nobody can see, such as the edge of the level, click **Picture** in the Scene dock and press the eye beside it. The Shape still notices the player.

Fade to another colour

1. Click **LevelExit** and press **Open Its State Machine...** under its settings. The NodeWorthy panel shows its template, which is two states, **Waiting** and **Leaving**, and the arrow between them.
2. Click the **Leaving** box. The **Actions** tab on the right shows its one action, **Change Scene**.
3. Click the name **Change Scene** to open its settings, and pick a colour beside **Fade Color**, such as white for a flash of light.
4. Every Level Exit in this scene runs the same template, so they all fade to that colour now.

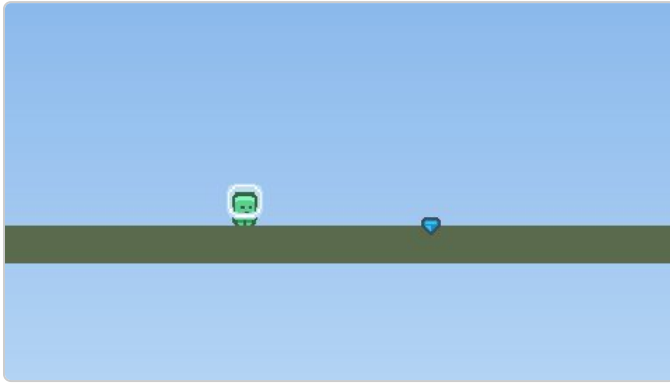
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The player walks in and nothing happens	No scene is picked. A warning in the Output panel, at the bottom of the editor, says that Change Scene has no scene to go to and names Goes To . Click the exit and pick a scene under Going .
The scene you want is not in the Goes To list	The list holds the scenes saved in your own project folders, and it is read when the exit is clicked. Save the other scene, click any other node, then click the exit again.
The player arrives at the start of the next scene and not at the spot you named	The name in Arrive At matches no node there, and a warning in the Output panel says there is no spot of that name. Check the spelling against the marker's name in the other scene. Another warning appears when a 2D player is sent to a Marker3D, or a 3D player to a Marker2D, so use the marker that matches the player.
The player arrives with full health and an empty Inventory	Each scene has its own copy of the player, and nothing is carried between them until Carries Over Between Scenes is ticked and both players run the same template. The Output panel says so when something was carried and the player in the new scene did not take it. Follow the steps under Keep the player's health and items from level to level.
A player you built yourself walks through it and is not noticed	The exit waits for the player, which to NodeWorthy means a node in Godot's group called player. Every ready-made hero is in it. For your own, press Open Its State Machine... , click Waiting , open Event On Collision , press the button beside Only Group and pick your player from the scene.

See also [Main Menu](#) 111, [Pause Menu](#) 116, [Item Pickup](#) 91, [Ground](#) 62

Item Pickup THING

Something lying in the level for the player to collect. When the player touches it, the item picked for it goes into the player's Inventory and the pickup disappears. Adding one gives the player an Inventory if they have none.



Item Pickup as the game runs.

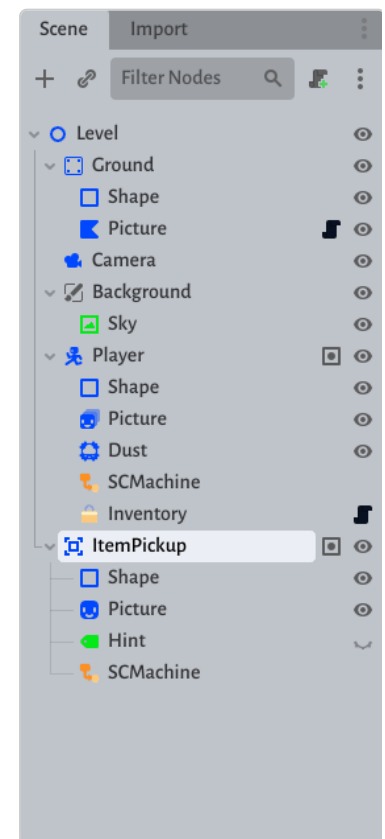
What it is good for

Use it for a key, a potion, a sword on the ground, or anything else the player should carry and use later. Each pickup holds its own item, so one template serves every kind. For loot the player opens with a key press, with gold and a chance of each item, the **Chest** is the better pick. An **Enemy** has Loot settings of its own for what it drops.

Add it

1. Start with a level. Its player can be there already or be added afterwards, because the pickup gives its item to whichever player touches it.
2. In the Scene dock, the list of nodes at the top left, right-click the top node and choose **NodeWorthy** → **Add Thing** → **Item Pickup**.
3. A small purple square lands 200 pixels to the right of the node you clicked, or 4 metres in 3D, and the player gets an **Inventory** node under them if they had none. A player added after the pickup is given one as they are added. Drag the pickup to where it should lie.
4. Click **ItemPickup** in the Scene dock. At the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, open the section **Item** and press its button, which reads **Pick an item...** until something is picked.
5. Choose **New item...**, type a name such as Key and press **Make it**, which saves the item in your project's items folder and picks it. To use an item you already have, choose **Pick an item...** from the same list.
6. Press the button again and choose **Open Key** to give the item a **Description** and an **Icon** in the Inspector. The Inspector's back arrow returns to the pickup.
7. To see what the player carries, click the **Inventory** under the player and press **Show It On Screen**, which adds an Inventory Grid to the screen.
8. Press F6, or the **Run Current Scene** button at the top right of Godot.

Try it. Walk the player onto the purple square. It vanishes, and the item appears in the Inventory Grid in the bottom left corner of the screen, if you added one. Nothing has to be pressed. To make the player press a key for it,



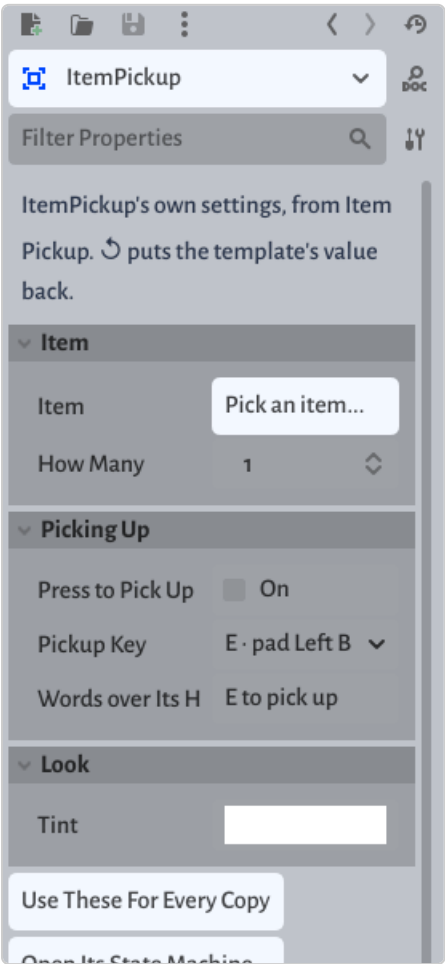
The Scene dock once it is added.

tick **Press To Pick Up** in its Inspector, walk up to it, and press **E** when "E to pick up" shows.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
ItemPickup	Area2D	The pickup. An Area2D is Godot's node for a space that is not solid and notices what walks into it. It stands on the collision layer called Pickups and watches for the player. In 3D it is an Area3D.
Shape	CollisionShape2D	How close the player has to come, a square 20 pixels across. In 3D it is a box 0.45 metres across.
Picture	Sprite2D	The purple square you see, which follows the Shape in the editor. A Polygon2D is Godot's node for a flat coloured shape. In 3D it is a MeshInstance3D, Godot's node that draws a 3D shape.
Hint	Label	The words "E to pick up" over it, hidden unless Press To Pick Up is ticked and the player is near. A Label is Godot's node for a line of words. In 3D it is a Label3D that always faces the camera.
SCMachine	Machine	The Machine, which is the node that runs the pickup's template. The template is saved as Item Pickup in your project's nodeworthy folder, unless you have picked another folder for templates, and every Item Pickup in the scene runs the same one.
Inventory	Inventory	Added under the player when they had none. It holds what the player carries, and the pickup puts its item here.

Its settings



Its settings in the Inspector.

Click **ItemPickup** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot’s own settings, in sections that fold.

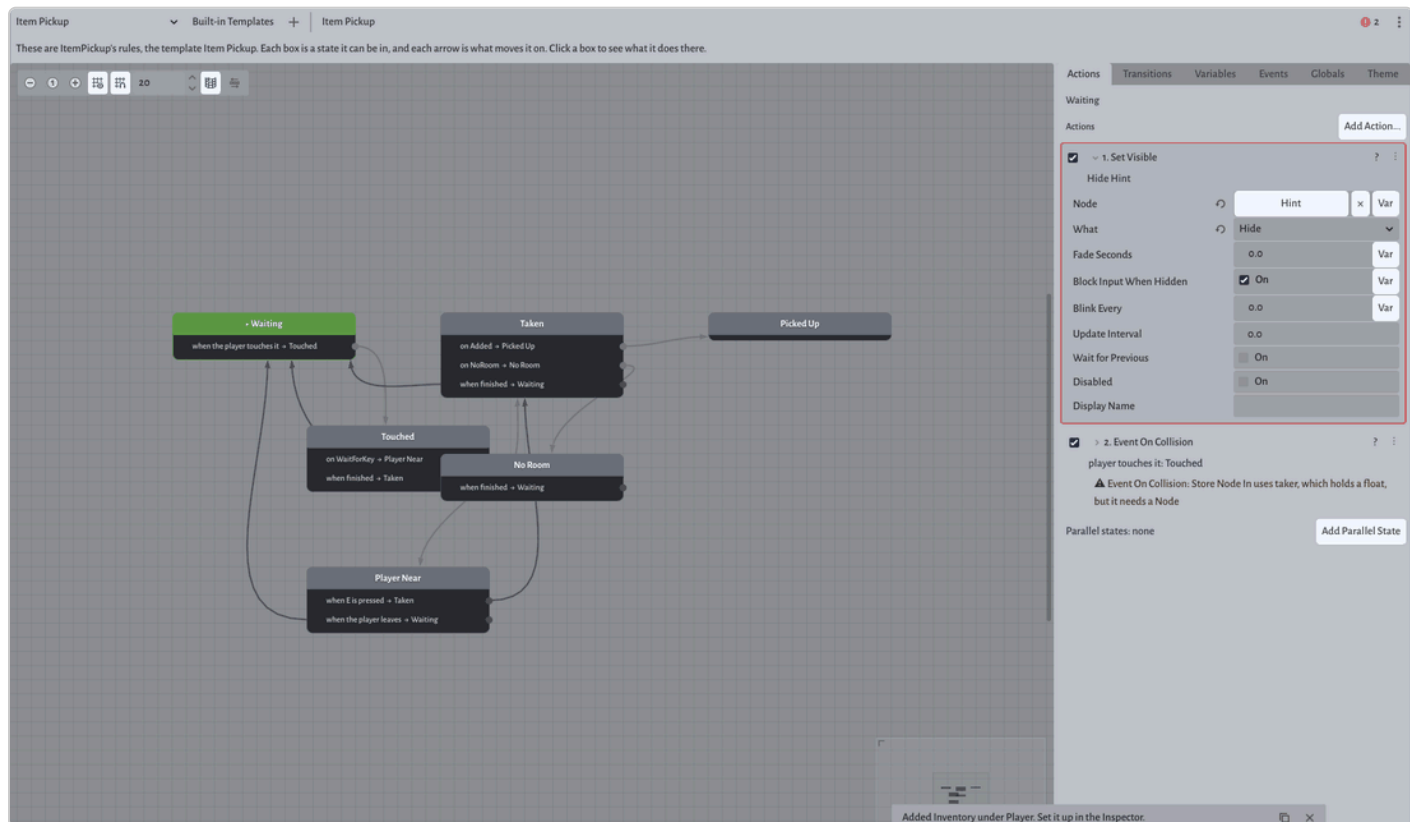
A change here changes this one only. The arrow that appears beside a changed setting puts the template’s value back. **Use These For Every Copy**, under the settings, makes them the template’s own, for every copy.

SETTING	WHAT IT DOES	STARTS AS
ITEM		
Item	What goes into the player's Inventory. It starts empty, and a pickup with no item stays on the floor when touched and says so in the Output panel. Press its button to pick an item from your project or to make one with New item... , and each pickup keeps its own choice.	nothing picked
How Many	How many of the item the player gets from this one pickup. It starts at 1, and a larger number suits a bundle of arrows. At 0 it gives nothing, stays on the floor and prints a warning.	1
PICKING UP		
Press to Pick Up	Off to start with, so the player picks it up by walking into it, as suits gems and coins. Tick it and the player walks up to it instead and presses Pick Up Key , while "E to pick up" shows over it. Walking away hides the words again.	off

SETTING	WHAT IT DOES	STARTS AS
Pickup Key	The key, and the gamepad button that goes with it, that picks it up while the player is near, when Press To Pick Up is ticked. It starts as E with the left face button (X or Square). The words over it follow it, so E to pick up becomes F to pick up. Choices: E · pad Left Button (X / Square), Mouse · Left Button · pad Right Trigger, C · pad Right Button (B / Circle), E · pad Top Button (Y / Triangle), F · pad Left Button (X / Square), Space · pad Bottom Button (A / Cross), Enter · pad Bottom Button (A / Cross), J · pad Left Button (X / Square), K · pad Right Button (B / Circle), Shift · pad Right Bumper, Ctrl · pad Right Stick Press, Escape · pad Start, Tab · pad Select / Back, P · pad Start.	E · pad Left Button (X / Square)
Words over Its Head	What shows over its head while the player is near. It is kept on its Hint node.	E to pick up
LOOK		
Tint	A colour laid over its picture, so one set of art makes a skeleton, a ghost and a boss. White shows the picture as drawn. It is kept on its Picture node.	white

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.



Its template in the NodeWorthy panel.

STATE	WHAT HAPPENS IN IT
Waiting	Watches for the player. When the player touches the pickup it keeps who touched it in the variable taker and sends the event Touched , which leads to the state of the same name.

STATE	WHAT HAPPENS IN IT
Touched	The player has walked into it. With Press To Pick Up off it goes straight on to Taken . With it ticked it goes to Player Near instead.
Player Near	Shows "E to pick up" over it and waits. Pick Up Key leads to Taken , and the player walking away hides the words and goes back to Waiting .
Taken	Runs Add Item , which puts How Many of the Item into the Inventory of the player kept in taker. When all of it went in, Added leads to Picked Up . With no room, NoRoom leads to No Room . With no item picked, How Many at 0 or no Inventory on the player, Add Item says which in the Output panel, at the bottom of the editor, and FINISHED leads back to Waiting .
Picked Up	Runs Queue Free , which removes the pickup from the level. It has no way out, because the pickup is gone.
No Room	Runs Print , which says in the Output panel that the pickup stays on the floor because the Inventory has no room for its item. FINISHED leads back to Waiting , so walking onto it again tries again.

Make it your own

A second pickup that holds something else

1. In the Scene dock, the list of nodes at the top left, right-click the top node and choose **NodeWorthy** → **Add Thing** → **Item Pickup** once more. It lands a little further along than the first, never on top of it, and shares the first one's template, so the two behave alike.
2. Click the new one. Under **Item**, press the button and pick or make another item, and set **How Many**.
3. In 2D, open **Look** and pick a **Tint**, so the player can tell the two apart before either has a picture of its own.

Give it your own picture

1. The picture is sized to the height of the pickup's Shape, which starts at 20 pixels. For a bigger pickup, click **Shape** under it and drag its handles out first.
2. Right-click **ItemPickup** and choose **NodeWorthy** → **Looks** → **Set Texture...** and pick a picture from anywhere on your computer. It is copied into your project's pictures folder.
3. In 3D the same menu offers **Looks** → **Set Model...** for a model of your own.

A pickup that can be taken again and again

1. Click **ItemPickup** and press **Open Its State Machine...** under its settings. The NodeWorthy panel shows its template, which is four states, **Waiting**, **Taken**, **Picked Up** and **No Room**.
2. Click the **Picked Up** box. In the **Actions** tab on the right, find **Queue Free**, whose line reads Destroy Self. Press : on it and choose **Remove**.
3. Right-click the **Picked Up** box and choose **Add Transition**. In the **Transitions** tab press **When** and pick **When everything in this state has finished**, then pick **Waiting** under **To State**.
4. Run the game. Stepping onto the pickup gives the item, and stepping off and on again gives another.
5. Every Item Pickup in the scene runs this template, so they all stay now. To change only one, right-click its **SCMachine** node first and choose **NodeWorthy** → **New Template for This Machine...** from the menu.

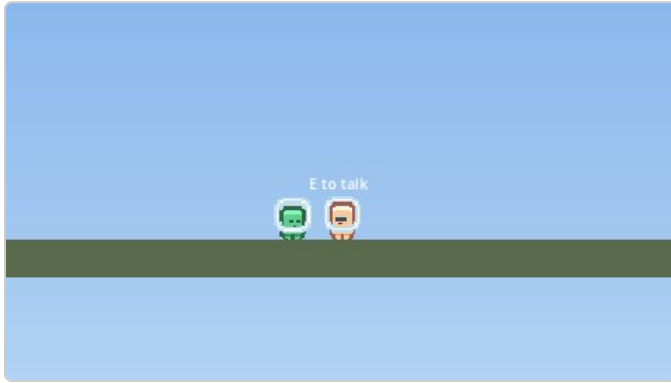
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
It stays on the floor when the player walks onto it	Its item did not go in, and the Output panel, at the bottom of the editor, says why. With no item picked, Add Item says its Item holds no item, so click the pickup and pick one under Item . With How Many at 0 it says that nothing is put in. With no Inventory on the player it says there is none, so right-click the player and choose NodeWorthy → Give It → Inventory . Then walk off the pickup and onto it again.
The item is taken and nothing shows on the screen	An Inventory holds things without drawing them. Click the Inventory under the player and press Show It On Screen , which adds an Inventory Grid.
The player's Inventory is full	An Inventory with Slots set can run out of room. The pickup then stays on the floor, and the Output panel says there is no room for its item. Once the player has made room, walking off the pickup and onto it again picks it up. With How Many above 1, when only some of them fit, those go in and the pickup stays with its full count. Leave Slots at 0, which holds as much as it needs, unless running out of room is part of the game.

See also **Inventory** 185, **Inventory Grid** 141, **Chest** 104, **Enemy** 74

Villager THING

Someone for the player to talk to. It stands still until the player walks up, shows "E to talk" over its head, and on **E** plays the dialogue you picked and then hands over a quest if you gave it one. It works the moment it is added, and it has nothing to say until you pick a dialogue.



Villager as the game runs.

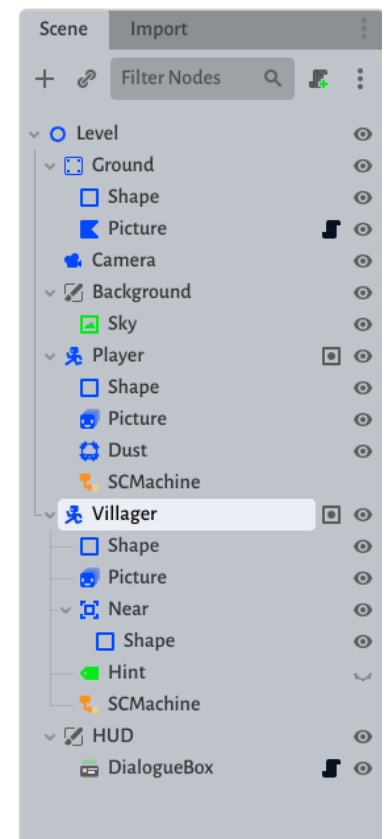
What it is good for

A shopkeeper, a farmer with a job to give, a guard with a warning, or anyone in a town the player can walk up to, in 2D, in 2D seen from above, or in 3D. It does not walk and it does not fight. An **Enemy** whose **Attacks** is set to **Villagers** goes after it. For something the player opens for loot, add a **Chest**.

Add it

1. Start with a level that has ground and a player, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts ground and a **Platform Hero (2D)** in it.
2. In the Scene dock, the panel at the top left that lists the nodes of the scene, right-click the top node and choose **NodeWorthy** → **Add Thing** → **Villager**.
3. A character called **Villager** lands to one side of the node you right-clicked, standing on the ground. Drag it in the view to where you want it.
4. Look at the Scene dock again. A **DialogueBox** has been added to the screen as well, because the Villager's words need somewhere to show. In the editor it shows an example line from a guard, and in the game it is hidden until someone talks.
5. Click **Villager** in the Scene dock. Its settings are at the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, under **Talking**.
6. Press the box beside **Dialogue**, choose **New dialogue...**, type a name and press **Make it**. The Villager now has one line to say.
7. Press F6, or the Run Current Scene button at the top right of Godot.

Try it. Walk up to the Villager. "E to talk" shows over its head, and hides again if you walk away. Press **E** and a box along the bottom of the screen shows who is speaking and their words, a letter at a time, while the player is held still. Press **Enter** to read on, and when the talk ends the words over its head come back.



The Scene dock once it is added.

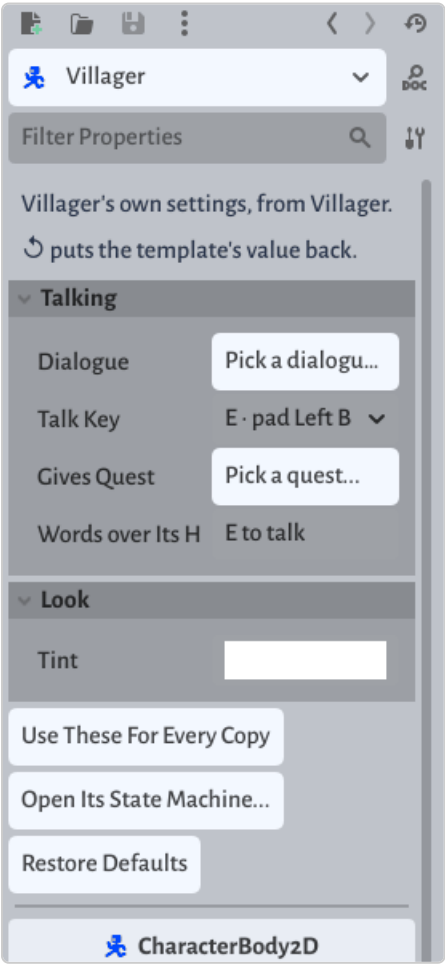
What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Villager	CharacterBody2D	The Villager itself. A CharacterBody2D is Godot's node for a character that is stopped by walls, and in 3D it is a CharacterBody3D. It is solid and on the World layer, as walls and ground are, so the player cannot walk through it. It is also in the group villager, which is how an Enemy set to attack Villagers finds it.
Shape	CollisionShape2D	What is solid about it, a box 24 pixels wide and 40 tall. In 3D it is a CollisionShape3D, 0.8 metres wide and 1.8 tall. Click it and drag its handles in the view to resize the Villager, and the plain picture follows.
Picture	AnimatedSprite2D	What you see. Seen from the side it is a little helmeted character from the free art that comes with NodeWorthy, and in a game seen from above a girl in purple, who hops a little with each step and breathes while she stands. An AnimatedSprite2D is Godot's node for a picture that steps through frames. In 3D it is an animated person tinted green, which gestures as it talks. Looks in its right-click menu puts art of your own in its place.
Near	Area2D	The space round the Villager that notices the player. An Area2D is Godot's node for a space that knows what is inside it, and in 3D it is an Area3D.
Shape	CollisionShape2D	Under Near . A circle 70 pixels in radius, or a ball 2 metres in radius in 3D. This is how close the player must come, so make it bigger to be noticed from further off.
Hint	Label	The words over its head, "E to talk", hidden until the player is near. A Label is Godot's node for words. In 3D it is a Label3D that always turns to face the camera.
SCMachine	Machine	NodeWorthy's Machine node, which runs the Villager's template. The template is the drawing of states and actions that makes it behave, and it is saved in your project's nodeworthy folder, unless you have picked another folder for templates.
DialogueBox	Dialogue Box	Not under the Villager. It is put on the screen, along the bottom, when a Villager goes into a scene that has none. It goes under the first CanvasLayer the scene draws over the game. A CanvasLayer is Godot's node for things drawn on the screen and not in the world, and one called HUD is made for the box when the scene has none.
QuestList	Quest List	Not under the Villager, and not there at first. It is put in the top right corner of the screen the first time you pick a quest in Gives Quest , if the scene has none. It lists the quests under way and ticks their steps off.
Sun	DirectionalLight3D	Only in a 3D scene that has no light yet. A DirectionalLight3D is Godot's light that shines one way over the whole scene, as the sun does. Without it a 3D game runs almost black.
Sky	WorldEnvironment	Only in a 3D scene that has no light yet. A WorldEnvironment is Godot's node for the sky and the light that comes from it.

The controls

PRESS	WHAT IT DOES
E, or the left face button on a gamepad (X or Square)	Starts the talk while the player stands near the Villager.
Enter, the Space bar, a click on the box, or the bottom button on a gamepad (A or Cross)	Shows the rest of the line at once. Pressed again, it moves on to the next line.
The arrow keys, or the D-pad or a stick on a gamepad	Move between the answers when a line offers some. Enter, a click, or the bottom button picks one.

Its settings



Click **Villager** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot’s own settings, in sections that fold.

A change here changes this one only. The arrow that appears beside a changed setting puts the template’s value back. **Use These For Every Copy**, under the settings, makes them the template’s own, for every copy.

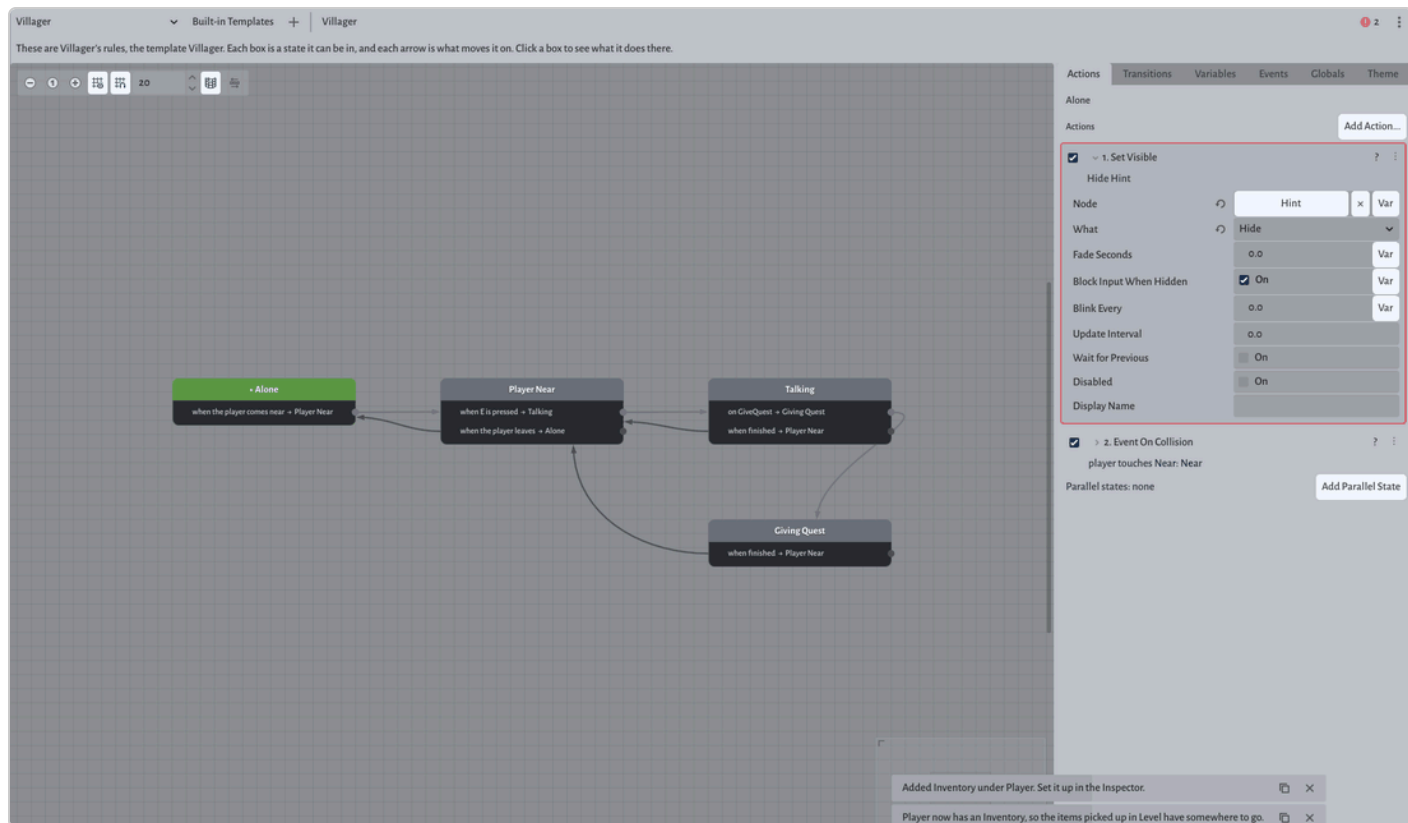
Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
TALKING		
Dialogue	The dialogue it plays when the player presses the key. It starts empty, and until you pick one, pressing E shows nothing. Press it to pick a dialogue already in your project or to make one with New dialogue.... Each Villager keeps its own, even though they share one template.	nothing picked

SETTING	WHAT IT DOES	STARTS AS
Talk Key	The key, and the gamepad button that goes with it, that starts the talk. It starts as E with the left face button, and the list offers fourteen pairs, such as F with the left button. The words over its head follow it, so E to talk becomes F to talk. Words of your own that do not begin with a key's name are left as you typed them. Choices: E · pad Left Button (X / Square), Mouse - Left Button · pad Right Trigger, C · pad Right Button (B / Circle), E · pad Top Button (Y / Triangle), F · pad Left Button (X / Square), Space · pad Bottom Button (A / Cross), Enter · pad Bottom Button (A / Cross), J · pad Left Button (X / Square), K · pad Right Button (B / Circle), Shift · pad Right Bumper, Ctrl · pad Right Stick Press, Escape · pad Start, Tab · pad Select / Back, P · pad Start.	E · pad Left Button (X / Square)
Gives Quest	A quest it hands the player when the talk ends. It starts empty, which gives none. Pick a quest, or make one with New quest... , and a Quest List goes on the screen by itself so the player can see it. A quest already under way or done is left as it is, so talking again does not start it twice.	nothing picked
Words over Its Head	What shows over its head while the player is near. It is kept on its Hint node.	E to talk
LOOK		
Tint	A colour laid over its picture, so one set of art makes a skeleton, a ghost and a boss. White shows the picture as drawn. It is kept on its Picture node.	white

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.



Its template in the NodeWorthy panel.

STATE	WHAT HAPPENS IN IT
Alone	The words over its head are hidden and the Villager waits. When the player walks into Near , the event Near is sent and leads to Player Near .
Player Near	The words over its head show. Pressing the Talk Key sends Talk , which leads to Talking . Walking out of Near sends Gone , which leads back to Alone .
Talking	The words hide and a Start Dialogue action plays the dialogue picked in Dialogue , holding the player still until it ends. Then a check asks whether Gives Quest holds a quest. If it does, GiveQuest leads to Giving Quest , and if it does not, FINISHED leads back to Player Near .
Giving Quest	A Start Quest action begins the quest picked in Gives Quest . It is over at once, and FINISHED leads back to Player Near , where the Villager waits for the next press.

Make it your own

Give it something to say

1. Click **Villager** in the Scene dock.
2. At the top of the Inspector, under **Talking**, press the box beside **Dialogue**, which reads **Pick a dialogue...**, and choose **New dialogue...**
3. Type a name, such as Old Farmer, and press **Make it**. The window says where it is saved, at first your project's conversations folder, and **Folder...** beside it picks another folder, which the next dialogue goes in too. The dialogue is picked for this Villager and opened as a graph in the NodeWorthy panel, where templates and dialogues are drawn, with one line to start from.
4. Click that line's box. At the side, type who is speaking in **Speaker** and what they say in **Text**.
5. Press **+ Line** for the next thing they say. It is joined on after the line you had picked.
6. To let the player answer, pick a line, press **Add Answer** and type the answer. Press **+ Line** again to make the line that answer leads to. A line or an answer whose arrow leads nowhere ends the talk.
7. Press **◀ Templates** to go back to the Villager's template, then run the scene, walk up and press **E**.
8. To change the words later, press **Dialogue** again and choose the line that begins **Open**.

Give it a quest to hand out

1. Click **Villager** in the Scene dock.
2. Under **Talking**, press the box beside **Gives Quest**, which reads **Pick a quest...**, and choose **New quest...**
3. Type the quest's name, such as Find The Lost Cat, and press **Make it**. The window says where it is saved, at first your project's quests folder, and **Folder...** beside it picks another folder. The quest is picked for this Villager and opened in the Inspector with a first step.
4. In the first step, type what the player reads in **Text**, and choose in **Done When** how the game knows the step is done, such as **An Enemy Is Beaten** or **The Player Reaches A Place**.
5. Press **Add Step** for each further thing to do.
6. Look at the Scene dock. A **QuestList** has been added to the screen, in the top right corner.
7. Run the scene and talk to the Villager. When the talk ends, the quest appears in the list, and its steps are ticked off as you play.

Give it your own picture or model

1. In a 2D game, right-click **Villager** in the Scene dock and choose **NodeWorthy → Looks → Set Texture...**
2. Pick a picture anywhere on your computer. **Drawn facing**, at the bottom of the window, says which way the character looks in the picture.
3. The picture is copied into your project's pictures folder, put in place of its picture, and sized to the Villager's height. One undo gives its picture back.
4. For a Villager that moves as it stands, choose **Import Sprite Sheet...** in the same menu. Name one row idle, and that row plays while it waits.
5. In a 3D game, choose **NodeWorthy → Looks → Set Model...** and pick a .glb, .gltf or .fbx file. Say which way the model faces and press **Use It**. It is copied into your project's models folder and sized to the Villager's height.

Change the key and the words over its head

1. Click **Villager** in the Scene dock.
2. Under **Talking**, pick another pair in **Talk Key**, such as **F** with the left button on a gamepad.
3. The words over its head now read F to talk. To say something else, type it in **Words Over Its Head** and press **Enter**.
4. The arrow that appears beside **Talk Key** once it is changed puts the template's key back.

Let the player walk about while it talks

1. Click **Villager** and press **Open Its State Machine...** under its settings. The NodeWorthy panel shows its template, the drawing of states that makes it behave, as four boxes joined by arrows.
2. Click the **Talking** box. The **Actions** tab on the right lists its three actions, a card each.
3. Click the name of the **Start Dialogue** card, whose line begins Have dialogue. Its settings open.
4. Untick **Hold The Player Still**.
5. Run the scene. The player can now walk while the box is on the screen.
6. Every Villager in the scene shares this template, so they all change. To change one only, first right-click its **SCMachine** in the Scene dock and choose **NodeWorthy** → **New Template for This Machine...**

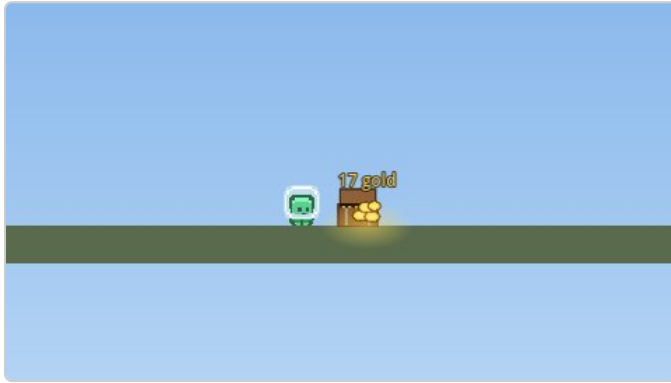
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
"E to talk" shows, but pressing E shows nothing	No dialogue is picked in Dialogue , or the one picked has no lines. Godot's Output panel, at the bottom of the editor, says Start Dialogue has no dialogue to play. Pick one, or make one with New dialogue....
The Output panel says there is no Dialogue Box in the scene	The DialogueBox that came with the Villager was deleted. Right-click the top node and choose NodeWorthy → Add To Screen → Dialogue Box .
The words over its head name a key that is not the Talk Key	The words follow the key only while they begin with a key's name and the word to, as E to talk does. Words of your own are left alone, so type new ones in Words Over Its Head , under Talking .
Nothing shows when a character you made yourself walks up	Near notices only a body that is in Godot's group player and on the Player or World layer. Every player from Add Player is both. Put yours on the layer with NodeWorthy → Collision Layer → Player , and add it to the group player in Godot's Node dock, which lists a node's groups.
The quest is not given a second time	A quest already under way or done is left as it is. Only a quest that failed or was given up begins again when the Villager gives it.

See also **Dialogue Box** 171, **Quest List** 146, **Quest Log** 150, **Enemy** 74, **Chest** 104

Chest THING

A chest the player opens for loot. It is solid and closed until the player walks up and presses **E**, or clicks it, and then it opens, spills gold and any items you gave it onto the floor, and stays open. It works the moment it is added, holding 5 to 20 gold.



Chest as the game runs.

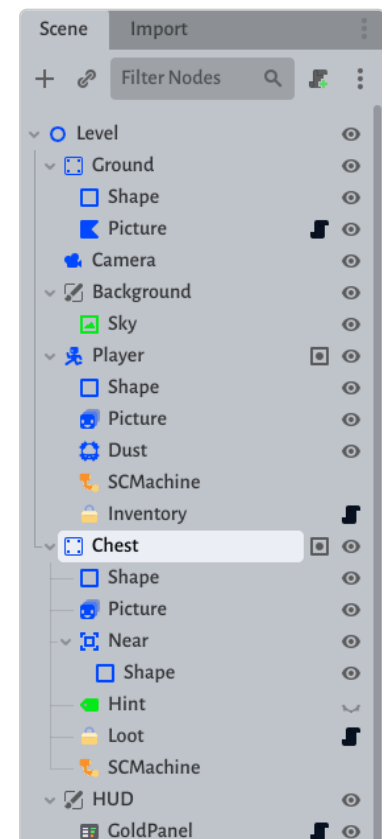
What it is good for

Rewards hidden in a level, in a platformer, a dungeon seen from above or a 3D world. Each Chest in a level can hold different things, picked at random from a list of your own. For one thing that is taken by touching it, such as a coin or a key lying in the open, an **Item Pickup** is the better pick. An **Enemy** has the same **Loot** settings, for what it drops when beaten.

Add it

1. Start with a level that has ground and a player, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts ground and a **Platform Hero (2D)** in it.
2. In the Scene dock, the panel at the top left that lists the nodes of the scene, right-click the top node and choose **NodeWorthy** → **Add Thing** → **Chest**.
3. A closed chest called **Chest** lands to one side of the node you right-clicked, standing on the ground. Drag it in the view to where you want it. The player is given an **Inventory** node under them if they had none, so the items a chest drops can be picked up.
4. Click **Chest** in the Scene dock. Its settings are at the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, in sections that fold. **Loot** holds what it drops.
5. Press F6, or the Run Current Scene button at the top right of Godot.

Try it. Walk up to the chest and "E to open" shows over it. Press **E**. The chest opens and a pile of gold is thrown out towards the player with its amount over it, somewhere from 5 to 20. Gold that lands at the player's feet is picked up at once, and the rest by walking over it. Press **E** again to see that an open chest stays open.



The Scene dock once it is added.

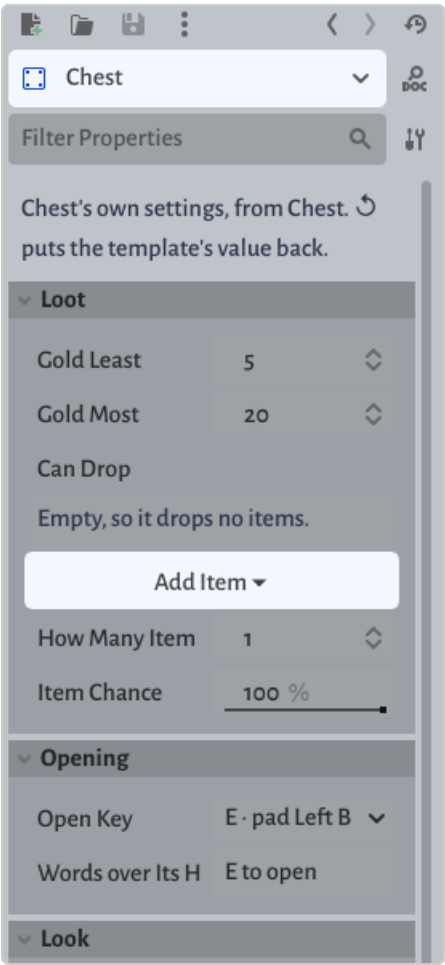
What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Chest	StaticBody2D	The Chest itself. A StaticBody2D is Godot's node for something solid that does not move, and in 3D it is a StaticBody3D. It is on the World layer, so it blocks the way, and on the Pickups layer too, so a player that walks where the mouse is clicked can click it.
Shape	CollisionShape2D	What is solid about it, a box 44 pixels wide and 28 tall. In 3D it is a CollisionShape3D 1 metre wide, 0.6 tall and 0.7 deep.
Picture	AnimatedSprite2D	The drawn chest. An AnimatedSprite2D is Godot's node for a picture with named animations, and this one has two, closed and open , of one picture each. In 3D it is a MeshInstance3D, a plain brown box.
Hinge	Node3D	Only in 3D. An empty node along the back edge of the box, which the lid turns about when the chest opens.
Lid	MeshInstance3D	Only in 3D, under Hinge . The slab on top of the box that swings up.
Near	Area2D	The space round the Chest that notices the player. An Area2D is Godot's node for a space that knows what is inside it, and in 3D it is an Area3D.
Shape	CollisionShape2D	Under Near . A circle 70 pixels in radius, or a ball 1.8 metres in radius in 3D, which is how close the player must stand to open it with the key.
Hint	Label	The words over it, "E to open", hidden until the player is near. A Label is Godot's node for words. In 3D it is a Label3D that always turns to face the camera.
Loot	Inventory	What the chest holds. An Inventory is NodeWorthy's node for what something carries. As the level starts it takes items at random from Can Drop , and everything in it spills out when the chest opens.
SCMachine	Machine	NodeWorthy's Machine node, which runs the Chest's template. The template is the drawing of states and actions that makes it behave, and it is saved in your project's nodeworthy folder, unless you have picked another folder for templates.
Inventory	Inventory	Not under the Chest. It is added under the player when they had none, and it is where the items a chest drops go when the player walks over them. A player added after the Chest is given one as they are added.
GoldPanel	Stat Panel	Not under the Chest. When the scene has a player whose template has a variable called gold, and nothing on the screen shows it, the player's gold is put on the screen. It joins a Stat Panel already there, or a new one called GoldPanel is made. It sits under the scene's screen layer, a CanvasLayer, which is Godot's node for things drawn over the game, and one called HUD is made when the scene has none.
Sun	DirectionalLight3D	Only in a 3D scene that has no light yet. A DirectionalLight3D is Godot's light that shines one way over the whole scene, as the sun does.
Sky	WorldEnvironment	Only in a 3D scene that has no light yet. A WorldEnvironment is Godot's node for the sky and the light that comes from it.

The controls

PRESS	WHAT IT DOES
E, or the left face button on a gamepad (X or Square)	Opens the chest while the player stands near it.
A click on the chest	Opens it once the player has walked up to it, in a game where the player walks where the mouse is clicked, which is what the Click To Move action does.

Its settings



Click **Chest** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot's own settings, in sections that fold.

A change here changes this one only. The arrow that appears beside a changed setting puts the template's value back. **Use These For Every Copy**, under the settings, makes them the template's own, for every copy.

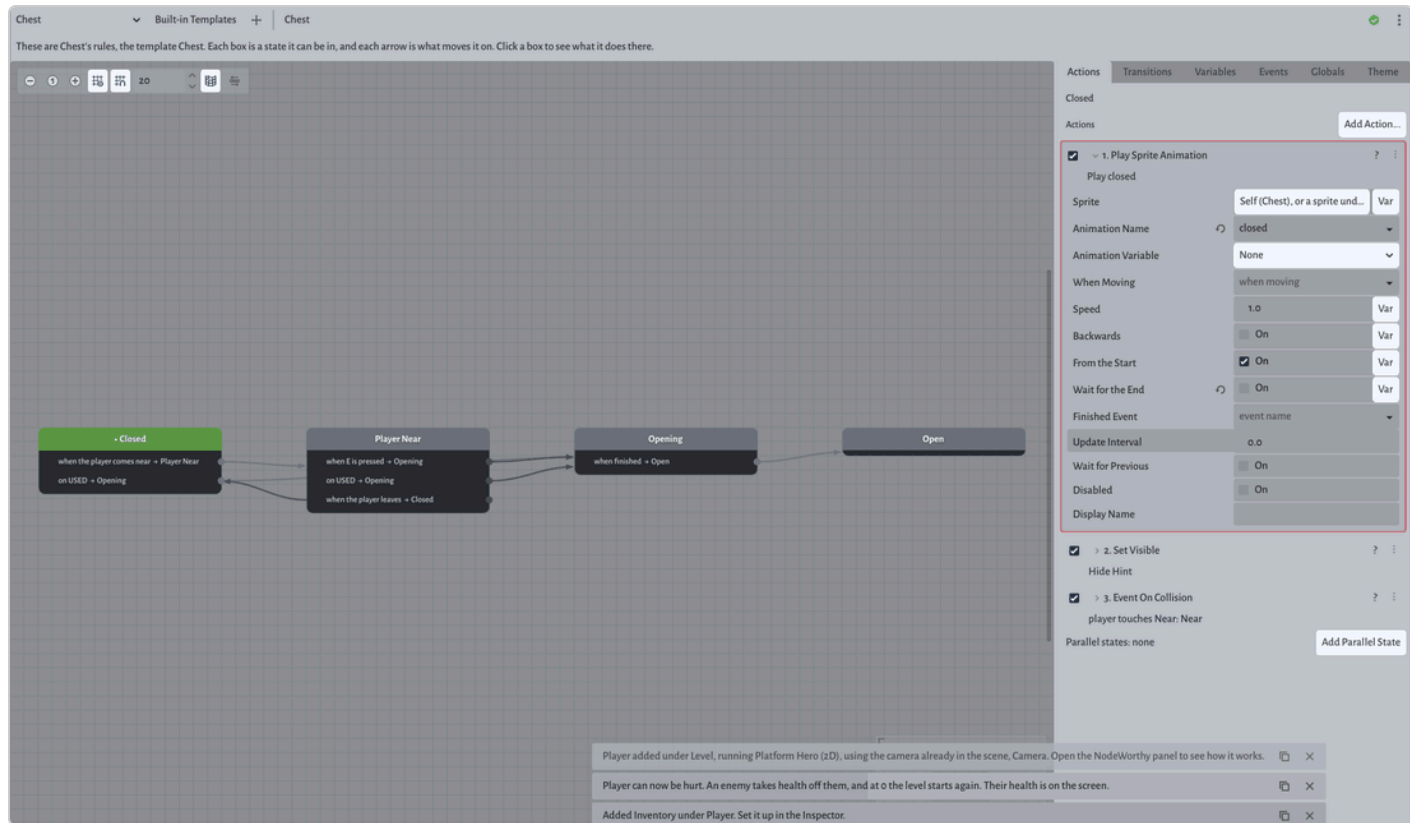
Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
LOOT		
Gold Least	The least gold the chest can hold. It starts at 5. The chest picks a whole number from this to Gold Most when it opens, so set the two the same for a chest that always gives the same amount.	5
Gold Most	The most gold the chest can hold. It starts at 20. At 0 the chest drops no gold at all, whatever Gold Least says. Raising it above 0 also puts the player's gold on the screen, when the player has gold and nothing shows it yet.	20

SETTING	WHAT IT DOES	STARTS AS
Can Drop	The items it can hold. As the level starts it takes How Many Items of them at random, rarer ones less often (set each item's Rarity), and drops them when it opens. Empty gives gold only. It is kept on its Loot node.	empty
How Many Items	How many items it takes from Can Drop. 0 holds none. It is kept on its Loot node.	1
Item Chance	How likely each of those items is really there, from 0 to 100, where 35 drops one about one time in three. It is kept on its Loot node.	100%
OPENING		
Open Key	The key, and the gamepad button that goes with it, that opens the chest while the player stands beside it. It starts as E with the left face button. The words over the chest follow it, so E to open becomes F to open. Words of your own that do not begin with a key's name are left as you typed them. A click opens it whatever this is set to, with a player that walks where it is clicked. Choices: E · pad Left Button (X / Square), Mouse - Left Button · pad Right Trigger, C · pad Right Button (B / Circle), E · pad Top Button (Y / Triangle), F · pad Left Button (X / Square), Space · pad Bottom Button (A / Cross), Enter · pad Bottom Button (A / Cross), J · pad Left Button (X / Square), K · pad Right Button (B / Circle), Shift · pad Right Bumper, Ctrl · pad Right Stick Press, Escape · pad Start, Tab · pad Select / Back, P · pad Start.	E · pad Left Button (X / Square)
Words over Its Head	What shows over its head while the player is near. It is kept on its Hint node.	E to open
LOOK		
Tint	A colour laid over its picture, so one set of art makes a skeleton, a ghost and a boss. White shows the picture as drawn. It is kept on its Picture node.	white

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.



Its template in the NodeWorthy panel.

STATE	WHAT HAPPENS IN IT
Closed	The picture shows closed and the words over it are hidden. In 3D there is no picture to play, and the lid sits shut. When the player walks into Near , the event Near leads to Player Near . A click from a player that walks where it is clicked sends USED , which leads straight to Opening .
Player Near	The words over it show. Pressing the Open Key sends Open , and Open or USED leads to Opening . Walking away sends Gone , which leads back to Closed .
Opening	The words hide, the picture plays open , and a Drop Loot action throws out everything Loot holds and a pile of gold somewhere from Gold Least to Gold Most . In 3D the lid swings up over 0.4 seconds in place of the picture. FINISHED leads to Open .
Open	The chest stays open. This state has no actions and no way out as it comes, so a chest cannot be opened twice.

Make it your own

Choose what it drops

1. Click **Chest** in the Scene dock. At the top of the Inspector, look under **Loot**.
2. Under **Can Drop** press **Add Item** and pick an item. Your own items are at the top. To make one, choose **New Item...**, type a name such as Potion and press **Make it**. The items NodeWorthy's own games and recipes use, pictures and all, are under **What came with NodeWorthy**.
3. Do the same for each item the chest might hold. Each is a row with its picture, its name and how rare it is. Press the name to give the item a picture and a rarity, and press **X** to take it out. Item files dragged onto the list from the FileSystem dock, the panel at the bottom left that lists the files of your project, go in too.
4. Set **How Many Items** to how many of them it takes as the level starts. It starts at 1, and 0 takes none.
5. Set **Item Chance** to how likely each of those is really there. 100 is always, and 35 is about one time in three.
6. Run the scene and open the chest. Rarer items turn up less often, and how rare an item is you set in its **Rarity**.
7. Each Chest keeps its own list, so click the next Chest and give it another.

Set how much gold it gives

1. Click **Chest** in the Scene dock.
2. Under **Loot**, set **Gold Least** and **Gold Most**. A boss's chest might give 50 to 100.
3. For a chest with no gold, set **Gold Most** to 0.
4. To make these the numbers every Chest starts with, press **Use These For Every Copy** under the settings.

Give it your own closed and open pictures

1. Draw two pictures of the same size, one of the chest shut and one of it open. Save them as PNG files whose last word says which, such as chest_closed.png and chest_open.png.
2. Right-click **Chest** in the Scene dock and choose **NodeWorthy** → **Looks** → **Import Sprite Sheet...** **Set Texture...** is greyed out for a Chest, because one still picture would stop the two animations its states play.
3. In the file window, pick both pictures at once and open them. They are copied into your project's pictures folder.
4. A window opens for each picture in turn. For a single picture, **Across** and **Down** are both 1. Check that **Row 1** reads closed for the shut chest and open for the open one, which it takes from the file's name.
5. Leave **Keep the animations it has** ticked and press **Use It** in each window.
6. The Chest now shows your closed picture and changes to the open one when it opens. The names must be closed and open, because those are the two its states play.
7. **As tall as its shape** fits each picture to the chest's solid box. Choose **As big as it is drawn** in the same window to keep a picture the size you drew it, standing on the bottom of the box.
8. In 3D the chest has no pictures. **NodeWorthy** → **Looks** → **Set Model...** puts a model in place of its box and hides the plain **Lid** under **Hinge**, so nothing of the box shows over your model.

Put one certain thing in a chest, such as a key

1. Click **Loot**, under the **Chest**, in the Scene dock.
2. In the Inspector, under **Starts With**, press **Add Item** and pick the item. Pick it again to add one more of it.
3. Whatever is listed there is always in the chest, beside anything it takes from **Can Drop**, and all of it spills out when the chest opens.
4. Each Chest has a **Loot** of its own, so one can hold the key and the next a potion.

Play a sound as it opens

1. Click **Chest** and press **Open Its State Machine...** under its settings. The NodeWorthy panel shows its template, the drawing of states that makes it behave, as four boxes named **Closed**, **Player Near**, **Opening** and **Open**.
2. Click the **Opening** box. The **Actions** tab on the right lists its three actions, a card each.
3. Press **Add Action...**, type sound, pick **Play Sound** and press **Add**. The new card opens.
4. Press the box beside **Stream**, which reads **Pick a sound...**, choose **From my computer...** and pick a short sound. It is copied into your project's sounds folder.
5. Run the scene and open the chest.
6. Every Chest in the scene shares this template, so they all play the sound.

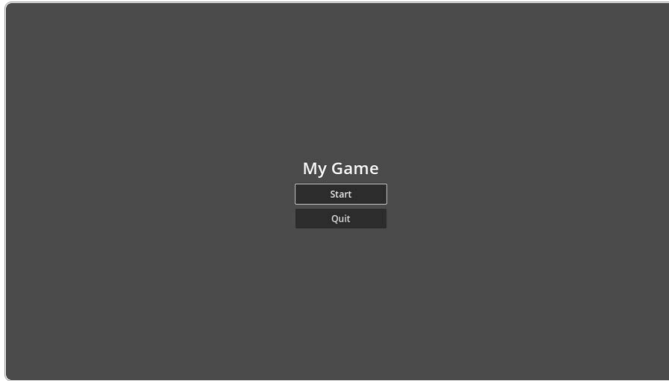
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The chest gives gold and never an item	Can Drop is empty, How Many Items is 0, or Item Chance is low. Add items to Can Drop under Loot , and set Item Chance to 100 while you test.
The chests hold fewer items the longer the game is played, and after a while none	Not Seen This Run is ticked on a chest's Loot . It leaves out any item that has already turned up since the game was started, and items turn up again only after a New Run action has run. It starts unticked, so each chest takes from its own Can Drop every time the level starts. Click Loot under the Chest and untick it in the Inspector, or run New Run as each run begins.
The player walks over an item and it stays on the floor	The player has no Inventory to put it in, and Godot's Output panel, at the bottom of the editor, says so, or the one it has is full. A player is given one when a Chest or an Item Pickup is added, or when an item is put in Can Drop . For a player whose Inventory was deleted, right-click the player and choose NodeWorthy → Give It → Inventory .
The player walks over the gold and it stays on the floor	The player's template has no number variable called gold, and Godot's Output panel, at the bottom of the editor, says so. Every ready-made hero has one. A player of your own needs it added. Click the player, press Open Its State Machine... , open the Variables tab, choose int, press Add Variable and name it gold.
The words over it name a key that is not the Open Key	The words follow the key only while they begin with a key's name and the word to, as E to open does. Words of your own are left alone, so type new ones in Words Over Its Head , under Opening .

See also [Inventory](#) 185, [Item Pickup](#) 91, [Enemy](#) 74, [Stat Panel](#) 122, [Inventory Grid](#) 141

Main Menu MENU

A title screen. It puts a title, a **Start** button and a **Quit** button in the middle of the screen, worked with the mouse, the keyboard or a gamepad. **Start** fades to the scene you pick, and **Quit** closes the game.



Main Menu as the game runs.

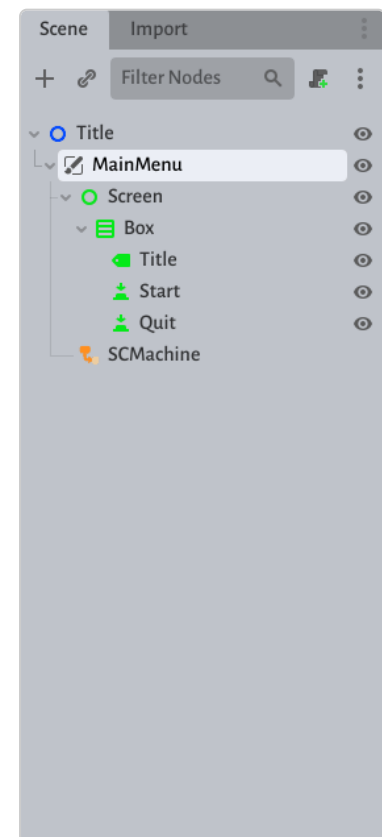
What it is good for

The first thing the player sees, in any game, 2D or 3D. Give it a scene of its own and make that the scene the game opens on. To stop the game part way through a level, add a **Pause Menu** to the level. To go from one level to the next, add a **Level Exit**.

Add it

1. Make a scene for the menu. Choose **Scene** → **New Scene**, press **2D Scene** in the Scene dock, the panel at the top left that lists the nodes of the scene, and save it with **Scene** → **Save Scene** as menu.tscn. A menu can go in any scene, even one with nothing 2D or 3D in it.
2. Right-click the top node in the Scene dock and choose **NodeWorthy** → **Add Menu** → **Main Menu**.
3. A **MainMenu** appears in the Scene dock, and the view shows the title My Game over the two buttons.
4. Click **MainMenu**. Its settings are at the top of the Inspector, the panel on the right that shows the settings of whatever is clicked.
5. Under **Starting**, press **First Level** and pick the scene **Start** goes to. The list names every scene saved in your project, so save your level first.
6. Under **Showing**, type your game's name in **Title**.
7. Press F6, or the Run Current Scene button at the top right of Godot.

Try it. The title and both buttons show, with **Start** already highlighted. Press **Enter**, or click **Start**, and the screen fades to your first level over half a second. Run it again and press the down arrow and **Enter** to choose **Quit**. Run from Godot, that stops the game and leaves Godot open.



The Scene dock once it is added.

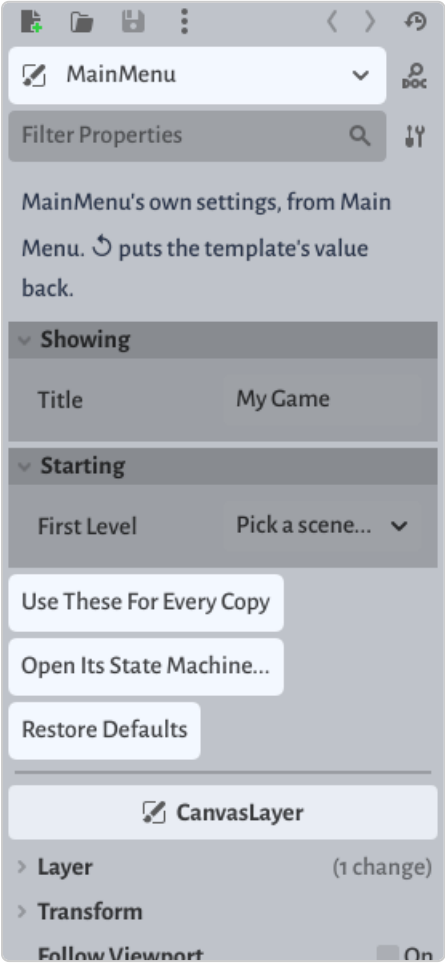
What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
MainMenu	CanvasLayer	The menu itself. A CanvasLayer is Godot's node for things drawn on the screen and not in the world, so the menu stays put whatever a camera does. It is on layer 10, which draws it over the game.
Screen	Control	Under MainMenu . A Control is Godot's node for anything on the screen, and this one is empty and stretched over the whole screen. It has no colour of its own, so whatever else is in the scene shows behind the menu.
Box	VBoxContainer	Under Screen . A VBoxContainer is Godot's node that stacks what is under it from top to bottom. It sits in the middle of the screen and holds the title and the buttons.
Title	Label	Under Box . The words at the top, My Game to begin with, in large letters. A Label is Godot's node for words.
Start	Button	Under Box . The button that starts the game, 260 pixels wide and 56 tall. A Button is Godot's node for something to press.
Quit	Button	Under Box . The button that closes the game.
SCMachine	Machine	NodeWorthy's Machine node, which runs the menu's template. The template is the drawing of states and actions that makes the buttons do something, and it is saved in your project's nodeworthy folder, unless you have picked another folder for templates.

The controls

PRESS	WHAT IT DOES
The up and down arrow keys	Move the highlight between Start and Quit .
Enter or the Space bar	Presses the highlighted button.
The mouse	Click a button to press it.
The D-pad or the left stick on a gamepad	Moves the highlight between Start and Quit .
The bottom button on a gamepad (A or Cross)	Presses the highlighted button.

Its settings



Its settings in the Inspector.

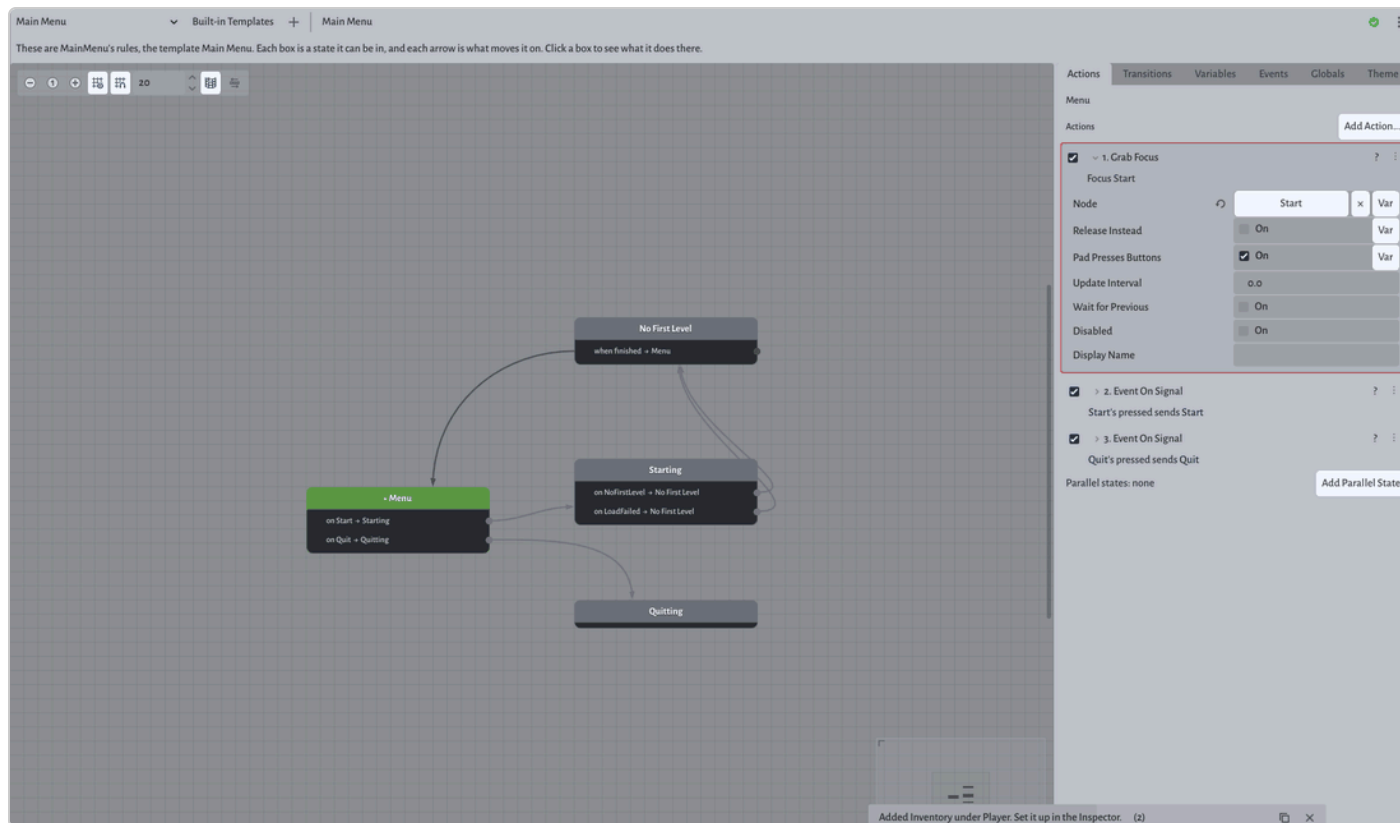
Click **MainMenu** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot's own settings, in sections that fold.

A change here changes this one only. The arrow that appears beside a changed setting puts the template's value back. **Use These For Every Copy**, under the settings, makes them the template's own, for every copy.

SETTING	WHAT IT DOES	STARTS AS
SHOWING		
Title	The words at the top of the screen. It is kept on its Title node.	My Game
STARTING		
First Level	The scene Start goes to. It starts empty and reads Pick a scene... , and until you pick one Start goes nowhere, says so in the Output panel, and leaves the menu working. The list names every scene saved in your project, so save your level before you look for it. The Change Scene action in the Starting state reads this setting, so this is the place to pick it. Choices: Pick a scene..., Chest, Enemy, Exit, Ground, Level, Main Menu, Pickup, Platform Hero, Platform Hero 3d, Side On Hero, Terrain, Third Person Hero, Top Down Hero, Top Down Hero 3d, Villager.	Pick a scene...

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.



Its template in the NodeWorthy panel.

STATE	WHAT HAPPENS IN IT
Menu	Start is given the highlight, so the keyboard and a gamepad work at once. Pressing Start sends the event Start , which leads to Starting , and pressing Quit sends Quit , which leads to Quitting .
Starting	A check first asks whether First Level holds a scene. With none, NoFirstLevel leads to No First Level . With one, a Change Scene action goes to it, fading the screen out and back in over half a second each. A scene that is no longer where it was sends LoadFailed , which also leads to No First Level .
No First Level	A Print action says in Godot's Output panel, at the bottom of the editor, that Start has nowhere to go, and to pick a scene in First Level . FINISHED leads back to Menu , so both buttons go on working.
Quitting	A Quit Game action closes the game. Run from Godot, it stops the game and leaves Godot open.

Make it your own

Pick where Start goes

1. Open your first level and save it with **Scene** → **Save Scene**, so it is a file in your project.
2. Open the menu's scene and click **MainMenu** in the Scene dock.
3. At the top of the Inspector, under **Starting**, press **First Level** and pick the level by its name.
4. If the level is not in the list, click another node and then **MainMenu** again, so the list is read afresh.
5. Run the menu's scene and press **Start**.

Make the game open on the menu

1. Find the menu's scene file, such as menu.tscn, in Godot's FileSystem dock, the panel at the bottom left that lists the files of your project.
2. Right-click the file and choose **Set as Main Scene**.
3. Press F5, or the Run Project button at the top right of Godot. The game now opens on the menu, and so does the finished game you give to other people.
4. F6 still runs whichever scene is open, which is how you test a level by itself.

Change the title and the words on the buttons

1. Click **MainMenu** in the Scene dock. Under **Showing**, type your game's name in **Title** and press **Enter**.
2. To change a button's words, click **Start** or **Quit** in the Scene dock and change **Text** in the Inspector, which is Godot's setting for the words on a button.
3. Leave the names of the nodes as they are. The template finds each button by its name in the Scene dock.

Make the fade slower or faster

1. Click **MainMenu** and press **Open Its State Machine...** under its settings. The NodeWorthy panel, where templates are drawn, shows its template, four boxes named **Menu**, **Starting**, **No First Level** and **Quitting**.
2. Click the **Starting** box. The **Actions** tab on the right shows its two actions, a card each. The second is named **Change Scene**.
3. Click the card's name to open its settings.
4. Change **Fade Seconds**. It starts at 0.5, which is half a second out and half a second back in. 0 cuts straight to the level, and 2 is a slow fade.
5. Run the menu's scene and press **Start**.

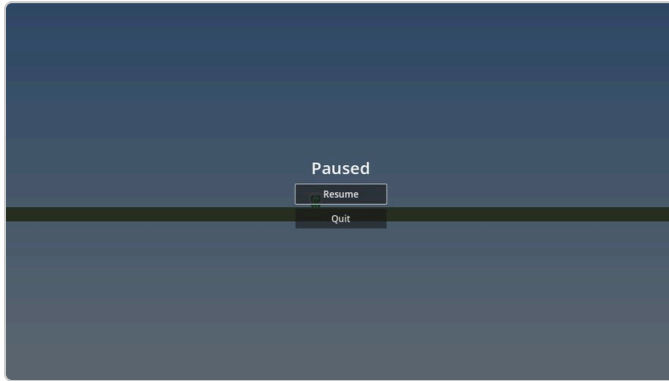
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
Start does nothing	No scene is picked in First Level , or the scene picked has been moved or deleted. Godot's Output panel, at the bottom of the editor, says Start has nowhere to go. The menu goes on working, so Quit still closes the game. Stop the game, click MainMenu , pick First Level and run it again.
Your level is not in the First Level list	The list names the scenes saved in your project, outside the addons folder. Save the level with Scene → Save Scene , then click another node and click MainMenu again.
The game still opens on a level and not on the menu	Godot opens the project's main scene. Right-click the menu's scene file in the FileSystem dock and choose Set as Main Scene , then run with F5.
The level carries on behind the menu	The menu was added to a level's scene. It covers the screen but does not stop the game, and it has no background of its own. Give the menu a scene of its own.

See also [Pause Menu](#) 116, [Level Exit](#) 86

Pause Menu MENU

A pause screen for a level. **Escape**, or **Start** on a gamepad, freezes the game, darkens the screen and shows **Resume** and **Quit**. The same key, or **Resume**, lets the game carry on from where it stopped.



Pause Menu as the game runs.

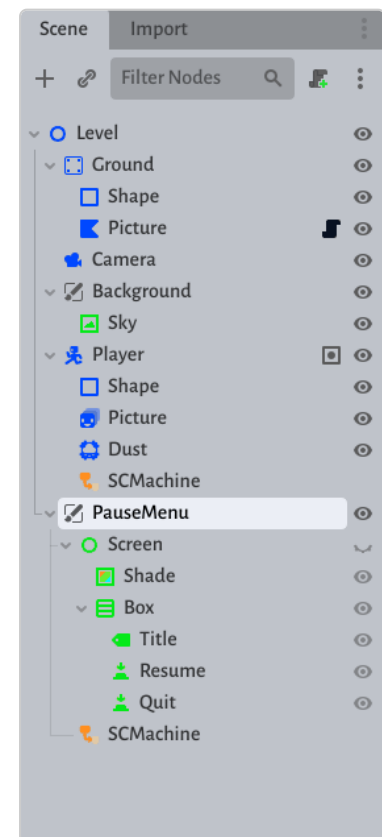
What it is good for

Any level the player might need to stop, in 2D or 3D. It belongs to the scene it is added to, so each level's scene needs one, and a scene takes only one. For the screen the game opens on, use a **Main Menu**.

Add it

1. Open a level's scene, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a player in it.
2. In the Scene dock, the panel at the top left that lists the nodes of the scene, right-click the top node and choose **NodeWorthy** → **Add Menu** → **Pause Menu**.
3. A **PauseMenu** appears in the Scene dock. Nothing new shows in the view, because its **Screen** is hidden until the game is paused.
4. Press F6, or the Run Current Scene button at the top right of Godot.

Try it. Press **Escape** while you play. Everything stops where it is, the screen darkens, and the word Paused shows over **Resume** and **Quit**, with **Resume** highlighted. Press **Escape** again, or **Enter**, and the game carries on. Pause again and choose **Quit**, which closes the game.



The Scene dock once it is added.

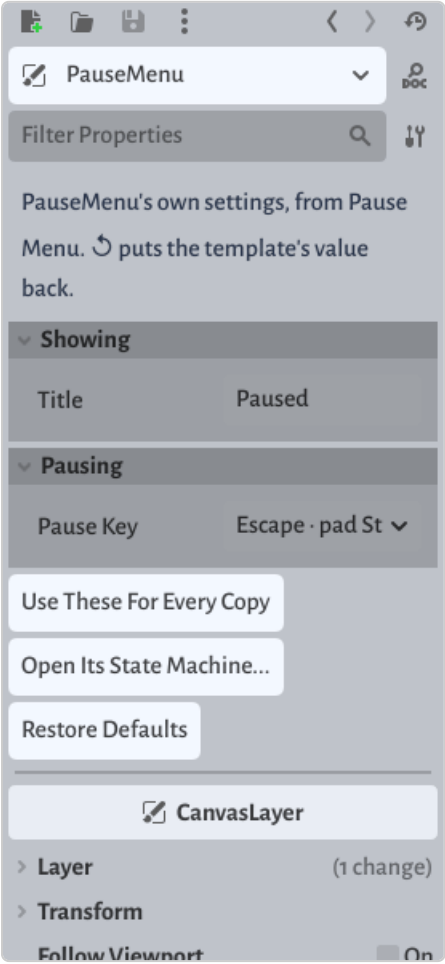
What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
PauseMenu	CanvasLayer	The menu itself. A CanvasLayer is Godot's node for things drawn on the screen and not in the world, so the menu stays put whatever a camera does. It is on layer 20, which draws it over the game. It keeps running while everything else is paused.
Screen	Control	Under PauseMenu . A Control is Godot's node for anything on the screen. This one is stretched over the whole screen, holds everything the player sees of the menu, and is hidden until the game is paused.
Shade	ColorRect	Under Screen . A ColorRect is Godot's node for a block of colour. This one is black and a little over half see-through, which darkens the game behind the menu.
Box	VBoxContainer	Under Screen . A VBoxContainer is Godot's node that stacks what is under it from top to bottom. It sits in the middle of the screen and holds the title and the buttons.
Title	Label	Under Box . The word Paused, in large letters. A Label is Godot's node for words.
Resume	Button	Under Box . The button that lets the game carry on, 260 pixels wide and 56 tall. A Button is Godot's node for something to press.
Quit	Button	Under Box . The button that closes the game.
SCMachine	Machine	NodeWorthy's Machine node, which runs the menu's template. The template is the drawing of states and actions that pauses the game and works the buttons, and it is saved in your project's nodeworthy folder, unless you have picked another folder for templates.

The controls

PRESS	WHAT IT DOES
Escape, or Start on a gamepad	Pauses the game and shows the menu. Pressed again, it hides the menu and the game carries on.
The up and down arrow keys, or the D-pad or the left stick on a gamepad	Move the highlight between Resume and Quit while the game is paused.
Enter or the Space bar, or the bottom button on a gamepad (A or Cross)	Presses the highlighted button.
The mouse	Click a button to press it. The pointer is shown while the game is paused, even in a game that hides it.

Its settings



Its settings in the Inspector.

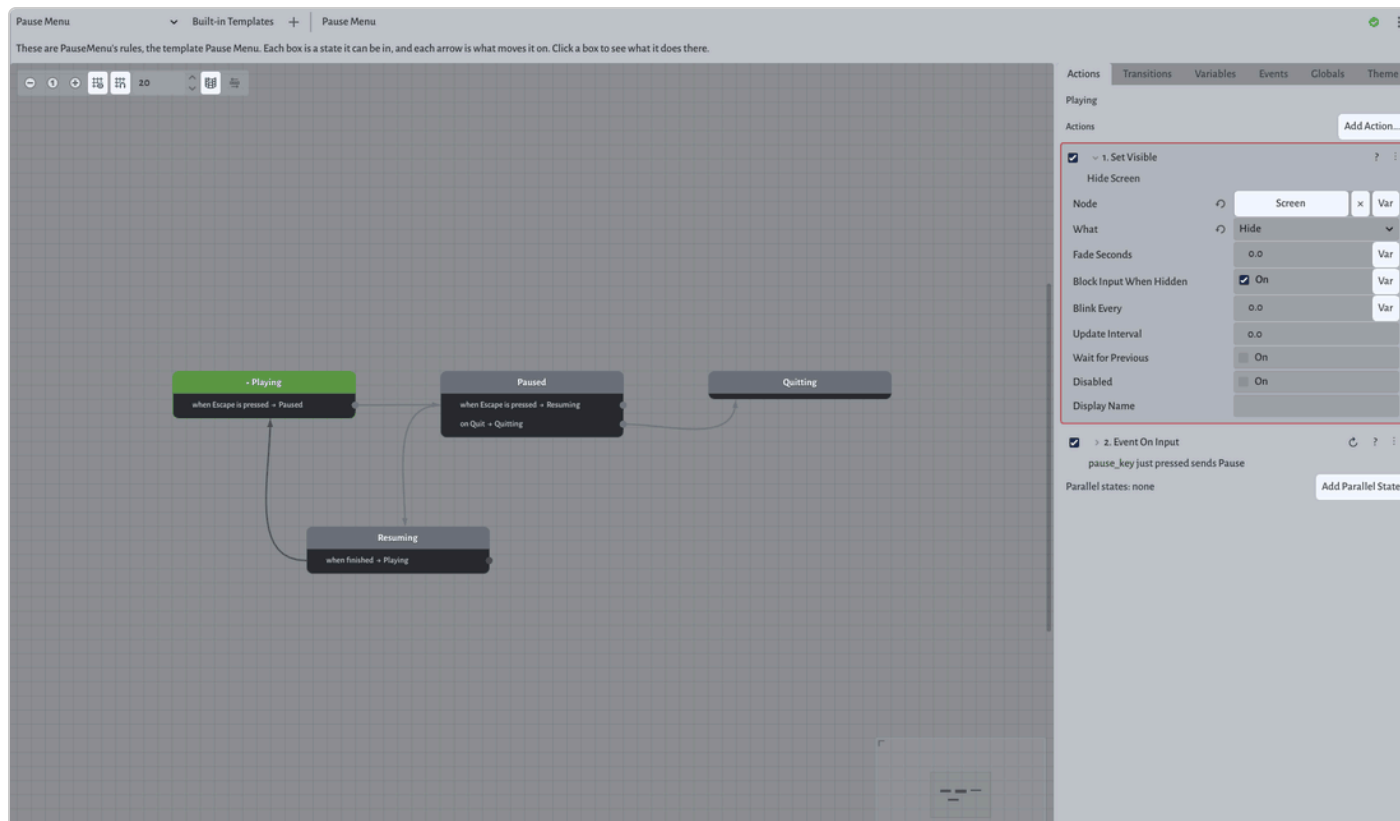
Click **PauseMenu** in the Scene dock and these are at the top of the Inspector, the panel on the right, above Godot's own settings, in sections that fold.

A change here changes this one only. The arrow that appears beside a changed setting puts the template's value back. **Use These For Every Copy**, under the settings, makes them the template's own, for every copy.

SETTING	WHAT IT DOES	STARTS AS
SHOWING		
Title	The words at the top of the screen. It is kept on its Title node.	Paused
PAUSING		
Pause Key	The key, and the gamepad button that goes with it, that pauses the game and lets it carry on again. It starts as Escape with Start . The list offers eleven pairs, and P with Start is there for a game that uses Escape for something else. Choices: E · pad Left Button (X / Square), Mouse - Left Button · pad Right Trigger, C · pad Right Button (B / Circle), E · pad Top Button (Y / Triangle), F · pad Left Button (X / Square), Space · pad Bottom Button (A / Cross), Enter · pad Bottom Button (A / Cross), J · pad Left Button (X / Square), K · pad Right Button (B / Circle), Shift · pad Right Bumper, Ctrl · pad Right Stick Press, Escape · pad Start, Tab · pad Select / Back, P · pad Start.	Escape · pad Start

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.



Its template in the NodeWorthy panel.

STATE	WHAT HAPPENS IN IT
Playing	Screen is hidden, however it was left in the editor. The game runs and the menu waits for the Pause Key . Pressing it sends Pause , which leads to Paused .
Paused	A Pause Game action freezes everything in the game except the PauseMenu and its Machine, and shows the mouse pointer. Screen is shown and Resume is given the highlight. Pressing Resume or the Pause Key sends Resume , which leads to Resuming , and pressing Quit sends Quit , which leads to Quitting .
Resuming	Screen is hidden and the game is let go again, with the mouse pointer put back as the game had it. FINISHED leads back to Playing .
Quitting	A Quit Game action closes the game. Run from Godot, it stops the game and leaves Godot open.

Make it your own

Change the pause key

1. Click **PauseMenu** in the Scene dock.
2. At the top of the Inspector, under **Pausing**, pick another pair in **Pause Key**, such as **P** with **Start** on a gamepad.
3. Run the scene and press the new key. The same key pauses and carries on.

Change the words

1. Click **PauseMenu** in the Scene dock. Under **Showing**, type new words in **Title**, such as Game Paused, and press **Enter**.
2. To change a button's words, click **Resume** or **Quit** in the Scene dock and change **Text** in the Inspector, which is Godot's setting for the words on a button.
3. Leave the names of the nodes as they are. The template finds each button by its name in the Scene dock.

Make Quit go back to the main menu

1. Click **PauseMenu** and press **Open Its State Machine...** under its settings. The NodeWorthy panel, where templates are drawn, shows its template, four boxes named **Playing**, **Paused**, **Resuming** and **Quitting**.
2. Click the **Quitting** box. The **Actions** tab on the right shows its one action, a card named **Quit Game**.
3. On that card press **:** and choose **Remove**.
4. Press **Add Action...**, type scene, pick **Change Scene** and press **Add**. The new card opens.
5. Press the box beside **Scene**, which reads **Pick a scene...**, and pick your menu's scene.
6. Leave **Resume First** ticked. It lets the game run again before the change, and without it the menu would open frozen.
7. Click **Quit** in the Scene dock and change its **Text** in the Inspector to Main Menu, so the button says what it now does.
8. A Pause Menu added to another level's scene has a template of its own, so make the same change there.

Make the screen darker or lighter behind the menu

1. Click **Shade**, under **Screen**, in the Scene dock.
2. In the Inspector press **Color** and choose another colour. The slider marked A is how see-through it is, and further to the right is darker.
3. To see the menu in the view while you work, click the eye beside **Screen** in the Scene dock. It can be left showing, because the menu hides itself as the level starts.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
Pause Menu is greyed out under Add Menu	The scene already has one, and pointing at the greyed line names it. Two would both answer Escape , one pausing the game as the other let it carry on, so a scene takes only one.
Escape does nothing in another level	A Pause Menu belongs to the scene it was added to. Open the other level's scene and add one there. Each one added to a new scene gets a template of its own, named Pause Menu 2 and so on.

See also **Main Menu** 111, **Level Exit** 86

NODES FOR THE SCREEN

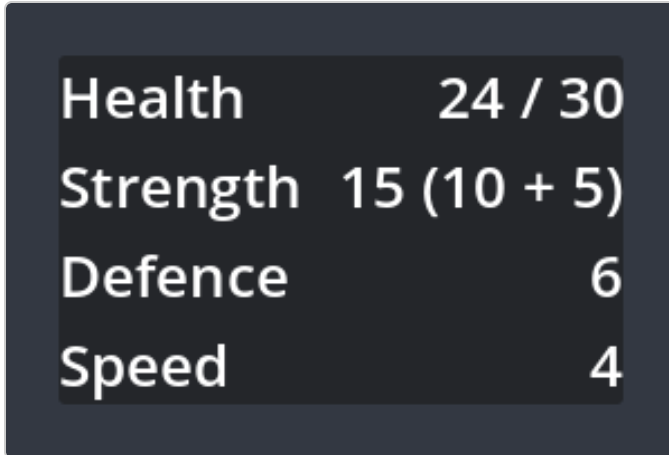
Nodes that go on the screen

These nodes draw on the screen, over the game, and never move with the camera. They show the player what is going on, such as health, a map, what is carried, the quests under way and who is speaking. They are under **NodeWorthy** → **Add To Screen** when you right-click the top node of a level.

Stat Panel	122
Bar	127
Globe	132
Icon Meter	137
Inventory Grid	141
Quest List	146
Quest Log	150
Minimap	155
Crosshair	162
Skill Bar	166
Dialogue Box	171
Item Collection	175

Stat Panel NODE

A panel that lists a character's numbers on the screen, such as Strength 15, Defence 8 and Health 20 / 30. It reads them straight from the character's template, so nothing has to keep it up to date. The moment it is added it shows the player's numbers, in the editor as they start and in the game as they change.



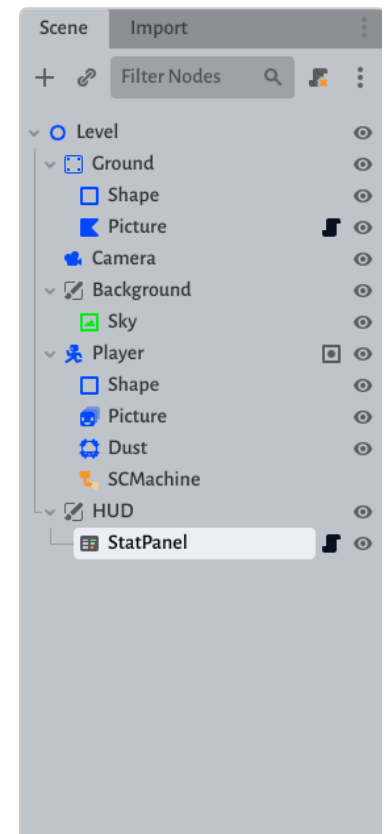
Stat Panel in a running game.

What it is good for

A stat sheet in a role-playing game, the gold and health in a corner of any game, or a panel to watch a hero's numbers while you tune them. A number that an item or an effect is changing shows what it started as and what was added, which no other screen node does. It writes numbers as words, so for health drawn as hearts pick the **Icon Meter**, for a draining globe the **Globe**, and for a boss the **Bar**.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it. The ready-made players are in the player group, a label Godot lets you hang on nodes, and that is how the panel finds the player.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Stat Panel**.
3. A node called **StatPanel** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
4. The editor shows it in the 2D view, even in a 3D game, because the screen is flat. The panel sits in the top left corner.
5. Before anything is set it lists the numbers the player's template keeps, one to a line, each under its variable's name with capitals. The player's settings, such as Walk Speed, are left out, because they are set once and are no stats.
6. Click **StatPanel** and look at the Inspector, the panel on the right that shows the settings of whatever is clicked. Under **Stats**, tick the numbers you want, a setting among them if you like, and the rest drop off the panel.
7. A level may already have a Stat Panel. **NodeWorthy** → **Give It** → **Health** on the player adds one called **HealthPanel** that shows health, and a level where



The Scene dock once it is added.

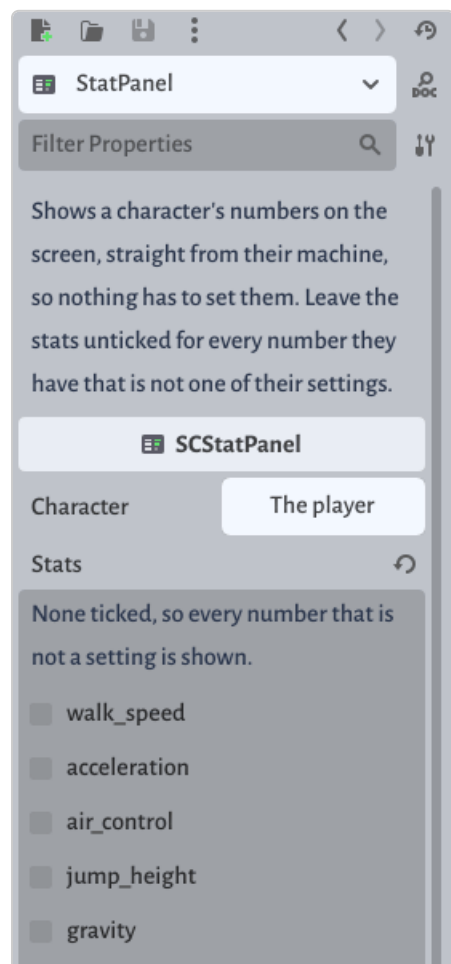
something drops gold gets one called **GoldPanel**, or the number is added to a Stat Panel already there. They are ordinary Stat Panels with one number ticked.

Try it. Press **F6**, or the **Run Current Scene** button at the top right of Godot. The panel shows the same lines over the game. When a number changes, such as gold picked up from a **Chest**, its line changes at once and flashes green going up or red going down.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.
StatPanel	Stat Panel	The panel itself. A PanelContainer is Godot's box with a background that grows to fit what is inside it, so the panel gets taller as it lists more numbers. It takes its background and its font from the game's look.

Its settings



Click **StatPanel** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Character	Whose numbers the panel lists, whether the player, an enemy or a shopkeeper. Empty, the box reads The player and the node follows whichever node is in the player group, which every ready-made player joins. Press the box and pick another character in the window that lists the scene's nodes, and press the X beside it to go back to the player.	empty
Stats	Which numbers are listed, as a row of tick boxes in the Inspector, one for each number the character's template keeps, with the ticked ones shown from top to bottom in the template's order. With none ticked every number is shown, less the character's settings and any that only serves as another number's limit. A character with an Experience node under it shows its experience as how far it is to the next level, as in XP 40 / 100.	empty
Show Maximum	When a number's Max is another variable, the two share a line, as in Health 20 / 30, and the limit gets no line of its own. Untick it and each number is listed on its own. It starts ticked.	on
Show Changes	While the game runs, a number that gear or an effect is changing shows what it started as and what was added, as in 15 (10 + 5). Untick it to show only the 15. It starts ticked.	on

SETTING	WHAT IT DOES	STARTS AS
Font Size	How big the words are. At 0, the start, the panel uses the text size of the game's look, which is set in the Theme tab of the NodeWorthy panel. A bigger number makes the whole panel bigger with it.	0
Up Color	The colour a number flashes when it goes up, a soft green to start with. The flash fades away in under a second.	0.55, 0.95, 0.6, 1
Down Color	The colour a number flashes when it goes down, a soft red to start with. The flash fades away in under a second.	1, 0.45, 0.45, 1

Make it your own

Show only the numbers that matter

1. Click **StatPanel** in the Scene dock.
2. In the Inspector, under **Stats**, tick the numbers to show, such as health and gold. The line above the ticks, which said that with none ticked every number that is not a setting is shown, now says they are shown top to bottom in this order.
3. The panel in the 2D view is drawn again as you tick, so you can see the result without running the game.

Show an enemy's or a boss's numbers

1. Click **StatPanel**, then press the **Character** box in the Inspector, which reads The player.
2. In the window that lists the scene's nodes, pick the character, such as an **Enemy**.
3. The panel now lists that character's numbers. The character must run a template, which means it has a Machine node under it.

Move it to another corner

1. Click **StatPanel** and drag it in the 2D view to where you want it.
2. To keep it in a corner on every screen size, open **Layout** in the Inspector, set **Layout Mode** to **Anchors** and pick a place in **Anchors Preset**, such as **Top Right**. The panel moves to that corner and keeps to it whatever size the game's window is.
3. The panel is as big as its lines need. To make it bigger, raise **Font Size**.

See what an effect adds to a number

1. Click the player in the Scene dock and press **Open Its State Machine...** under its settings at the top of the Inspector. The NodeWorthy panel shows the player's template.
2. Click the **Playing** box, then press **Add Action...** in the **Actions** tab on the right.
3. Type Modify Stat in the search box, pick **Modify Stat** and press **Add**.
4. In **Variable Name** pick one of the player's numbers, such as gold, which most ready-made heroes keep. Leave **Change** at **Add**, set **Amount** to 5 and type boots in **Source**.
5. Run the scene. The Gold line reads 5 (0 + 5), which is the number now, then what it started as and what the boots added.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The panel says No character . Put the player in the player group, or pick one under Character	Nothing in the scene is in the player group and Character is empty. The ready-made players join the group by themselves. For a player you built yourself, press Character and pick it in the scene.
The panel says No stats to show yet	The character has a Machine, but its template keeps no number variables. Open the template and add a number in its Variables tab.
The panel lists numbers you do not want	With nothing ticked under Stats the panel shows every number the template keeps except its settings, and a ready-made hero keeps a few working numbers of its own beside its gold. Tick the ones you want and the rest drop off. A setting, such as Walk Speed, is shown only when it is ticked.
A line shows a question mark where the number should be	A name ticked under Stats is no longer a variable of the character's template, usually after the variable was renamed. The Inspector lists it as no such number variable. Untick it there.
A yellow warning sign shows beside StatPanel in the Scene dock	Hover over the sign to read why. It appears when the character it follows has no Machine, which is the node that runs a template, so there are no numbers to read. It also appears when Character points at a node that is no longer in the scene. Press Character and pick a character that runs a template.

See also **Icon Meter** 137, **Globe** 132, **Bar** 127, **Experience** 195, **Inventory Grid** 141

Bar NODE

A long bar on the screen that follows one number of one character, with a name written over it. Added from the menu it sits along the top of the screen in the middle, the way a boss's health bar does. It follows the player's health until you point it at someone else. With **Over Its Head** ticked it rides over a character in the game, which is how an enemy gets a health bar.



Bar in a running game.

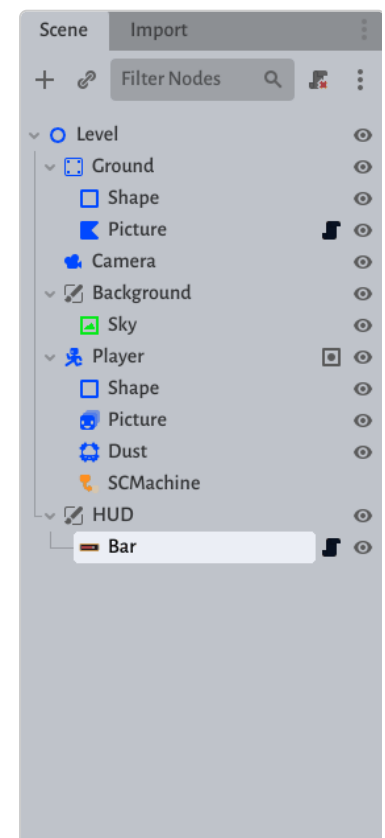
What it is good for

A boss fight, a shield, a rage meter, or any number big enough to deserve a bar of its own. With **Appears Within** set it comes up only once the player walks close to the character, and it goes when the character is gone. For the player's own health the **Icon Meter** or the **Globe** usually reads better, and the **Stat Panel** is the one for plain numbers.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it. Add someone with health for the bar to follow, such as an **Enemy** from **NodeWorthy** → **Add Thing** → **Enemy**, which comes with a health of 3.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Bar**.
3. A node called **Bar** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
4. The editor shows it in the 2D view, even in a 3D game. It is drawn along the top of the screen in the middle, dark red and full. The editor always draws it full, because there is no number to read until the game runs.
5. Click **Bar**. In the Inspector, the panel on the right that shows the settings of whatever is clicked, press the **Character** box, which reads The player, and pick the **Enemy** in the window that lists the scene's nodes.
6. Leave **Shows** at health, and type a name in **Title**, such as The Warden. The name is written over the bar.

Try it. Give the player a way to hurt the enemy, such as **NodeWorthy** → **Give It** → **Gun** on the player, then press **F6** to run the scene. The bar is full, with the name over it. Shoot the enemy with **F** and the bar drains smoothly to the new level, leaving a pale strip for a moment where the health was. When the enemy dies and is removed, the bar goes with it.

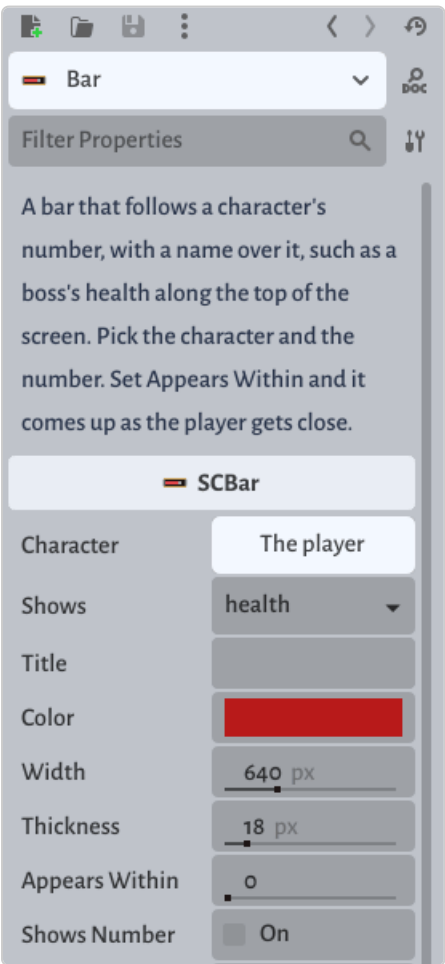


The Scene dock once it is added.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.
Bar	Bar	The bar itself. A Control is Godot's plain node for anything drawn on the screen, and the bar draws its frame, its fill and its title on it.

Its settings



Click **Bar** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Character	Whose number the bar follows. Empty, the box reads The player and the node follows whichever node is in the player group, which every ready-made player joins. Press the box and pick another character in the window that lists the scene's nodes, and press the X beside it to go back to the player.	empty
Shows	The character's number variable the bar follows. Press the little arrow beside the box and pick it from the list, which holds the numbers your templates keep. It starts as health. The variable's Max is a full bar.	health

SETTING	WHAT IT DOES	STARTS AS
Title	Words written over the bar in the middle, such as a boss's name. Left empty, nothing is written and the bar takes up less height.	empty
Color	The colour of the filled part, a dark red to start with. The empty part behind it is a much darker shade of the same colour, and the frame keeps its own colours.	0.72, 0.1, 0.1, 1
Width	How long the bar is, in pixels, from 60 to 2000. It starts at 640. This is what sets the bar's length, and dragging its handles in the 2D view does not stretch it.	640.0
Thickness	How tall the bar is, in pixels, from 4 to 120. It starts at 18, a slim strip. Raise it for a chunkier bar.	18.0
Appears Within	At 0, the start, the bar shows all the time. With a distance set, it stays hidden until the player comes that close to the character, in pixels in a 2D game or metres in a 3D one, and once it has come up it stays up. It is for a character other than the player. The player's own bar has nobody to come close to, so it shows all the time, and a warning sign beside it says so.	0.0
Shows Number	Writes the number and its most on the bar, as in 180 / 250. It starts unticked. In the editor it writes 1 / 1, since there is no number to read until the game runs.	off
Over Its Head	Off to begin with, and the bar stays where you put it on the screen. Ticked, it leaves that place and rides just above its Character in the game, wherever the character goes, in a 2D game and a 3D one, and it hides while the character is off the screen. This is how an enemy gets a health bar over its head. The quickest way to one is NodeWorthy → Give It → Health Bar on the enemy, which adds a small bar with this already ticked.	off

Make it your own

Put a health bar over an enemy's head

1. In the Scene dock, the list of nodes at the top left of Godot, right-click the **Enemy** and choose **NodeWorthy** → **Give It** → **Health Bar**. A small bar called **HealthBar** appears under the Enemy, with the Enemy picked as its **Character**, **Shows** set to health and **Over Its Head** ticked.
2. Press **F6**. The bar rides just above the Enemy's head, drains as the Enemy is hurt, hides while the Enemy is off the screen, and goes when the Enemy does. It works the same in a 2D game and a 3D one.
3. To change how it looks, click **HealthBar** and set **Color**, **Width** and **Thickness** in the Inspector, the panel on the right. Tick **Shows Number** to write the health on it.
4. Right-click the Enemy and choose **Duplicate** for a second enemy, and the copy has a bar of its own, because the bar sits under the Enemy.
5. A character with no health yet is given health too, the same as **NodeWorthy** → **Give It** → **Health** would.
6. A Bar you added yourself with **Add To Screen** does the same once you pick the enemy under **Character** and tick **Over Its Head**. It stays under **HUD** then, so each enemy needs a bar of its own.

Make it a boss's bar that comes up as the player walks in

1. Click **Bar**, press **Character** and pick the boss in the scene.
2. Type the boss's name in **Title**.
3. Set **Appears Within** to how close the player must come, such as 400 in a 2D game, where it is in pixels, or 12 in a 3D game, where it is in metres.
4. Run the scene. The bar is hidden until the player comes that close, and then stays on the screen until the boss is gone.

Follow another number, such as a shield

1. Click the character in the Scene dock and press **Open Its State Machine...** under its settings at the top of the Inspector. The NodeWorthy panel shows its template.
2. Open the **Variables** tab. If the number is not there yet, pick float and press **Add Variable**, then name it shield and give it a starting value.
3. Open the variable and set its **Max**, such as 50. The bar needs a **Max** to know what a full bar is.
4. Click **Bar**, press the little arrow beside **Shows** and pick shield from the list. Pick a blue in **Color** so it is not taken for health.

Move it and change its size

1. Click **Bar** and drag it in the 2D view to where you want it.
2. Set its length with **Width** and its height with **Thickness**. Godot's sizing handles do not stretch the bar.
3. To keep it against another edge on every screen size, open **Layout** in the Inspector and pick a place in **Anchors Preset**, such as **Center Bottom**. It starts at **Center Top**.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The bar does not show in the game	The character has no Machine, or its template has no variable with the name in Shows , so the bar hides itself. In the second case a yellow warning sign shows beside Bar in the Scene dock, and the Output panel says so once in the game. A ready-made player has no health until it is given one with NodeWorthy → Give It → Health , or until an Enemy is added to the level. Pick the number again from the list beside Shows .
Appears Within is set and the bar shows all the time	Character is empty, so the bar follows the player, and the distance is measured from the player to the character. The player has nobody to come close to, so the bar is shown from the start, and a warning sign beside it says so. Pick the character whose bar this is, or set Appears Within back to 0.
The Output panel says the number has no Max	The bar needs to know what full is. Without a Max it measures the number against the most it has held so far. Open the character's template and, in the Variables tab, give the variable a Max , such as 100 or max_health.
A Set Visible action shows the bar and it hides again	A Set Visible that hides the bar keeps it hidden until another shows it. Showing it does not overrule the bar, which stays hidden while its character is gone or has no such number, and until the player has come within Appears Within .

WHAT YOU SEE	WHY, AND WHAT TO DO
A yellow warning sign shows beside Bar in the Scene dock	Hover over the sign to read why. It appears when the character it follows has no Machine, which is the node that runs a template, so there are no numbers to read. It appears when Character points at a node that is no longer in the scene, when the character's template has no variable of the name in Shows , and when Appears Within is set while Character is empty. Press Character and pick a character that runs a template, or pick the number again under Shows .
A HealthBar under a 3D enemy cannot be seen in the 3D view	A Bar is drawn on the screen and not in the 3D world, so Godot's 3D view does not show it. Run the scene and it is over the enemy's head. In a 2D level the bar is put above the head in the editor as well.

See also **Globe** 132, **Icon Meter** 137, **Stat Panel** 122, **Enemy** 74, **Hurt Box** 180

Globe NODE

A round glass globe that fills with liquid to show a number, with the number written across it. The first one added sits in the bottom left corner, red, and follows the player's health. A second goes in the bottom right corner, blue, and follows the player's mana.



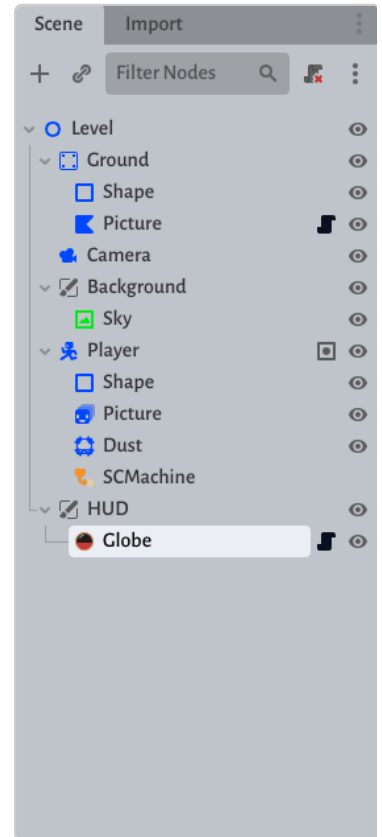
Globe in a running game.

What it is good for

Action role-playing games, where health and mana sit in the two bottom corners of the screen, and any game with a dark fantasy look. It drains smoothly when the number drops, and the surface sways while it is not full. Pick the **Icon Meter** for hearts and the **Bar** for a boss, and add a **Skill Bar** to spend the mana the blue globe shows.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it. The player needs a health to show, so right-click the player and choose **NodeWorthy** → **Give It** → **Health** if it has none. That also puts the health on the screen as a number, unless something already shows it.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Globe**.
3. A node called **Globe** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
4. The editor shows it in the 2D view, even in a 3D game. It sits in the bottom left corner, red and full, reading 100 / 100. Those are the **Value** and **Maximum** it was made with, and the player's own numbers take over when the game runs.
5. Choose **NodeWorthy** → **Add To Screen** → **Globe** again for a second one. It is called **Globe2**, goes in the bottom right corner, is blue, and has **Shows** set to mana. Any globe after that stands along the bottom beside the first, red, with **Shows** at health, for you to point at another number.
6. The blue globe needs a number called mana on the player. Until the player's template has one, it stays full at 100 / 100, and a yellow warning sign beside **Globe2** in the Scene dock says why.



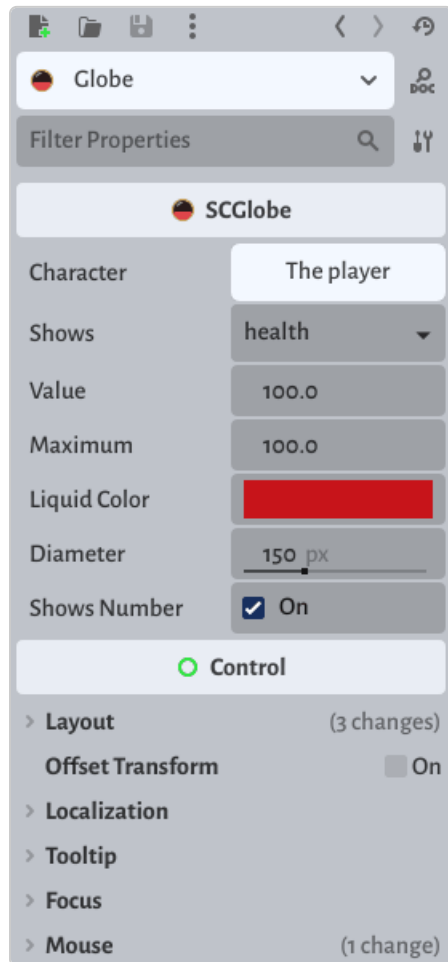
The Scene dock once it is added.

Try it. Press **F6**, or the **Run Current Scene** button at the top right of Godot. The red globe reads the player's health over its most, such as 3 / 3. Let an **Enemy** reach the player and the liquid drains to the new level, with a bright line rippling along its surface.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.
Globe	Globe	The globe itself. A Control is Godot's plain node for anything drawn on the screen, and the globe draws its glass, its liquid and its number on it.

Its settings



Click **Globe** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Character	Whose number the globe shows. Empty, the box reads The player and the node follows whichever node is in the player group, which every ready-made player joins. Press the box and pick another character in the window that lists the scene's nodes, and press the X beside it to go back to the player.	empty
Shows	The character's number variable the globe follows, such as health or mana. Press the little arrow beside the box and pick it from the list, which holds the numbers your templates keep. It starts as health. Cleared, the globe follows nobody and shows Value , for a machine to set.	health
Value	The number shown. It starts at 100 and is what the editor draws. While Shows names a variable the game writes that variable here about ten times a second, so you set it yourself only when Shows is empty.	100.0
Maximum	The number that fills the globe to the top. It follows the variable's Max when the variable has one. When the variable has no Max it stays at what is set here, which is 100 to start with. Left at 100 with such a variable, the Output panel says so once as the game runs.	100.0
Liquid Color	The colour of the liquid, a deep red to start with. The glass behind it is a very dark shade of the same colour, so a strong colour reads best.	0.78, 0.08, 0.1, 1

SETTING	WHAT IT DOES	STARTS AS
Diameter	How wide and tall the globe is, in pixels, from 32 to 400. It starts at 150. The number written across it grows with the globe.	150.0
Shows Number	Writes the number and its most across the middle, as in 72 / 100, in whole numbers. It starts ticked. Untick it for a globe that is read by its level alone.	on

Make it your own

Give the player mana for a blue globe

1. Add a second **Globe** from the menu. It comes blue, in the bottom right corner, with **Shows** set to mana.
2. Click the player in the Scene dock and press **Open Its State Machine...** under its settings at the top of the Inspector. The NodeWorthy panel shows the player's template.
3. Open the **Variables** tab, pick float and press **Add Variable**. Name the variable mana and give it a starting value of 100.
4. Open the variable and set its **Max** to 100, so the globe knows what full is.
5. Run the scene. The blue globe reads 100 / 100 and goes down whenever something spends the mana, such as a **Skill Bar**.

Let a machine set the level

1. Click the **Globe** and clear **Shows**, so that it shows its own **Value**.
2. Open the template of the machine that keeps the number. For the player, click it and press **Open Its State Machine....**
3. Click a state that runs all through the game, which for a ready-made player is **Playing**. Press **Add Action...**, type Set Property, pick **Set Property** and press **Add**.
4. In **Node** pick the **Globe**. In **Property** press the little arrow and pick Value.
5. Beside **Value** press **Var** and pick the variable that holds the number. It should be a float, the kind of number the globe's **Value** holds.
6. Press the  on the action's card so that it lights. The action then runs every frame and the globe keeps up with the variable. Set **Maximum** on the globe to the most the number can be.

Move it, size it and recolour it

1. Click the **Globe** and drag it in the 2D view to where you want it.
2. Set its size with **Diameter** and its liquid with **Liquid Color**, such as green for stamina.
3. To keep it in another corner on every screen size, open **Layout** in the Inspector and pick a place in **Anchors Preset**, such as **Top Left**.

When it goes wrong

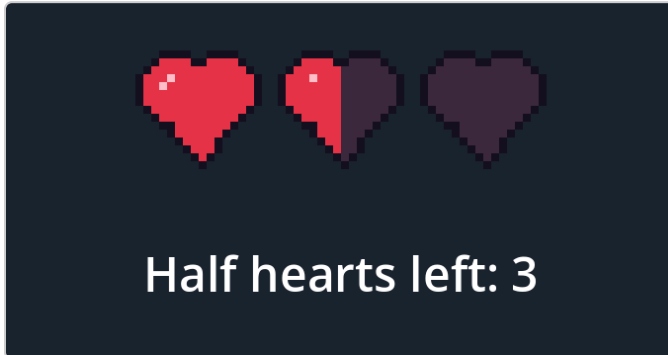
WHAT YOU SEE	WHY, AND WHAT TO DO
The globe stays full at 100 / 100 while the game runs	The character has no variable with the name in Shows , so the globe goes on showing Value and Maximum as they were set. This is what a second, blue globe does until the player has a variable called mana. A warning sign beside the globe in the Scene dock says so, and the Output panel says it once in the game. Add the variable in the template's Variables tab, or pick another number from the list beside Shows .

WHAT YOU SEE	WHY, AND WHAT TO DO
The globe is nearly empty though the character is at full health	The variable has no Max , so the globe measures it against Maximum , which starts at 100. The Output panel says so once while Maximum is still 100. Give the variable a Max in the Variables tab, or set Maximum to the most the number can be.
Typing in Value changes nothing in the game	While Shows names a variable the globe copies that variable into Value over and over. Clear Shows to set Value yourself or from a machine.
A yellow warning sign shows beside Globe in the Scene dock	Hover over the sign to read why. It appears when the character it follows has no Machine, which is the node that runs a template, so there are no numbers to read. It appears when Character points at a node that is no longer in the scene, and when the character's template has no variable of the name in Shows . Press Character and pick a character that runs a template, or pick the number again under Shows . With Shows cleared no character is needed and the sign is not shown.

See also **Bar** 127, **Icon Meter** 137, **Skill Bar** 166, **Stat Panel** 122

Icon Meter NODE

A row of icons that shows a number, such as hearts for health, skulls for lives or shells for ammunition. The moment it is added it draws three plain red hearts of its own. It follows a character's number once you pick one in **Shows**.



Icon Meter in a running game.

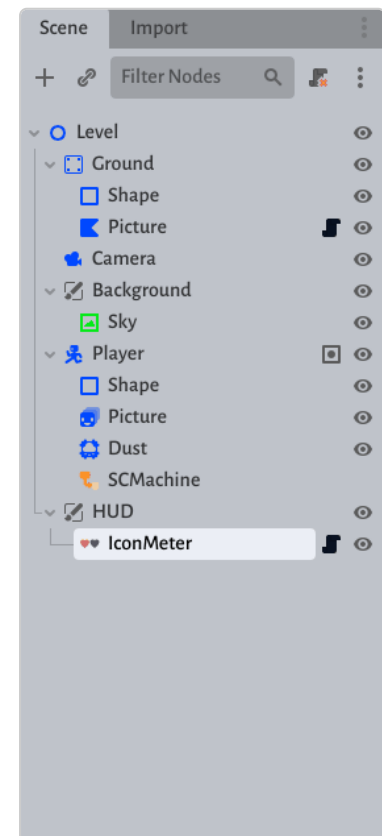
What it is good for

Health as hearts in an adventure or a platformer, and lives, keys or ammunition in any game where the number is small enough to count at a glance. One icon can stand for more than one point, so a heart can be lost half at a time. For a big number such as a boss's 250 health pick the **Bar**, and for a number written as words the **Stat Panel**.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it. The player needs a health to show, so right-click the player and choose **NodeWorthy** → **Give It** → **Health** if it has none. That gives it a health of 3.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Icon Meter**.
3. A node called **IconMeter** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
4. The editor shows it in the 2D view, even in a 3D game. It sits in the top left corner and draws three plain red hearts, which it does whenever it has no pictures of its own.
5. Click **IconMeter**. In the Inspector, the panel on the right that shows the settings of whatever is clicked, press the ▼ beside **Shows** and pick health from the list of numbers your templates keep.
6. Set **Per Icon** to 1, so that each heart is one point of health. At its starting 2, a health of 3 is drawn as one full heart and one half heart.

Try it. Press **F6**, or the **Run Current Scene** button at the top right of Godot. The meter draws one heart for each point of the player's health. Let an **Enemy** reach the player and the last heart turns dark.

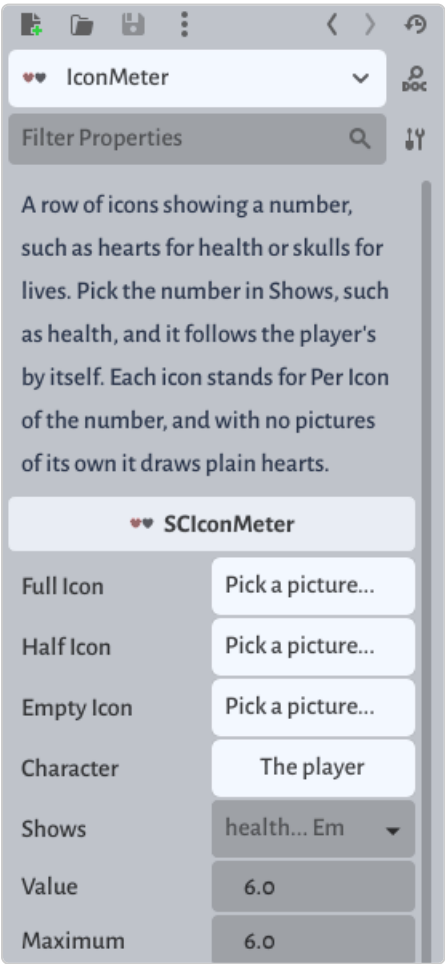


The Scene dock once it is added.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.
IconMeter	Icon Meter	The meter itself. A Control is Godot's plain node for anything drawn on the screen, and the meter draws its row of icons on it.

Its settings



Click **IconMeter** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Full Icon	The picture drawn for an icon that is full, such as a whole heart. While all three picture boxes are empty the meter draws plain hearts of its own. While this one is empty and another has a picture, a full icon is still drawn as a plain heart, so set this one first.	empty
Half Icon	The picture for an icon that is partly full, such as half a heart. Left empty, a part-filled icon is drawn with Full Icon , so 5 out of 6 looks the same as 6 out of 6.	empty

SETTING	WHAT IT DOES	STARTS AS
Empty Icon	The picture for an icon the number no longer reaches, such as a grey heart. Left empty, nothing is drawn there, so lost hearts vanish from the end of the row.	empty
Character	Whose number the meter shows. Empty, the box reads The player and the node follows whichever node is in the player group, which every ready-made player joins. Press the box and pick another character in the window that lists the scene's nodes, and press the X beside it to go back to the player.	empty
Shows	The character's number variable to follow, such as health, picked from a list of the numbers your templates keep. It starts empty, which means the meter shows Value and waits for a machine to set it.	empty
Value	The number shown. It starts at 6. While Shows names a variable the game writes that variable here about ten times a second.	6.0
Maximum	The most the number can be. Divided by Per Icon it gives how many icons are drawn, so 6 at 2 each is three hearts. It follows the variable's Max when the variable has one. When the variable has none and this is still 6, the Output panel says so once as the game runs.	6.0
Per Icon	How much of the number one icon stands for. It starts at 2, which makes each heart two halves, so 5 out of 6 shows two full hearts and a half one. Set it to 1 for lives, keys or shells, where each icon is one.	2.0
Icon Size	How wide and tall each icon is drawn, in pixels. It starts at 32 by 32. Your pictures are stretched to this size.	32, 32
Gap	The space between icons, in pixels, across and down. It starts at 4.	4.0
Per Row	How many icons go on a row before a new row starts under it. It starts at 10. Set it to 0 to keep every icon on one long row.	10

Make it your own

Use your own heart pictures

1. Click **IconMeter** in the Scene dock.
2. Press **Full Icon**, which reads **Pick a picture....** Choose **From my computer...** to copy a picture into the project, or **Pick a picture...** for one the project already has.
3. Do the same for **Empty Icon**, with a grey or hollow heart, so that lost hearts stay on the screen as empty ones.
4. If a heart can be half lost, give **Half Icon** a picture of half a heart and leave **Per Icon** at 2.

Show lives, keys or ammunition

1. Click **IconMeter**, press the ▼ beside **Shows** and pick the number, such as lives. It has to be a number variable in the character's template.
2. Set **Per Icon** to 1, so that each icon is one life.
3. If the variable has no **Max**, set **Maximum** to the most the player can hold, which decides how many icons there are.
4. Give it pictures to suit, such as a skull in **Full Icon**.

Let a machine set the number

1. Click **IconMeter** and leave **Shows** empty, so that it shows its own **Value**.
2. Open the template of the machine that keeps the number. For the player, click it and press **Open Its State Machine...** under its settings at the top of the Inspector.
3. Click a state that runs all through the game, which for a ready-made player is **Playing**. Press **Add Action...**, type Set Property, pick **Set Property** and press **Add**.
4. In **Node** pick **IconMeter**. In **Property** press the little arrow and pick **Value**.
5. Beside **Value** press **Var** and pick the variable that holds the number. It should be a float, the kind of number the meter's **Value** holds.
6. Press the  on the action's card so that it lights. The action then runs every frame and the icons keep up with the variable.

Move it and size it

1. Click **IconMeter** and drag it in the 2D view to where you want it.
2. Set the size of the icons with **Icon Size** and the space between them with **Gap**.
3. To keep it in a corner on every screen size, open **Layout** in the Inspector, set **Layout Mode** to **Anchors** and pick a place in **Anchors Preset**, such as **Top Left**.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The hearts never change in the game	Shows is empty, so nothing is followed, or it names a variable the character does not have, in which case the meter goes on showing Value , a yellow warning sign shows beside IconMeter in the Scene dock, and the Output panel says so once in the game. Pick the variable from the list beside Shows .
There are fewer hearts than expected	The number of icons is Maximum divided by Per Icon , rounded up. A health of 3 at the starting Per Icon of 2 is two hearts, the second one half full. Set Per Icon to 1 for one heart a point.
My pictures are set but the full hearts are plain red ones	Only Half Icon or Empty Icon has a picture. A full icon with no picture of its own is drawn as a plain heart, so that the number still shows. Set Full Icon .
Lost hearts disappear instead of turning grey	The meter has pictures of your own but Empty Icon has none, so nothing is drawn past the number. Give it a picture of an empty heart.
A yellow warning sign shows beside IconMeter in the Scene dock	Hover over the sign to read why. It appears when the character it follows has no Machine, which is the node that runs a template, so there are no numbers to read. It appears when Character points at a node that is no longer in the scene, and when the character's template has no variable of the name in Shows . Press Character and pick a character that runs a template, or pick the number again under Shows . The sign shows only while Shows names a variable.

See also **Globe** 132, **Bar** 127, **Stat Panel** 122, **Hurt Box** 180

Inventory Grid NODE

A grid of slots that draws what an **Inventory** is carrying, with a count on any stack of more than one. Added to a level it sits in the bottom left corner and shows the player's pack straight away. Choosing an item in it uses the item or puts it on.



Inventory Grid in a running game.

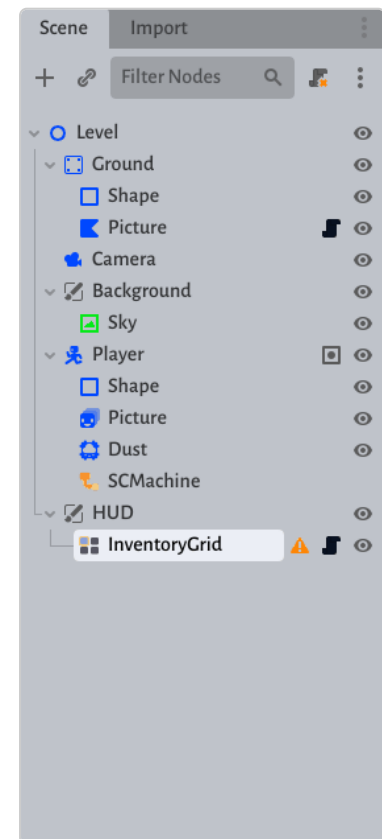
What it is good for

The player's pack, a chest that has been opened, a shop's shelf, or what a character is wearing. One grid shows one inventory, so a pack and the things worn are two grids. For a page of every item in the game, found or not, pick the **Item Collection**, and for a potion on a hotkey the **Skill Bar**.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it. The player needs something to carry things in, so right-click the player and choose **NodeWorthy** → **Give It** → **Inventory**. A player that already has an **Inventory** node under it, such as one brought by an **Item Pickup**, needs nothing more.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Inventory Grid**. Clicking the **Inventory** and pressing **Show It On Screen** at the top of the Inspector does the same.
3. A node called **InventoryGrid** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
4. The editor shows it in the 2D view, even in a 3D game, in the bottom left corner, as a row six slots wide. With an empty pack it draws six empty slots, so there is something to see.
5. In the editor it draws what the inventory starts with. To put something in, click the **Inventory** under the player and fill in **Starts With** in the Inspector, the panel on the right that shows the settings of whatever is clicked.

Try it. Press **F6** and pick something up, for instance from an **Item Pickup**. It appears in the first slot, and items that stack share a slot, with a count in its corner once there is more than one. Click a slot to move the highlight to it and click it again to choose the item, which drinks a potion or puts on a sword.



The Scene dock once it is added.

The first click on the grid only brings the highlight up, even on the first slot, so a potion is never drunk by one stray click.

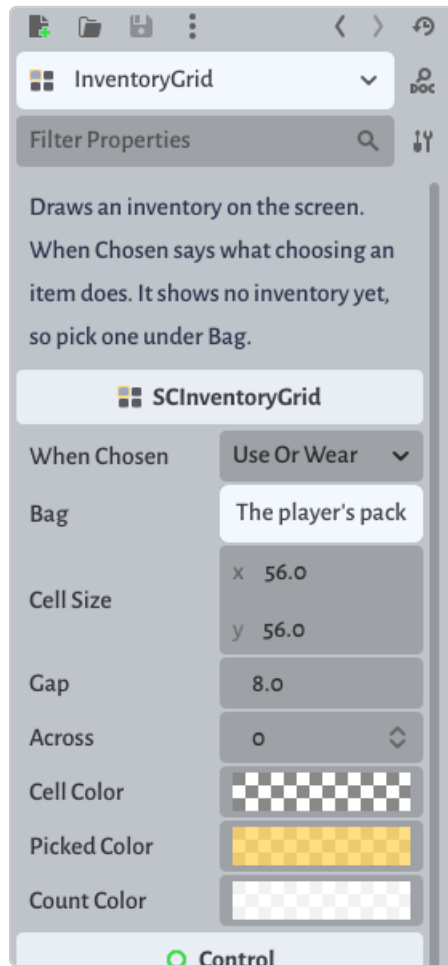
What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.
InventoryGrid	Inventory Grid	The grid itself. A Control is Godot's plain node for anything drawn on the screen, and the grid draws its slots, pictures and counts on it. It can take the focus, which is Godot's word for being the one thing on the screen that key presses go to.

The controls

PRESS	WHAT IT DOES
A click on a slot	Moves the highlight to that slot and gives the grid the keyboard. A click on the slot the highlight is already on chooses the item in it.
The arrow keys	Move the highlight one slot, while the grid has the keyboard. A held key keeps moving. At an edge of the grid the press moves on to whatever sits beside the grid on the screen.
Enter or the Space bar	Chooses the item under the highlight. On an empty slot nothing is chosen, and the press is left for anything else waiting for it, such as a Dialogue Box moving on.
The D-pad or the left stick on a gamepad	Moves the highlight one slot for each push.
The bottom button on a gamepad	Chooses the item under the highlight, with nothing to set up.
Resting the pointer on an item	Shows its name and its description.

Its settings



Click **InventoryGrid** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
When Chosen	What choosing an item does. Use Or Wear , which a new grid starts with, drinks a potion, puts on a sword, takes it off again when it is chosen where it is worn, and uses an item that is used with a button. Nothing leaves it to a template, as a shop that sells what is chosen needs, and the grid sends its chosen signal either way. Choices: Use Or Wear, Nothing.	Use Or Wear
Bag	Which inventory the grid draws. Empty, the box reads The player's pack, and the grid shows an Inventory on its own parent if there is one, or else the one under the player. Press it and pick a chest, a shop or an Inventory node itself to show that one.	empty
Cell Size	How wide and tall each slot is drawn, in pixels. It starts at 56 by 56. An item's picture is fitted inside the slot and keeps its shape.	56, 56
Gap	The space between slots, in pixels, across and down. It starts at 8.	8.0
Across	How many slots go on one row. At 0, the start, the grid fits as many as its width holds, so making it wider in the 2D view adds columns. A number such as 6 fixes it at six across whatever its width.	0

SETTING	WHAT IT DOES	STARTS AS
Cell Color	The colour of each slot's background, a faint white to start with. A game look made in the Theme tab of the NodeWorthy panel can give slots a look of its own, which is then drawn in place of this colour.	1, 1, 1, 0.06
Picked Color	The colour of the frame round the slot the highlight is on, a warm gold to start with. The frame is drawn only while the grid has the keyboard.	1, 0.85, 0.42, 0.85
Count Color	The colour of the count on a stack of more than one, white to start with. It also colours an item's name where the item has no picture, and, more faintly, the slot names of a worn inventory.	1, 1, 1, 0.9

Make it your own

Show a chest's or a shop's inventory

1. Add another grid with **NodeWorthy** → **Add To Screen** → **Inventory Grid**.
2. Click it and press the **Bag** box in the Inspector, which reads The player's pack.
3. In the window that lists the scene's nodes, pick the chest or the shop, or the **Inventory** node under it.
4. The new grid goes just above the first one, clear of it. Drag it wherever it should be in the 2D view.

Show what the player is wearing

1. Right-click the player and choose **NodeWorthy** → **Give It** → **Inventory** for a second inventory.
2. Click the new **Inventory**. In the Inspector, under **Worn**, press **Add Slot** under **Slot Names** and pick Weapon, and add any other slots your game wears things in the same way.
3. Press **Show It On Screen** at the top of the Inspector. A grid is added that draws one slot for each slot name, with the name written in it while it is empty. It goes just above the first grid, so both can be seen and clicked.
4. Run the scene and choose a sword in the pack. It moves to the Weapon slot, and choosing it there takes it off again.

Make it six slots wide, and move it

1. Click **InventoryGrid** and set **Across** to 6. The slots are laid out six to a row, with more rows under them as the pack fills.
2. Set **Cell Size** for bigger or smaller slots.
3. Drag the grid in the 2D view to where you want it. It is fixed to the bottom left corner and grows upwards as rows are added.

Let the keyboard reach it from the start

1. Click the player in the Scene dock and press **Open Its State Machine...** under its settings at the top of the Inspector. The NodeWorthy panel shows the player's template.
2. Click the **Playing** box, then press **Add Action...** in the **Actions** tab on the right.
3. Type Grab Focus in the search box, pick **Grab Focus** and press **Add**.
4. In the action's **Node** pick **InventoryGrid**.
5. Run the scene. The highlight is on the first slot from the start, the arrow keys move it, and **Enter** chooses. A hero that also walks with the arrow keys walks as the highlight moves.

Let a template decide what choosing does

1. Click **InventoryGrid** and set **When Chosen** to **Nothing**.
2. Open the template that should decide, such as a shopkeeper's, with **Open Its State Machine...**
3. Click the state it waits in, press **Add Action...** and add **Event On Signal**.
4. In **Node** pick **InventoryGrid**, and in **Signal Name** pick chosen. In **Event Name** type a name for the event, such as Bought. In **Store Argument In** pick a Resource variable to hold the item, or make one with **New variable...**
5. Add a transition that waits for that event and leads to a state that does the work. An action there, such as **Remove Item**, reads the item from the variable when you press **Var** beside its **Item**.

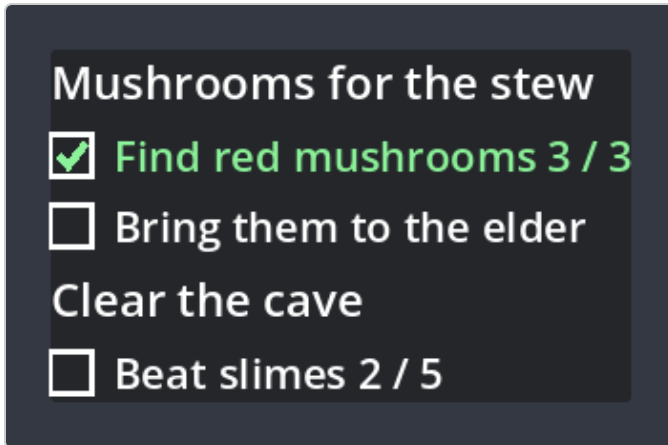
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The grid says No inventory to show . Put an Inventory under the player	There is no Inventory for it to draw, and a yellow warning sign shows beside it in the Scene dock. Right-click the player and choose NodeWorthy → Give It → Inventory , or press Bag and pick an inventory.
The arrow keys do nothing	The grid listens to the keyboard only while it has the focus. Click the grid in the game, or add a Grab Focus action pointed at it.
The slots are stacked in one column	With Across at 0 the grid fits as many slots across as its width holds, and a narrow grid holds one. Drag its right edge wider in the 2D view, or set Across to the number you want.
Choosing an item does nothing	Something that is only carried has no use. Click the item's file and set What It Is to Used Up for a potion or Worn for a sword. When an item cannot be used or worn, the reason is written in Godot's Output panel.

See also **Inventory** 185, **Item Pickup** 91, **Chest** 104, **Item Collection** 175, **Dialogue Box** 171

Quest List NODE

A panel that lists the quests under way and their steps, with a box against each step that is ticked when the step is done, and a count where a step needs more than one, as in Find mushrooms 2 / 3. Added from the menu it sits in the top right corner. It keeps itself up to date and hides while there are no quests to show.



Quest List in a running game.

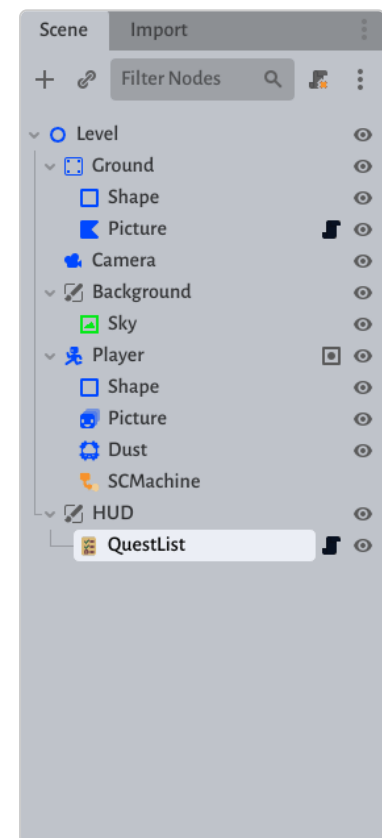
What it is good for

A quest tracker in a corner of any game with quests, missions or challenges. With descriptions and finished quests switched on it becomes a journal page. For a full screen the player opens with a key, with buttons to track a quest or give it up, add the **Quest Log** as well, and add a **Minimap** to point the way.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it. The list needs a quest before it has anything to show in the game, and the quickest giver is a **Villager** with a quest picked in **Gives Quest**.
2. Picking a quest in a **Villager's Gives Quest**, or adding a **Start Quest** action, puts a Quest List on the screen by itself when the scene has none. To add one by hand, go on to the next step.
3. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Quest List**.
4. A node called **QuestList** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
5. The editor shows it in the 2D view, even in a 3D game, in the top right corner. It draws an example there, Mushrooms for the stew with two steps, so that you can see its size and its look. The example is never shown in the game.
6. If **Quest List** is greyed out in the menu, the scene already has one, and a scene needs only one. Hover over the greyed line to read where it is, then click that node in the Scene dock.

Try it. Press **F6**, or the **Run Current Scene** button at the top right of Godot. The list is hidden at first. Start a quest, for instance by talking to the **Villager**,



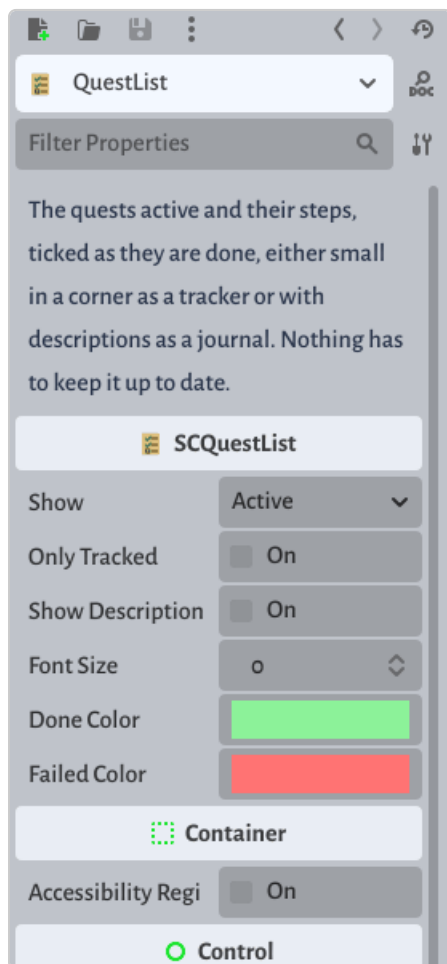
The Scene dock once it is added.

and it appears with the quest's title and its steps. Each step's box is ticked and turns green as the step is done, and when every step is done the quest leaves the list.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.
QuestList	Quest List	The list itself. A PanelContainer is Godot's box with a background that grows to fit what is inside it, so the list gets taller as quests are added. It takes its background and its font from the game's look.

Its settings



Click **QuestList** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Show	Which quests are listed. Active lists only quests under way, which makes a tracker. Active And Done adds finished quests, marked (done), and Every One adds failed and abandoned quests too, which makes a journal. Choices: Active, Active And Done, Every One.	Active
Only Tracked	Ticked, the list shows only the quests the player is tracking. Every quest is tracked from the moment it begins, and the Track Quest action or the buttons in a Quest Log stop and start tracking one. It starts unticked, which lists every quest under way.	off
Show Descriptions	Ticked, each quest's Description is written under its title, a little fainter, and each step's under the step. It starts unticked, which keeps a corner tracker short.	off
Font Size	How big the words are. At 0, the start, the list uses the text size of the game's look, which is set in the Theme tab of the NodeWorthy panel. Titles are drawn a little bigger than steps.	0
Done Color	The colour of a step that is done and of its tick, and of a finished quest's title. It starts as a soft green.	0.55, 0.95, 0.6, 1
Failed Color	The colour of the title of a quest that was failed or abandoned, a soft red to start with. It is seen only when Show is Every One .	1, 0.45, 0.45, 1

Make it your own

Turn it into a journal

1. Click **QuestList** in the Scene dock.
2. Set **Show** to **Every One**, so that finished, failed and abandoned quests stay on the list.
3. Tick **Show Descriptions**, so that each quest and each step says what it is about.
4. Drag it to the middle of the screen in the 2D view, since a journal takes more room than a tracker.

Move it and make it wider

1. Click **QuestList** and drag it in the 2D view to where you want it.
2. It grows to fit its words and starts 260 pixels wide. Drag the handle on its left edge to make it narrower, down to what its words need.
3. To keep it against another corner on every screen size, pick a place in **Anchors Preset** under **Layout**, such as **Top Left**. It starts at **Top Right**.

Let the player choose which quests are listed

1. Click **QuestList** and tick **Only Tracked**.
2. Add a quest log with **NodeWorthy** → **Add To Screen** → **Quest Log** on the top node.
3. Run the scene, start two quests and press **Q** to open the log.
4. Click a quest in the log and press **Stop Tracking**. Close the log, and that quest is gone from the list. It is still under way, and its steps still count.

Begin a quest when a chest is opened

1. Click a **Chest** in the Scene dock and press **Open Its State Machine...** under its settings at the top of the Inspector. The NodeWorthy panel shows its template.
2. Click the **Opening** box, then press **Add Action...** in the **Actions** tab on the right.
3. Type Start Quest in the search box, pick **Start Quest** and press **Add**.
4. In the action's **Quest** pick the quest to begin.
5. Run the scene and open the chest. The quest appears in the list with its steps.

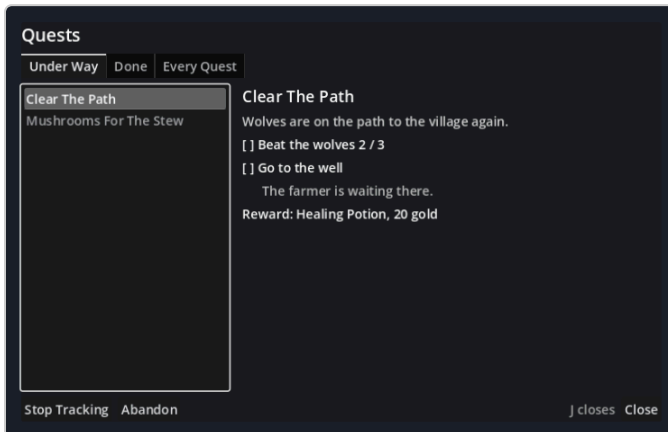
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The list is missing when the game runs	It hides itself while no quest is under way. Start one with a Villager's Gives Quest , a Start Quest action, or an answer in a dialogue.
Quest List is greyed out in the menu	A scene needs only one, and a second would sit on top of the first. Hover over the greyed line to read where the one in the scene is.
A quest is under way but not listed	The quest has Hidden ticked, which keeps it off every list, or Only Tracked is ticked and the quest is not being tracked.
Finished quests vanish from the list	Show is Active , which lists only quests under way. Set it to Active And Done to keep finished ones, marked (done).
The editor shows a quest about mushrooms that is not mine	The editor always draws the same example, because quests exist only while the game runs. Your own quests show when you run the scene.

See also **Quest Log** 150, **Minimap** 155, **Villager** 97, **Dialogue Box** 171

Quest Log NODE

The player's quest log, a screen of its own that opens over the game with the **Q** key. It lists the quests down one side and sets out the one picked beside them, with its description, its steps, its reward, and buttons to track it or give it up. The game stands still while it is open.



Quest Log in a running game.

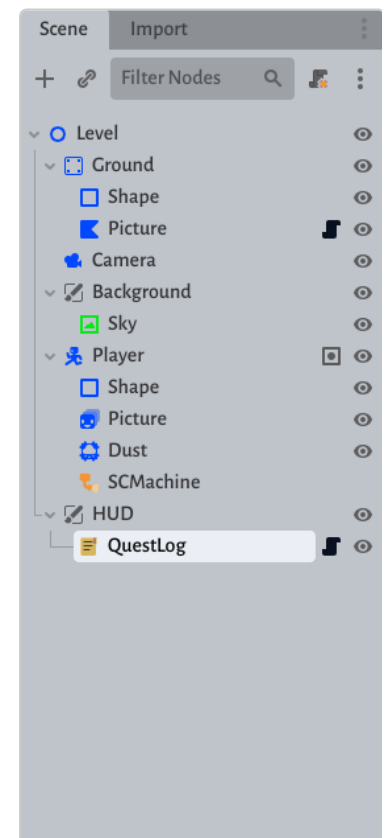
What it is good for

Any game with more quests than a corner tracker can hold, where the player wants to read a quest again, choose which ones to follow, or drop one. Keep a **Quest List** on the screen for the steps in hand and a **Minimap** to point the way, since both follow the tracking the player sets here.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it. The log has something to show once the game has a quest, such as one given by a **Villager**.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Quest Log**.
3. A node called **QuestLog** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
4. The editor shows it in the 2D view, even in a 3D game. It covers most of the screen, 80 pixels in from the sides and 60 from the top and bottom, and draws an example with two quests so that you can see how it is laid out.
5. It covers the other parts of the screen while you work. Click the eye beside **QuestLog** in the Scene dock to hide it in the editor. It still opens in the game.
6. If **Quest Log** is greyed out in the menu, the scene already has one, and a scene needs only one. Hover over the greyed line to read where it is, then click that node in the Scene dock.

Try it. Press **F6**, start a quest, then press **Q**. The log opens over the game, which stops. Click a quest on the left to read it on the right, press **Stop Tracking** and its name goes fainter in the list, then press **Q** or **Escape** to close the log and carry on.



The Scene dock once it is added.

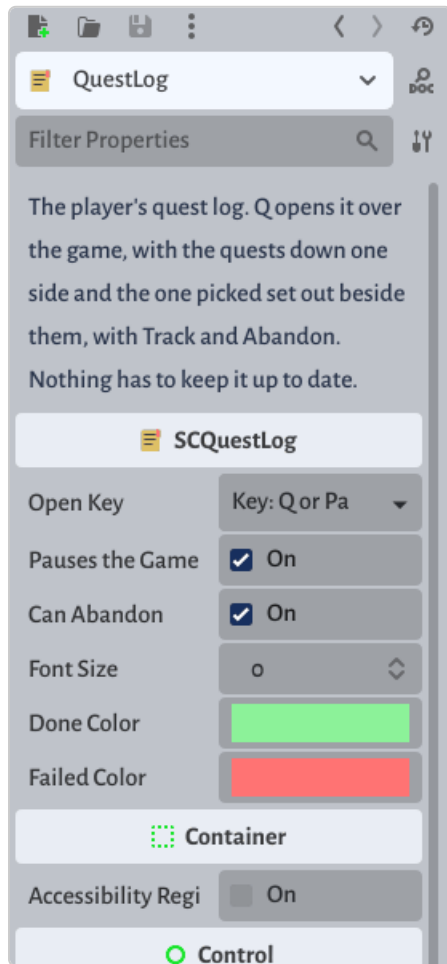
What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.
QuestLog	Quest Log	The log itself. A PanelContainer is Godot's box with a background, and the log builds its heading, its tabs, its list and its buttons inside it when the game starts. It takes its look from the game's look.

The controls

PRESS	WHAT IT DOES
Q , or the Select button on a gamepad, which some gamepads call Back	Opens the log, and closes it again. This is the Open Key setting.
Escape	Closes the log.
The up and down arrow keys	Move through the list of quests. The list has the keyboard as soon as the log opens with a quest in it.
A click	Picks a quest in the list, changes between the Under Way , Done and Every Quest tabs, or presses Track , Stop Tracking , Abandon or Close .

Its settings



Click **QuestLog** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Open Key	The key, or gamepad button, that opens the log and closes it again. It starts as the Q key, which no ready-made player or thing uses, or the Select button on a gamepad, which some gamepads call Back. To change it, press the little arrow beside the box and pick a key, a mouse button or a gamepad button from the list. Tick the first line of the list before picking and the one picked is added to what is there, so that either opens the log.	Key: Q or Pad: Select / Back
Pauses the Game	Ticked, the start, the whole game stops while the log is open and carries on when it closes. Untick it for a game that should go on behind the log. If something else had already paused the game, the log leaves it paused when it closes.	on
Can Abandon	Ticked, the start, the log has an Abandon button for a quest under way. The first press changes the button to Press again to give it up, and the second gives the quest up. Untick it to leave the button out.	on
Font Size	How big the words are. At 0, the start, the log uses the text size of the game's look, which is set in the Theme tab of the NodeWorthy panel. It sizes every word of the log, the list of quests, the tabs and the buttons among them, with the heading a little bigger. The example in the 2D view follows as you change it.	0

SETTING	WHAT IT DOES	STARTS AS
Done Color	The colour of a finished quest in the list and of a step that is done, a soft green to start with.	0.55, 0.95, 0.6, 1
Failed Color	The colour of a failed or abandoned quest in the list, a soft red to start with.	1, 0.45, 0.45, 1

Make it your own

Open it with another key

1. Click **QuestLog** in the Scene dock.
2. In the Inspector, press the little arrow beside the **Open Key** box and pick the key you want from the list, such as the L key.
3. Run the scene and press the new key. The words at the bottom of the log, which say which key closes it, follow the change.

Keep the game running behind it

1. Click **QuestLog** and untick **Pauses The Game**.
2. Run the scene and open the log. Enemies and timers carry on while the player reads.

Make it smaller

1. Click **QuestLog** and press the **2D** button at the top of Godot if the 2D view is not showing.
2. Drag the handles on its edges inwards. It keeps the same distance from each edge of the screen on every screen size.

Put a quest in the log from a chest

1. Click a **Chest** in the Scene dock and press **Open Its State Machine...** under its settings at the top of the Inspector. The NodeWorthy panel shows its template.
2. Click the **Opening** box, then press **Add Action...** in the **Actions** tab on the right.
3. Type Start Quest in the search box, pick **Start Quest** and press **Add**.
4. In the action's **Quest** pick the quest to begin.
5. Run the scene, open the chest and press **Q**. The quest is in the log, on the **Under Way** tab.

When it goes wrong

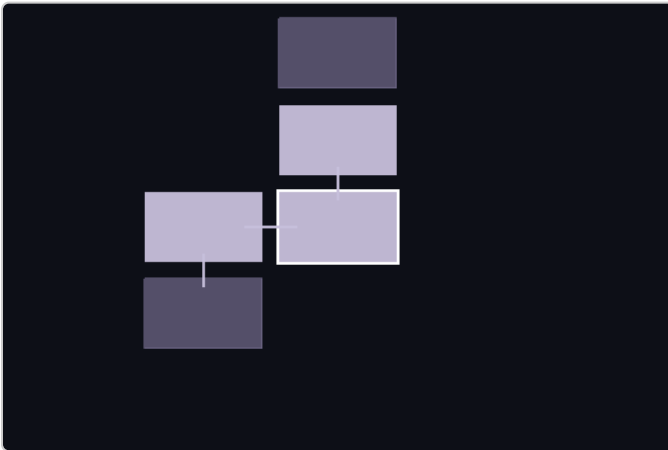
WHAT YOU SEE	WHY, AND WHAT TO DO
Q opens the log and does something else in your game as well	No ready-made player or thing uses Q , so something of your own listens for it too, such as a skill. Change the letter in Open Key to one your game does not use.
The log covers everything in the 2D view	It is drawn in the editor, with an example in it, so that you can see it. Click the eye beside QuestLog in the Scene dock to hide it while you work. It still opens in the game.
A quest is missing from the log	A quest with Hidden ticked is never in it, and a quest that has not begun is on no tab. The Under Way tab shows only quests under way, so look on Every Quest for one that was failed or given up.
Quest Log is greyed out in the menu	A scene needs only one. Hover over the greyed line to read where the one in the scene is.

WHAT YOU SEE	WHY, AND WHAT TO DO
No key opens the log	Open Key has been cleared, or the words in it were changed by mistake. Press the round arrow that Godot shows beside a changed setting to put back the words it started with.

See also **Quest List** 146, **Minimap** 155, **Villager** 97, **Chest** 104

Minimap NODE

A small map in the bottom right corner of the screen. In any level it is a radar, with the player in the middle, red dots for the enemies near them, green for the people to talk to, gold for what there is to pick up or open, a white square for the way out, and a mark for each thing a tracked quest wants now, such as the place to go, the one to talk to and the enemies to beat. Right-click it and choose **NodeWorthy** → **Take Map Picture**, or press the button of that name at the top of its Inspector, and it shows the level itself under the marks, pictured from above. On a floor laid out by the **Build Dungeon** action it becomes a map of the dungeon's rooms.



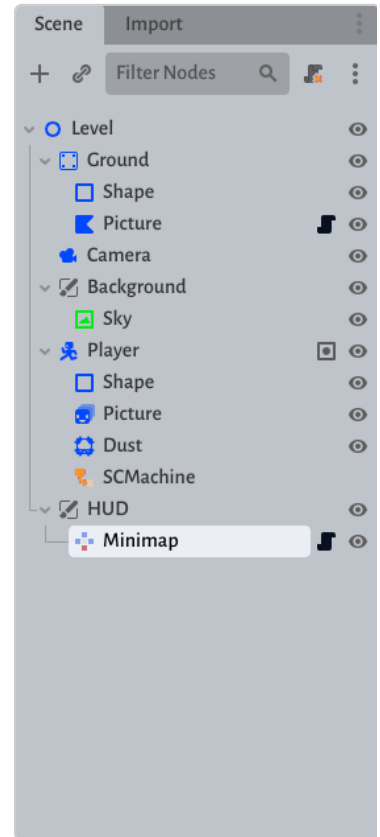
Minimap in a running game.

What it is good for

Games with quests, where the player needs to know which way to go, games with a level big enough to get lost in, and dungeon games built with **Build Dungeon**. Without a picture the radar still marks the enemies, people, items, way out and quest targets, but on a plain circle with no walls or ground. With a picture of the level the player sees the ground, the paths and the buildings round them. Pair it with a **Quest List**, which says what each quest's mark is for, and a **Quest Log**, where the player chooses which quests are tracked.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Minimap**.
3. A node called **Minimap** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
4. The editor shows it in the 2D view, even in a 3D game, in the bottom right corner, 180 pixels wide and tall. Its **Player** setting is already pointed at the player, found by the player group.
5. In the editor it draws a dark circle with a white arrowhead in the middle for the player, a gold diamond for a quest's mark, a gold arrowhead at the edge for one too far away to fit, and a red, a gold and a green dot for an enemy, an item and a person near the player. All five marks are examples, so their colours can be seen.
6. Click **Minimap** in the Scene dock. At the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, are **Take Map Picture**, **Use My Own Picture...** and **Save As A Picture....** Press **Take Map Picture** and the map shows the level from above, with the player's arrowhead on it. The example marks go, and the real ones show over the picture when the scene runs.
7. If **Minimap** is greyed out in the menu, the scene already has one, and a scene needs only one. Hover over the greyed line to read where it is, then click that node in the Scene dock.



The Scene dock once it is added.

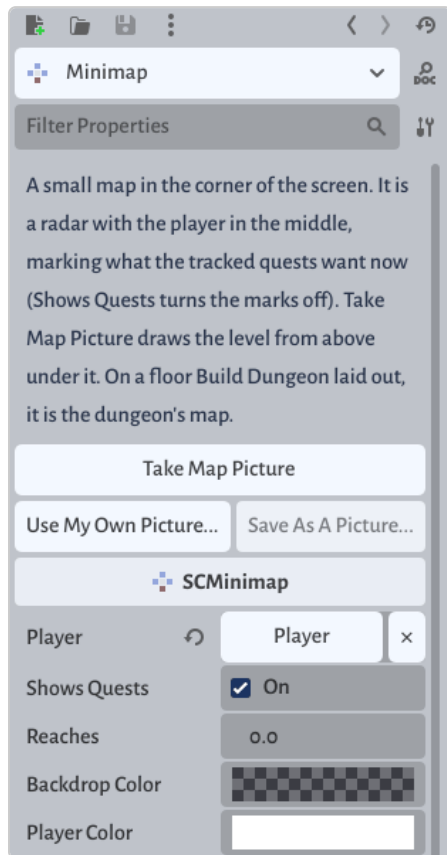
Try it. Press **F6**, or the **Run Current Scene** button at the top right of Godot. With no quest under way the map is the circle, the player's arrowhead and the dots of what is near, over the level's picture if one was taken. Walk about, and the picture slides under the arrowhead, which points the way the player is facing. Start a quest with a step that sends the player to a place, and a diamond in the quest's colour shows where that place is. Walk away until it is further off than the map reaches, and the mark slides to the edge and becomes an arrowhead pointing the way.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Minimap	Minimap	The map itself. A Control is Godot's plain node for anything drawn on the screen, and the map draws its circle, the level's picture and its marks on it, as big as the node's box. A picture taken of the level is kept in the maps folder inside your project's nodeworthy folder.

Its settings



Click **Minimap** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Player	The node the map is centred on. The menu fills it in with the player when it adds the map. Empty, the box reads The player and the map uses whichever node is in the player group.	empty
Shows Quests	Ticked, the start, the radar marks what each tracked quest wants now. Unticked, no quest is marked, but the enemies, people, items and way out still are, each with a tick of its own under Marks .	on
Reaches	How far from the player the radar sees, measured to its edge, in pixels in a 2D game and metres in a 3D one. At 0, the start, it reaches 600 pixels, or 40 metres. A bigger number shows more of the level and packs the marks closer together.	0.0
Backdrop Color	The colour of the circle behind the radar, a dark blue you can see through to start with. With a picture of the level it shows only where the picture ends.	0.05, 0.06, 0.1, 0.6
Player Color	The colour of the arrowhead that stands for the player. It starts white.	1, 1, 1, 1

SETTING	WHAT IT DOES	STARTS AS
Shape	Round , the start, draws the map as a circle, as a radar is. Square draws it as a square, which suits a map of the whole level, and a quest's mark or the way out too far away sits on its square edge. Choices: Round, Square.	Round
Shows Enemies	On to start with. The enemies near the player show as red dots while they are alive. Off, no enemy is marked. An enemy further off than the map reaches is not shown.	on
Shows People	On to start with. The people near the player there are to talk to, the villagers, show as green dots.	on
Shows Items	On to start with. What there is to pick up or open near the player shows in gold, a dot for an item pickup or loot on the ground and a square for a chest, until it is taken.	on
Shows Way Out	On to start with. The way out of the level, the Level Exit, shows as a white square, and when it is further off than the map reaches it sits at the edge as an arrowhead pointing the way.	on
Enemy Color	The colour of the enemies' dots, red to start with.	0.95, 0.28, 0.25, 1
People Color	The colour of the people's dots, green to start with.	0.4, 0.9, 0.45, 1
Item Color	The colour of the items' and chests' marks, gold to start with.	1, 0.82, 0.25, 1
Way Out Color	The colour of the way out's mark, white to start with.	1, 1, 1, 1
Map Texture	The picture of the level drawn under the marks. Take Map Picture at the top of the Inspector fills it in with the level pictured from straight above, and Use My Own Picture... with one of your own. Empty, the start, the map is a plain radar. It is under Map Picture in the Inspector with the settings that follow, which show once there is a picture.	empty
Map Covers	The part of the level the picture shows, as where its top left corner is and how wide and tall it is, in pixels in a 2D game and metres in a 3D one, where the second number is how far north or south. Take Map Picture fills it in. Change it only to line a picture of your own up with the level.	Rect2(0, 0, 0, 0)
Shows	Around The Player , the start, slides the picture under the player as they move and shows as far as Reaches . The Whole Level shows all of the picture at once, with the player's arrowhead moving over it. Choices: Around The Player, The Whole Level.	Around The Player
Turns with Player	Unticked, the start, north stays at the top of the map and the player's arrowhead turns to show which way they face. Ticked, the picture turns instead, so the way the player faces is always up the map. In a 3D game that is the way the camera looks, and in a 2D game the way the player last moved. It shows only with Around The Player .	off
Room Size	On a dungeon floor, how wide and tall each room is drawn, in pixels. It starts at 18 by 11. It is under Dungeon Floor in the Inspector with the settings that follow, and none of them changes the radar.	18, 11
Gap	On a dungeon floor, the space between rooms on the map, in pixels. It starts at 3.	3.0

SETTING	WHAT IT DOES	STARTS AS
Visited Color	On a dungeon floor, the colour of the rooms the player has been in, and of the short bridges that join rooms. It starts as a pale lilac.	0.78, 0.75, 0.86, 0.95
Known Color	On a dungeon floor, the colour of rooms the player knows about but has not entered, which are the rooms joined to one already visited. It starts dimmer than Visited Color .	0.45, 0.42, 0.55, 0.7
Current Color	On a dungeon floor, the colour of the frame round the room the player is in. It starts white.	1, 1, 1, 1
Boss Color	On a dungeon floor, the colour of the small square that marks the boss room once it is known. It starts red.	0.9, 0.25, 0.35, 1
Treasure Color	On a dungeon floor, the colour of the small square that marks the treasure room once it is known. It starts gold.	0.98, 0.8, 0.3, 1
Shop Color	On a dungeon floor, the colour of the small square that marks the shop once it is known. It starts green.	0.4, 0.8, 0.5, 1
Secret Color	On a dungeon floor, the colour of the small square that marks the secret room. A secret room stays off the map until the player has found it. It starts violet.	0.6, 0.45, 0.95, 1

Make it your own

Make it see further, or less far

1. Click **Minimap** in the Scene dock.
2. Set **Reaches** to the distance from the player to the edge of the map. In a 2D game try 1200 to see twice as far as it does to start with, and in a 3D game try 80.
3. Run the scene. A quest's mark or the way out that sat at the edge as an arrowhead now shows inside the circle, and enemies, people and items further off are marked too.

Show the level under the map

1. Open the level and add the ground, walls, tiles and buildings first, as the picture shows the level as it is when it is taken.
2. Click **Minimap** in the Scene dock.
3. Press **Take Map Picture** at the top of the Inspector. The level is pictured from straight above, leaving out the player, enemies, anything that moves and anything picked up, and the map in the 2D view shows it at once.
4. Run the scene. The ground round the player slides under the arrowhead as they walk.
5. After changing the level, press **Take Map Picture Again**. **Undo** puts back the picture before.

Show the whole level at once

1. Take a picture of the level as above.
2. Set **Shows** under **Map Picture** to **The Whole Level**, and **Shape** to **Square**.
3. Drag a corner of the Minimap in the 2D view to make it as big as the level should look.
4. Run the scene. The whole level is on the map, and the arrowhead moves over it as the player walks.

Turn the map as the player turns

1. Take a picture of the level as above, with **Shows** at **Around The Player**.
2. Tick **Turns with Player** under **Map Picture**.
3. Run the scene and turn. The way the player faces stays at the top of the map and the picture turns under it, as maps in many 3D games do.

Draw a map of your own

1. Take a picture of the level, then press **Save As A Picture...** and save it somewhere you can find it.
2. Paint over it in any picture editor, for example to draw it as an old parchment map. Keep its width and height as they are.
3. Press **Use My Own Picture...** and pick the painted picture. It covers the same part of the level as the one it was painted over, so everything lines up.
4. A picture drawn from nothing works too. Set **Map Covers** to the part of the level it shows.

Move it, and make it bigger or smaller

1. Click **Minimap** and drag it in the 2D view to where you want it.
2. Drag a corner handle outwards to make it bigger, or inwards to make it smaller. The circle fills the node's box.
3. To keep it in another corner on every screen size, pick a place in **Anchors Preset** under **Layout**, such as **Top Right**. It starts at **Bottom Right**.

Change its colours

1. Click **Minimap** and set **Backdrop Color** and **Player Color** for the circle and the player's arrowhead. The map in the 2D view follows as you pick.
2. A quest's mark takes the quest's own colour. Click the quest's file in the FileSystem dock, the list of the project's files at the bottom left, and set its **Color** in the Inspector.
3. Run the scene to see the change.

Let the player choose which quests are marked

1. Add a quest log with **NodeWorthy** → **Add To Screen** → **Quest Log** on the top node.
2. Run the scene, start a quest and press **Q** to open the log.
3. Click the quest and press **Stop Tracking**. Close the log, and the quest's marks are gone from the map. Press **Track** in the log to bring them back.

Begin a quest when a chest is opened

1. Click a **Chest** in the Scene dock and press **Open Its State Machine...** under its settings at the top of the Inspector. The NodeWorthy panel shows its template.
2. Click the **Opening** box, then press **Add Action...** in the **Actions** tab on the right.
3. Type Start Quest in the search box, pick **Start Quest** and press **Add**.
4. In the action's **Quest** pick the quest to begin.
5. Run the scene and open the chest. The quest begins, tracked from the start, and the map marks what its first step wants.

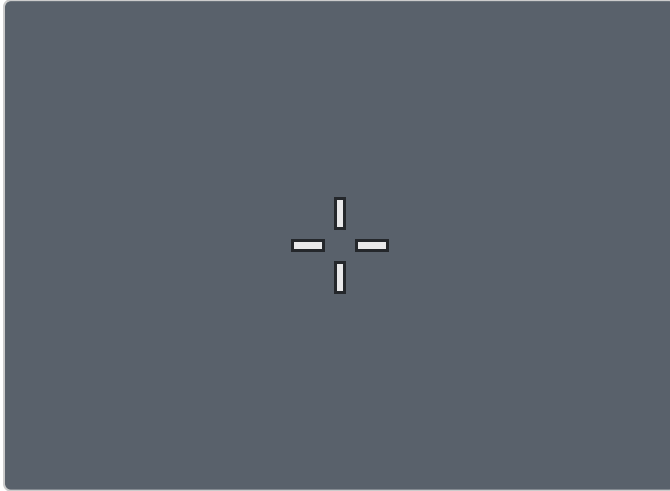
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The map is an empty circle	The radar marks the enemies, people, items and chests within Reaches of the player, the way out however far off it is, and what each tracked quest wants now. With none of those near, no Level Exit in the level and no tracked quest with a step that names a place, someone to talk to or an enemy to beat, there is nothing to mark. Your own enemies, people and things get their dots and squares only when they are in the same group as the ready-made ones, which are called enemies, villager, pickups, chests and exits. Without a picture it draws no walls. Press Take Map Picture to show the level under it.
A quest's mark is missing	The quest is not tracked, it has Hidden ticked, the step is already done, or the quest has In Order ticked and that step's turn has not come. A step that sends the player to a place is marked only when a node of that name is in the level.
In a 3D game the marks do not turn as the camera turns	The radar turns only with a picture of the level and Turns with Player ticked. Otherwise up on the map is always the same direction in the level, whichever way the player or the camera faces.
The picture shows the level as it was	A picture is taken once and kept. After moving walls, painting tiles or adding buildings, press Take Map Picture Again .
Take Map Picture says there is nothing to take a picture of	The level has no ground, walls, tiles or pictures yet, only characters and things that move, which are left out. Add the level's ground first.
A picture of my own is in the wrong place	It covers what Map Covers says. Paint over a picture taken here and keep its size, or change Map Covers until it lines up with the level.
Minimap is greyed out in the menu	A scene needs only one. Hover over the greyed line to read where the one in the scene is.
Godot's warning sign shows beside the Minimap in the Scene dock	Rest the pointer on the sign to read why. The map centres on the player, so the level needs one, added with NodeWorthy → Add Player , or pick who it follows under Player . It belongs on the screen, under a CanvasLayer where NodeWorthy → Add To Screen puts it, or it slides away as the camera moves. A Map Picture needs Map Covers set too. Fix it and the sign goes.

See also [Quest List](#) 146, [Quest Log](#) 150, [Villager](#) 97, [Enemy](#) 74

Crosshair NODE

The aiming mark in the middle of a shooter's screen. Put on the HUD, it fills the screen and draws a white cross in the very centre when the game runs, so nothing needs placing. A machine opens it, tints it and shows its hit marks by setting its properties.



Crosshair in a running game.

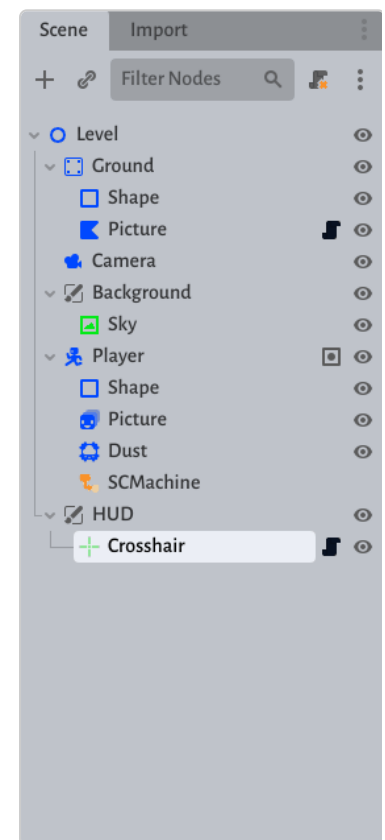
What it is good for

First-person and third-person shooters, and any game where the player aims at the middle of the screen. It says how steady the aim is by opening and closing, says what is under it by its colour, and confirms a hit. A top-down or side-on game that aims with the mouse pointer has no use for it.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 3D Level...** with **Third Person Hero (3D)** picked under **The Player** in its window.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Crosshair**.
3. A node called **Crosshair** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
4. It starts as four short white marks with a gap in the middle and a dark edge round them. Click **Crosshair** and pick another shape in **Look** in the Inspector, the panel on the right that shows the settings of whatever is clicked.
5. The editor shows it in the 2D view, even in a 3D game, in the centre of the rectangle that stands for the game's screen. Press **F6** to see it over the game.
6. If **Crosshair** is greyed out in the menu, the scene already has one, and a scene needs only one. Hover over the greyed line to read where it is, then click that node in the Scene dock.

Try it. Press **F6**, or the **Run Current Scene** button at the top right of Godot. The cross sits in the exact centre of the screen and stays there whatever size



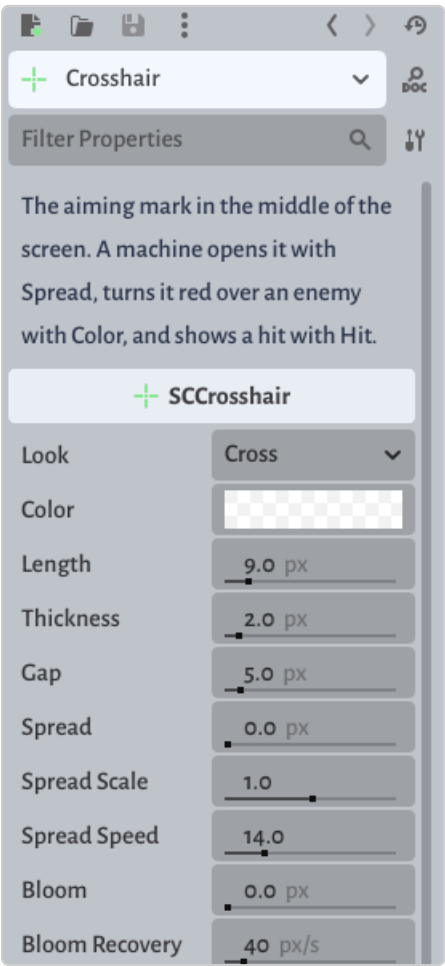
The Scene dock once it is added.

the window is. It does nothing more until a machine sets **Spread**, **Bloom**, **Color** or **Hit**. For a quick look at what they do, stop the game, type 20 into **Spread** in the Inspector and watch the marks ease apart in the 2D view, then set it back to 0.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.
Crosshair	Crosshair	The mark itself. A Control is Godot's plain node for anything drawn on the screen. Under a CanvasLayer the Crosshair stretches itself over the whole screen each time the game starts, and draws in the middle of it.

Its settings



Click **Crosshair** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Look	The shape of the mark. Cross is four marks with a gap in the middle, Cross And Dot adds a dot in the centre, Dot is the dot alone, and Circle is a ring that grows as the mark opens, the way a shotgun's does. Choices: Cross, Cross And Dot, Dot, Circle.	Cross
Color	The colour of the marks, white and very slightly see-through to start with. A machine can change it to say what is under the aim, such as red over an enemy.	1, 1, 1, 0.9
Length	How long each of the four marks is, in pixels. It starts at 9. With Circle it adds to the size of the ring.	9.0
Thickness	How thick the marks are, in pixels. It starts at 2. The dot grows with it.	2.0
Gap	The space between the centre and where each mark begins when the mark is fully closed, in pixels. It starts at 5.	5.0
Spread	How far the marks open beyond Gap , in pixels. It starts at 0. A machine sets it higher while the player runs or jumps and lower while they stand still, and the marks ease to the new place by themselves.	0.0
Spread Scale	How much of Spread is shown, as a share, where 1 is all of it. Aiming down the sights can set 0.3 to keep the marks tight without touching the Spread that walking and running set.	1.0
Spread Speed	How quickly the marks follow a change in Spread . It starts at 14. A higher number snaps open and shut, and a lower one drifts.	14.0
Bloom	A kick open on top of Spread , in pixels. A machine sets it for each shot, and it closes again by itself at Bloom Recovery , so holding fire keeps the marks open and letting go lets them settle.	0.0
Bloom Recovery	How fast Bloom closes again, in pixels a second. It starts at 40, so a kick of 20 is gone in half a second.	40.0
Outline	Draws a dark edge round the marks, so that they can be seen against a bright wall or sky. It starts ticked.	on
Outline Color	The colour of that edge, a black you can partly see through to start with.	0, 0, 0, 0.6
Hit	How many seconds the hit marks show, which are four short slanting marks round the centre. A machine sets it when a shot lands, and it counts down to 0 by itself, fading out at the end.	0.0
Hit Color	The colour of the hit marks. It starts white.	1, 1, 1, 1

Make it your own

Change its shape and colour

1. Click **Crosshair** in the Scene dock.
2. Pick a shape in **Look**, such as **Cross And Dot**.
3. Set **Color**, and make the marks longer, thicker or further apart with **Length**, **Thickness** and **Gap**.

Open it while the hero sprints

1. Click the player, a **Third Person Hero (3D)** in this example, in the Scene dock and press **Open Its State Machine...** under its settings at the top of the Inspector. The NodeWorthy panel shows its template.
2. Double-click the **Playing** box, which is marked , to see the states inside it.
3. Click **Sprinting**, press **Add Action...**, type Set Property, pick **Set Property** and press **Add**.
4. In **Node** pick **Crosshair**. In **Property** press the little arrow and pick Spread.
5. Beside **Value** press the button that says what kind of value it is, choose float, and type 14.
6. Do the same on **On The Ground**, with a value of 0.
7. Run the scene and hold **Shift** while walking. The marks ease open as the hero sprints and close again when it slows to a walk.

Show a hit when a shot lands

1. Open the template of whatever does the shooting, and click the state it goes to when a shot has hit something.
2. Add a **Set Property** action there. In **Node** pick **Crosshair**, in **Property** pick Hit, and set **Value** to a float of 0.2.
3. Each time that state is entered, the four hit marks show round the centre for a fifth of a second and fade.

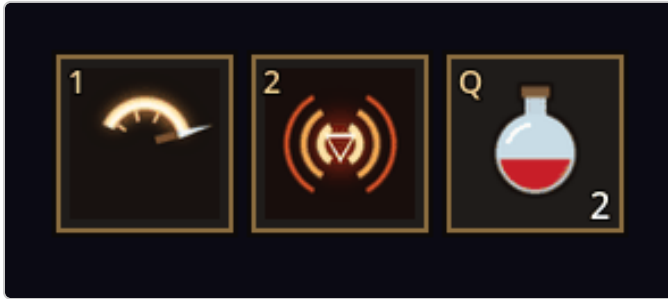
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
Moving it in the 2D view changes nothing in the game	Under a CanvasLayer the Crosshair stretches over the whole screen each time the game starts and draws in the middle of it. It is made to stay in the centre.
In the game it is not in the centre of the screen	It was added under a panel or another Control, where it keeps the size of its own box and draws in the middle of that. In the Scene dock, drag it onto HUD .
A Set Property action does nothing to it	The Value of a new Set Property starts as a tick box. Spread , Bloom and Hit hold numbers, so set the kind of value to float first. When the kinds do not match, nothing is set and the Output panel says so.
The marks start wide open, or the hit marks flash, every time the game starts	A number was left in Bloom or Hit in the Inspector. The editor shows them as set, so that you can see how they look, and the game counts them down from there. Set both back to 0.

See also **Third Person Hero (3D)** 37, **Enemy** 74, **Hit Box** 200, **Hurt Box** 180

Skill Bar NODE

A row of skill slots along the bottom of the screen, each with its picture and its key. When the player presses a skill's key or clicks its slot, and the skill is ready and the mana is there, the mana is paid, the skill's event is sent to the player's machine, and the slot darkens and sweeps back as it comes ready again. Until you add a skill the editor draws one empty slot with the words No skills yet, and the game draws nothing.



Skill Bar in a running game.

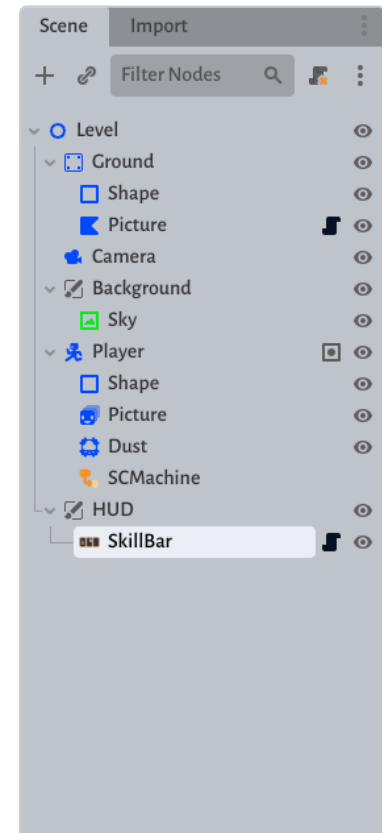
What it is good for

Action role-playing games, and any hero with special moves on the number keys, such as a whirl of the sword, a heal or a fireball. A skill can also stand for an item, such as a potion on a hotkey, and then its slot shows how many are carried. The bar only says when a skill is used, and what the skill does is a state in the character's template, so expect to open the template. Pair it with a **Globe** that shows the mana.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Skill Bar**.
3. A node called **SkillBar** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
4. It is placed along the bottom of the screen in the middle, as one empty slot with the words No skills yet beside it.
5. Click **SkillBar**. In the Inspector, the panel on the right that shows the settings of whatever is clicked, press **Add Skill** under **Skills**. A slot appears on the bar in the 2D view, with the word Skill in it and a 1 in its corner for its key.
6. Fill in the skill's row. Give it a **Skill Name**, an **Icon**, a **Cooldown** in seconds and an **Event Name**, such as HEAL.
7. Press **Add Skill** again for the next skill, which is given the 2 key.

Try it. Press **F6** and press **1**, or click the slot. The slot flashes, goes dark, and the dark part sweeps away like a clock hand while the seconds left are counted down on it. Press **1** again before it is ready and the slot's edge flashes red. Nothing else happens in the game until a state in the player's template waits for the skill's event.



The Scene dock once it is added.

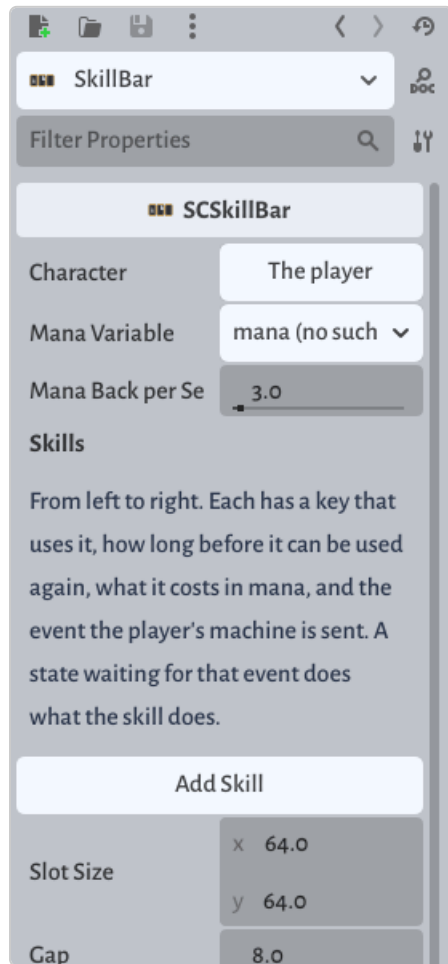
What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.
SkillBar	Skill Bar	The bar itself. A Control is Godot's plain node for anything drawn on the screen, and the bar draws its slots on it. It also listens for each skill's key and for clicks on its slots.

The controls

PRESS	WHAT IT DOES
Each skill's Key	Uses that skill. The first skill added is given the 1 key, the second the 2 key, and so on.
A click on a slot	Uses that skill, the same as its key.
Resting the pointer on a slot	Shows the skill's name, its key, its description, what it costs in mana and how long it takes to come ready again.

Its settings



Click **SkillBar** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Character	Whose skills these are, which means whose mana pays for them and whose machine is sent each skill's event. Empty, the box reads The player and the node follows whichever node is in the player group, which every ready-made player joins. Press the box and pick another character in the window that lists the scene's nodes, and press the X beside it to go back to the player.	empty
Mana Variable	The character's number variable that skills are paid from, picked from the list of that character's own numbers, whatever template the NodeWorthy panel shows. It starts as mana. When the character has no variable of that name, or this is left empty, every skill costs nothing. In the first case a warning sign shows beside the bar as soon as a skill has a Mana Cost , and the Output panel says so once when such a skill is used.	mana
Mana Back per Second	How much mana comes back by itself each second, up to the mana variable's Max . It starts at 3. At 0 none comes back, and none comes back while the variable has no Max .	3.0

SETTING	WHAT IT DOES	STARTS AS
Skills	The skills, from left to right, as rows in the Inspector, with an Add Skill button under them and a X on each row to take it away. Each skill has a Skill Name , a Description , an Icon , a Key , a Cooldown , a Mana Cost , the Event Name it sends, and Uses Item for a skill that needs an item. The Key is picked from a list of keys, mouse buttons and gamepad buttons, which the little arrow beside its box opens. The Event Name has a list too, of the events the templates already wait for, and a new one is typed.	empty
Slot Size	How wide and tall each slot is drawn, in pixels. It starts at 64 by 64. A skill's picture is fitted inside the slot and keeps its shape.	64, 64
Gap	The space between slots, in pixels. It starts at 8.	8.0

Make it your own

Make a skill do something

1. Click **SkillBar** and, in the skill's row, type HEAL in **Event Name**. The player needs a health for this example, from **NodeWorthy** → **Give It** → **Health**.
2. Click the player in the Scene dock and press **Open Its State Machine...** under its settings at the top of the Inspector. The NodeWorthy panel shows the player's template.
3. Press **+** in the panel's toolbar and choose **New State....** Type Healing and press **Create**.
4. Click the **Healing** box, press **Add Action...** and add **Set Variable**. In **Variable Name** pick health, set **Operation** to **Add** and leave **Value** at 1.
5. Right-click the **Playing** box, choose **Add Transition**, and drag the new row's port onto **Healing**. Click the row, press **When** on the **Transitions** tab, and pick **When SkillBar sends HEAL**. If that line is not in the list, choose **Name an event of your own...** and type HEAL.
6. Right-click **Healing**, choose **Add Transition**, and drag its port onto **Playing**. Click the row, press **When**, and pick **When everything in this state has finished**.
7. Run the scene, take a hit, and press **1**. The player's health goes up by 1, and the skill cannot be used again until its **Cooldown** has passed.

Make a skill cost mana

1. Open the player's template with **Open Its State Machine...** and go to the **Variables** tab.
2. Pick float and press **Add Variable**. Name the variable mana, give it a starting value of 100, then open it and set its **Max** to 100.
3. Click **SkillBar** and set the skill's **Mana Cost**, such as 25.
4. Run the scene. Each use takes 25 off the mana, the mana creeps back at **Mana Back Per Second**, and a skill the mana cannot pay for is dimmed blue.

Put a potion on a key

1. Click **SkillBar** and press **Add Skill**.
2. In the new row, press **Uses Item** and pick the potion. The slot now shows how many the character carries, and with none the skill cannot be used.
3. Give it an **Event Name**, such as DRINK.
4. The bar does not take the potion out of the pack. In the state that waits for the event, add a **Use Item** action with the same potion picked, which takes one out and does what the potion does.

Move it and size it

1. Click **SkillBar** and drag it in the 2D view to where you want it.
2. Set the size of the slots with **Slot Size** and the space between them with **Gap**.
3. To keep it against another edge on every screen size, open **Layout** in the Inspector and pick a place in **Anchors Preset**. It starts at **Center Bottom**.

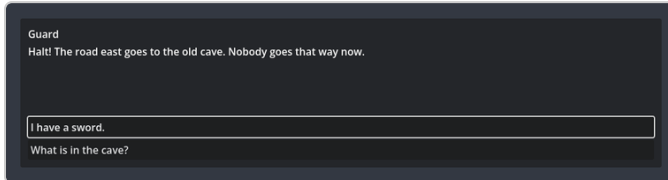
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The bar is one empty slot that says No skills yet	It has no skills. Click it and press Add Skill in the Inspector. In the game a bar with no skills draws nothing.
The key works and the slot sweeps, but nothing happens in the game	The bar only sends the skill's event to the character's machine. A state in the character's template has to wait for that event and do the work.
A skill with a Mana Cost is used for nothing	The character has no variable with the name in Mana Variable , which starts as mana. A yellow warning sign beside SkillBar says so, and the Output panel says it once in the game. Add a number variable called mana to its template, with a Max .
Mana never comes back	The mana variable has no Max , so the bar brings none back, and says so once in the Output panel. Give it a Max in the Variables tab, such as 100 or max_mana.
A slot's edge flashes red when its key is pressed	The skill is still coming ready, the mana cannot pay for it, or it uses an item and none is carried.

See also [Globe](#) 132, [Inventory](#) 185, [Inventory Grid](#) 141, [Stat Panel](#) 122

Dialogue Box NODE

The box a dialogue is shown in, along the bottom of the screen. It shows who is speaking, their picture, their words a letter at a time, and the player's answers as buttons under them. It is hidden until a **Start Dialogue** action plays a dialogue in it, and it hides again when the dialogue ends.



Dialogue Box in a running game.

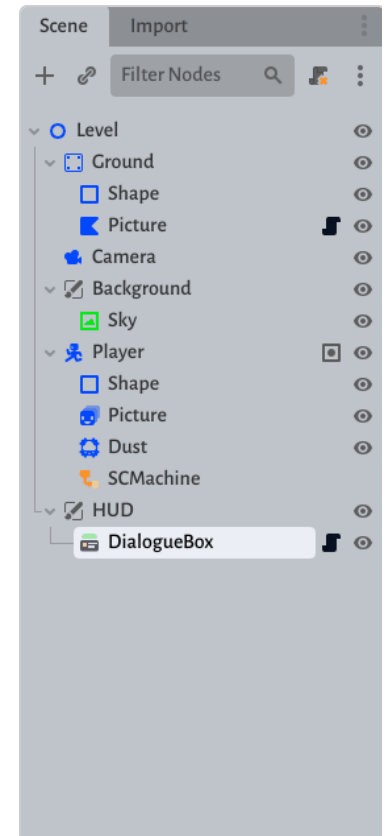
What it is good for

Any game where characters talk, from a villager's greeting to a shopkeeper or a quest giver with answers to choose. One box serves every dialogue in the scene. A ready-made **Villager** brings one with it, so add it by hand only for someone you build yourself.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it.
2. Adding a **Villager**, or a **Start Dialogue** action, to a scene with no Dialogue Box adds one for you. To add one by hand, go on to the next step.
3. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Dialogue Box**.
4. A node called **DialogueBox** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
5. The editor shows it in the 2D view, even in a 3D game, along the bottom of the screen, 40 pixels in from the sides and the bottom. It draws an example there, a guard's line with two answers under it, so that you can see its look. The example is never shown in the game.
6. If **Dialogue Box** is greyed out in the menu, the scene already has one, and a scene needs only one. Hover over the greyed line to read where it is, then click that node in the Scene dock.

Try it. Press **F6** and the box is hidden. Walk up to a **Villager** that has a dialogue picked in **Dialogue** and press **E**, and the box appears with the words typed out a letter at a time. Press **Enter** once to show the rest of the line at once, and again to move on. When a line has answers, move between them with the up and down arrow keys and press **Enter** to choose.



The Scene dock once it is added.

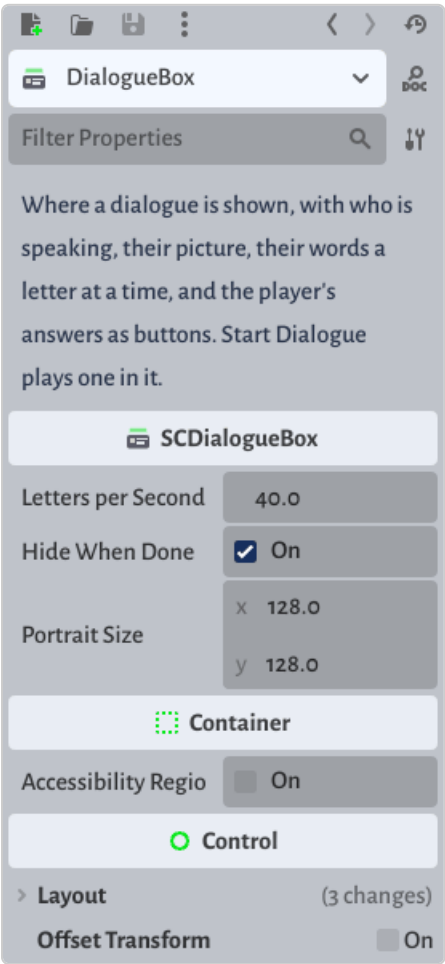
What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.
DialogueBox	Dialogue Box	The box itself. A PanelContainer is Godot's box with a background, and the Dialogue Box builds the speaker's name, the picture, the words and the answer buttons inside it when the game starts. It takes its panel, its font and its buttons from the game's look.

The controls

PRESS	WHAT IT DOES
Enter or the Space bar	Shows the rest of the line at once. Pressed again, it moves on to the next line, or chooses the answer the highlight is on.
A click on the box	Does the same as Enter . A click on an answer chooses that answer.
The up and down arrow keys, or Tab	Move between the answers, round from the last to the first and never off them.
The bottom button on a gamepad, A or Cross	Shows the rest of the line, moves on, or chooses the answer the highlight is on.
The D-pad or a stick on a gamepad	Moves between the answers.

Its settings



Click **DialogueBox** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Letters per Second	How fast the words are typed out, in letters each second, 40 to start with. At 0 every line appears whole. A speaker whose Actor file has a Letters Per Second of its own uses that speed for its lines.	40.0
Hide When Done	Ticked, the start, the box hides when the dialogue ends. Unticked, the last line stays on the screen after the dialogue is over.	on
Portrait Size	How big the speaker's picture is drawn, in pixels. It starts at 128 by 128. A line with no picture shows none, and its words take the full width.	128, 128

Make it your own

Move it or make it taller

1. Click **DialogueBox** in the Scene dock and press the **2D** button at the top of Godot if the 2D view is not showing.
2. Drag the handle on its top edge upwards to make it taller, or drag the whole box to another place.
3. It is fixed to the bottom edge and stretches from side to side, so it fits every screen size.

Give it your game's look

1. Open the **Theme** tab, the last tab on the right of the NodeWorthy panel.
2. Press one of the ready-made looks, such as **Parchment** or **Pixel**, and change any of its choices.
3. Press **Use For The Whole Game**. The Dialogue Box takes its panel, its font and its buttons from that look, and so does every other panel in the game.

Slow the typing down

1. Click **DialogueBox** and set **Letters Per Second** to 20 for a slow, steady voice, or to 0 to show each line whole.
2. Run the scene and talk to someone to see the difference.

Have a chest say something when it is opened

1. Click a **Chest** in the Scene dock and press **Open Its State Machine...** under its settings at the top of the Inspector. The NodeWorthy panel shows its template.
2. Click the **Open** box, then press **Add Action...** in the **Actions** tab on the right.
3. Type Start Dialogue in the search box, pick **Start Dialogue** and press **Add**.
4. Press **Dialogue** in the new action and choose **New dialogue...** Give it a name. The dialogue opens as a graph in the NodeWorthy panel, with one line to start from.
5. Click the first line and type what is said. Press **◀ Templates** to go back to the chest's template.
6. Run the scene and open the chest with **E**. Once it is open, the words are typed out in the Dialogue Box. Press **Enter** to move on, and the box closes.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
Nothing shows when the game runs	The box is hidden until a Start Dialogue action plays a dialogue in it, and it hides again afterwards. That is how it is meant to work.
The Output panel says there is no Dialogue Box in the scene	A Start Dialogue action ran in a scene without one. Right-click the top node and choose NodeWorthy → Add To Screen → Dialogue Box .
Keys stop working in an Inventory Grid or a menu while someone talks	While a dialogue goes on the box has the focus, so Enter and the arrow keys go to it. Whatever had the focus has it back when the dialogue ends.
A second dialogue cuts the first one off	One box shows one dialogue. A dialogue started while another is still going takes the box over, and the first one ends.
Dialogue Box is greyed out in the menu	A scene needs only one. Hover over the greyed line to read where the one in the scene is.

See also **Villager** 97, **Quest List** 146, **Inventory Grid** 141, **Chest** 104

Item Collection NODE

A grid of item pictures for a collection page. Items that have ever been found are drawn in full colour and the rest as dark shapes still to find, and resting the pointer on one shows its name and description. Until you list some items or name a folder of them it has nothing to draw, and says so in the editor.



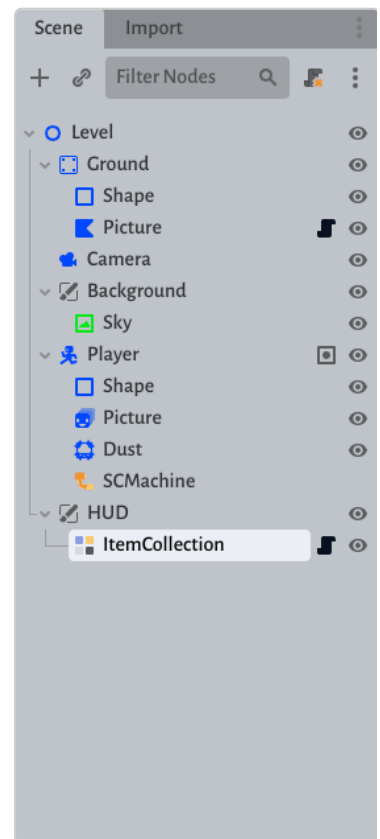
Item Collection in a running game.

What it is good for

A page in a menu that shows everything the player has discovered across all their runs, in a game built on finding items. With **Show** set to **Held This Run** it lists only what the player has been given this run, for a pause menu. It is for looking, not for choosing, so to let the player use what they carry pick the **Inventory Grid**.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it. You also need some items, each with an **Icon**, made with **New Item...** on the + menu of the NodeWorthy panel.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click the top node and choose **NodeWorthy** → **Add To Screen** → **Item Collection**.
3. A node called **ItemCollection** appears under a node called **HUD**. HUD is a CanvasLayer, which is Godot's node for things drawn on the screen over the game. If the scene already has a HUD, or another CanvasLayer of its own drawn over the game, the new node goes under that one.
4. It sits in the top left corner of the 2D view, clear of the edges, six items wide. With no items yet it draws one empty square and the words No items to show.
5. Click **ItemCollection**. In the Inspector, the panel on the right that shows the settings of whatever is clicked, add your items to **Items**, one to a row. For many items, leave **Items** empty and pick the folder they are kept in under **Folder**. The items are drawn in the 2D view as you add them, every one in full colour, so you can lay the page out before running the game.
6. Drag the handles of its box in the 2D view to the width of the page you want. It fits as many items across as the box is wide, and the box grows downwards by itself to hold every row.



The Scene dock once it is added.

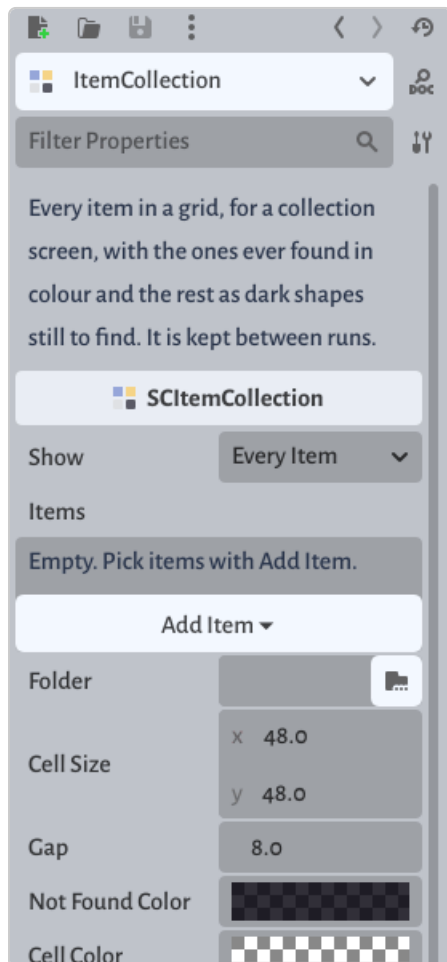
Try it. Press **F6**, or the **Run Current Scene** button at the top right of Godot. Every listed item that has not been found yet is drawn as a dark shape. Rest the pointer on one and it says Not found yet. Once an item goes into the

player's **Inventory**, from an **Item Pickup** or a **Chest** for instance, or a **Give Item** action hands it to the player, that item is drawn in colour, in this run and in every run after it.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HUD	CanvasLayer	Made only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn on the screen, which stay put while the camera moves. A level's Background and a ready-made menu are CanvasLayers too, and a screen node added from the level is not put under those.
ItemCollection	Item Collection	The collection itself. A Control is Godot's plain node for anything drawn on the screen, and the collection draws its squares and pictures on it, in the editor as well as in the game.

Its settings



Click **ItemCollection** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Show	Which items are drawn. Every Item draws all the items in Items and Folder , the found ones in colour and the rest dark. Held This Run leaves both lists aside and draws the items given to the player this run, in the order they were given, all in colour. Choices: Every Item, Held This Run.	Every Item
Items	The items to draw, in the order listed. An item listed twice is drawn once. For a game with many items leave this empty and name a Folder .	empty
Folder	A folder in your project. Every item file in it, and in the folders inside it, is drawn after the ones in Items , in the order of the file names. A new item saved there appears on the page without being added here.	empty
Cell Size	How wide and tall each item's square is, in pixels. It starts at 48 by 48. The picture is fitted inside and keeps its shape.	48, 48
Gap	The space between squares, in pixels, across and down. It starts at 8.	8.0
Not Found Color	The colour an item's picture is tinted with before the item has been found. It starts almost black, so that little more than the shape shows.	0.05, 0.04, 0.08, 0.85
Cell Color	The colour of the square behind each item, a faint white to start with.	1, 1, 1, 0.06

Make it your own

Fill it from a folder

1. Click **ItemCollection** in the Scene dock.
2. Press the folder button beside **Folder** and pick the folder your items are in, such as the items folder that **New Item...** saves into.
3. Run the scene. Every item in that folder and the folders inside it is on the page, in the order of the file names.

Mark an item as found when a chest is opened

1. Click a **Chest** in the Scene dock and press **Open Its State Machine...** under its settings at the top of the Inspector. The NodeWorthy panel shows its template.
2. Click the **Opening** box, then press **Add Action...** in the **Actions** tab on the right.
3. Type Give Item in the search box, pick **Give Item** and press **Add**.
4. Press **Item**, which reads **Pick an item...**, and pick the item. In **Machine** pick the player, so that the item is given to the player and not to the chest.
5. Run the scene and open the chest. The item is now drawn in colour on the page, and stays in colour the next time the game is run.

List what the player holds on the pause menu

1. Add a pause menu with **NodeWorthy** → **Add Menu** → **Pause Menu** if the scene has none.
2. Click **ItemCollection** and set **Show** to **Held This Run**. It then lists what **Give Item** actions have handed over this run, which is not the same as what an **Inventory** holds.
3. In the Scene dock, drag **ItemCollection** onto **Screen** under **PauseMenu**. **Screen** is hidden until the game is paused, and everything under it shows and hides with it.
4. Click **Screen**'s eye in the Scene dock to show it while you drag the collection to a clear spot in the 2D view, then click the eye again to hide it.

Size it

1. Click **ItemCollection** and drag the handles of its box in the 2D view. A wider box fits more items on a row, and the box is always tall enough for every row.
2. Set **Cell Size** for bigger or smaller squares, and **Gap** for the space between them.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The page says No items to show	Items and Folder are both empty, so there is nothing to draw. Add items to Items , or pick a Folder . With Show at Held This Run the editor has nothing to draw either, because nothing is given until the game runs.
Nothing shows in the game	Items and Folder are both empty, or Show is Held This Run and nothing has been given yet. An item with no Icon is drawn as its name once it is found, and as a dark square before that.
An item the player picked up is still dark	An item is found when it goes into an Inventory under the node in the player group, or when a Give Item action hands it over. Every ready-made player is in that group. A player of your own has to be put in it, in the Groups tab of Godot's Node dock, and an item put into a chest's or a shop's Inventory is not found until the player has it.
Items found while testing are still found the next day	The collection is kept on the computer between runs, and it only ever fills up. That is how the finished game behaves too, so expect it while you test.

See also [Inventory Grid](#) 141, [Inventory](#) 185, [Item Pickup](#) 91, [Chest](#) 104, [Pause Menu](#) 116

NODES A CHARACTER CARRIES

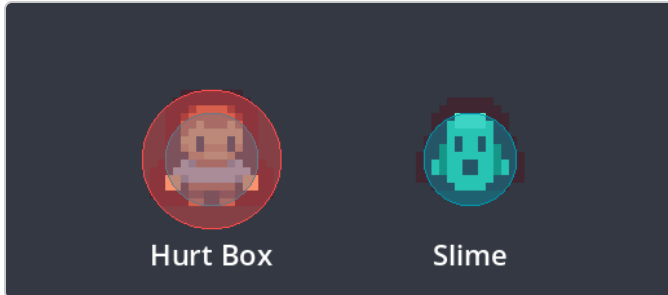
Nodes a character carries

These nodes have nothing to show. Each gives the character it sits under something it did not have, such as health, a bag of items, eyes and ears, or experience and levels. They are under **NodeWorthy** → **Give It** when you right-click a character.

Hurt Box	180
Inventory	185
Perception	191
Experience	195
Hit Box	200
Data Node	205
Performance Overlay	210

Hurt Box NODE

A **Hurt Box** makes the thing it sits under able to be hurt. It comes with **Give It** → **Health**, which also gives the character a health of 3 and a **Dead** state, and gives the player a small panel that shows the health. From then on a **Hit Box**, a bullet or a swung weapon that reaches it takes health off it.



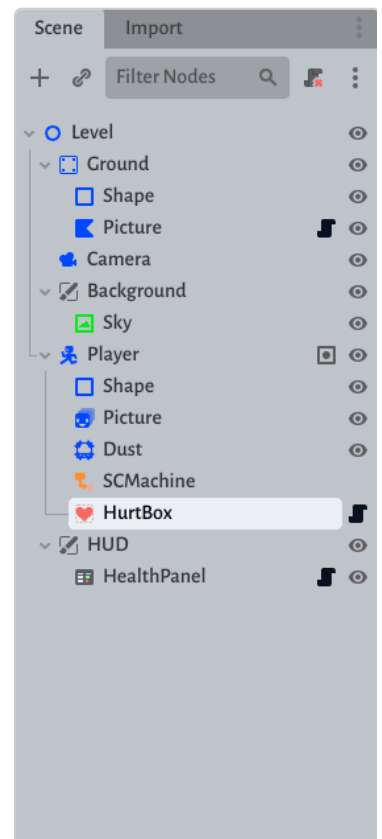
Hurt Box in a running game.

What it is good for

Anything that can be hurt or broken, such as the player, an enemy, a barrel or a boss. It pairs with the **Hit Box**, which is the part that does the hurting. The ready-made **Enemy** already has one, and a player is given health by itself the moment an Enemy is added to its level, so you add it yourself mostly to characters and things of your own.

Add it

1. Start with a level that has a player in it and no enemy yet, such as one made with **NodeWorthy** → **Start A 2D Level...** with **Enemies** set to 0 in its window, which puts a **Platform Hero (2D)** in it. The character must run a template, which every ready-made hero and thing does.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click **Player** and choose **NodeWorthy** → **Give It** → **Health**.
3. A node called **HurtBox** appears under the player, and a message at the bottom right of the editor says the player has health.
4. Look at the top of the Scene dock. A CanvasLayer called **HUD** has been added if the level had none drawn over the game. A CanvasLayer is Godot's node for things drawn over the game. Under it is a **HealthPanel** that shows the player's health.
5. Click **Player**. At the top of the Inspector, the panel on the right that shows the settings of whatever is clicked, a **Health** section now holds **Max Health**, which starts at 3, and the Hurt Box's **Invincibility Time**.
6. Press **Open Its State Machine...** under those settings. The template in the NodeWorthy panel has a new state called **Dead**, and the states at its top level lead to it on **DIED**.
7. If **Health** is greyed out in the menu, the node has no Machine yet. Choose **NodeWorthy** → **Add State Machine...** on it first.



The Scene dock once it is added.

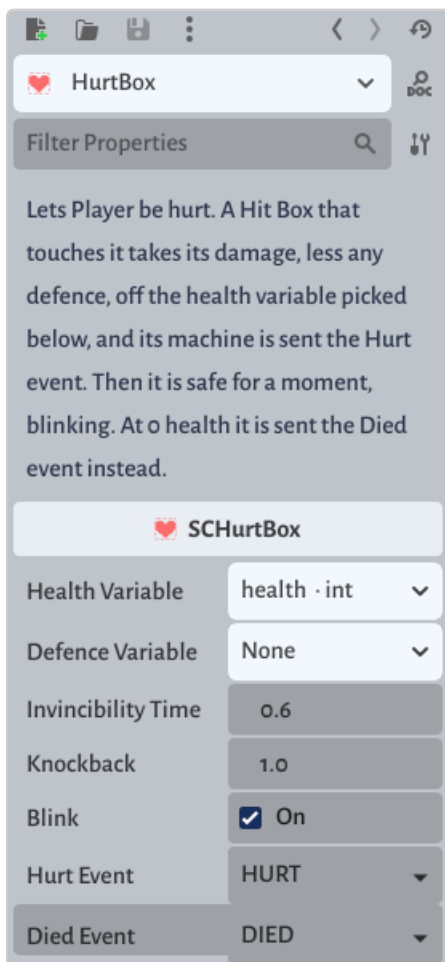
Try it. Press **F6**. The health shows at the top left of the screen as Health 3 / 3. Add an **Enemy** with **NodeWorthy** → **Add Thing** → **Enemy** and let it touch you. Each touch takes 1 health, knocks the player back, tints it red for an

instant in 2D, and makes it blink for a moment while it cannot be hurt again.
At 0 the player's **Dead** state waits a second and starts the level again.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HurtBox	Hurt Box	The part that can be hit. When the game starts it makes an Area2D, or an Area3D in a 3D game, in the shape of the character's own collision shape and a little bigger. An Area is Godot's node for a region that notices what overlaps it. That area is made while the game runs and never shows in the Scene dock.
HUD	CanvasLayer	Added to the top node for the player only, and only when the scene has no CanvasLayer drawn over the game yet. A CanvasLayer is Godot's node for things drawn over the game, such as health and a score.
HealthPanel	Stat Panel	A Stat Panel set to show health, added under the HUD for the player only. It is left out when a Stat Panel in the scene already shows health. When the scene has a Stat Panel that shows other numbers, health is added to that one and no new panel is made.

Its settings



Click **HurtBox** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Health Variable	The number variable of the character's template that a hit is taken from. It starts as health, which Give It → Health makes with a Min of 0 and a Max of max_health. Pick another from the list only when your template keeps health under another name. If the template has no such variable, a hit takes nothing and the Output panel, Godot's panel at the bottom that prints messages, says so.	health
Defence Variable	A number variable taken off every hit before it lands, as armour does. It starts empty, which takes nothing off. With a defence of 4 a hit of 7 takes 3. A hit always takes at least 1, however high the defence.	empty
Invincibility Time	How long it cannot be hurt again after a hit, in seconds, shown as Invincibility Time . It starts at 0.6, and Give It → Health sets 0.15 on anything that is not the player, so that a quick gun does not waste its bullets. Longer is kinder to the player. At 0 something standing on spikes is hurt many times a second.	0.6
Knockback	How hard a hit pushes it away from whatever hit it. 1, where it starts, is a good push, 0 is none and 2 is twice as hard. It is multiplied by the Knockback of the Hit Box that landed the hit.	1.0
Blink	Ticked, as it starts, the character goes faint and comes back many times a second while it is safe, so the player can see that the hit landed and that it cannot be hurt yet. Untick it for a thing that should not flicker, such as a barrel.	on
Hurt Event	The event its character's machine is sent when it is hurt and still has health left. It starts as HURT and carries how much was taken. A transition waiting for it can play a flinch or send an enemy for cover. Emptied, nothing is sent.	HURT
Died Event	The event its character's machine is sent, in place of the hurt event, when its health reaches 0. It starts as DIED , which is what the transitions to the Dead state wait for. Change it and those transitions must be changed to match.	DIED
XP Worth	The experience given to whoever beats it, shown as XP Worth . It starts at 0, which gives none. Whoever lands the last hit gets it, when that character has an Experience node under it or a number variable called xp in its template.	0.0
Store Attacker In	A Node variable of the character's template that is given whatever hit it, each time a hit lands. It starts empty, which keeps nothing. With it filled, a state can turn to face the attacker or go after it.	empty
Flash Color	The colour a 2D character is tinted for an instant when a hit lands, shown as Flash Color . It starts as a light red and fades back to the character's own colour in under a fifth of a second. Made fully see-through, there is no flash. It does nothing in 3D.	1, 0.35, 0.35, 1
Shows Damage	Ticked, each hit shows how much it took as a number that rises and fades over the character, in 2D and in 3D. It starts unticked.	off
Damage Color	The colour of those numbers, shown as Damage Color . It starts white. White over an enemy and red over the player reads well.	1, 1, 1, 1
Hurt Sound	A sound played where the character is each time it is hurt, such as a grunt or bones knocking. It starts empty, which plays none.	empty
Died Sound	A sound played in place of the hurt sound on the hit that takes its health to 0. It starts empty, and then the hurt sound plays for that hit as well.	empty

SETTING	WHAT IT DOES	STARTS AS
Shape	The shape of the area that can be hit. Same As Body , where it starts, copies the character's own collision shape a little bigger, so nothing needs setting up. Circle makes a circle, or a ball in 3D, of Radius moved by Offset , for a weak spot or for a thing with no collision shape of its own. Choices: Same As Body, Circle.	Same As Body
Radius	For Circle only, and hidden otherwise. How big the circle is, in pixels in 2D and metres in 3D. It starts at 16 pixels in a 2D game and at half a metre in a 3D one.	16.0
Offset	For Circle only, and hidden otherwise. How far the circle sits from the character's own position, as two numbers in 2D and three in 3D. It starts at 0, which centres the circle on that position.	0, 0, 0

How it works

Press **Open Its State Machine...** under its settings and the NodeWorthy panel shows its template, the drawing of states that makes it do what it does. Nothing is hidden, and every state is made of ordinary actions you can change.

STATE	WHAT HAPPENS IN IT
Dead	Added to the character's template by Give It → Health , together with a transition to it on DIED from every top-level state that had none. The player's Dead runs Wait for 1 second and then Change Scene set to reload the current scene with a half-second fade, so the level starts again. Anyone else's Dead runs Queue Free , which removes them from the game.

Make it your own

Make a character take more hits

1. Click the character in the Scene dock.
2. At the top of the Inspector open the **Health** section and raise **Max Health**, for example from 3 to 10. It starts each level with that much.
3. Two enemies that share one template can each have their own **Max Health**. Press **Use These For Every Copy** under the settings to give this one's numbers to all of them.

Make it safe for longer after a hit

1. Click the character. Under **Health** at the top of the Inspector, raise **Invincibility Time**, for example to 1.5 seconds for a forgiving game.
2. Or click the **HurtBox** under it in the Scene dock and change **Invincibility Time** there, which is the same number.
3. Run the scene and get hit. The character blinks for that long, and nothing hurts it until the blinking stops.

Give it armour

1. Click the **HurtBox** under the character.
2. Press the box beside **Defence Variable** and choose **New variable....** Keep the name defence and confirm.
3. Right-click the character, choose **NodeWorthy** → **Open Its State Machine...**, and on the **Variables** tab set the **Starting Value** of defence, for example to 1.
4. Run the scene. Every hit now takes 1 less, and never less than 1.

Change what happens when it dies

1. Click the character and press **Open Its State Machine...** under its settings, or right-click it and choose **NodeWorthy → Open Its State Machine...**
2. Click the **Dead** box in the NodeWorthy panel. Its actions are listed on the **Actions** tab on the right.
3. For the player, click the **Wait** card and change **Seconds** from 1 to 2 for a longer pause before the level starts again.
4. For an enemy or a barrel of your own, press **Add Action...** and add **Drop Loot**. Set its **Gold Least** and **Gold Most**, or list items in its **Items**, and drag its card above **Queue Free** so it leaves something behind before it goes.

Show each hit as a number

1. Click the character. Under **Health** at the top of the Inspector tick **Shows Damage**.
2. Click its **HurtBox** and set **Damage Color**, for example red over the player.
3. Run the scene. Each hit's number rises over the character and fades.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
Health is greyed out in the Give It menu.	Health needs a template to add its Dead state to, and this node has no Machine yet. Choose NodeWorthy → Add State Machine... on it first, then NodeWorthy → Give It → Health .
The Output panel says it was hit, but its machine has no variable called health.	The Hurt Box was added on its own, without Give It → Health , or Health Variable names a variable the template has not got. Add a number variable for the health on the template's Variables tab, or pick the right one in Health Variable .
The Output panel says it is set to the same shape as its body, but the body has no collision shape.	With Shape on Same As Body the Hurt Box copies the CollisionShape2D or CollisionShape3D directly under its character, which is Godot's node for a body's shape. Give the character one, or set Shape to Circle and give it a Radius .
Health reaches 0 and the character does not die.	DIED is sent only when a hit through the Hurt Box takes the last health. A Set Variable action that lowers health sends nothing. Use the Hurt action to take health in a template, which goes through the Hurt Box.
Hits stop landing for a moment after each hit.	That is the Invincibility Time . While it lasts every hit is ignored, whoever it comes from, and the character blinks. Lower it for an enemy that a fast weapon should hurt with every shot.
Godot's warning sign shows beside the Hurt Box in the Scene dock	Rest the pointer on the sign to read why. A Hurt Box works from where its parent is, so it goes under a 2D or 3D character, not a plain Node. With Shape left as the body's own, the character needs a collision shape, or set Shape to Circle . A character with no Machine has no health to take off, so right-click it and pick NodeWorthy → Give It → Health . Fix it and the sign goes.

See also **Hit Box** 200, **Stat Panel** 122, **Enemy** 74, **Experience** 195, **Icon Meter** 137

Inventory NODE

An **Inventory** is what something is carrying. It draws nothing on the screen. Added under a player, a chest or a shopkeeper, it holds stacks of items, and the item actions put things in and take them out.



Inventory in a running game.

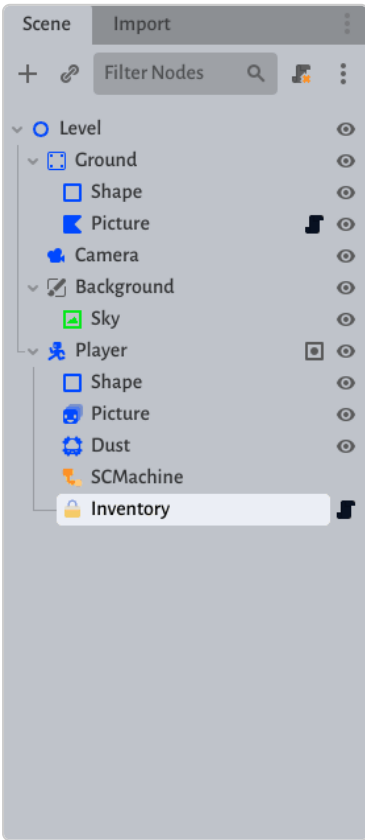
What it is good for

Any game where things are picked up, kept, spent or sold, such as keys for doors, potions and arrows. One goes under everything that holds things, so a player, a chest and a shop shelf each carry their own. To see what it holds while the game runs, add an **Inventory Grid**. For a chest that opens and spills its loot with nothing to build, pick the **Chest**, which comes with an Inventory of its own called **Loot**.

Add it

1. Start with a level that has a player in it, such as one made with **NodeWorthy** → **Start A 2D Level...**, which puts a **Platform Hero (2D)** in it.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click **Player** and choose **NodeWorthy** → **Give It** → **Inventory**.
3. A node called **Inventory** appears under the player and is selected. A message at the bottom right of the editor says where it was added.
4. Look at the Inspector, the panel on the right that shows the settings of whatever is clicked. Its top line says what the Inventory is for, and the line under it says that nothing shows this inventory on the screen yet.
5. Press **Show It On Screen**. An **Inventory Grid** is added to the screen, in the bottom left corner, and it draws this Inventory.
6. Click **Inventory** in the Scene dock again. Under **Starts With** press **Add Item** and pick an item. If the list says there are no items yet, make one first with **+** and then **New Item...** in the NodeWorthy panel.

Try it. Press **F6**, or the Run Current Scene button at the top right of Godot. The grid in the bottom left corner shows what you put in **Starts With**, with a count on any stack that holds more than one. With nothing in **Starts With** the grid is empty, so add an **Item Pickup** to the level, give it an item and walk onto it to watch the item arrive.

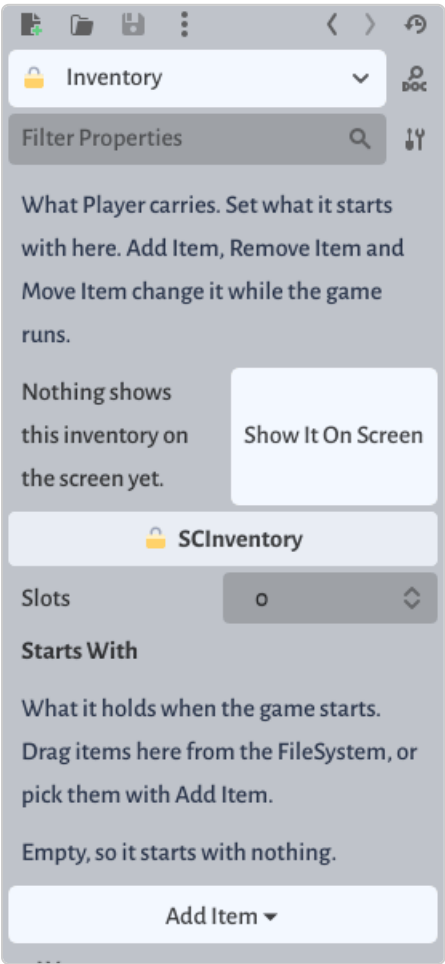


The Scene dock once it is added.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Inventory	Inventory	What its parent carries. It is a plain Node, which is Godot's simplest node, with no place in the world and nothing to draw. That is why it can sit under a player, a chest, or a shop that is nothing more than a Node.

Its settings



Click **Inventory** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Slots	How many stacks it can hold, where a stack is one kind of item piled together. It starts at 0, which means no limit, and that suits most games. Set it to 8 and a ninth kind of item is refused, while more of an item it already holds still fits until that stack reaches the item's Max Stack .	0
Starts With	What it holds the moment the game starts, a row for each item with how many of it. In the Inspector it is the Starts With list, filled with Add Item or by dragging item files in from the FileSystem dock, Godot's list of files at the bottom left. It starts empty. The rows are copied as the game starts, so playing never changes what the scene says it holds.	empty
Slot Names	The places things are worn in, such as Weapon, Armour and Trinket, found under Worn in the Inspector. Left empty, as it starts, the Inventory is a pack that holds stacks of anything. With names in it the Inventory holds one thing in each named slot and takes only items whose Slot matches one of them. What it wears from the start counts as worn, so a sword in Starts With adds its stat changes to the wearer as the scene begins.	empty

SETTING	WHAT IT DOES	STARTS AS
Fills From	A set of items to take from at random as it enters the scene, on top of anything in Starts With . It is found under Filled at random and starts empty, which means the Inventory holds only what you put in it. Rarer items are taken less often. With one item of each rarity in the set, a common one is taken about 60 times in a hundred, an uncommon one 25, a rare one 12 and a legendary one 3.	empty
Takes	How many times it takes an item from Fills From . It starts at 1. Set it to 3 for a chest that holds three things, and to 0 for one that takes nothing at random.	1
Fill Chance	How likely each of those takes is to put an item in, as a percentage from 0 to 100. At 100, where it starts, every take gives an item. At 35, which the ready-made Enemy uses for its loot, an item is there about one time in three.	100.0
Not Seen This Run	Left unticked, as it starts, the Inventory takes from Fills From every time the level starts, so a chest that lists a potion always has the chance of one. Ticked, an item that has already turned up this run is left out, so two chests do not hold the same thing, and once every item in the set has turned up a later Inventory takes nothing at all. Tick it for a game played in runs, and begin each run with the New Run action.	off

Make it your own

Start the player with three potions

1. Make the item if you have none. In the NodeWorthy panel press **+**, choose **New Item...**, call it Potion and press **Create**.
2. Click **Inventory** under **Player** in the Scene dock.
3. In the Inspector, under **Starts With**, press **Add Item** and pick Potion. A row appears with the potion and a count of 1.
4. Change the count on that row to 3.
5. Run the scene. The potions are in the grid from the first moment. An item's **Max Stack** says how many share one slot, and it starts at 1, so three potions take three slots until you raise it.

Limit how much can be carried

1. Click the **Inventory** in the Scene dock.
2. Set **Slots** to the number of stacks it may hold, such as 6.
3. Run the scene and pick things up. Once six stacks are in use a new kind of item no longer fits, while more of an item already held still goes onto its stack.
4. Whatever hands an item over decides what becomes of one that did not fit. An **Add Item** action sends its **No Room Event** then, so give that event a name and lead it to a state that shows a message and keeps the item.
5. The ready-made **Item Pickup** does this already. When its item does not fit it stays on the floor and says so in the Output panel, and walking onto it again once there is room picks it up.

Let the player wear a sword

1. Right-click **Player** and choose **NodeWorthy** → **Give It** → **Inventory** a second time. The player is given an Inventory to wear things in by itself the first time something is worn, but one you add yourself can be shown on the screen.
2. Click the new Inventory. In the Inspector, under **Worn**, press **Add Slot** under **Slot Names** and pick Weapon in its list, or type a slot of your own.
3. Press **Show It On Screen** at the top of its Inspector, so a second grid draws the slot, just above the first.
4. Make an item whose **What It Is** is set to **Worn** and whose **Slot** is Weapon.
5. Run the scene, pick the sword up and choose it in the first grid. It leaves the pack and appears in the Weapon slot. Choosing it there takes it off again.

Put an item in from a template

1. Right-click the thing that should hand the item over, such as a lever or a villager that runs a template, and choose **NodeWorthy** → **Open Its State Machine....** Its template opens in the NodeWorthy panel.
2. Click the box of the state in which the item should arrive. On the **Actions** tab on the right press **Add Action....**
3. Type add item in the search box, click **Add Item** and press **Add**.
4. In the action's settings pick the item in **Item** and set **How Many**.
5. Press **Bag** and pick the player's **Inventory** in the scene. Left empty, **Bag** means the Inventory under the node this template runs on.

Fill a chest of your own at random

1. Give the chest an Inventory with **NodeWorthy** → **Give It** → **Inventory** and click it.
2. Under **Filled at random**, press **Add Item** under **Fills From** and pick each item it may hold, one at a time.
3. Set **Takes** to how many it should hold, and **Fill Chance** to how likely each one is.
4. Tick **Not Seen This Run** only if an item should turn up in one chest and no other. Then a **New Run** action has to run at the start of each run, or the chests come up empty.
5. Run the scene. The chest rolls what it holds once, as the level starts.

When it goes wrong

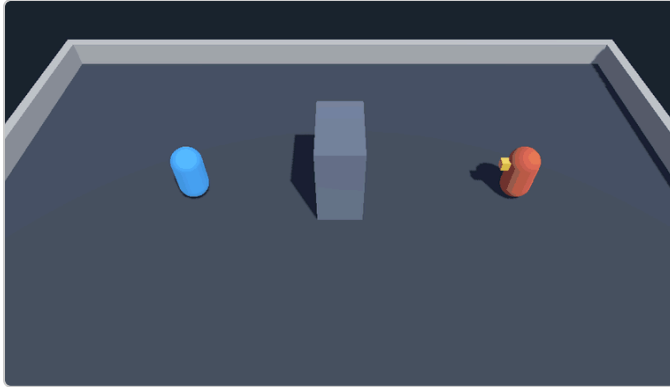
WHAT YOU SEE	WHY, AND WHAT TO DO
The Output panel, Godot's panel at the bottom that prints messages, says there is no Inventory to put anything in.	An item action such as Add Item ran on a node with no Inventory under it and nothing picked in its Bag setting. Right-click that node and choose NodeWorthy → Give It → Inventory , or press Bag on the action and pick the Inventory you mean.
The game runs and nothing shows what is carried.	An Inventory draws nothing. Click it and press Show It On Screen at the top of the Inspector, which adds an Inventory Grid that draws it.
A chest or an enemy that fills itself at random comes up empty.	Not Seen This Run is ticked and every item in Fills From has already turned up. Starting the scene again does not start a new run. Untick Not Seen This Run , which is how it starts, or add the New Run action to the state that begins a fresh run.
An item will not go into an Inventory that has Slot Names filled in.	An Inventory that wears things takes one item for each slot, and only an item whose Slot matches one of its Slot Names . It also refuses an item while that slot is taken. Keep a second Inventory with no Slot Names for what is carried.

WHAT YOU SEE	WHY, AND WHAT TO DO
Right-clicking the Inventory offers only one greyed line.	An Inventory is a part of the thing it sits under, so its menu only says where to click. Right-click its parent for the NodeWorthy menu, and click the Inventory itself to change its settings in the Inspector.

See also [Inventory Grid](#) 141, [Item Pickup](#) 91, [Chest](#) 104, [Enemy](#) 74, [Item Collection](#) 175

Perception NODE

Perception is what a character can notice. Added under a guard or an animal, it looks for what you pick five times a second, works out whether that is in range, inside its view and not behind a wall, and remembers where it was last seen. It does nothing you can see by itself, because the AI actions and checks are what act on what it finds.



Perception in a running game.

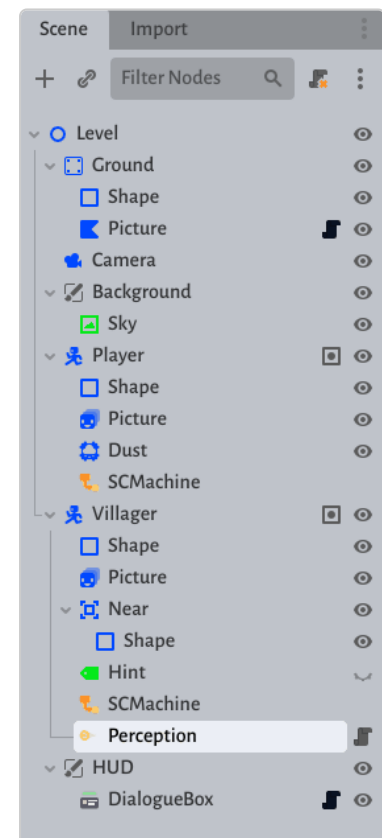
What it is good for

Any character that should react to another, such as a guard that chases the player, a deer that runs away or a shopkeeper who turns to greet you. One node holds the range, the view and the memory, so every AI action on that character agrees about who it is dealing with. The ready-made **Enemy** already has one, so pick the Enemy for a guard that works at once, and add Perception yourself to a character of your own.

Add it

1. Start with a level that has a player and a character that should notice it, such as a **Villager** added with **NodeWorthy** → **Add Thing** → **Villager**. The ready-made **Enemy** comes with a Perception, so it needs none added, and **Perception** is greyed out in its **Give It** menu, because a character's second one is never read.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click the character and choose **NodeWorthy** → **Give It** → **Perception**.
3. A node called **Perception** appears under it and is selected. In a 2D level its **Sight Range** is set to 400 pixels for you, and in a 3D level it keeps 20 metres.
4. In the Inspector, the panel on the right that shows the settings of whatever is clicked, find **Watch Group**. It starts on the player and the box shows the player's name. Press it to pick something else from the scene.
5. Set **FOV** to how wide the character should see, and leave the rest as it is for now.
6. Perception only notices. To make the character act on it, open its template and add a transition that waits for **When it sees the player**, as the first change below shows.

Try it. Perception has nothing to look at, so watch it through a character that uses it. Add an **Enemy** with **NodeWorthy** → **Add Thing** → **Enemy**, press **F6** and walk the player towards it. The Enemy comes after you once you are inside its **Sight Range**, and when you stay out of its sight it goes to where it



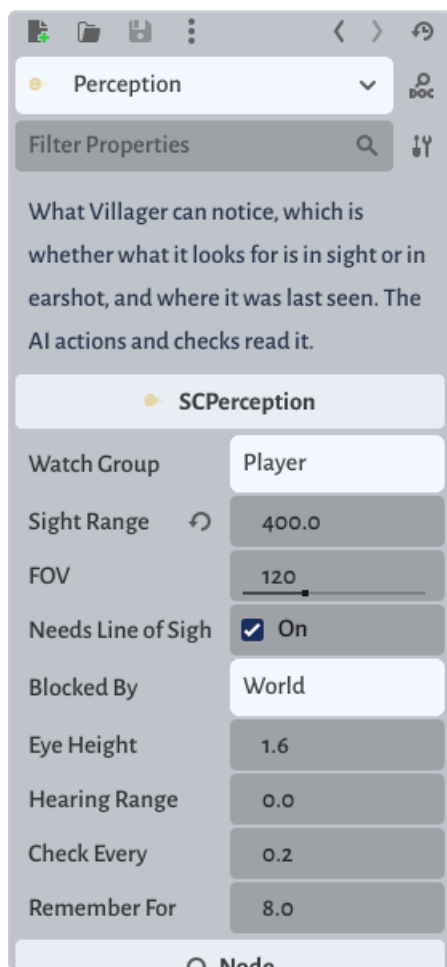
The Scene dock once it is added.

last saw you, looks round and gives up. While the game runs, pick the Enemy's machine under **Debug** in the NodeWorthy panel to watch its state go from **Standing Guard** to **Noticed** and **Chasing**.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Perception	Perception	The character's eyes, ears and memory. It is a plain Node, which is Godot's simplest node, so it draws nothing. It looks out from where its parent is and the way its parent faces, which is why it must sit directly under a 2D or 3D character.

Its settings



Click **Perception** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Watch Group	What it looks for, shown as Watch Group in the Inspector. It starts on the player. Press the box to pick something else from the scene, which can be one node by its name, every copy of a scene, or everything under one node. A Look For Target action sets it to its own Look For while its state runs.	player

SETTING	WHAT IT DOES	STARTS AS
Sight Range	How far it can see, in pixels in 2D and metres in 3D. Added from the right-click menu it starts at 400 in a 2D level and 20 in a 3D one. Raise it for a lookout on a tower, and lower it for a sleepy guard the player can creep close to.	20.0
FOV	How wide its view is, in degrees, centred on the way its character faces. The Inspector shows it as FOV . It starts at 120, roughly what a person notices, which leaves a blind spot behind the character. 360 sees all round, and 60 is a narrow cone the player can slip past.	120.0
Needs Line of Sight	Ticked, as it starts, anything behind a wall, a floor, a crate or another solid body stays unseen. Another character standing in the way does not hide anything. Untick it for a game with no walls to hide behind, and the check is skipped.	on
Blocked By	Which physics layers block its sight, ticked by name. It starts on World, which is walls, ground and anything you add yourself. The ready-made player and enemies stand on layers of their own and never block it. A wall you have put on another layer blocks nothing until that layer is ticked here.	1
Eye Height	How far above the character's position its eyes are, in metres. It is used in 3D only and starts at 1.6, about the height of a person's eyes, so that sight is not cast along the floor. Lower it for a dog or anything that crawls. In 2D the character looks from the middle of its shape, or from near the top of it in a game seen from the side.	1.6
Hearing Range	How close something must be to be noticed whichever way the character faces and whatever is in the way, as if it were heard. It is measured like Sight Range and starts at 0, which switches hearing off. It is also how far away the character hears a Make Noise action, so at 0 a guard is deaf to a thrown bottle.	0.0
Check Every	The seconds between one look and the next. It starts at 0.2, a fifth of a second, which is quick enough for anything that walks. A smaller number reacts a little sooner and costs more work. A larger one suits a crowd of distant characters.	0.2
Remember For	How many seconds it keeps its target in mind after losing sight of it, before it gives up and forgets. It starts at 8. At 0 it forgets the moment sight is broken. A long time, such as the 60 the ready-made Enemy uses, leaves the giving up to the template.	8.0

Make it your own

Make a character of your own react when it sees the player

1. Right-click the character in the Scene dock and choose **NodeWorthy** → **Open Its State Machine....** Its template opens in the NodeWorthy panel.
2. Right-click the box of the state in which it should keep watch and choose **Add Transition**.
3. On the **Transitions** tab press **When** and pick **When it sees the player**. A **Look For Target** action is added to the state, and the character is given a Perception if it had none.
4. Pick the state the transition goes to, such as one that holds a **Chase** action.
5. Run the scene and walk the player into the character's view.

Make a guard see further

1. Click the guard in the Scene dock. For a ready-made **Enemy**, open the **Seeing** section at the top of the Inspector and raise **Sight Range**.
2. For a character of your own, click the **Perception** under it and raise **Sight Range** there.
3. Run the scene and walk towards the guard from far off. It reacts from further away than before.

Give a guard a narrow cone and a blind spot

1. Click the guard. Under **Seeing**, or on its **Perception**, set **FOV** to 90. The ready-made **Enemy** starts at 360, which sees all round.
2. A guard sees the way it faces. The ready-made Enemy turns as it walks, so its blind spot is always behind it.
3. Run the scene and walk up behind the guard. It does not notice you until it turns round.

Let a guard hear the player close behind it

1. Click the guard. Under **Seeing**, or on its **Perception**, set **Hearing Range** to a short distance, such as 80 pixels in 2D or 2 metres in 3D.
2. Run the scene and creep up behind the guard. Inside that distance it notices you whichever way it faces, even through a wall.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The guard never sees the player, although the player is close.	Sight is a cone centred on the way the character faces. Until something turns it, a 2D character faces right and a 3D one faces away from the screen, so a character that never turns looks one way for ever. Set FOV to 360, or tick Turn To Face on its Patrol , Wander or Chase action so that it faces the way it goes.
The guard sees the player through a wall.	Only the layers ticked in Blocked By block sight, and it starts on World alone. Right-click the wall and choose NodeWorthy → Collision Layer → World , or tick the wall's layer in Blocked By . Check as well that Needs Line Of Sight is ticked.
A 2D character sees almost nothing.	A Perception added from Godot's own Add Child Node window keeps a Sight Range of 20, which is 20 metres in 3D and only 20 pixels in 2D. Set it to a few hundred. One added with NodeWorthy → Give It → Perception in a 2D level is set to 400 for you.
A Can See or Target Lost check always answers no, and the Output panel at the bottom of Godot names the character.	The character the check asks about has no Perception under it. Right-click that character and choose NodeWorthy → Give It → Perception .
A guard does not come when a Make Noise action goes off near it.	Its Hearing Range is 0, which is deaf, or the noise is further away than the Hearing Range . Raise Hearing Range on the guard's Perception until the noise is inside it, and check that the noise is inside the action's own Max Distance as well.
Godot's warning sign shows beside the Perception node in the Scene dock	Rest the pointer on the sign to read why. Perception looks from its parent and the way that parent faces, so it goes under a 2D or 3D character, not a plain Node. In a level, something has to be in the group it watches for, the player to start with. Add the player, or pick what it watches for under Watch Group . Fix it and the sign goes.

See also **Enemy** 74, **Villager** 97, **Hit Box** 200, **Hurt Box** 180

Experience NODE

Experience gives the character it sits under levels. It watches that character's xp variable, and once the xp reaches what the next level needs the level goes up, the character's stats grow, its health is filled, and every machine in the scene is sent **LEVEL_UP**.



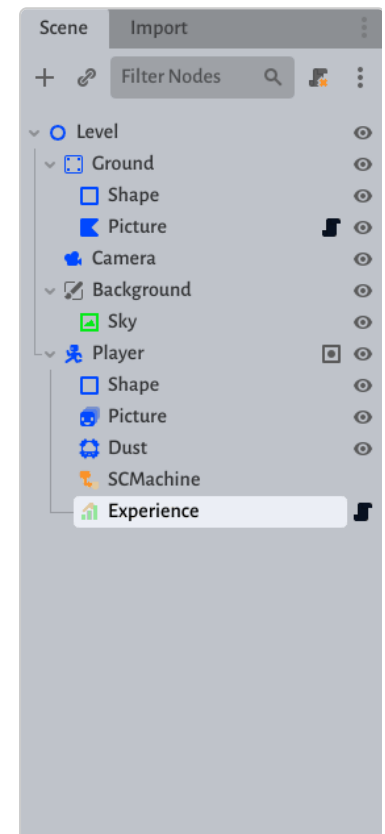
Experience in a running game.

What it is good for

Games where the player grows stronger by beating enemies and finishing quests, such as an action role-playing game or a dungeon crawl. It goes under the player as a rule, though any character that runs a template can have one. It shows nothing itself, so pair it with a **Stat Panel** to see the level and the XP, and with a **Hurt Box** on each enemy, whose **XP Worth** is where most XP comes from.

Add it

1. Start with a level that has a player in it, such as a **Platform Hero (2D)** added with **NodeWorthy** → **Add Player**.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click **Player** and choose **NodeWorthy** → **Give It** → **Experience**. A node called **Experience** appears under the player and is selected.
3. The same step gives the player's template the two number variables Experience counts in, xp, which starts at 0, and level, which starts at 1. A message at the bottom right of the editor says so. A template that already keeps a variable of either name keeps its own, and one Undo takes the node and the new variables away together. Give the player health first, with **NodeWorthy** → **Give It** → **Health**, and each level also adds 5 to its max_health.
4. In the Inspector, the panel on the right that shows the settings of whatever is clicked, **XP Variable** and **Level Variable** name those two. To count in other variables, press the box beside either and pick one of the character's number variables, or choose **New variable...** to make one.
5. Right-click the top node of the scene and choose **NodeWorthy** → **Add To Screen** → **Stat Panel**. In its **Stats** tick xp and level, so both show on the screen.



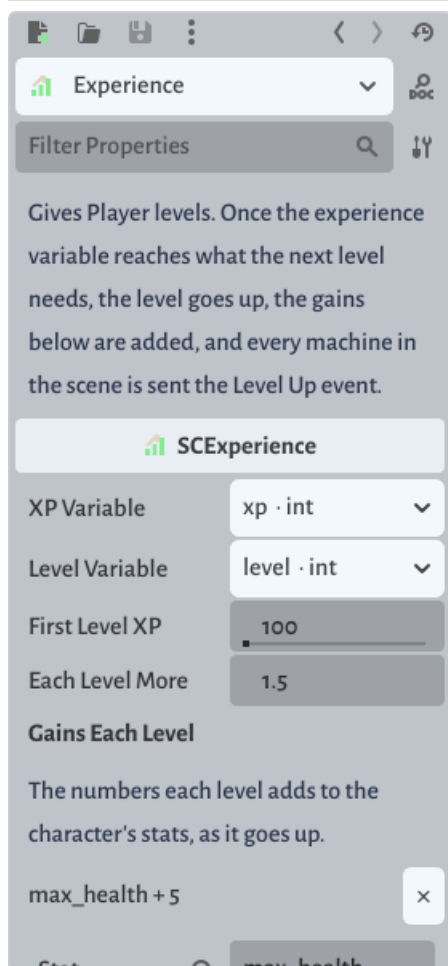
The Scene dock once it is added.

Try it. Press **F6**. The Stat Panel shows the level, and the XP against what the next level needs, as in XP 0 / 100. Give an enemy an **XP Worth** of 50 and beat two of them, for instance with a **Gun** from the **Give It** menu. The level goes to 2, the XP starts again from what was left over, and the panel now counts towards 150.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
Experience	Experience	The node that counts XP into levels. It is a plain Node, which is Godot's simplest node, and it draws nothing. It works on the variables of the Machine beside it, so it sits directly under the character whose template holds xp and level.

Its settings



Click **Experience** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
XP Variable	The number variable of the character's template that experience is added to, shown as XP Variable . It starts as xp. Give It → Experience makes it when the template has none of that name. Pick another from the list of the character's own number variables, or make one there with New variable Anything that adds to that variable counts, whether a beaten enemy, a quest's reward or a Set Variable action.	xp
Level Variable	The number variable that holds the level. It starts as level, and a level below 1 is read as 1. Give It → Experience makes it with a Starting Value of 1. Give a variable you make yourself a Starting Value of 1 too, so that a Stat Panel shows the right level from the first moment.	level
First Level XP	The XP it takes to go from level 1 to level 2, shown as First Level XP . It starts at 100, which with enemies worth 35 each means three of them for the first level. Lower it for quick early levels.	100.0
Each Level More	How many times more XP each level needs than the one before. It starts at 1.5, so the levels need 100, then 150, then 225. At 1 every level needs the same. At 2 each needs twice the last, which makes high levels slow to reach. A number below 1 counts as 1.	1.5
Gains Each Level	What each level adds to the character's stats, shown as the rows under Gains Each Level . Given from the Give It menu it starts with 5 more max_health and 1 more strength, each only where the character's template keeps that number, so a hero with no health yet starts with no rows. Take every row away with its X and a level adds nothing, and the list stays empty when the scene is opened again. Add Gain adds a row, where you pick one of the character's number variables and whether the amount is added to it or multiplies it. A row is only useful when the template has a variable of that name, and a gain to a number the character has not got is said once in the Output panel when the level comes.	empty
Heal on Level Up	Ticked, as it starts, every level fills the health variable up to its Max , as a reward. It works when that Max is another variable, as the health from Give It → Health has max_health. Untick it for a harder game.	on
Health Variable	The variable that Heal on Level Up fills. It starts as health, which is the name Give It → Health uses. Change it only if your template keeps health under another name.	health
Level Up Event	The event every machine in the scene is sent each time a level is gained, carrying the new level. It starts as LEVEL_UP . A state anywhere in the scene can wait for it to show a message, play a sound or flash the screen. Emptied, no event is sent, though the level still goes up.	LEVEL_UP
Top Level	The highest level there is. It starts at 0, which means there is no top. Set it to 20 and the character stops levelling at 20, while XP still gathers in its variable.	0

Make it your own

Give experience for beating an enemy

1. Give the player an Experience node with its xp and level variables, as the steps for adding it show.
2. Click the **Enemy** in the Scene dock. At the top of the Inspector, under **Worth**, set **XP Worth** to 50. On an enemy of your own, click its **HurtBox** and set **XP Worth** there.
3. Give the player a way to beat it, such as **NodeWorthy** → **Give It** → **Gun** on the player, which fires with the F key.
4. Run the scene and beat two enemies. The second takes the XP to 100, the level goes to 2 and the player's health is filled.

Make levels come faster or slower

1. Click **Experience** under the player in the Scene dock.
2. Set **First Level XP** to what the first level should cost, such as 50.
3. Set **Each Level More** to how much harder each level is than the last. 1.2 keeps levels coming, and 2 makes each one twice as far off.
4. Set **Top Level** if the game should stop at a highest level.

Change what a level gives

1. Click **Experience** and find **Gains Each Level** in the Inspector.
2. Click a row to change its amount, or press **Add Gain** for a new one.
3. On the new row pick the stat to grow from the list of the character's number variables, such as speed, and set how much is added.
4. If the stat you want is not in the list, add it first as a number variable on the **Variables** tab of the character's template.

Show a message when the player levels up

1. Right-click the **HUD** node in the Scene dock, choose Godot's **Add Child Node**, type Label and press **Create**. Type Level up into its **Text** in the Inspector.
2. Right-click the new Label and choose **NodeWorthy** → **Add State Machine...**, and give the template a name such as Level Message.
3. In the NodeWorthy panel press **+** and choose **New State...** twice, calling the states Hidden and Showing.
4. Click Hidden, press **Add Action...** and add **Set Visible** with **What** on **Hide**. Click Showing and add **Set Visible** with **What** on **Show**, then add **Wait** with **Seconds** at 1.5.
5. Right-click Hidden and choose **Add Transition**. On the **Transitions** tab press **When**, pick **When Experience sends LEVEL_UP**, and send it to Showing.
6. Right-click Showing, choose **Add Transition**, press **When** and pick **When everything in this state has finished**, and send it to Hidden.
7. Run the scene and gain a level. The words show for a second and a half.

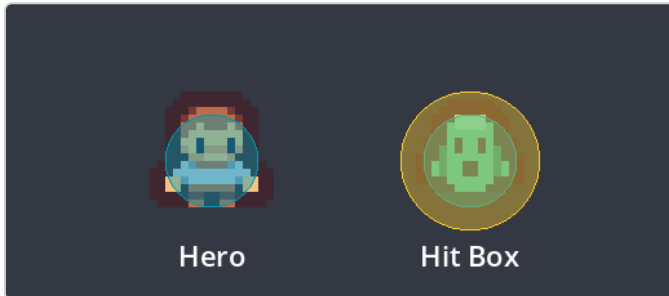
When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
The level never goes up.	Experience watches the variable named in XP Variable , and does nothing while the character's template has no such variable or the character has no Machine. A yellow warning sign beside Experience in the Scene dock says which, and the Output panel says it once in the game. This is how an Experience node added from Godot's own windows starts, and what renaming the variable leaves behind. Press XP Variable and choose New variable... , or add a number variable called xp on the Variables tab.
Enemies give no XP.	Each enemy's XP Worth starts at 0, which gives none. Click the enemy and set XP Worth under Worth . The XP goes to whoever landed the last hit, and only when that character has an Experience node or an xp variable.
A level is gained but health is not filled.	Heal on Level Up fills health up to its Max , and only when that Max is another variable, as the max_health from Give It → Health is. Give the health variable a Max that names a variable.
The character stops gaining levels while XP keeps growing.	Top Level is above 0 and the character has reached it. Set Top Level to 0 for no top.
A level is gained but no stat grows.	Gains Each Level has no rows. Experience comes with gains only for max_health and strength, and only where the character kept those numbers when it was given. Click Experience , press Add Gain and pick the stat to grow.

See also **Hurt Box** 180, **Stat Panel** 122, **Enemy** 74, **Villager** 97

Hit Box NODE

A **Hit Box** is the part that does the hurting. Added under a character or a trap, it takes health off whatever it touches that has a **Hurt Box**, knocks it back, and never hurts its own character.



Hit Box in a running game.

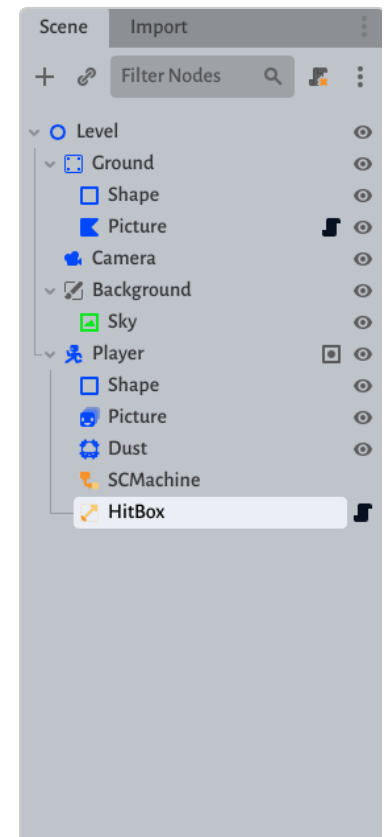
What it is good for

Anything that hurts to touch, such as a slime's body, spikes, a saw blade or a bullet scene of your own. For a sword or a punch the **Attack** action does the swing for you, so a Hit Box is needed there only when you build the swing yourself. The ready-made **Enemy** already has one that hurts the player, so add a Hit Box to characters and traps of your own.

Add it

1. Start with a level that has a player and an enemy in it, such as one with **NodeWorthy** → **Add Player** → **Platform Hero (2D)** and **NodeWorthy** → **Add Thing** → **Enemy**. These steps make the player hurt the enemy by walking into it, which is the quickest way to see a Hit Box work.
2. In the Scene dock, the list of nodes at the top left of Godot, right-click **Player** and choose **NodeWorthy** → **Give It** → **Hit Box**.
3. A node called **HitBox** appears under the player and is selected.
4. In the Inspector, the panel on the right that shows the settings of whatever is clicked, press the box beside **Hurts Group**. It reads Anything with a Hurt Box. Pick the enemy from the list under In this scene.
5. Leave **Damage** at 1 and **Shape** on **Same As Body**, so the Hit Box takes the player's own collision shape.
6. If a yellow warning sign shows beside **HitBox** in the Scene dock, what it is set to hurt has no Hurt Box. Point at the sign to read what to do.

Try it. Press **F6** and walk into the enemy. Each touch takes 1 of its 3 health and knocks it back, and after three it fades away. Its own touch hurts you as well, so expect to lose some health. To see the areas themselves, tick **Visible Collision Shapes** in Godot's **Debug** menu at the top of the editor and run again. Hit Boxes are drawn gold and Hurt Boxes red.

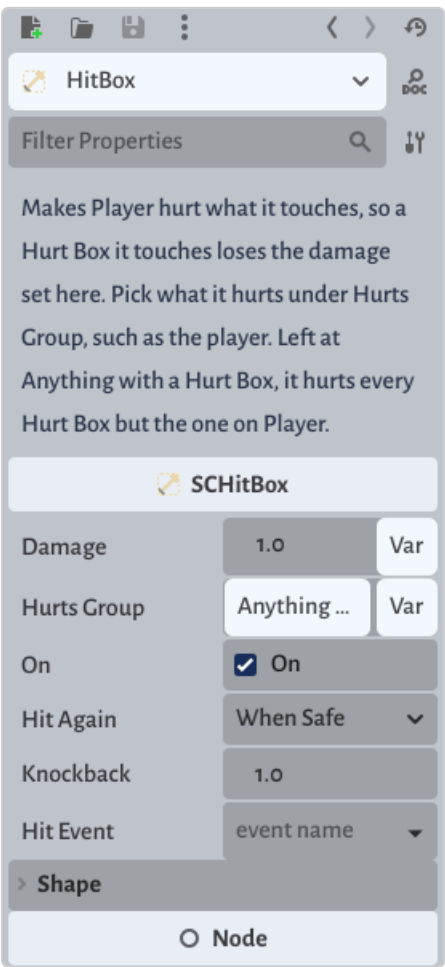


The Scene dock once it is added.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
HitBox	Hit Box	The part that hurts. When the game starts it makes an Area2D, or an Area3D in a 3D game, which is Godot's node for a region that notices what overlaps it. That area sits on a physics layer of its own, the last one, so it meets Hurt Boxes and nothing else. It is made while the game runs and never shows in the Scene dock.

Its settings



Click **HitBox** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Damage	How much health one hit takes, before the target's defence is taken off. It starts at 1, which against the 3 health from Give It → Health is three hits. A hit always takes at least 1. A Damage of 0 or less is no hit at all, with no push and no flash.	1.0
Damage Variable	A number variable of the character's template to read the damage from, in place of the number in Damage . In the Inspector it is the Var button beside Damage . Press it and pick a variable such as strength, and the damage grows as that variable does. Left empty, as it starts, Damage is used.	empty

SETTING	WHAT IT DOES	STARTS AS
Hurts Group	What it hurts, shown as Hurts Group and picked from the scene. It starts on Anything with a Hurt Box, which hurts everything that can be hurt except its own character. Pick the player for an enemy or a trap. Pick the enemies for something the player carries, so that enemies do not hurt one another.	empty
Hurts Group Variable	A variable of the character's template to read what it hurts from, in place of the pick in Hurts Group . In the Inspector it is the Var button beside Hurts Group . It is how the ready-made Enemy lets its Attacks setting decide who its touch hurts. Left empty, as it starts, Hurts Group is used.	empty
On	Whether it is hurting now. It starts ticked, which suits a body or a trap that always hurts. For a swing, keep it unticked and have a Set Property action switch it on as the swing starts and off as it ends. Each time it is switched on, everything can be hit afresh.	on
Hit Again	How it hits the same thing again. When Safe , where it starts, hits again as soon as the one it hurt is no longer safe, which suits a slime's body or spikes. Next Swing hits each thing once and then not again until On has been switched off and on, which suits a sword. Choices: When Safe, Next Swing.	When Safe
Knockback	How hard it pushes back what it hits. 1, where it starts, is a good push, 0 is none and 2 is twice as hard. It is multiplied by the Knockback of the Hurt Box it hits, so a heavy boss whose Hurt Box has a Knockback of 0 never moves.	1.0
Hit Event	An event sent to its own character's machine each time it lands a hit, carrying what it hit. It starts empty, which sends nothing. A bullet's machine can wait for it and run Queue Free , so the bullet is gone once it has hit something.	empty
Shape	Same As Body , where it starts, copies its character's own collision shape a little bigger, so a slime hurts with its whole body. Circle makes a circle, or a ball in 3D, of Radius moved by Offset , for a reach in front of the character or a blast round it. Choices: Same As Body, Circle.	Same As Body
Radius	For Circle only, and hidden otherwise. The size of the circle, in pixels in 2D and metres in 3D. Added with Give It → Hit Box it starts at 16 pixels in a 2D level and half a metre in a 3D one. The ready-made Enemy uses 30 pixels, or 1 metre in 3D, so its touch reaches a little past its body.	16.0
Offset	For Circle only, and hidden otherwise. How far the circle sits from the character's own position, as two numbers in 2D and three in 3D. It starts at 0. Move it to put the circle where a blade reaches, or over the middle of a tall character.	0, 0, 0

Make it your own

Make an enemy's touch hurt more

1. Click the **Enemy** in the Scene dock.
2. At the top of the Inspector open the **Fighting** section and raise **Damage**. Its Hit Box reads that number, and hurts only while **How It Fights** is **Touches**.
3. Raise **Touch Knockback** as well for a harder push.
4. On a character of your own, click its **HitBox** and change **Damage** and **Knockback** there.

Make spikes that hurt only the player

1. Build the spikes as an Area2D with a CollisionShape2D under it, or as any body with a collision shape. An Area2D is Godot's node for a region that notices what enters it, and a CollisionShape2D is the node that gives it its shape.
2. Right-click the spikes and choose **NodeWorthy** → **Give It** → **Hit Box**.
3. Click the new **HitBox**, press **Hurts Group** and pick the player.
4. Leave **Hit Again** on **When Safe**. The player is hurt on touching the spikes and again each time its **Invincibility Time** runs out, for as long as it stays on them.
5. Make sure the player has health, with **NodeWorthy** → **Give It** → **Health** on the player.

Make damage grow with a strength variable

1. Open the character's template with **NodeWorthy** → **Open Its State Machine...** and, on the **Variables** tab, add a number variable called strength with a **Starting Value** of 1.
2. Click the character's **HitBox** in the Scene dock.
3. Press **Var** beside **Damage** and pick strength from the list.
4. Whatever raises strength now raises the damage, such as a level gained through an **Experience** node.

Build a swing out of your own states

1. Click the character's **HitBox**. Untick **On**, set **Hit Again** to **Next Swing**, set **Shape** to **Circle**, and give it a **Radius** and an **Offset** that put the circle in front of the character.
2. Right-click the character and choose **NodeWorthy** → **Open Its State Machine....** Its template opens in the NodeWorthy panel.
3. Click the box of the state that should swing. On the **Actions** tab press **Add Action...** and add **Set Property**. Pick the HitBox under **Node**, pick on under **Property**, and set **Value** to true.
4. Add **Wait** with **Seconds** at 0.2, which is how long the swing lasts.
5. Add a second **Set Property** on the HitBox's on with **Value** at false, and tick its **Wait For Previous** so that it runs after the wait.
6. Give the state a transition back to where it came from, with **When** on **When everything in this state has finished**.

When it goes wrong

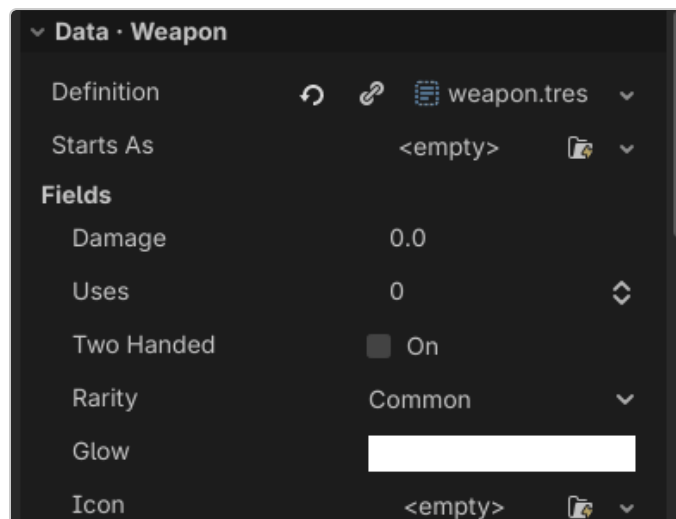
WHAT YOU SEE	WHY, AND WHAT TO DO
A yellow warning sign beside the HitBox says that something has no Hurt Box.	A hit only lands on a Hurt Box. The sign names each thing the Hit Box is set to hurt that cannot be hurt. Right-click each one named and choose NodeWorthy → Give It → Health .
It hurts other enemies as well as the player.	Hurts Group is still on Anything with a Hurt Box, which hurts everything but its own character. Press it and pick the player.
It touches its target and nothing happens.	Check that On is ticked, that Damage is above 0, and that the target is what Hurts Group names. With a variable picked through Var beside Damage , check that the variable is above 0 as well. A hit also does nothing while the target is still safe from the last one.

WHAT YOU SEE	WHY, AND WHAT TO DO
The Output panel at the bottom of Godot says it is set to the same shape as its body, but the body has no collision shape.	Same As Body copies the CollisionShape2D or CollisionShape3D directly under the Hit Box's parent. Give the parent a collision shape, or set Shape to Circle .
The Output panel says it goes under a 2D or 3D node.	A Hit Box works from where its parent is. Under a Label, another screen node or a plain Node it has no place, so it makes no area and hurts nothing. Put it under the body or the area that should hurt.
Godot's warning sign shows beside the Hit Box in the Scene dock	Rest the pointer on the sign to read why. A Hit Box works from where its parent is, so it goes under a 2D or 3D node, such as a sword or a character. With Shape left as the body's own, that node needs a collision shape, or set Shape to Circle . Fix it and the sign goes.

See also **Hurt Box** 180, **Enemy** 74, **Experience** 195, **Perception** 191

Data Node NODE

A **Data Node** gives one thing values of its own, such as this goblin's Health and Damage or that chest's Gold. You pick a Data Definition, which is the list of fields a kind of thing has, and the fields appear at the top of the thing's Inspector, the panel on the right that shows the settings of whatever is clicked.



Data Node in a running game.

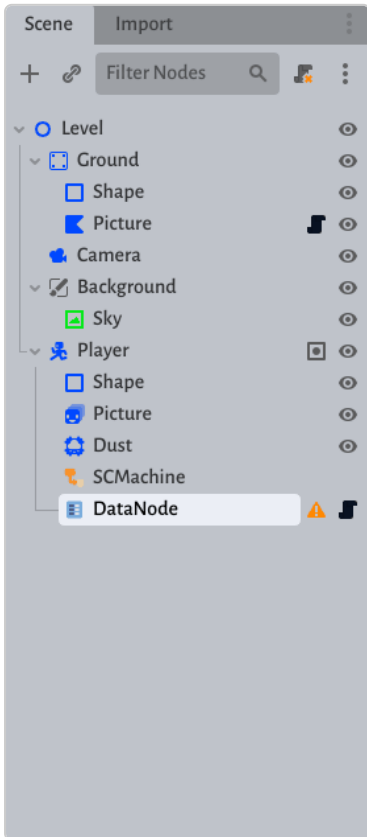
What it is good for

Games with many things of one kind whose numbers differ, such as enemy types, weapons or shop goods. Use it when two copies of one thing must each keep a value of their own that changes during play. A value that only one template needs is simpler as a variable on its **Variables** tab, and the ready-made **Enemy** and **Chest** already keep their numbers as settings at the top of the Inspector.

Add it

1. Start with a level that has an enemy or another character in it. You also need a Data Definition. In Godot's FileSystem dock, the list of files at the bottom left, right-click a folder of your own and choose **Create New** → **NodeWorthy Data Definition...**, and call it Enemy Type.
2. Its grid opens in the NodeWorthy panel. Press **Add Field**, name the field Damage and leave its **Kind** at Number.
3. In the Scene dock, the list of nodes at the top left of Godot, right-click the character and choose **NodeWorthy** → **Give It** → **Data Node**.
4. A node called **DataNode** appears under it, with a yellow warning sign beside it because it has no definition yet.
5. In the Inspector press the empty box beside **Definition**, choose **Quick Load** and pick your definition file. Dragging the file from the FileSystem dock onto the box does the same.
6. The warning sign goes and the definition's fields appear below. Type this character's own values into them.
7. Click the character itself. Its Inspector now has a section at the top whose title starts with **Data**, and it holds the same fields.

Try it. A Data Node shows nothing in the game. To see it work, open the character's template, add **Get Data** to its first state with **From** on **Data On A Node**, and read Damage into a variable. Run the scene, pick the character's machine under **Debug** in the NodeWorthy panel, and the **Variables** tab shows the value that was read, in green.

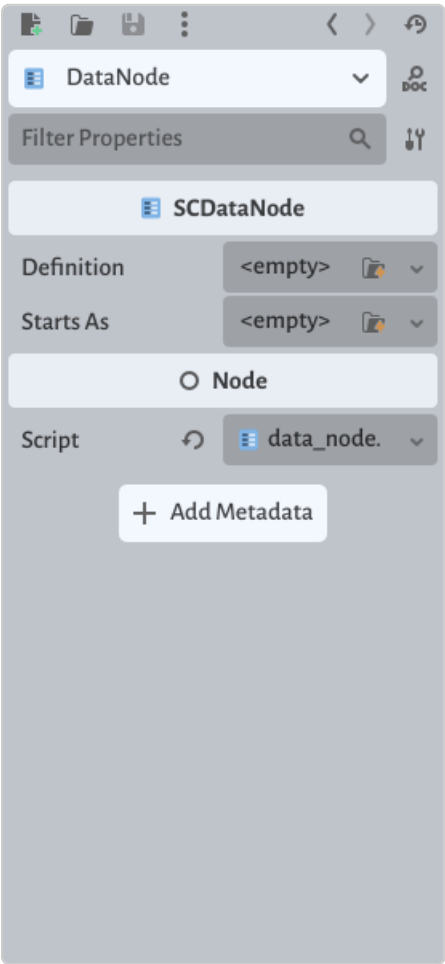


The Scene dock once it is added.

What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
DataNode	Data Node	Holds one record's values for the thing it sits under. It is a plain Node, which is Godot's simplest node, and it draws nothing. The data actions find it by looking directly under the node their machine works on.

Its settings



Click **DataNode** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Definition	The Data Definition whose fields this node holds, shown as Definition . It starts empty, and until one is picked the node has no fields and shows a warning sign. Add a field to the definition later and it appears here as well. A field this node has not set uses the value the definition starts it at.	empty
Starts As	A Data record of the same definition to take starting values from, such as a Goblin file. It starts empty. With a record picked, every field this node has not set uses the record's value, and the fields it has set keep their own. A record made from another definition gives the node nothing, and a warning sign beside the node says so.	empty

Make it your own

Give two goblins different damage

1. Give each goblin a Data Node with **NodeWorthy** → **Give It** → **Data Node** and pick the same definition in **Definition** on both.
2. Click the first goblin. In the section at the top of its Inspector whose title starts with **Data**, set Damage to 2.
3. Click the second goblin and set its Damage to 5.
4. Each keeps its own number, even when both run one template.

Start a goblin from a record

1. In the FileSystem dock right-click a folder of your own and choose **Create New** → **NodeWorthy Data...**, and call it Goblin.
2. In the Inspector pick the Enemy Type definition in its **Definition**, and fill in the fields that appear.
3. Click the goblin's **DataNode** in the Scene dock, press the box beside **Starts As**, choose **Quick Load** and pick the Goblin file.
4. Fields you have not typed into on the node now follow the Goblin record, so a change to the record changes every goblin that starts as it.

Read its value in the template

1. Right-click the character and choose **NodeWorthy** → **Open Its State Machine...** Its template opens in the NodeWorthy panel.
2. Click the box of the state that needs the value. On the **Actions** tab press **Add Action...**, type get data, click **Get Data** and press **Add**.
3. Set **From** to **Data On A Node**. Leave **Node** empty, which means the node this template runs on.
4. Pick Damage in **Field**.
5. Press **Store In** and choose **New variable...**, calling it damage. Actions further down the state can now read damage.

Take 1 off this goblin's own Health

1. Add a Health field to the definition with **Add Field**, and type a Health for each goblin.
2. In the goblin's template, in the state it enters when it is hit, press **Add Action...** and add **Set Data**. Its **From** starts on **Data On A Node**.
3. Pick Health in **Field**, set **Operation** to **Add** and **Value** to minus 1.
4. To act when it runs out, add a **Data** check that asks whether Health is **Less Or Equal** to 0, and lead its event to a state that removes the goblin.

Read a weapon from a table

1. A table of many records is read with **Find Row**, which keeps one row in a variable. A Data Node is for values that belong to one thing. The two use the same definitions.
2. In the NodeWorthy panel press **+** and choose **New Data Table...**, call it Weapons, and pick a Weapon definition in its **Definition** in the Inspector. Press **Add Row** for each weapon and type its values into the grid.
3. In a state of the template, press **Add Action...** and add **Find Row**. Pick the Weapons table in **Table**, set **How** to **Named** and type Axe in **Row Name**. Press **Store In** and choose **New variable...**, calling it weapon.
4. Add **Get Data** under it. Set **From** to **A Variable**, pick weapon in the **Data Variable** box that appears, pick Damage in **Field**, and store it in a number variable.
5. Run the scene. The variable holds the axe's Damage, and changing the table changes what is read.

When it goes wrong

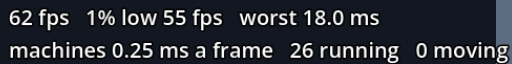
WHAT YOU SEE	WHY, AND WHAT TO DO
A yellow warning sign beside the DataNode says to pick a Data Definition.	The node has no Definition yet, so it has no fields. Click it and pick a definition file in Definition .

WHAT YOU SEE	WHY, AND WHAT TO DO
The Output panel, Godot's panel at the bottom that prints messages, says a node has no Data node under it.	A Get Data , Set Data or Data check with From on Data On A Node ran on a node that has none. Right-click that node and choose NodeWorthy → Give It → Data Node , or pick the right node in the action's Node .
The record picked in Starts As changes nothing.	The record was made from another Data Definition than the one in Definition , so none of its values are used. A yellow warning sign beside the node says so, and the Output panel says it when the game runs. Pick a record of the same definition.
Values changed while playing are back where they started on the next run.	A Data Node starts every run from what the scene says. What Set Data changes lasts while the game runs and is not written into any file.
Data Node is greyed out in the Give It menu.	The thing already has one and needs only one, because the data actions and the Inspector use the first Data Node directly under a node. Put every field the thing needs into one definition.

See also **Enemy** 74, **Chest** 104, **Inventory** 185, **Stat Panel** 122

Performance Overlay NODE

A **Performance Overlay** is a small readout in the top left corner of the running game that shows how fast it runs. It shows frames a second, the 1% low, the worst frame, how long the machines take in each frame, how many machines are running and how many nodes are being kept moving.



Performance Overlay in a running game.

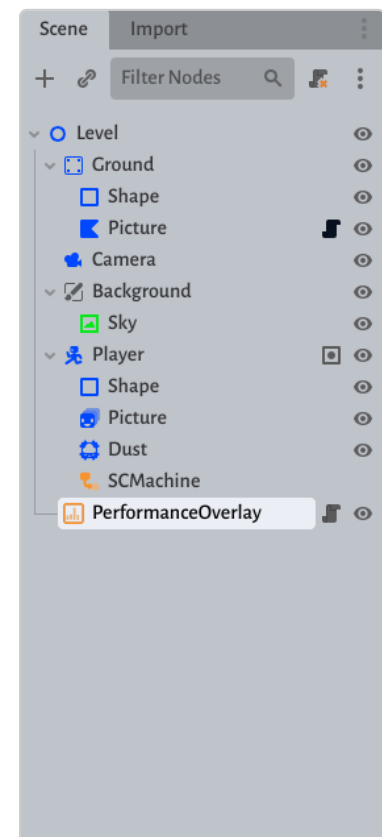
What it is good for

While you build, whenever the game feels rough, or when you add a crowd of enemies or bullets and want to know what it costs. It is a tool for you and not a part of the game, so take it out before you give the game to anyone. It only measures, and changes nothing about how the game plays. To show the player a number of the game's own, such as health, use a **Stat Panel**.

Add it

1. Start with a level in which something runs a template, such as a player added with **NodeWorthy** → **Add Player**. The Performance Overlay is not in the right-click menu. It is added from the **Add Action** window.
2. Click the player in the Scene dock, the list of nodes at the top left of Godot, and press **Open Its State Machine...** under its settings at the top of the Inspector, the panel on the right that shows the settings of whatever is clicked. Its template opens in the NodeWorthy panel.
3. Click any state's box, and on the **Actions** tab on the right press **Add Action....**
4. Type performance in the search box. **Performance Overlay** is listed under **Nodes To Add**, after the actions.
5. Click it and press **Add To Scene**. A node called **PerformanceOverlay** is added under the top node of the scene and selected. Nothing is added to the state.
6. Press **F6**, or the Run Current Scene button at the top right of Godot.

Try it. Two lines of white text appear in the top left corner. The first gives the frames a second, the 1% low and the worst frame in milliseconds. The second gives the machines' time for a frame, how many machines are running and how many nodes are moving. Press **F3**, or the backtick key under Escape, to hide it and show it again.



The Scene dock once it is added.

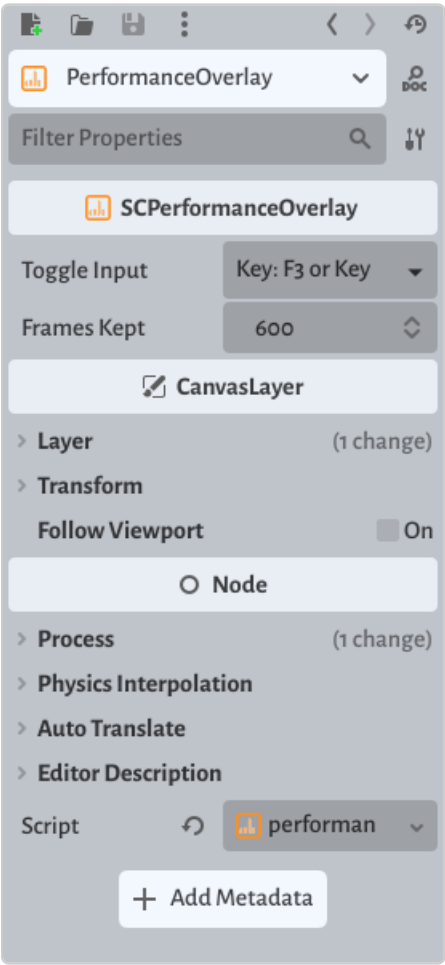
What you get

IN THE SCENE DOCK	IT IS A	WHAT IT IS FOR
PerformanceOverlay	Performance Overlay	A CanvasLayer is Godot's node for things drawn over the game. This one sits on layer 100, so it draws over your own screens, and it makes its text when the game starts. It keeps running while the game is paused.

The controls

PRESS	WHAT IT DOES
The F3 key, or the backtick key under Escape	Shows or hides the panel while the game runs. The numbers are gathered either way.

Its settings



Click **PerformanceOverlay** in the Scene dock and these are in the Inspector, the panel on the right, under its own name.

Its settings in the Inspector.

SETTING	WHAT IT DOES	STARTS AS
Toggle Input	The keys that show and hide the panel while playing, shown as Toggle Input . It starts as the F3 key or the backtick key, and either one works. There are two because on a Mac the F keys work the screen's brightness and its windows unless the keyboard is set otherwise, so F3 alone never reaches the game there. Emptied, no key is listened for and the panel stays as it is. To use other keys, press the little arrow beside the box and pick from the list of keys, mouse buttons and gamepad buttons.	Key: F3 or Key: Backtick
Frames Kept	How many of the latest frames the numbers are worked out from, from 60 to 3600. It starts at 600, which at 60 frames a second is the last ten seconds. More gives steadier numbers, and fewer follows a change sooner. Changed while the game runs, the numbers start afresh from that moment.	600

Make it your own

Keep it hidden until you ask for it

1. Click **PerformanceOverlay** in the Scene dock.
2. In the Inspector untick **Visible**, which is one of Godot's own settings for the node.
3. Run the scene. Nothing shows until you press **F3** or the backtick key, and the numbers have been gathered all the while.

Find out whether a crowd of machines is slowing the game

1. Run the scene with the crowd in it and read the second line. It gives the machines' share of a frame in milliseconds and how many are running. At 60 frames a second a whole frame is a little under 17 milliseconds.
2. If the machines take a large part of that, click one of the crowd in the Scene dock and press **Open Its State Machine...** under its settings.
3. Click the state the crowd spends most of its time in. On the **Actions** tab look for a **Check** that runs every frame and does not need to.
4. Click that card and set its **Update Interval** to 0.2, so it asks five times a second.
5. Run again and compare the machines' time with what it was.

Read steadier numbers, or quicker ones

1. Click **PerformanceOverlay** and change **Frames Kept**.
2. Set it to 1800 for numbers that settle over half a minute at 60 frames a second.
3. Set it to 120 to see a change within a couple of seconds.

Take it out before you ship

1. Click **PerformanceOverlay** in the Scene dock of every scene you added it to.
2. Press the Delete key and save the scene.

When it goes wrong

WHAT YOU SEE	WHY, AND WHAT TO DO
It is not in the NodeWorthy right-click menu.	Among NodeWorthy's own menus the Performance Overlay is offered only in the Add Action window, under Nodes To Add . Open any template, press Add Action... and search for performance.
Pressing F3 does nothing on a Mac.	On a Mac the F keys work the brightness and the windows unless the keyboard is set otherwise. Press the backtick key under Escape, which does the same.
The panel has gone and no key brings it back.	Toggle Input has been emptied, so no key is listened for. Click PerformanceOverlay and tick Visible again, or press the round arrow beside Toggle Input to put back what it started as.
Frames a second looks fine and the game still stutters.	Read the 1% low and the worst frame. A game that averages 60 with a 1% low of 30 feels rough, and a smooth one keeps the two close.

See also **Enemy** 74, **Perception** 191, **Hit Box** 200