
EgoVenture

Release 1.0.0

Dennis Ploeger

May 27, 2023

CONTENTS

1	Introduction	3
1.1	Getting started	3
1.2	Basic Structure	19
1.3	The game state	20
1.4	Scenes and navigation	20
1.5	Inventory handling	21
1.6	Hotspots	23
1.7	Effects, music and background	24
1.8	Configuration	25
1.9	Hint system	25
1.10	Tools	26
1.11	Theming EgoVenture	26
1.12	Updating EgoVenture	30
1.13	EgoVenture API reference	33
2	Development	83
3	Logo	85



INTRODUCTION

EgoVenture is a [Godot](#) framework written in Gdscript for creating first person point and click adventure games like the [Carol Reed games](#) by [MDNA games](#) for mouse- and touch-based games.

Actually, this engine was originally made by MDNA games together with deep entertainment and is used for the Carol Reed series since 2021.

It's streamlined for their games, but may be of use for other developers as well. Thus we're releasing it as Open Source under the MIT license.

If you want to see all the features, *EgoVenture* provides, check out the [egoventure-example-game](#) “[Carol Refurbished](#)”.

1.1 Getting started

Hello and welcome to *EgoVenture*.

EgoVenture is a framework for creating first person point and click games. It handles and streamlines the most basic stuff and makes it easy to develop games like this.

The *EgoVenture* project provides different packages:

- The [EgoVenture example game](#), which is made by MDNA games and showcases all features of *EgoVenture*. You can download the game and open it in Godot to check out how things are done.
- The [EgoVenture game template](#), which is used as a starting point for a new game, that is based on *EgoVenture*. This template includes all required plugins, a default game configuration, a basic game state and initialization code and the recommended folder structure to get you started right away.
- The individual Godot addons, that form the *EgoVenture* features:
 - The [EgoVenture core plugin](#) which holds the basic features
 - [Speedy Gonzales](#) for easy custom mouse cursor handling
 - [Parrot](#) used for voice dialog handling

In the following sections we will guide you how to start creating your first game using the *EgoVenture* framework.

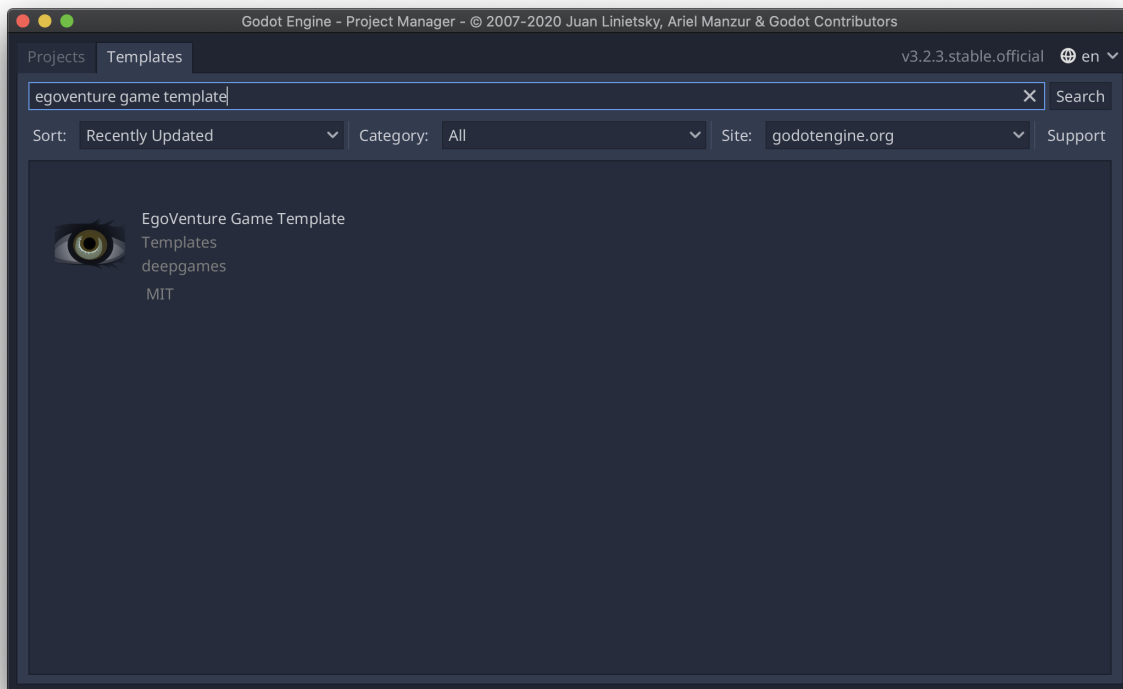
1.1.1 Downloading Godot

Go to <https://godotengine.org/download> and download [Godot 3.3.4](#) for your operating system and install it.

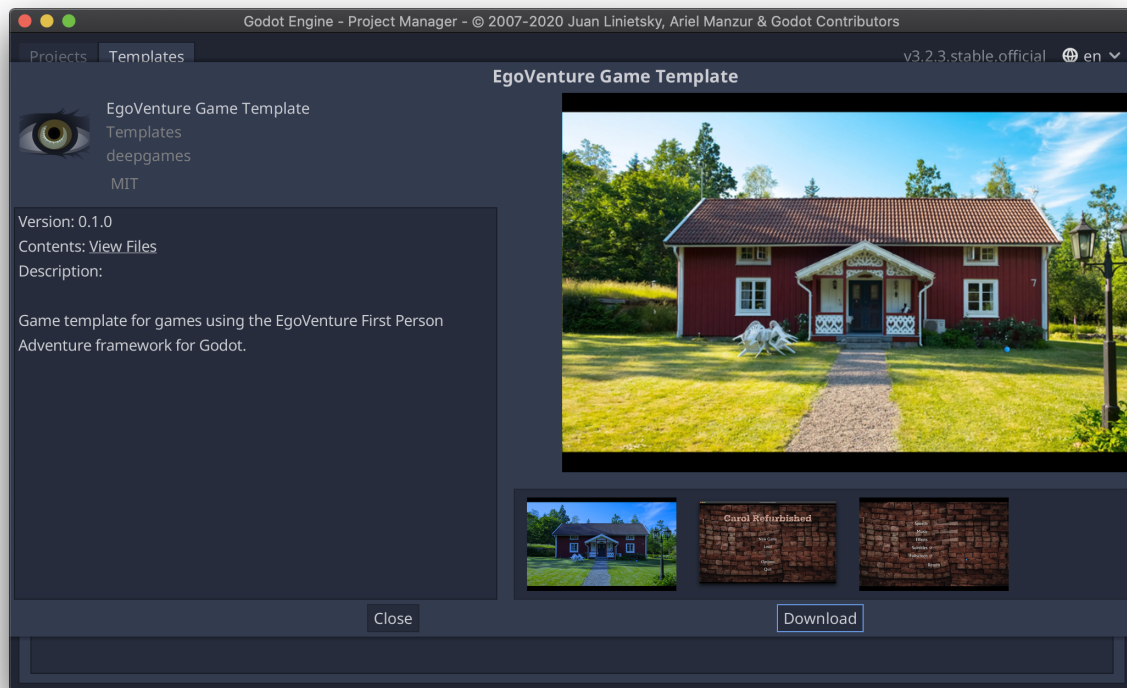
Important: EgoVenture currently doesn't support Godot 3.4 yet. We plan to have it available with one of the next releases.

1.1.2 Create a new Project

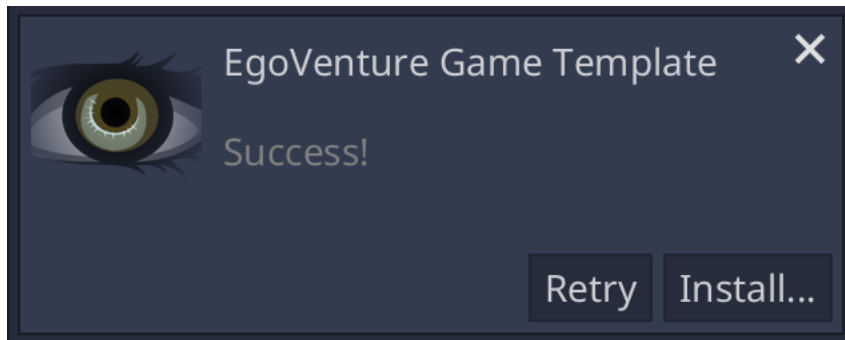
Start Godot, and search for the *EgoVenture Game Template* in the Templates tab:



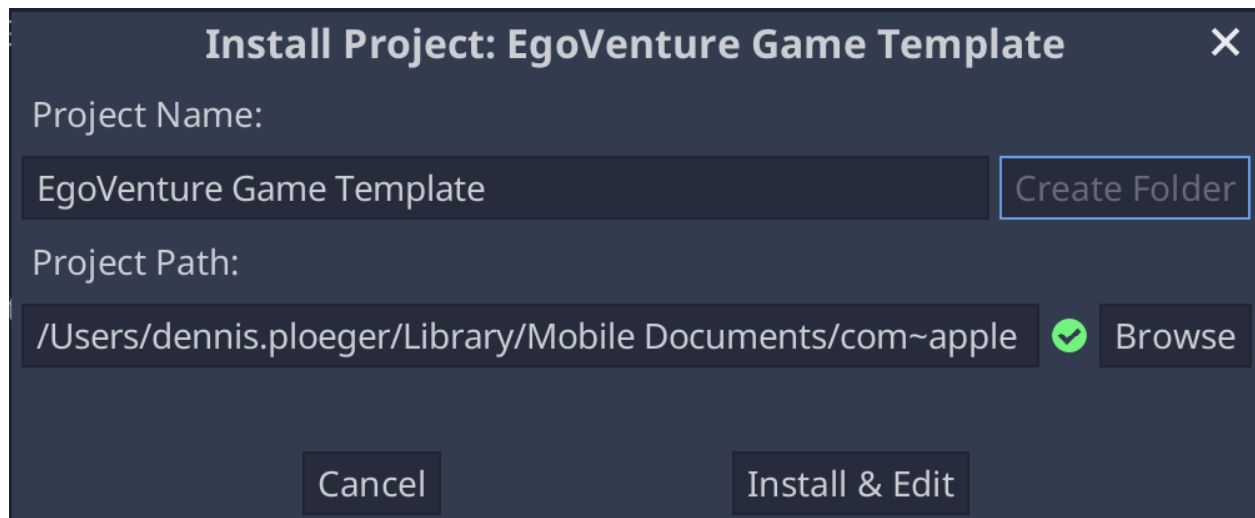
Click on “EgoVenture Game Template”.



Click on “Download” to download the game template to your computer. After finishing, click install:



Enter the name of your new game and the folder where it should be created:



Install Project: EgoVenture Game Template X

Project Name:

EgoVenture Game Template Create Folder

Project Path:

/Users/dennis.ploeger/Library/Mobile Documents/com~apple ✓ Browse

Cancel Install & Edit

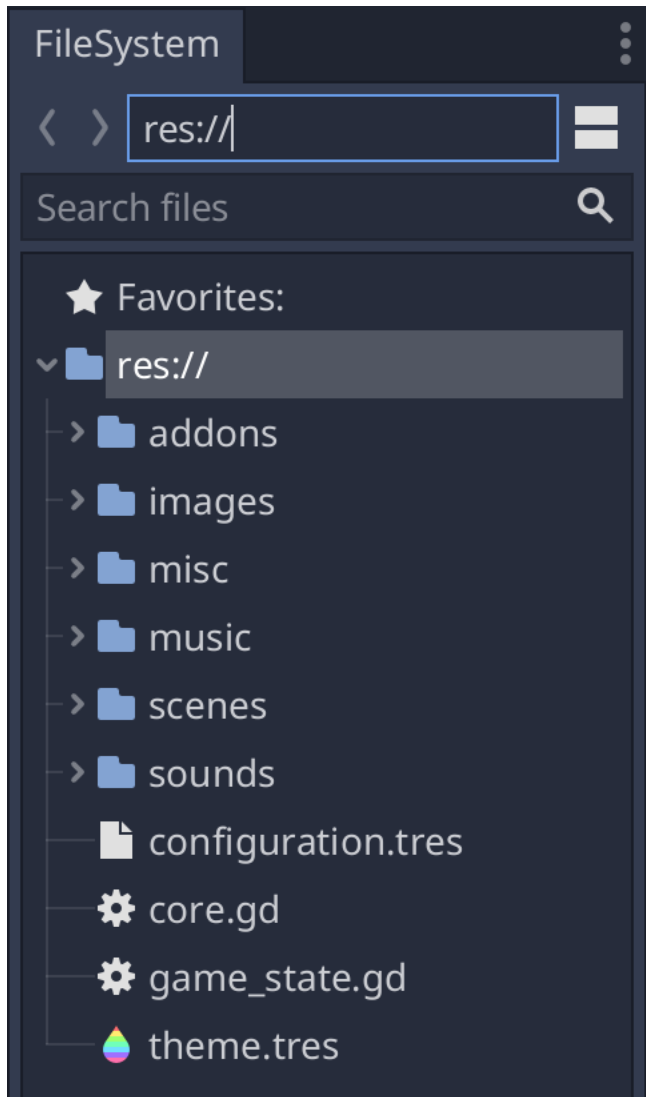
Click on “Install & Edit” to start the editor.

The *EgoVenture* game template includes a small set of assets (i.e. mouse cursor icons, a basic game configuration and theme) that helps you getting started. After you’ve clicked on “Install & Edit”, these assets are automatically processed by Godot.

After that process finished, **close Godot and open it again**. Godot seems to loose the links to the provided assets if you don’t.

1.1.3 Default folder structure

Afterwards check out the default folder structure of the game template:



- **addons:** Holds the compatible versions of the required addons *EgoVenture*, *Parrot* and *Speedy Gonzales*
- **characters:** Holds all dialog characters if you want to take advantage of *Parrot*'s multi-character feature
- **dialogs:** Your game's *Parrot* dialogs
- **inventory:** The inventory items and their corresponding images
- **images:** The root folder for all visual resources in this project. Currently, there are two subfolders there:
 - **misc:** Holds various images like icons and button images
 - **mouse:** Holds the default mouse cursor images
- **misc:** Holds everything that doesn't match the other folders. Currently, it holds one subfolder:
 - **fonts:** The used fonts
- **music:** Holds all music items
- **scenes:** Holds all locations and their corresponding scenes. Map scenes, for example, can directly reside in "scenes", while other scenes should be in a location subfolder. For more information check out the [scenes management documentation](#).

- **sounds:** Holds all background sounds and sound effects
- **voices:** The voice files of your *Parrot* dialogs

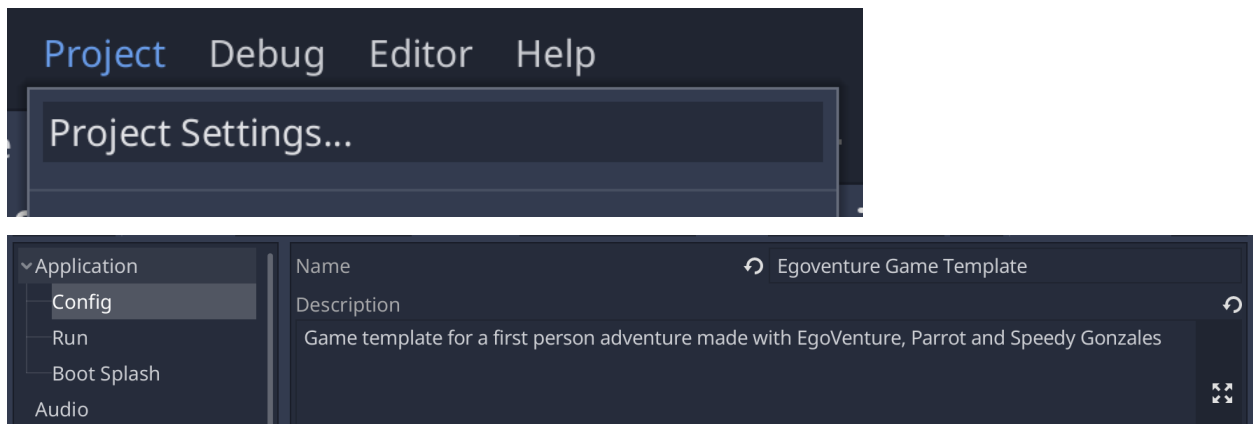
This folder structure is a recommendation for organizing the assets of the game, but you can obviously change it to your liking.

Additionally, these files reside in the root directory of the project:

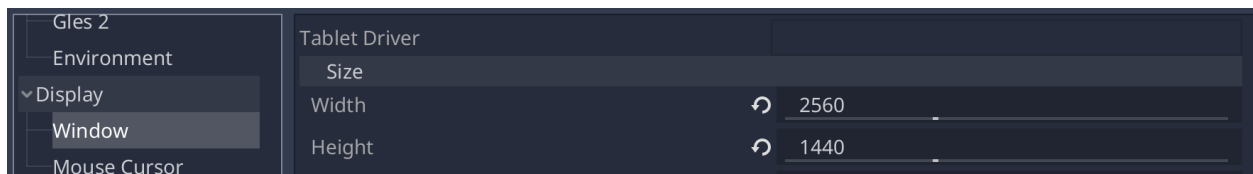
- **configuration.tres:** The default *game configuration*
- **core.gd:** The core singleton for the game template, that controls the basic game logic. Check out the *structure documentation* for details
- **game_state.gd:** An example *game state*
- **theme.tres:** The default *game theme*
- **hints.txt** (not visible in Godot): An example file for the *internal hints system*.
- **project.godot** (not visible in Godot): The project file of your new game
- Additional files like the README and license file from the game template (not visible in Godot). Those can safely be removed.

1.1.4 Setting up

It's recommended to first set the game name in the project settings. Click on the *Project* menu, then on *Project Settings...* On the left side, select the section *Config* under *Application* and change the *Name* and *Description* fields so that they match your game. This way, you can easily identify your game in the project manager when you start Godot.



Additionally, you might want to check the game resolution in the *Window* section under *Display*. EgoVenture defaults to the QHD resolution of 2560 by 1440 pixels.



If you plan to use a smaller resolution, you need to change this setting.

Afterwards, find the *game configuration* file **configuration.tres** in the file explorer in the lower left corner, doubleclick it and configure everything you need using the “Inspector” on the right side of the screen. All available configuration options are documented in the *game configuration* page of this documentation.

Also, check out the [default theme](#) by double clicking on the file `theme.tres` and change it to your liking. Check out the [theme documentation](#) for details about configuring a theme and what things you can change in *EgoVenture*.

1.1.5 Importing assets

Open the game folder in your operating system's file manager. There you will see the same folder structure as documented above, so you can directly copy game assets like images, music and sounds to their respective folders.

For scene images, it's recommended to group connected images of a specific area into *locations*, add a new folder under *images* for each location and put the images there.

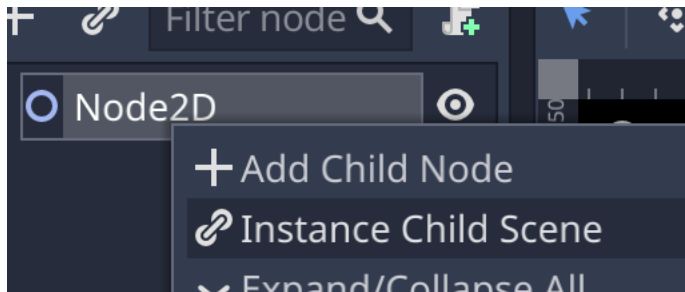
When you get back to Godot, it will import those assets into a standard format, so they can be used in all supported platforms. Check out the [Godot documentation about the import process](#) for details.

1.1.6 Designing scenes

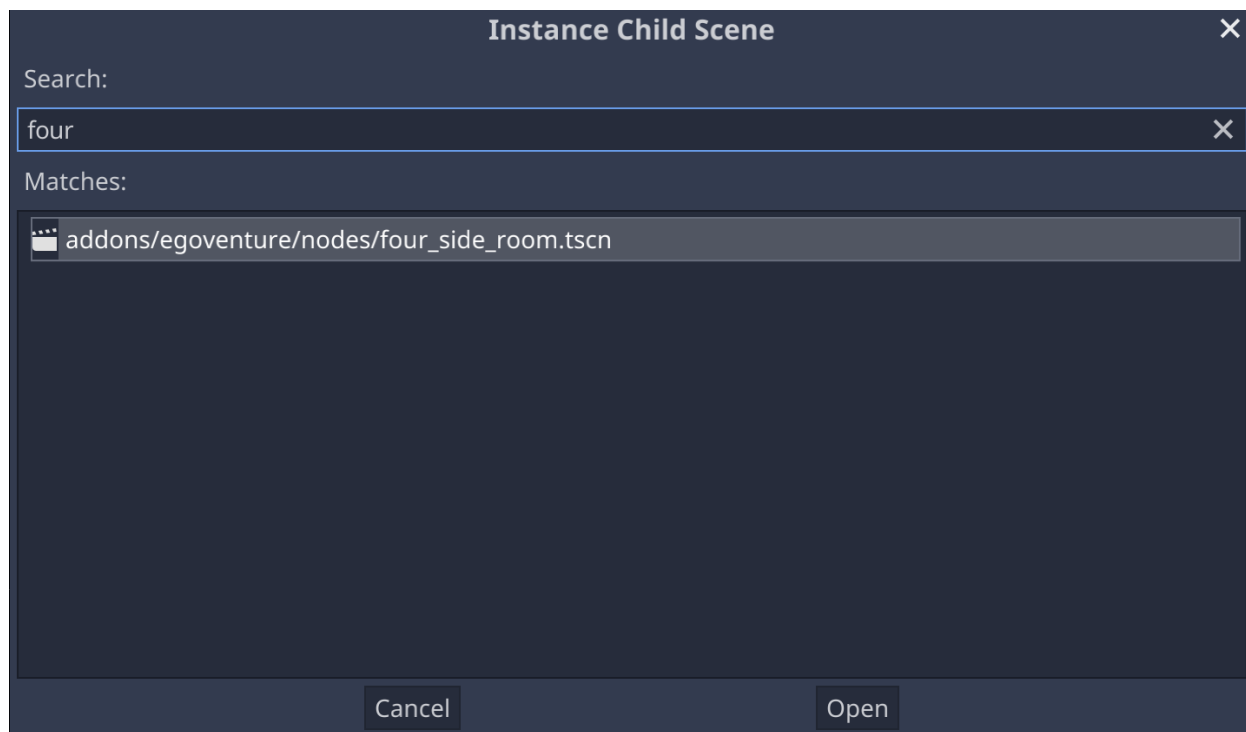
Select "New Scene" from the "Scene" menu to create a new scene. You can use all the provided Godot resource types to build up your specific scenes. *EgoVenture* has some special resources that aid you in creating the scenes for your game. For example, there is a special scene that supports four different images, each for one side and already includes features to switch around in this scenes.

Setting up the scene

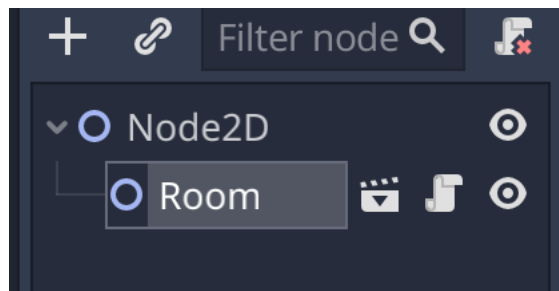
To use this, right-click the first node of the new scene, and select "Instance Child Scene".



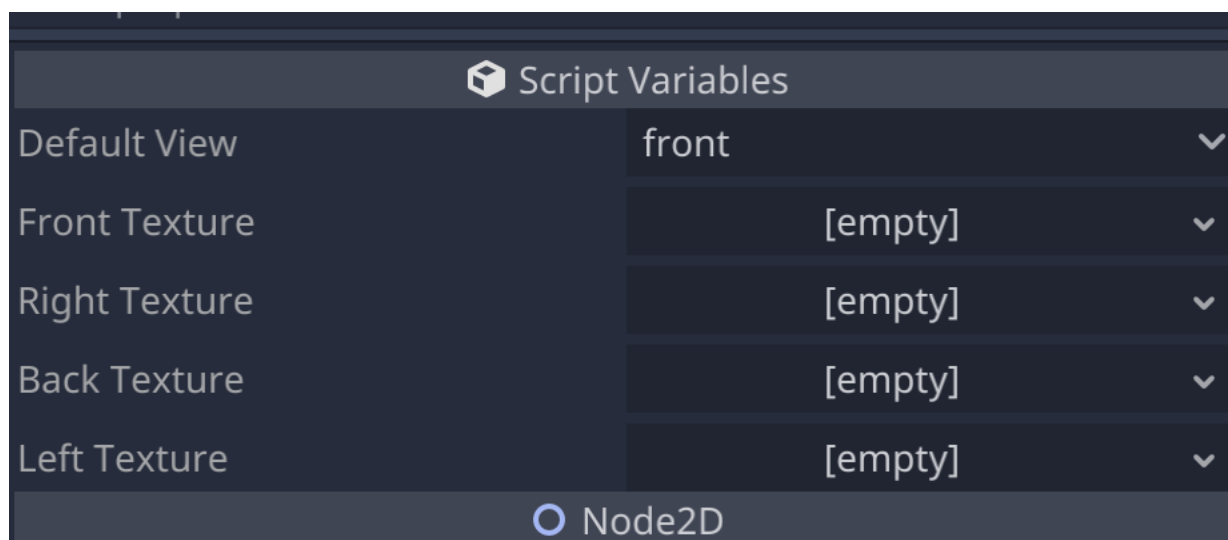
Search for "four" and select "Four Side Room". Click on "Open"



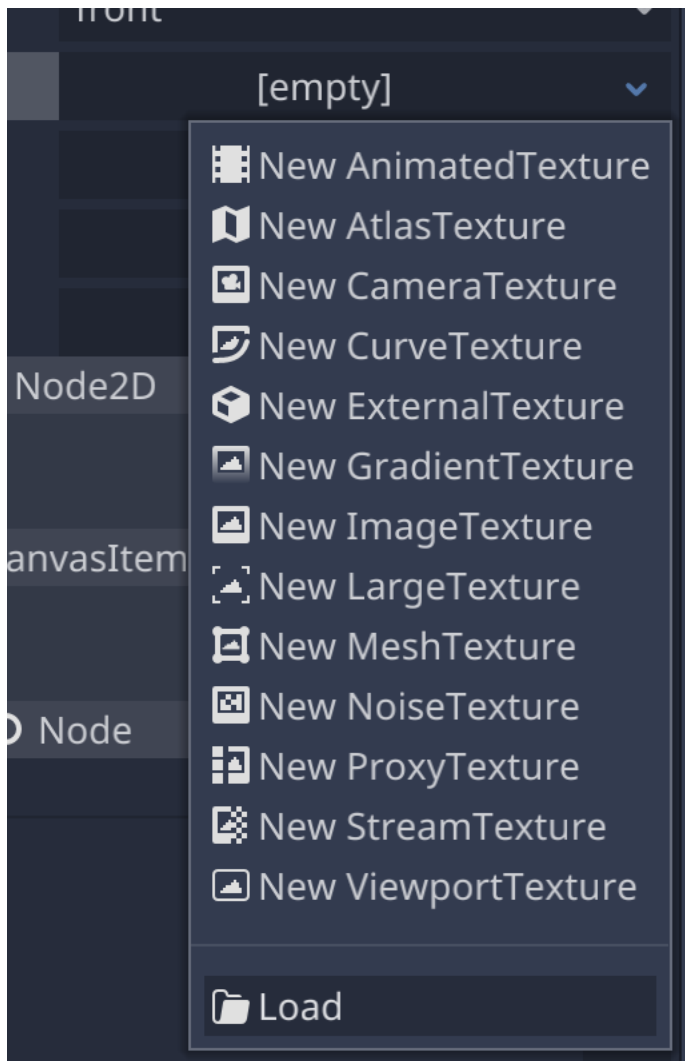
This will add a new node to your scene, which is an *instance* (basically a copy) of the four side room scene.



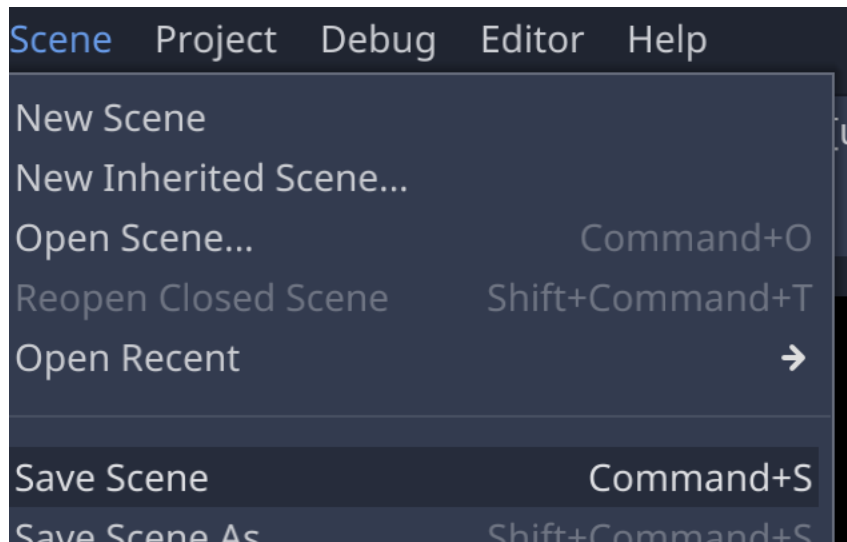
Look into the inspector on the right side. Here you can configure four images that each represent one side of this scene and select a default view that will be shown when the player enters the scene.



To select an image, click on the arrow of the respective view and select *Load*.



Select the image from the file browser and a small preview will be shown. Set an image for each side then save the scene using *Save Scene* from the *Scene* menu:



Use a meaningful name, so that you can reference it later when the player walks through the rooms.

Now it's time to playtest this scene. Click on the *Play Scene* button on the upper right part of Godot:

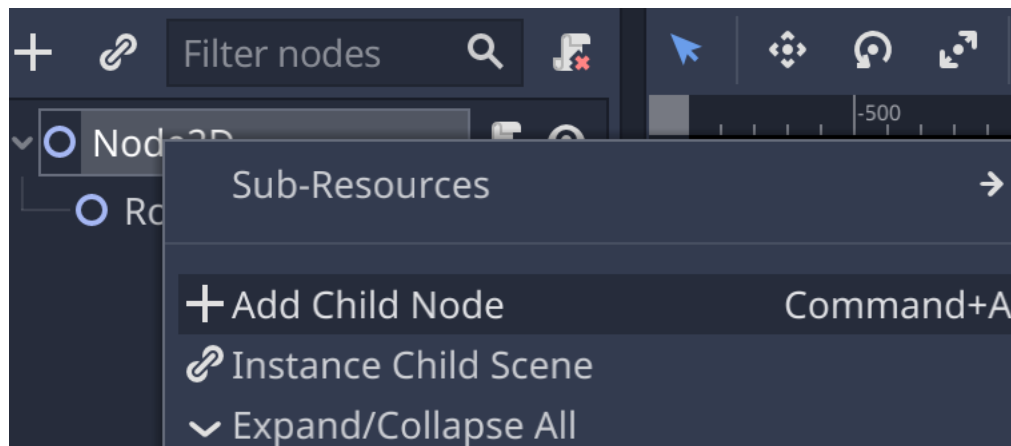


If all went right, you should see the default view you configured earlier and can click on the left and right side of the screen to “turn around” in your scene.

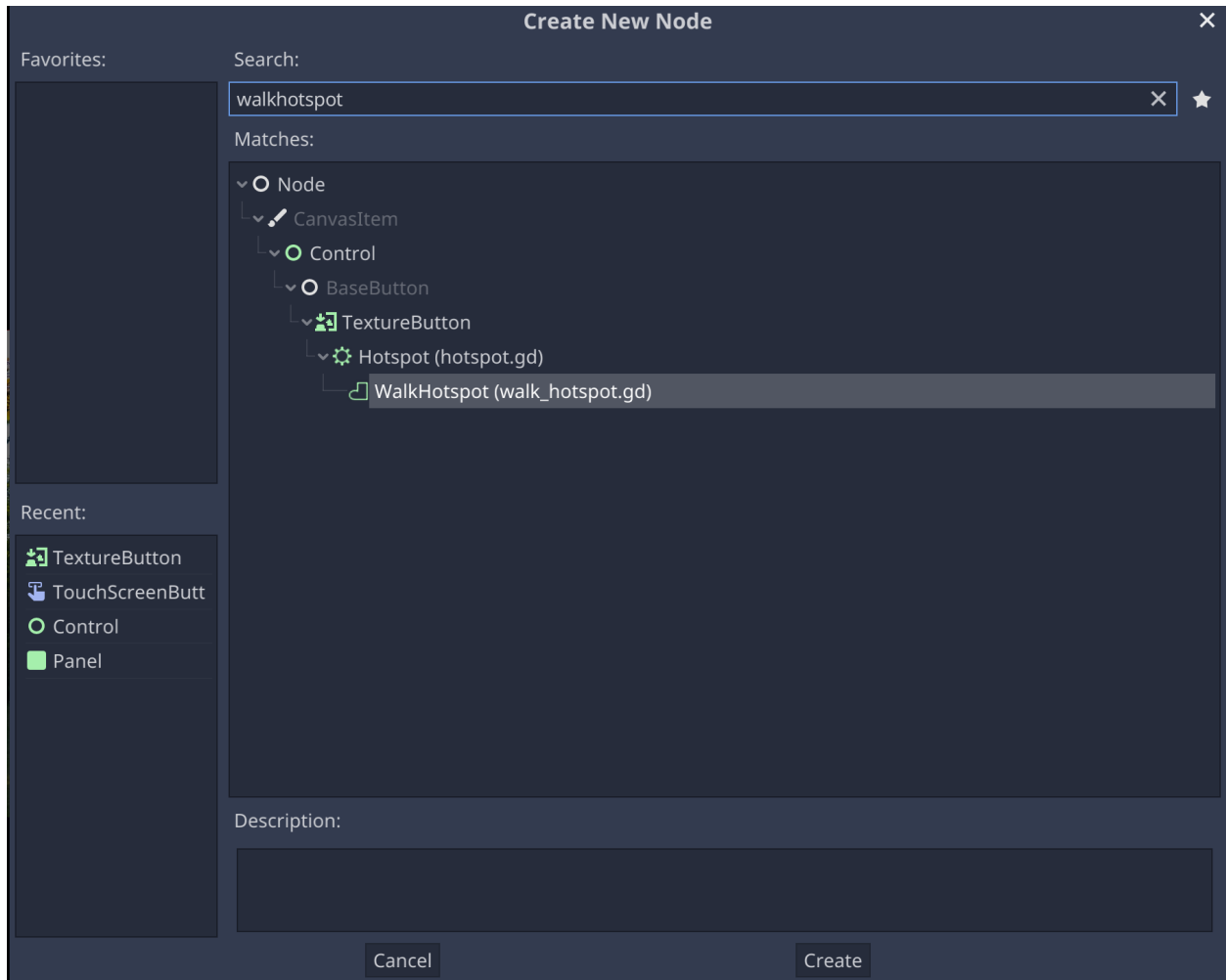
So, turning around now works. Now we should allow the player to advance to the next scene.

Adding a hotspot

Right click on the top node of your scene again. This time, click on *Add Child Node*

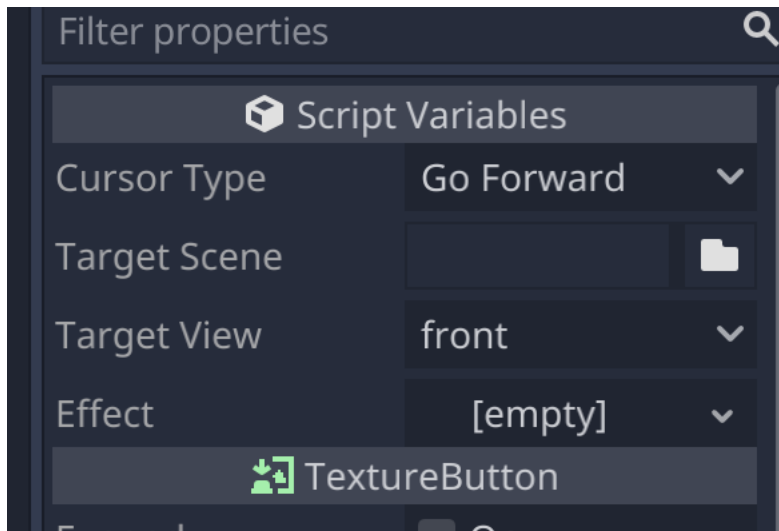


Search for “WalkHotspot” and click on *Create*:



A **WalkHotspot** is a standardized way of creating clickable areas where the player can move to another scene. This will create a rectangle in the scene view, that you have to move and resize to match the area the player can click:

In the inspector you can configure the target scene by selecting it from the file browser. You can also select the view that should be shown when entering the new scene, another mouse cursor, if you don't want to show the *Go Forward* cursor and select a sound effect that is played when the user clicks on the area and can be used when opening doors for example:



Use this methods to move around in your game.

Adding more interactivity

Now it get's a bit more complicated, as we will need to use Godot's scripting language *GScript* for the more advanced things like handling inventory items or modifying your scenes.

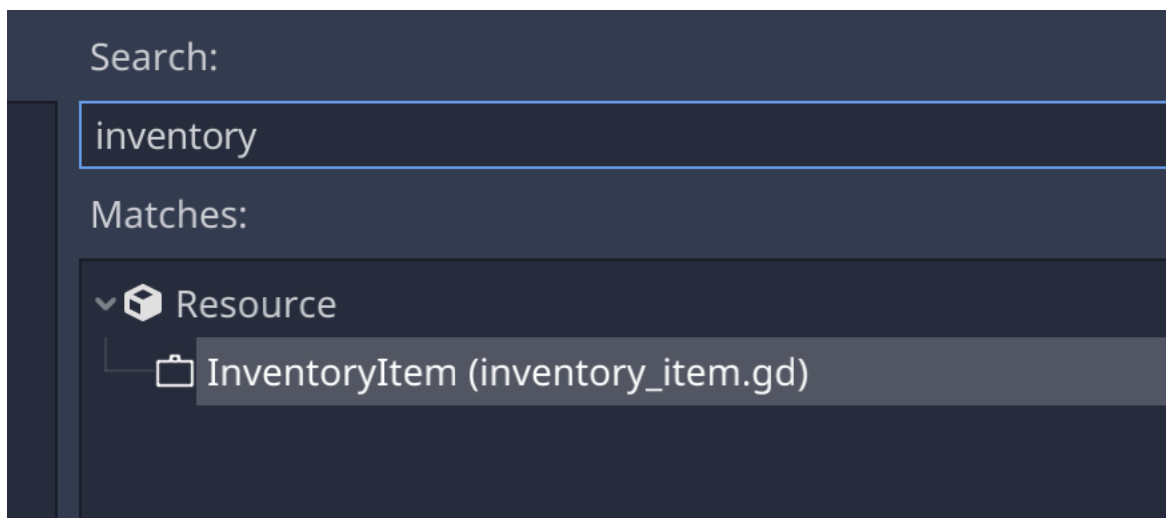
In this example we will be adding a new inventory item to the inventory.

First, put three images for the new inventory item in the folder `inventory`:

- A normal image, that is shown in the inventory
- A highlighted image that is shown when the user selects the item and moves it over an area where it can be used
- A big image, that will be shown when the user right-clicks the item in the inventory

After that create a new inventory item resource by right clicking the folder `inventory` in the file browser. Select *New Resource*:

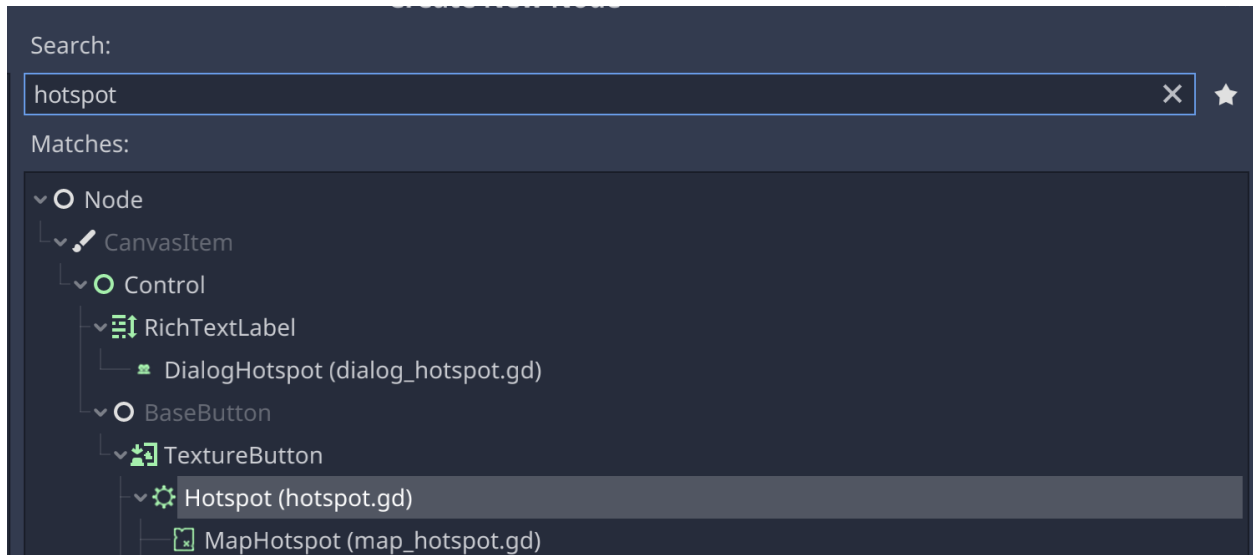
Search for *Inventory* and create a new `InventoryItem` resource:



Give the new resource file a meaningful name (e.g. `keys.tresor dog_whistle.tres`). In the inspector, set a title for your inventory item, a description that will be shown in the detail view and select the three images for the respective properties:

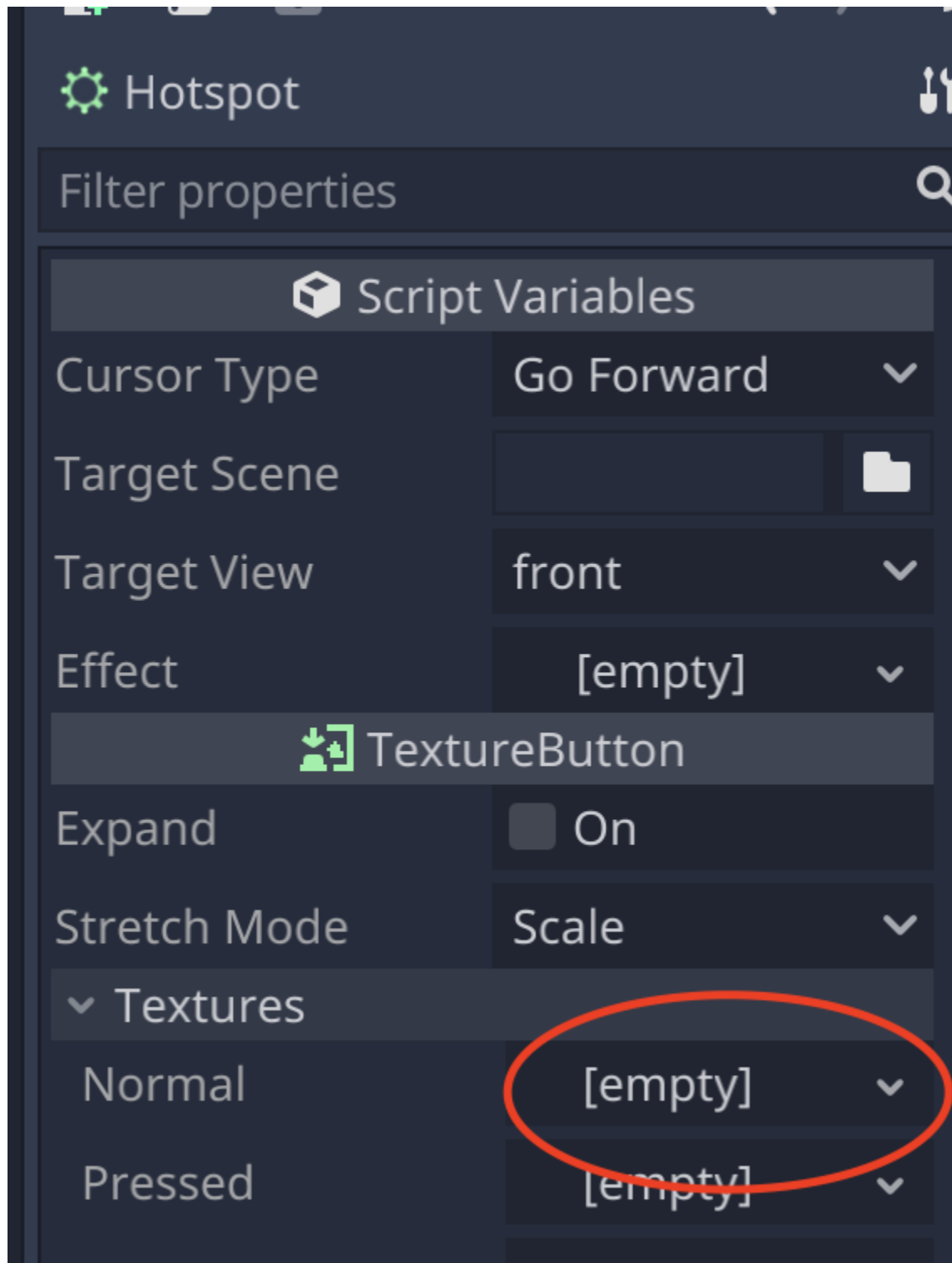
There are some more options here, but let's leave them for the moment.

Back to the scene. Right click the top node again and add a new child node. This time, select a Hotspot:

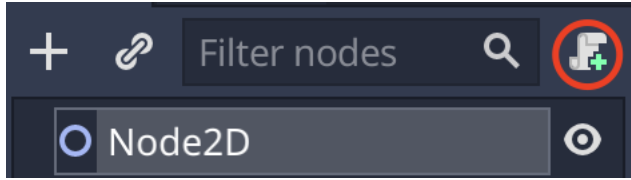


This time you only have to select a fitting mouse cursor for the hotspot (e.g. *Hand*) and move the rectangle somewhere in the scene where the player can “pick up” your inventory item by clicking on the area.

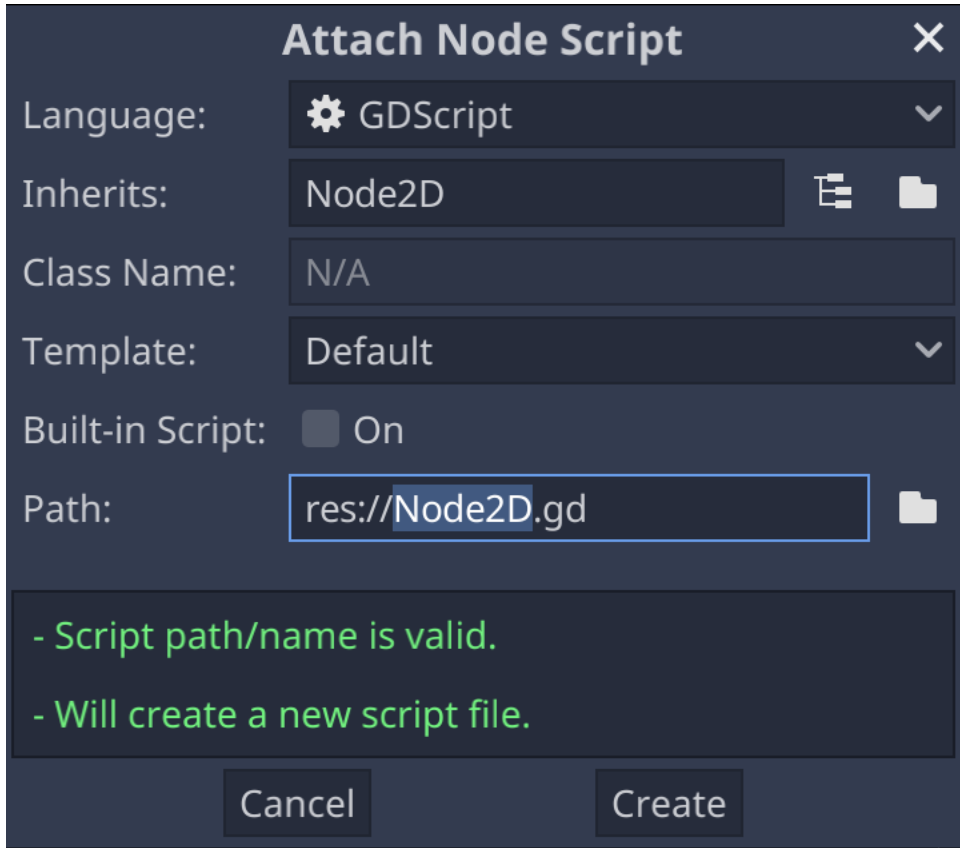
If the item is not part of your background image, you can use the “normal” texture property in the hotspot to add an image to the hotspot:



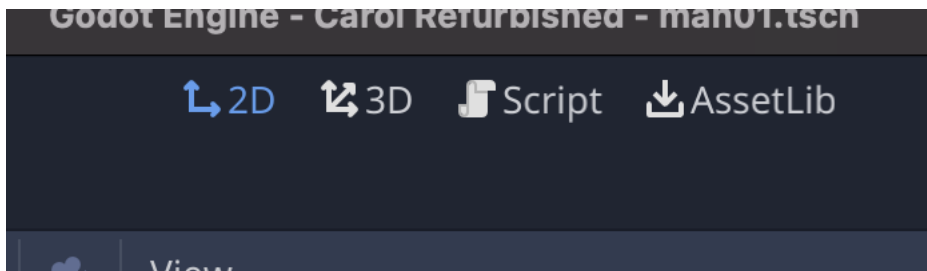
Now we need to add a new script for this scene. We will use Godot's own scripting language *GDScript* in this example. Select your top node using the left mouse button and use the *Attach Script* button to add a new script to your scene:



Enter the name of your new script. It's recommended to give the script the same name as the scene (with the .gd extension)

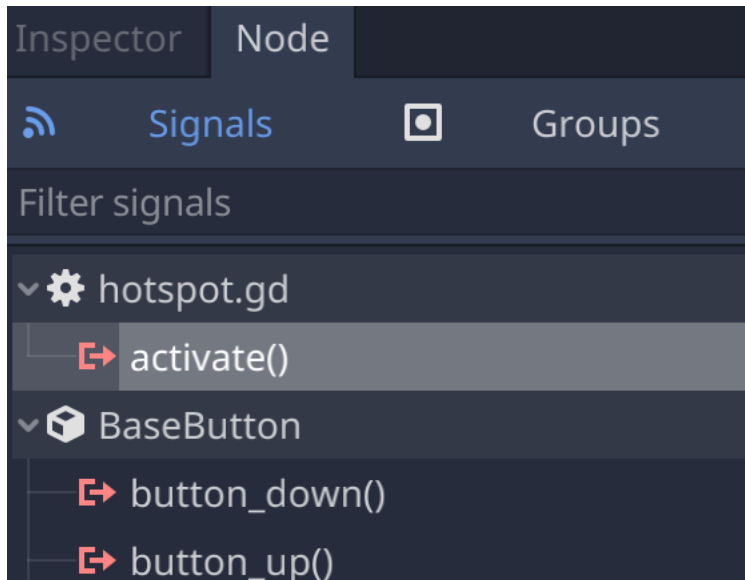


This will directly open the script in the built-in script editor. For now, just use the *2d* button in the mode selection over the editor:



Godot scripting is done using a system of *signals*. Nodes can emit signals on various actions and scripts can “listen to” these signals, and react to them.

Select the new hotspot and on the right side, switch from *Inspector* to *Node* to see the available signals. Double click on *activate*.



Enter a meaningful name for a new function, that will be created in the script of the scene and that will listen to the *activate* signal.

Usually, you can just accept the default and click on *Connect*. The script editor will be shown again and you're directly in the new function:

```
17
18
19 ▸ func _on_Hotspot_activate():
20     >| pass # Replace with function body.
21
```

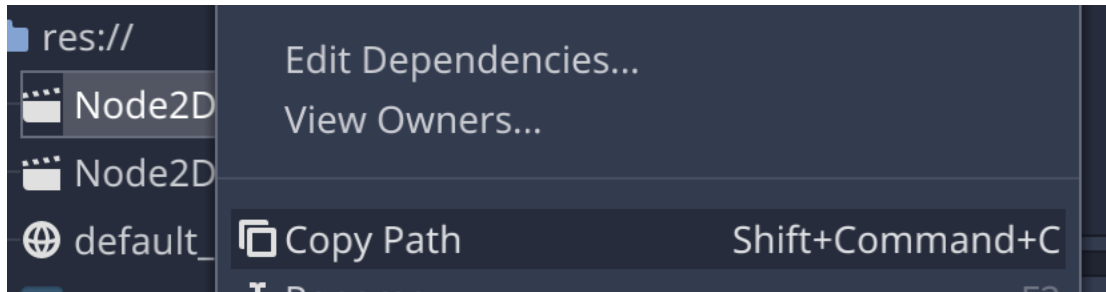
Now it comes to scripting, but don't be afraid. GDScript is very easy to understand and all the available functions of *EgoVenture* are well documented in the [API documentation](#).

To add a new item to the player's inventory, you need the *Inventory* class and specifically the `add_item` function. Also, you need the path to your inventory item.

Godot uses a so-called *virtual file system*, because the file systems of each platform, that Godot supports, differ in certain aspects. Paths in Godot always start with `res://` and the path to the specific resource after that. Each directory is separated using a forward slash `/`. So, for example, if you added the inventory item `keys.tres` in the `inventory` folder, the path would be:

```
res://inventory/keys.tres
```

To make it easier to use paths, you can right-click a file in the file browser and then click on *Copy Path*.



To add this item to the player's inventory, remove the line `pass` from the script and replace it with:

```
Inventory.add_item(preload("res://inventory/keys.tres"))
```

Don't get confused by the `preload` function. That is used for *loading* the inventory item itself and giving it to the `add_item` function and is required in this case.

That's it. Start the scene again and click on the area. The inventory bar should show your new item.

1.1.7 Where to go from here

Now that you know the basics, check out the rest of the documentation.

If anything goes wrong or you have questions, please join our Discord server for community support:

1.2 Basic Structure

This document outlines the basic design of *EgoVenture*.

EgoVenture makes heavy use of [singletons]([Singletons \(AutoLoad\) — Godot Engine \(stable\) documentation in English](#)). The central singletons are *EgoVenture*, which handles the most basic parts of the game, *Backpack* for inventory handling and *Boombox* for handling audio.

The whole game is made up of multiple Godot *scenes*, that are switched when walking through the environments. Multiple parts of the environments (called *locations*) are bound together using a map scene. To optimize for performance and memory consumption, *EgoVenture* contains a scene caching algorithm that precaches upcoming scenes based on an index in their filenames and also removes unneeded scenes after a few steps.

Various tools like special *hotspots* are provided to ease and streamline game development.

[Signals](#) are used to communicate with the different components.

Various parts that build up the game are created using standard [Godot resources](#). For example inventory items, the game's configuration or the *game state*, a standardized resource for storing variables about the progress in the game. Variables in that state can be used in the scenes to toggle sprites or hotspots.

Most tools use [Godot controls](#) to implement their actions. Also, the main menu is made up using controls and can completely be themed using [Godot themes](#) using *special theme overrides*.

Finally, there should be a core singleton for each game, that handles very basic stuff like initializing the state, and is used for configuring *EgoVenture* and handling signals.

1.2.1 Canvas layers

EgoVenture uses CanvasLayers to stack different views over one another. The bigger This list records the IDs for reference:

Layer	Resource	Description
1	inventory.tscn\$Inventory/Canvas	The inventory bar
90	detail_view.tscn\$DetailView	The inventory item detail view
99	parrot_dialog.tscn\$ParrotDialog	Parrot's dialog character and subtitle view
100	map_notification.tscn\$MapNotification	The overlay that shows up when the map notification is played
125	main_menu.tscn\$MainMenu	The main menu
126	waiting_screen.tscn\$WaitingScreen	The screen shown when filling the cache
127	menu_grab.tscn\$MenuGrab	A tool layer to capture right mouse clicks and show the menu
128	speedy.tscn\$MouseLayer	The custom mouse cursors

1.3 The game state

EgoVenture uses the concept of a *game state*. This state is a standard [Godot resource](#) that extends the included `BaseState` class and contains all variables the game developer needs to store information about the player's progress in the game.

The game state from the *EgoVenture Game Template* looks like this:

game_state.gd

```
# The game state
class_name GameState
extends BaseState

# Add variables about your game here
#export(bool) var met_stina = false
```

The three first lines are basically just setting up the state resource. After that, any standard variable of a [Godot built-in type](#) can be used to store game information. It's important to **export** each variable.

While each built-in type is supported, it is **recommended** to only use very basic types like `bool`, `int`, `float` and `string`.

For details see the [BaseState API-Docs](#).

1.4 Scenes and navigation

1.4.1 Scenes

While the player moves around in the game, each view is made up of a Godot scene that contains *Hotspots* which trigger code that changes the current scene.

This makes designing a game straight forward.

1.4.2 Caching and scene naming

Because changing scenes - especially with large images - takes time, *EgoVenture* includes a scene caching algorithm that preloads upcoming scenes in the background.

To achieve that, the algorithm expects that scene files names contain a numeric index.

Scene filenames need to start with a letter, an underscore or a dash. At some point it needs to include a number. After that number any other character **other than a number** can follow.

Here are some examples:

- `woods1b2.tscn` **invalid**, because two numbers are included
- `Woods12.tscn` **invalid**, because it contains an upper case character
- `woods1secondtree.tscn` **valid**

When the player enters a scene (i.e. `woods12.tscn`) it will do the following:

- Cache scenes `woods13` to `woods15.tscn` and `woods11` to `woods9.tscn` (if they exist)
- Remove scenes `woods8.tscn` and `woods16.tscn` (if they exist)

Note: the amount of scenes the caching algorithm caches, defaults to 3, but can be changed in the [Game configuration](#). Remember, that more scenes require more memory.

1.4.3 Locations

Because not all views can follow one another, *EgoVenture* includes a concept of *locations*. Locations allow scene files to be placed in a subdirectory. Together with `MapHotspots` and foreground caching, they allow seamless transition between ready-cached scenes.

1.4.4 Four-Side-Room scenes

Since most of the scenes that the player walks through, consist of a front, right, left and back view, *EgoVenture* includes a helper scene `four_side_room.tscn`, that can be instantiated in a scene to speed up creating walk-throughs.

After instantiating it in the specific scene, it can be configured which images will be displayed on the four sides.

The four sides will be shown around the center like a cut-open box.

The scene will take care of showing the images and navigating left and right.

Additional hotspots (e.g. for moving forward) can be placed over the four sides.

1.5 Inventory handling

EgoView includes a complete inventory management solution with its `Inventory` singleton. The inventory includes inventory items that can be used on `TriggerHotspots` and on other inventory items.

Each inventory item can be viewed in a detail view, which can optionally show scenes for more advanced usage.

All inventory items are displayed in an inventory bar, that either scrolls down from the top on mouse-based devices and can be toggled on touch devices.

Each item can be selected and show a specific mouse cursor (on mouse-controlled devices).

1.5.1 Inventory items

An inventory item is a Godot resource that describes various details of the item (see the *InventoryItem API doc*)

To create a new item, create a new resource in the *inventory* folder based on `InventoryItem`. In the inspector, configure all required properties.

1.5.2 Handling inventory items

To add a new inventory item somewhere in the game, call the following code:

```
Inventory.add_item(preload("res://inventory/myitem.tres"))
```

This will briefly show the inventory bar, add the inventory item to it and hide it again.

(This can be skipped by adding `false` as a second argument to `add_item`)

The player can click on the item in the inventory bar to select the item so it can be used with other items or hotspots.

It can be deselected in code using:

```
Inventory.release_item()
```

To remove an item (if it is dropped later), call the following code:

```
Inventory.remove_item(preload("res://inventory/myitem.tres"))
```

This will remove the item from the inventory bar (and also deselects it if it is currently selected)

1.5.3 Using inventory items on one another

Inventory item resources include a list of other items that can be used with other items. If an item is included in that list, the mouse cursor will show the active image of the inventory item when hovered over it.

If the player eventually clicks on that item while having selected the other item, the signal `triggered_inventory_item` will be emitted by `Inventory`.

It is recommended, that those signals are caught in the game's core singleton and reacted on like this:

`core.gd`

```
(...)
func _ready():
    (...)
    Inventory.connect("triggered_inventory_item", self, "_on_triggered_inventory_item")

func _on_triggered_inventory_item(first_item: InventoryItem, second_item: InventoryItem):
    match first_item.title:
        "Key":
            if second_item.title == "Box":
                (...)
            ...
```

Note: The key in the example can be used with the box and vice-versa, so the developer needs to take care of both cases.

1.5.4 Using inventory items on hotspots

The easiest way to combine inventory items with hotspots in the game is to use the `TriggerHotspot` nodes. See the *TriggerHotspot documentation* for details.

1.6 Hotspots

EgoVenture includes various clickable areas called *hotspots* to ease, speedup and streamline game development.

1.6.1 Hotspot

`Hotspot` is a swiss-army knife hotspot for various things. It allows to specify a *type*, which defines the mouse cursor when hovering over it and sets the image for the hotspot indicator.

If a `target_scene` is specified, clicking on the hotspot will change the current scene to that and thus makes the hotspot useful for navigating around. Four scenes that utilize the four-side-room scene, a `target_view` in that scene can also be specified.

The `pressed` signal can be used to manually execute code instead of changing a scene (e.g. triggering game actions). For details, refer to the *Hotspot API doc*.

1.6.2 LookHotspot

A `LookHotspot` triggers a *Parrot* dialog when clicked. The dialog resource need to be configured on the hotspot.

The mouse cursor and hotspot indicator is an image for a “look” action.

For details, refer to the *LookHotspot API doc*.

1.6.3 MapHotspot

The `MapHotspot` is specifically used in map scenes. Additionally to the `target_scene` and `target_view` from the `Hotspot`, it also needs a `target_location`.

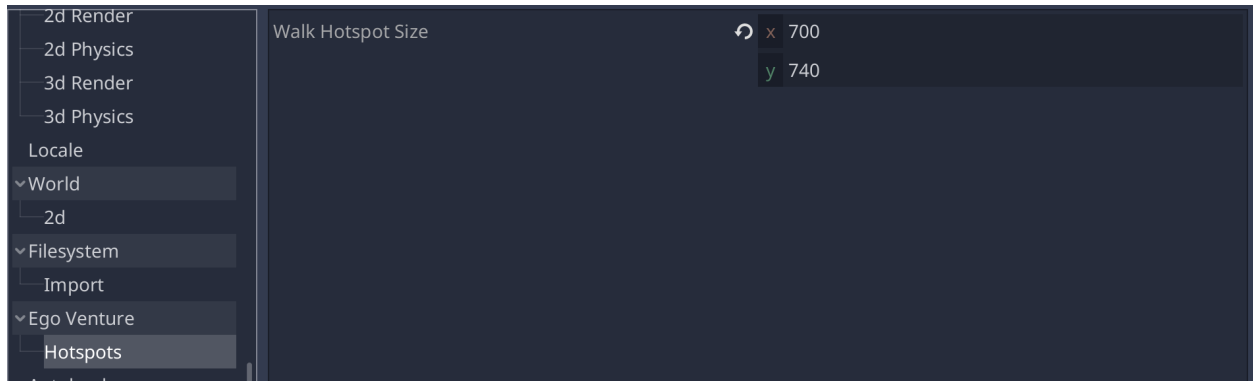
When clicked, it switches the location, starts a location-specific music, caches the scenes around the target scene and shows a loading image.

After the caching is done, the scene is changed.

For details, refer to the *MapHotspot API doc*.

1.6.4 WalkHotspot

The `WalkHotspot` is small utility hotspot that eases the process of additional creating navigation hotspots in the four-view-room scenes (e.g. forward/backward). They share a default size, which can be configured in the project settings under *EgoVenture/Hotspots*:



For details, refer to the [WalkHotspot API doc](#).

1.6.5 TriggerHotspot

The `TriggerHotspot` is used for areas on-screen that can be used with an inventory item. It holds a list of valid inventory items that will react to it by changing the mouse cursor to its active image.

When no inventory item is selected, the cursor for the “Use”-Type is used.

For details, refer to the [TriggerHotspot API doc](#).

1.6.6 Hotspot indicators

EgoVenture games include a feature to show all the hotspots in a scene using a touch button or a key. When the player executes that action, all images defined as hotspot indicators (look above) are displayed.

1.7 Effects, music and background

EgoVenture includes a streamlined way to play sound effects, music and background sounds with its `Boombox` singleton.

For this, *EgoVenture* comes with a predefined `AudioBus layout`, that is modified in the game’s option menu and is used in the resources of the `Boombox`.

Hint: If you’re not using the Egoventure Game Template, make sure to use `addons/egoventure/default_bus_layout.tres` as your default bus layout in the project settings.

See the [Boombox API docs](#) for details.

When using the `Boombox.play_music` function, `Boombox` includes an audio fader that will fade from the currently playing track to the new one.

Of course, when a game is loaded, the music and background sound that was playing in the saved game, is loaded as well.

1.8 Configuration

EgoVenture tries to streamline, standardize and speed-up game development progress and at the same time customize most of the game's aspects.

To be able to easily update to new versions of *EgoVenture* it is **not recommended** to change the sources of the addon itself and rather use the game configuration or *theming*.

If you can't customize a specific aspect, please file an [issue with us](#) and tell us what you need.

See the *Api-Doc of the GameConfiguration* resource type for available configuration options.

1.9 Hint system

EgoVenture includes a hint system called Notebook.

It is designed to be a list of goals with a list of associated hints that are progressed as the player advances in the game. When the player has done something or has been somewhere, a hint can be progressed and the next hint is shown in the notepad. Once all hints are progressed, the next goal is reached.

The hint progression are stored in save games.

1.9.1 The hints file

Goals and hints are stored in a file in CSV format with the extension TXT (due to Godot issue [#38957](#)).

The *EgoVenture Game Template* includes a sample file.

The hints file is referenced in the *game configuration* and loaded when the game is loaded.

1.9.2 Progressing hints

When the player advanced to a certain point in the game that would progress a hint, the following code needs to be called:

```
Notepad.progress_goal(goal_id: int, hint_id: int)
```

So to progress from the first goal, first hint to the second hint of the first goal, call:

```
Notepad.progress_goal(1, 1)
```

If all hints of a goal are progressed, Notepad automatically moves to the next goal.

For details, check out the *Api-Docs*.

1.10 Tools

EgoVenture includes some smaller tools that standardize specific things or just makes them easier.

1.10.1 Map notification

In the Carol Reed series of games, there are certain points where the player learns about a new location on the map. On these occasions an image is blinking and a sound is played.

The image and sound can be set in the *game configuration* and the notification can be triggered by calling:

```
MapNotification.notify()
```

1.11 Theming EgoVenture

1.11.1 Introduction

EgoVenture games typically feature a lot of images and graphics. All aspects of the game look can be customized using Godot's *theming features*.

A starter theme resource called *theme.tres* is already available in the *EgoVenture game template*.

This document describes the available theming settings.

1.11.2 Basic anatomy of the theme

Upon opening *theme.tres* using the file system browser in Godot, the inspector shows the theme like this:

The theme is categorized in different UI element types. Throughout the game, different aspects use different UI elements to achieve the desired result.

Additionally, the game uses Godot's default dialog boxes for confirmation and messages. Because of this, not only special game features are included in the theme.

In this document, each category and setting is described.

1.11.3 Default Font

The default font that is used throughout the game

1.11.4 Button

Subcategory	Setting	Description	Used in
Colors	Font Color	Color of the text on default buttons	Standard Dialogs
	Font Color Disabled	Color of the text when a button is disabled	Standard Dialogs
	Font Color Hover	Color of the text when hovering over default buttons	Standard Dialogs
	Font Color Pressed	Color of the text when pressing default buttons	Standard Dialogs
Fonts	Menu Button	Font for menu buttons	Main menu
	Menu Button Hover	Font for menu buttons when hovered	Main menu
Styles	Focus	Fill style of default buttons	Standard Dialogs
	Hover	Fill style used when hovering over default buttons	Standard Dialogs
	Menu Button Disabled	Fill style of menu buttons when disabled	Main menu
	Menu Button Hover	Fill style of menu buttons when hovered	Main menu
	Menu Button Normal	Fill style of menu buttons	Main menu
	Menu Button Pressed	Fill style of pressed menu buttons	Main menu
	Normal	Fill style of default buttons	Standard Dialogs
	Pressed	Fill style of default buttons	Standard Dialogs

1.11.5 Check Button

Used in the game options for the “Subtitles”-setting

Subcategory	Setting	Description	Used in
Icons	Off	The unchecked state image of the check button	Game options
	On	The checked state image of the check button	Game options
Styles	Focus	Fill style when the check button is focused	Game options
	Hover	Fill style when the check button is hovered	Game options
	Hover Pressed	Fill style when the check button is hovered and checked	Game options
	Normal	Fill style for the check button	Game options
	Pressed	Fill style for the check button is pressed	Game options

1.11.6 H Slider

Used in the game options for the “Volume”-settings

Subcategory	Setting	Description	Used in
Icons	Grabber	The image used for the grabber in the slider	Game options
	Grabber Highlight	The image used for the grabber when the slider is highlighted	Game options
	Tick	The image used for the ticks in the slider	Game options
Styles	Grabber Area	The fill for the grabber area	Game options
	Grabber Area Highlight	The fill for the grabber area when highlighted	Game options
	Slider	The fill for the slider	Game options

1.11.7 Label

Subcategory	Setting	Description	Used in
Colors	Detail View Font Color	The font color of the detail view description	Detail View
	Font Color	The font color of various items (based on labels) like the date labels, the menu titles, etc.	Various places
	Goals	The font color of the goals in the notepad	Notepad
	Hints	The font color of the hints in the notepad	Notepad
Fonts	Detail View	The font used for the description in the detail view	Detail view
	Goals	The font used for the goals in the notepad	Notepad
	Hints	The font used for the hints in the notepad	Notepad

1.11.8 Panel

Subcategory	Setting	Description	Used in
Styles	Detail View	The background used in the inventory item detail view	Detail view
	Dialog Panel	The background used in the Parrot dialogs	Dialogs
	Inventory Panel	The background of the inventory panel	Inventory
	Inventory Panel Touch	The background of the inventory panel for touch devices	Inventory
	Notepad Panel	The background behind the image of the notepad	Notepad
	Saveslot Panel	The design of an empty saveslot	Save Slots

1.11.9 Progress Bar

Subcategory	Setting	Description	Used in
Styles	Bg	The background of the progress bar	Loading screen
	Fg	The foreground fill of the progress bar	Loading screen

1.11.10 Rich Text Label

Subcategory	Setting	Description	Used in
Colors	Dialog Hotspot Asked Font Color	The color of the font when a question of a dialog hotspot was already asked	Dialog Hotspots
	Dialog Hotspot Hover Font Color	The color of the font when a dialog hotspot is hovered	Dialog Hotspots
	Dialog Hotspot Font Color	The normal color of the dialog hotspot font	Dialog Hotspots
Fonts	Dialog Font	Font used in the parrot dialogs	Dialogs
	Dialog Hotspot Bold Font	Bold font used in dialog hotspots	Dialog Hotspots
	Dialog Hotspot Normal Font	Normal font used in dialog hotspots	Dialog Hotspots

1.11.11 Window Dialog

Subcategory	Setting	Description	Used in
Icons	Close	The icon used as a close icon in dialog boxes. Should be set to an empty image as we're not using it	Standard dialogs
	Close Highlight	The icon used as a highlighted close icon in dialog boxes. Should be set to an empty image as we're not using it	Standard dialogs
Styles	Panel	The window panel	

1.12 Updating EgoVenture

The main EgoVenture functionality is distributed as Godot addons. Addons can be simply updated by downloading the new version and overwriting the existing files.

Warning: If you use EgoVenture, you should **never** change files inside the following folders to be able to update to newer versions:

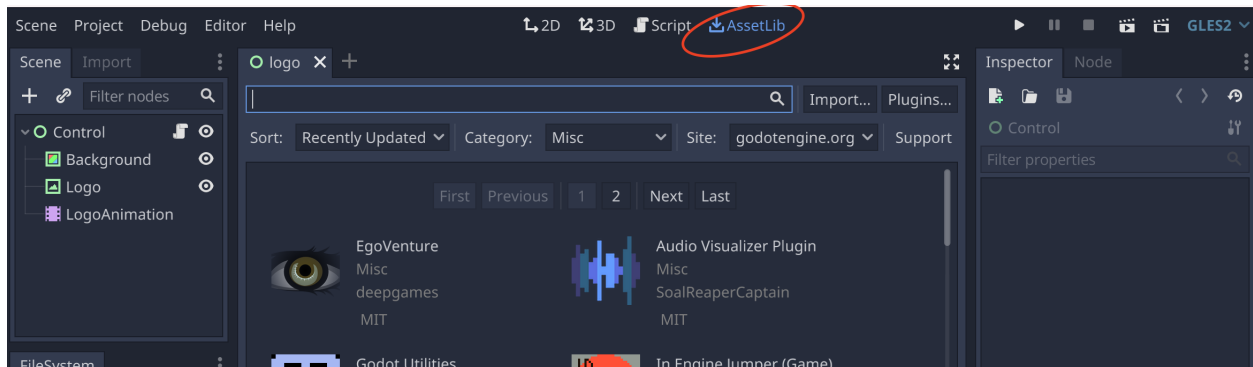
- addons/egoventure
- addons/parrot
- addons/speedy_gonzales

Before you start updating make sure, that you have a *current backup of your game*. Also, *read the release notes* of all intermediate releases from your current version to the version you want to update to. You can find the release notes in the following places:

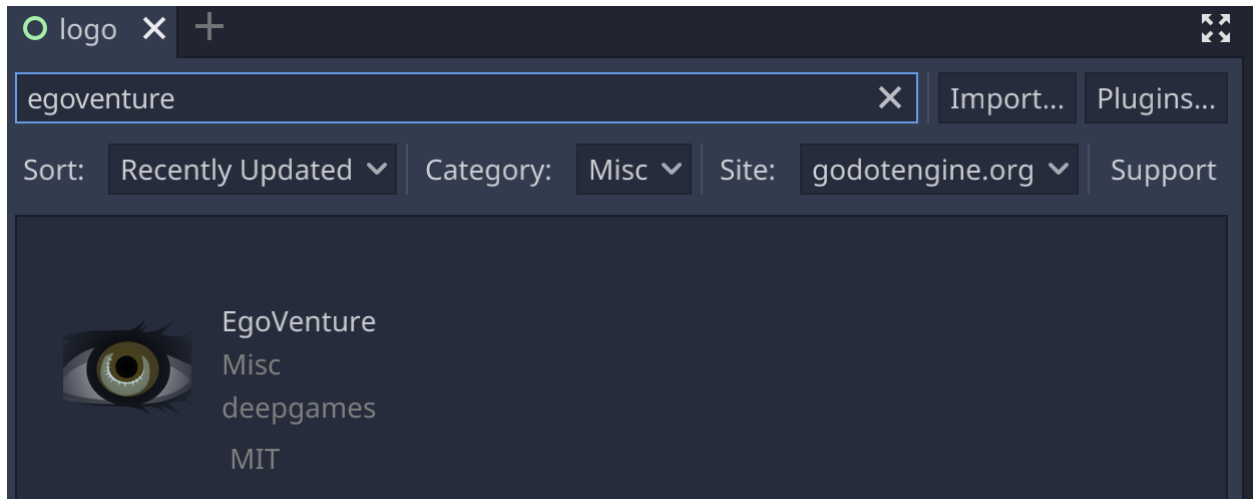
- [EgoVenture](#)
- [Parrot](#)
- [Speedy Gonzales](#)

Especially look for notes regarding dependency updates (e.g. From EgoVenture 0.24.0 on, Parrot 0.7.0 is required)

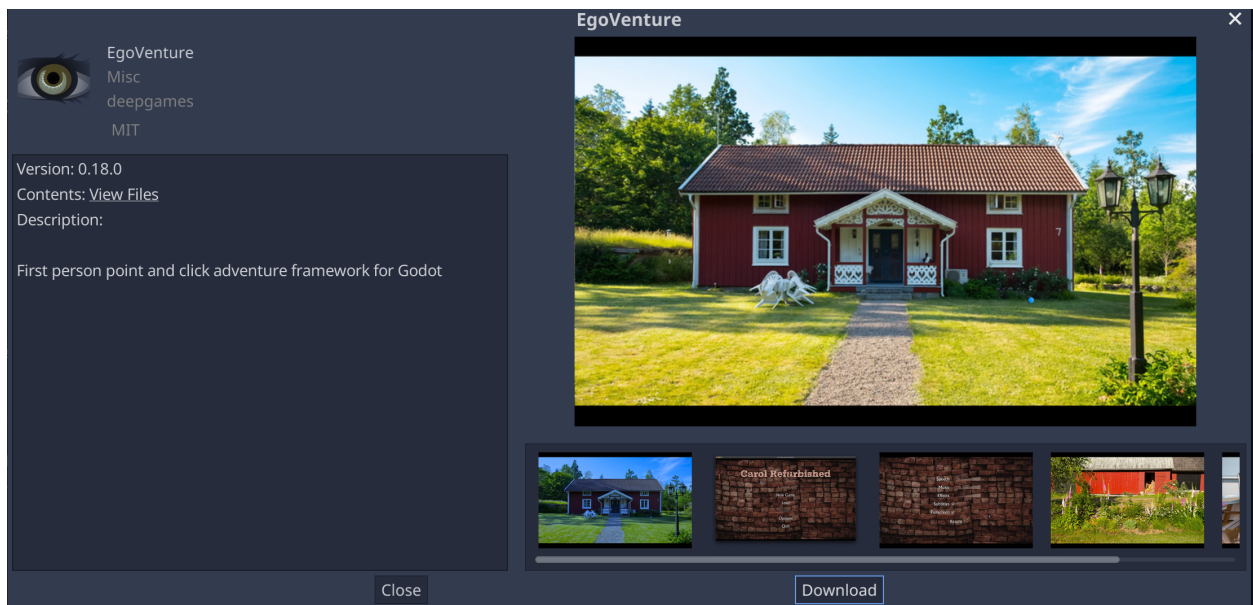
To update, switch to the Godot AssetLib view:



Then, search for the addon you want to update (EgoVenture, Parrot or Speedy Gonzales):



Click on the name of the Addon (EgoVenture in this example) to see some information about it:

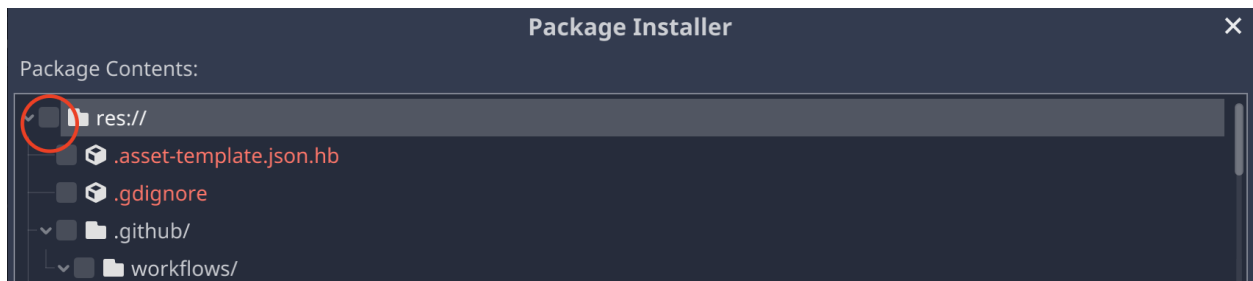


Check, that the given version is the one you'd like to update to (0.18.0 in this example). Click "Download". The addon will be downloaded now. Wait until it's finished:

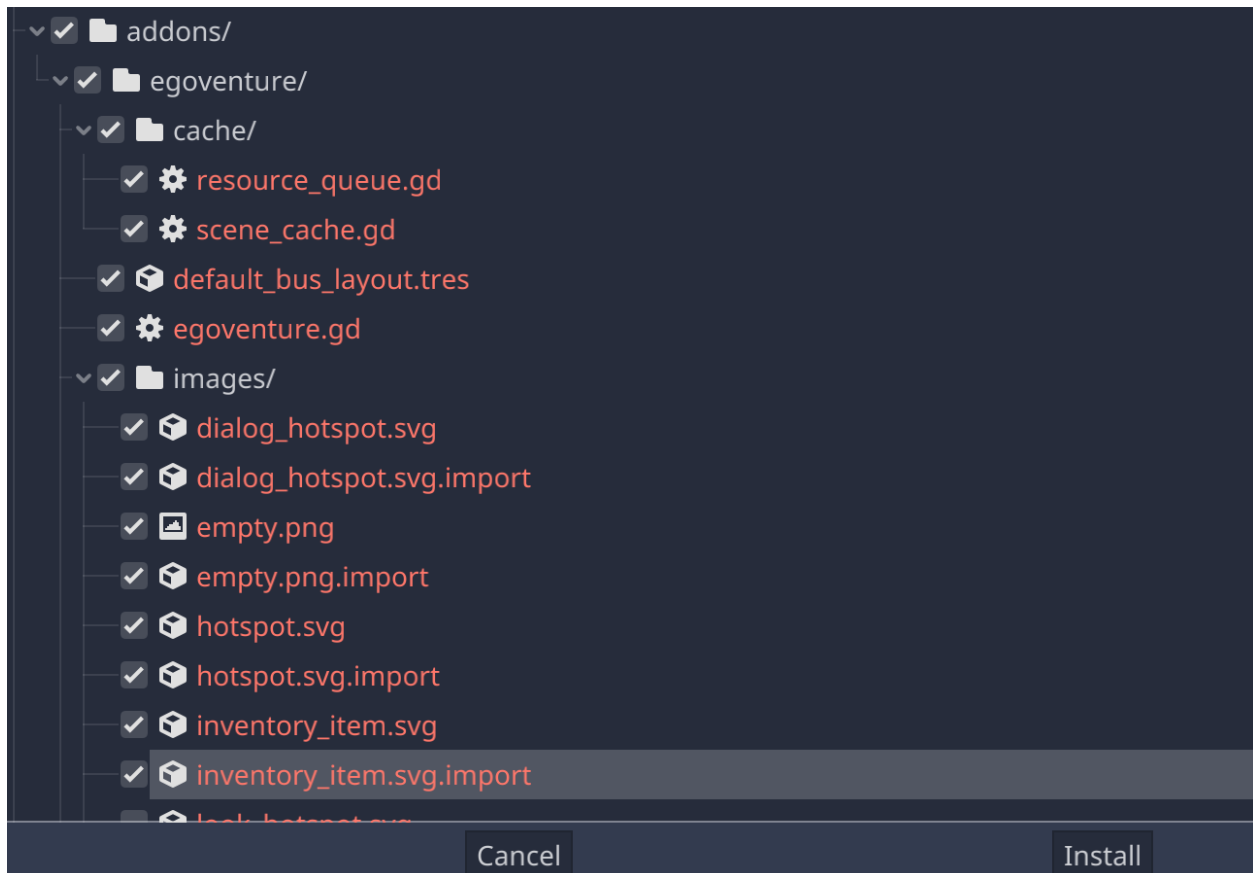


Click "Install..."

The next part is currently very cumbersome. First, remove the check mark at the topmost entry in the package installer:



Then select **every** folder and file inside the Addons' path (addons/egoventure in this example). This is because you the files already exists and Godot requires you to check all files that it should overwrite.



After that, click “Install” and the update will be installed.

1.13 EgoVenture API reference

1.13.1 BaseState

Extends: *Resource*

Description

A base state used as a base class in all game states

Property Descriptions

current_scene

```
export var current_scene: String = ""
```

The path of the currently shown scene

inventory_items

```
export var inventory_items: Array = []
```

Current list of inventory items

target_view

```
export var target_view: String = ""
```

Target view of the stored scene

target_location

```
export var target_location: String = ""
```

Target location of the stored scene

current_music

```
export var current_music: String = ""
```

Path to current music playing

current_background

```
export var current_background: String = ""
```

Path to current background playing

current_goal

```
export var current_goal: int = 1
```

Current notepad goal

goals_fulfilled

```
export var goals_fulfilled: Array = []
```

An array of FulfillmentRecords

overridden_cursors

```
export var overridden_cursors: Dictionary = {}
```

The currently overridden cursors

parrot_skip_enabled

```
export var parrot_skip_enabled: bool = true
```

Whether skipping in Parrot is enabled

1.13.2 Cursor

Extends: *Resource*

Description

The mouse cursor configuration

Property Descriptions

type

```
export var type = 1
```

cursor

```
export var cursor = "[Object:null]"
```

The mouse cursor image

cursor_hotspot

```
export var cursor_hotspot = "(0, 0)"
```

The mouse cursor hotspot

1.13.3 DialogHotspot

Extends: *RichTextLabel*

Description

A button that triggers a dialog

Property Descriptions

dialog

```
export var dialog: String = ""
```

The dialog to play

asked

```
export var asked: bool = false
```

Wether the question was already asked

visibility_state

```
export var visibility_state = ""
```

Show this hotspot depending on the boolean value of this state variable

Signals

- signal pressed(): Emitted when the button is clicked but no dialog is selected

1.13.4 EgoVentureMenuButton

Extends: *Button*

Description

A button for the EgoVenture main menu

Signals

- signal button_pressed(): Signal emitted when button is pressed (adds a sound effect before emitting)

1.13.5 EightSideRoom

Extends: *Node2D*

Description

A scene, that can be instantiated in a scene and features a room with up to eight different sides with automatic view navigation using a Camera2D

Constants Descriptions

TEXTURE_DISTANCE

```
const TEXTURE_DISTANCE: int = 100
```

Distance between textures to allow overlapping hotspot areas

VIEW_BACK

```
const VIEW_BACK: String = "back"
```

VIEW_BACKLEFT

```
const VIEW_BACKLEFT: String = "backleft"
```

VIEW_BACKRIGHT

```
const VIEW_BACKRIGHT: String = "backright"
```

VIEW_FRONT

```
const VIEW_FRONT: String = "front"
```

VIEW_FRONTLEFT

```
const VIEW_FRONTLEFT: String = "frontleft"
```

view constants

VIEW_FRONTRIGHT

```
const VIEW_FRONTRIGHT: String = "frontright"
```

VIEW_LEFT

```
const VIEW_LEFT: String = "left"
```

VIEW_RIGHT

```
const VIEW_RIGHT: String = "right"
```

VIEW_UNSET

```
const VIEW_UNSET: String = ""
```

An unset view

Property Descriptions

view_dict

```
var view_dict
```

Dictionary used to map view constant to index (and back)

default_view

```
export var default_view = "front"
```

The default/starting view of the four views

frontleft_texture

```
export var frontleft_texture = "[Object:null]"
```

The texture for the front view

front_texture

```
export var front_texture = "[Object:null]"
```

frontright_texture

```
export var frontright_texture = "[Object:null]"
```

right_texture

```
export var right_texture = "[Object:null]"
```

backright_texture

```
export var backright_texture = "[Object:null]"
```

back_texture

```
export var back_texture = "[Object:null]"
```

backleft_texture

```
export var backleft_texture = "[Object:null]"
```

left_texture

```
export var left_texture = "[Object:null]"
```

enable_navigation

```
export var enable_navigation = true
```

Whether navigation features are enabled in this room

current_view

```
var current_view
```

The current view shown to the player

Signals

- signal view_changed(old_view, new_view): Triggered when the user switches the view

1.13.6 FourSideRoom

Extends: *Node2D*

Description

A scene, that can be instantiated in a scene and features a room with four different sides with automatic view navigation using a Camera2D

Constants Descriptions

VIEW_BACK

```
const VIEW_BACK: String = "back"
```

The backwards view

VIEW_FRONT

```
const VIEW_FRONT: String = "front"
```

The front view

VIEW_LEFT

```
const VIEW_LEFT: String = "left"
```

The left view

VIEW_RIGHT

```
const VIEW_RIGHT: String = "right"
```

The right view

VIEW_UNSET

```
const VIEW_UNSET: String = ""
```

An unset view

Property Descriptions

default_view

```
export var default_view = "front"
```

The default/starting view of the four views

front_texture

```
export var front_texture = "[Object:null]"
```

The texture for the front view

right_texture

```
export var right_texture = "[Object:null]"
```

The texture for the right view

back_texture

```
export var back_texture = "[Object:null]"
```

The texture for the backwards view

left_texture

```
export var left_texture = "[Object:null]"
```

The texture for the left view

enable_navigation

```
export var enable_navigation = true
```

Whether navigation features are enabled in this room

current_view

```
var current_view
```

The current view shown to the player

Signals

- signal view_changed(old_view, new_view): Triggered when the user switches the view

1.13.7 FulfillmentRecord

Extends: *Resource*

Description

A record whether the player how far the player has fulfilled a certain goal

Property Descriptions

goal_id

```
export var goal_id: int = 0
```

The id of the corresponding goal

fulfilled

```
export var fulfilled: Array = []
```

An array of hints that were fulfilled

1.13.8 GameConfiguration

Extends: *Resource*

Description

The configuration of an MDNA game based on MDNA core

Property Descriptions

design_theme

```
var design_theme: Theme
```

The theme holding all design configurations

design_logo

```
var design_logo: Texture
```

The game's logo

design_cursors

```
var design_cursors: Array
```

Cursors

menu_background

```
var menu_background: Texture
```

The menu background texture

menu_music

```
var menu_music: AudioStream
```

The music playing when the menu is opened

menu_switch_effect

```
var menu_switch_effect: AudioStream
```

A sound effect to play when the something is pressed

menu_button_effect_hover

```
var menu_button_effect_hover: AudioStream
```

A sound effect played when the mouse is over the menu button

menu_button_effect_click

```
var menu_button_effect_click: AudioStream
```

A sound effect played when the a menu button is clicked

menu_item_separation

```
var menu_item_separation: int = 30
```

The main menu item separation

menu_saveslots_background

```
var menu_saveslots_background: Texture
```

The background texture for the save slots

menu_saveslots_previous_image

```
var menu_saveslots_previous_image: Texture
```

The image for the “Previous page” button

menu_saveslots_next_image

```
var menu_saveslots_next_image: Texture
```

The image for the “Next page” button

menu_saveslots_empty_color

```
var menu_saveslots_empty_color: Color = "0,0,0,0.55"
```

The color used for empty save slots

menu_saveslots_free_text

```
var menu_saveslots_free_text: String = "SAVESLOTS_FREE"
```

The text shown under the free save slot

menu_options_background

```
var menu_options_background: Texture
```

The background of the options menu

menu_options_speech_sample

```
var menu_options_speech_sample: AudioStream
```

The sample to play when the speech slider is changed

menu_options_effects_sample

```
var menu_options_effects_sample: AudioStream
```

The sample to play when the effect slider is changed

menu_options_locale_button_modulate

```
var menu_options_locale_button_modulate: Color = "1,1,1,0.2"
```

The modulate color for selected locale flags

menu_options_locale_button_modulate_selected

```
var menu_options_locale_button_modulate_selected: Color = "1,1,1,1"
```

The modulate color for selected locale flags

menu_options_hide_language_selection

```
var menu_options_hide_language_selection: bool = false
```

Hide the language selection from the options menu

menu_quit_confirmation

```
var menu_quit_confirmation: String = "DIALOG_QUIT"
```

The confirmation text for the quit confirmation prompt

menu_overwrite_confirmation

```
var menu_overwrite_confirmation: String = "DIALOG_OVERWRITE"
```

The confirmation text for the overwrite confirmation prompt

menu_restart_confirmation

```
var menu_restart_confirmation: String = "DIALOG_RESTART"
```

The confirmation text for the restart confirmation prompt

inventory_size

```
var inventory_size: int = 92
```

The vertical size of the inventory bar

inventory_texture_menu

```
var inventory_texture_menu: Texture
```

The texture for the menu button (on touch devices)

inventory_texture_notepad

```
var inventory_texture_notepad: Texture
```

The texture for the notepad button

inventory_texture_reveal

```
var inventory_texture_reveal: Texture
```

The texture for the hot spots reveal button (on touch devices)

inventory_texture_left_arrow

```
var inventory_texture_left_arrow: Texture
```

The texture for the left arrow of the inventory bar

inventory_texture_right_arrow

```
var inventory_texture_right_arrow: Texture
```

The texture for the right arrow of the inventory bar

notepad_hints_file

```
var notepad_hints_file: String
```

The path to the hints csv file

notepad_background

```
var notepad_background: Texture
```

The texture in the notepad screen

notepad_goals_rect

```
var notepad_goals_rect: Rect2
```

The notepad goals label rect

notepad_hints_rect

```
var notepad_hints_rect: Rect2
```

The notepad hints label rect

tools_map_image

```
var tools_map_image: Texture
```

The flashing map image

tools_map_sound

```
var tools_map_sound: AudioStream
```

The sound to play when flashing the map

tools_navigation_width

```
var tools_navigation_width: float
```

How wide the left and right navigation areas should be in the four room scene

tools_dialog_stretch_ratio

```
var tools_dialog_stretch_ratio: float = 2
```

The stretch ratio that influences the height of the subtitle panel. The bigger this value, the smaller the subtitle panel.

tools_music_fader_seconds

```
var tools_music_fader_seconds: float = 0.5
```

The number of seconds to fade between the two music channels

tools_background_fader_seconds

```
var tools_background_fader_seconds: float = 0.5
```

The number of seconds to fade between the two background channels

cache_scene_path

```
var cache_scene_path: String = "res://scenes"
```

The path where the scenes are stored

cache_scene_count

```
var cache_scene_count: int = 3
```

Number of scenes to precache before and after the current scene

cache_permanent

```
var cache_permanent: PoolStringArray
```

A list of scenes (as path to the scene files) that are always cached

cache_minimum_wait_seconds

```
var cache_minimum_wait_seconds: int = 4
```

The minimum time to show the loading indicator when precaching

cache_minimum_wait_skippable

```
var cache_minimum_wait_skippable: bool = false
```

Whether the minimum wait time can be skipped by left clicking

1.13.9 Goal

Extends: *Resource*

Description

A goal in the notepad hint system

Property Descriptions

id

```
export var id = 0
```

The numeric id of this goal

title

```
export var title = ""
```

The title of the goal

hints

```
export var hints = []
```

A list of hints

hints_fulfilled

```
export var hints_fulfilled = []
```

The last hint currently visible

1.13.10 Hotspot

Extends: *TextureButton*

Description

A simple hotspot which can also move to a target scene upon pressing

Property Descriptions

cursor_type

```
export var cursor_type = 1
```

The cursor type

target_scene

```
export var target_scene = ""
```

If set, changes to the given scene

target_view

```
export var target_view = ""
```

effect

```
export var effect = "[Object:null]"
```

If set, plays a sound effect when the hotspot is pressed and the scene is changed

visibility_state

```
export var visibility_state = ""
```

Show this hotspot depending on the boolean value of this state variable

show_indicator

```
export var show_indicator = true
```

Whether to show the hotspot indicator or not

Signals

- signal activate(): A signal that can be connected to for custom actions of this hotspot

1.13.11 InGameConfiguration

Extends: *Resource*

Description

A configuration for game-wide elements (like sound settings)

Property Descriptions**subtitles**

```
var subtitles: bool = true
```

Whether subtitles should be shown

fullscreen

```
var fullscreen: bool = true
```

Whether to show the game in full screen

speech_db

```
var speech_db: float = 0
```

The volume of the speech channel in db

music_db

```
var music_db: float = 0
```

The volume of the music channel in db

effects_db

```
var effects_db: float = 0
```

The volume of the effects channel in db

continue_state

```
var continue_state: BaseState
```

The continue state, that is saved automatically

locale

```
var locale: String = ""
```

The selected locale

1.13.12 InventoryItem

Extends: *Resource*

Description

An inventory item

Property Descriptions

title

```
export var title: String = ""
```

The title of the inventory item

description

```
export var description: String = ""
```

A description for the inventory item

image_normal

```
export var image_normal: Texture = "[Object:null]"
```

The image/mouse pointer for the inventory item

image_active

```
export var image_active: Texture = "[Object:null]"
```

The image/mouse pointer for the inventory item if it's selected

image_big

```
export var image_big: Texture = "[Object:null]"
```

The big image used in detail views

combineable_with

```
export var combineable_with: Array = []
```

The items this item can be combined with

detail_scene

```
export var detail_scene: String = ""
```

A scene to load for the detail view instead of the big image

detail_show_mouse

```
export var detail_show_mouse: bool = false
```

Whether to show the mouse cursor in the detail view

grabbable

```
export var grabbable: bool = true
```

Whether the item is selectable and useable on the screen. If set to false, clicking the item with either mouse button will show the (custom) detail view.

1.13.13 InventoryItemNode

Extends: *TextureButton*

Description

The inventory item as a graphic node.

Property Descriptions

item

```
var item: InventoryItem
```

The corresponding resource.

Method Descriptions

configure

```
func configure(p_item: InventoryItem)
```

Configure this item and connect the required signals.

**** Parameters ****

- `p_item`: The `InventoryItem` resource this item is based on.

show_detail

```
func show_detail()
```

Show the detail view.

Signals

- signal triggered_inventory_item(): Emitted, when another inventory item was triggered

1.13.14 LookHotspot

Extends: *TextureButton*

Description

A hotspot that triggers a Parrot dialog for “look” actions

Property Descriptions

dialog

```
export var dialog = ""
```

The dialog resource that should be played by Parrot

visibility_state

```
export var visibility_state = ""
```

Show this hotspot depending on the boolean value of this state variable

1.13.15 MapHotspot

Extends: *Hotspot < TextureButton*

Description

A hotspot supporting precaching of scenes and showing a loading screen and playing a tune while doing so

Property Descriptions

loading_image

```
export var loading_image = "[Object:null]"
```

The loading image to show while the scenes for the new location are cached

location_music

```
export var location_music = "[Object:null]"
```

The music that should be played while loading

location

```
export var location = ""
```

The new location (subdirectory of the scene files)

state_variable

```
export var state_variable = ""
```

Set the boolean value of this variable in the state to true

1.13.16 ResourceQueue

Property Descriptions

thread

```
var thread
```

mutex

```
var mutex
```

sem

```
var sem
```

time_max

```
var time_max
```

Milliseconds.

queue

```
var queue
```

pending

```
var pending
```

Method Descriptions**queue_resource**

```
func queue_resource(path, p_in_front = false)
```

cancel_resource

```
func cancel_resource(path)
```

get_progress

```
func get_progress(path)
```

is_ready

```
func is_ready(path)
```

get_resource

```
func get_resource(path)
```

thread_process

```
func thread_process()
```

thread_func

```
func thread_func(_u)
```

start

```
func start()
```

1.13.17 SceneCache

Extends: *Node*

Description

A rolling cache to preload scenes for a better performance Uses the ResourceQueue as described in the Godot documentation

Method Descriptions

_init

```
func _init(cache_count: int, scene_path: String, scene_regex: String, permanent_cache: PoolStringArray)
```

update_progress

```
func update_progress()
```

Update the current progress on the waiting screen and emit the queue_complete signal when we're done

get_scene

```
func get_scene(path: String) -> PackedScene
```

Retrieve a scene from the cache. If the scene wasn't already cached, this function will wait for it to be cached.

**** Parameters ****

- path: The path to the scene

update_cache

```
func update_cache(current_scene: String) -> int
```

Update the cache. Start caching new scenes and remove scenes, that are not used anymore

**** Parameters ****

- current_scene: The path and filename of the current scene **Returns** Number of cached scenes

Signals

- signal queue_complete(): All pending resources were loaded

1.13.18 TriggerHotspot

Extends: *TextureButton*

Description

A hotspot that reacts to inventory items and features a special mouse cursor when no item is selected

Property Descriptions

visibility_state

```
export var visibility_state: String = ""
```

Show this hotspot depending on the boolean value of this state variable

valid_inventory_items

```
export var valid_inventory_items: Array = []
```

The list of valid inventory items that can be used on this hotspot

show_indicator

```
export var show_indicator = true
```

Whether to show the hotspot indicator or not

Method Descriptions

on_mouse_entered

```
func on_mouse_entered()
```

If an inventory item was selected and it is in the list of valid inventory items, show it as active

Signals

- signal item_used(item): Emitted when a valid item was used on the hotspot

1.13.19 WalkHotspot

Extends: *Hotspot < TextureButton*

Description

A hotspot that can be used for forward/backward hotspots and has a predefined size

1.13.20 boombox.gd

Extends: *Node*

Description

Boombox - a singleton audio player framework

Constants Descriptions

VOLUME_MAX

```
const VOLUME_MAX: int = 0
```

The volume to fade to if the channel should be on

VOLUME_MIN

```
const VOLUME_MIN: int = -80
```

The volume to fade to if the channel should be off

Property Descriptions

ignore_pause

```
var ignore_pause: bool
```

Let Boombox ignore game pausing. So all sound will continue playing when a game is paused

active_music

```
var active_music: AudioStreamPlayer
```

The active music player

active_background

```
var active_background: AudioStreamPlayer
```

The active background player

Method Descriptions

reset

```
func reset()
```

Reset the settings. Stop all music, sounds and backgrounds Used when starting a new game

play_music

```
func play_music(music: AudioStream, from_position: float = 0)
```

Play a new music file, if it isn't the current one.

**** Parameters****

- music: An audiostream of the music to play

pause_music

```
func pause_music()
```

Pause playing music

resume_music

```
func resume_music()
```

Resume playing music

stop_music

```
func stop_music()
```

Stop the currently playing music

get_music

```
func get_music() -> AudioStream
```

Get the current music

is_music_playing

```
func is_music_playing() -> bool
```

Get whether boombox is currently playing music

play_background

```
func play_background(background: AudioStream, from_position: float = 0)
```

Play a background effect

**** Parameters ****

- background: An audiostream of the background noise to play

pause_background

```
func pause_background()
```

Pause playing background effect

resume_background

```
func resume_background()
```

Resume playing background effect

stop_background

```
func stop_background()
```

Stop playing a background effect

get_background

```
func get_background() -> AudioStream
```

Get the current background

is_background_playing

```
func is_background_playing() -> bool
```

Get whether boombox is currently playing background

play_effect

```
func play_effect(effect: AudioStream, from_position: float = 0)
```

Play a sound effect

**** Parameters ****

- effect: An audiostream of the sound effect to play make sure it's set to "loop = false" in the import settings

pause_effect

```
func pause_effect()
```

Pause playing the sound effect

resume_effect

```
func resume_effect()
```

Resume playing the sound effect

stop_effect

```
func stop_effect()
```

Stop playing a sound a effect

Signals

- signal effect_finished(): Emitted when a sound effect completed playing

1.13.21 check_cursor.gd

Extends: *Node*

Description

This is a singleton setting the cursor shape Needed as a workaround as Godot doesn't set the cursor in certain scenarios

Method Descriptions

activate

```
func activate()
```

deactivate

```
func deactivate()
```

1.13.22 cursors.gd

Extends: *Node*

Description

Enumerations

Type

```
const Type: Dictionary = {"CUSTOM1":13,"CUSTOM2":14,"CUSTOM3":15,"CUSTOM4":16,"DEFAULT"
↳":0,"DOWN":6,"EXIT":10,"GO_BACKWARDS":2,"GO_FORWARD":1,"HAND":12,"LOOK":8,"MAP":11,
↳":"SPEAK":9,"TURN_LEFT":4,"TURN_RIGHT":3,"UP":5,"USE":7}
```

The available types of cursors

CURSOR_MAP

```
const CURSOR_MAP: Dictionary = {"0":0,"1":3,"2":6,"3":15,"4":14,"5":10,"6":9,"7":2,"8
↳":16,"9":1,"10":11,"11":12,"12":13,"13":7,"14":8,"15":4,"16":5}
```

A map of cursor types to core input cursors

Method Descriptions

configure

```
func configure(configuration: GameConfiguration)
```

Configure the mouse cursors

override

```
func override(type, texture: Texture, hotspot: Vector2, target_position = null)
```

reset

```
func reset(type)
```

Reset the previously overridden cursor to its default form

**** Parameters ****

- type: The type to reset (based on the Type enum)

get_cursor_texture

```
func get_cursor_texture(type)
```

Return the texture of the specified hotspot type

**** Parameters ****

- type: The type to return the default texture of

Signals

- signal cursors_configured():

1.13.23 detail_view.gd

Extends: *CanvasLayer*

Description

The detail view of an inventory item

Property Descriptions

is_visible

```
var is_visible
```

Whether the DetailView is currently visible

Method Descriptions

show_with_item

```
func show_with_item(item: InventoryItem)
```

Show the item

**** Parameters ****

- item: The inventory item to display

hide

```
func hide()
```

Hide the panel

1.13.24 egoventure.gd

Extends: *Node*

Description

First person point and click adventure framework for Godot

Constants Descriptions**SCENE_REGEX**

```
const SCENE_REGEX: String = "[a-z_-]+(?<index>\\d+)\\D?.*$"
```

A regex to search for the scene index in a scene filename. e.g.: home04b.tscn has the index 4, castle12detail1.tscn has the index 12.

Property Descriptions**state**

```
var state: BaseState
```

The current state of the game

current_scene

```
var current_scene: String = ""
```

Path of the current scene

current_view

```
var current_view: String = ""
```

The current view of the four side room

target_view

```
var target_view: String = ""
```

The target view for the next room

current_location

```
var current_location: String = ""
```

Current location (subfolder in scenes folder)

game_started

```
var game_started: bool = false
```

Whether the game has started. Should be set to true in the first interactive scene

in_game_configuration

```
var in_game_configuration: InGameConfiguration
```

The in game configuration (sound and subtitles)

configuration

```
var configuration: GameConfiguration
```

The game's configuration

saves_exist

```
var saves_exist: bool = false
```

Whether at least one savegame exists

wait_timer

```
var wait_timer: Timer
```

A timer that runs down while a waiting screen is shown

interactive

```
var interactive: bool = true
```

Whether the game currently accepts input

is_touch

```
var is_touch: bool
```

Helper variable if we're on a touch device

Method Descriptions

configure

```
func configure(p_configuration: GameConfiguration)
```

Configure the game from the game's core class

**** Parameters ****

- p_configuration: The game configuration

change_scene

```
func change_scene(path: String)
```

Switch the current scene to the new scene

**** Arguments ****

- path: The absolute path to the new scene

set_parrot_skip_enabled

```
func set_parrot_skip_enabled(value: bool)
```

Set whether dialog line skipping is enabled in parrot

**** Arguments ****

- value: Whether skipping is enabled

save

```
func save(slot: int)
```

Save the current state of the game

**** Arguments ****

- slot: The save slot index

save_continue

```
func save_continue()
```

Save the “continue” slot

save_in_game_configuration

```
func save_in_game_configuration()
```

Save the in game configuration

load

```
func load(slot: int)
```

Load a game from a savefile

**** Arguments ****

-slot: The save slot index to load

load_continue

```
func load_continue()
```

Load the game from the continue state

set_audio_levels

```
func set_audio_levels()
```

Set the audio levels based on the in game configuration

update_cache

```
func update_cache(scene: String = "", blocking = false) -> int
```

Cache scenes for better loading performance

**** Arguments ****

- scene: The scene path and filename. If empty, will be set to the current scene
- blocking: Whether to display a waiting screen while caching

has_continue_state

```
func has_continue_state() -> bool
```

Check if a continue state exists

options_set_subtitles

```
func options_set_subtitles(value: bool)
```

Set the subtitle

**** Arguments ****

- value: Enable or disable subtitles

options_get_subtitles

```
func options_get_subtitles() -> bool
```

Get subtitle

Returns The current subtitle setting

options_set_speech_level

```
func options_set_speech_level(value: float)
```

Set the speech volume

**** Arguments ****

- value: The new value

options_get_speech_level

```
func options_get_speech_level() -> float
```

Return the current speech volume

Returns The current value

options_set_music_level

```
func options_set_music_level(value: float)
```

Set the music volume

**** Arguments ****

- value: The new value

options_get_music_level

```
func options_get_music_level() -> float
```

Return the current music volume

Returns The current value

options_set_effects_level

```
func options_set_effects_level(value: float)
```

Set the effects volume

**** Arguments ****

- value: The new value

options_get_effects_level

```
func options_get_effects_level() -> float
```

Return the current speech volume

Returns The current value

set_full_screen

```
func set_full_screen()
```

Set full screen according to game configuration

reset

```
func reset()
```

Reset the game to the default

wait_screen

```
func wait_screen(time: float)
```

Show a waiting screen for the given time

wait_skipped

```
func wait_skipped()
```

Called when the waiting screen was skipped

reset_continue_state

```
func reset_continue_state()
```

Reset the continue state

Signals

- signal game_loaded(): Emits when the game was loaded
- signal queue_complete(): Emits when the queue of the scene cache has completed
- signal requested_view_change(to): Emitted when a loaded game needs to change the target view but is already in the current scene
- signal waiting_completed(): Emitted when the waiting screen finished loading

1.13.25 inventory.gd

Extends: *Control*

Description

EgoVenture Inventory system

Property Descriptions

selected_item

```
var selected_item: InventoryItemNode
```

The currently selected inventory item or null

activated

```
var activated: bool = false
```

Whether the inventory is currently activated

just_released

```
var just_released: bool = false
```

Whether the inventory item was just released (to prevent other actions to be carried out)

ignore_pause

```
var ignore_pause: bool = false
```

Whether to ignore a game pause

Method Descriptions

configure

```
func configure(configuration: GameConfiguration)
```

Configure the inventory. Should be call by a game core singleton

**** Parameters ****

- configuration: The game configuration

disable

```
func disable()
```

Disable the inventory system

enable

```
func enable()
```

Enable the inventory system

add_item

```
func add_item(item: InventoryItem, skip_show: bool = false, allow_duplicate: bool =  
↳ false, position: int)
```

remove_item

```
func remove_item(item: InventoryItem)
```

Remove item from the inventory

**** Parameters ****

- item: Item to remove from the inventory

release_item

```
func release_item()
```

Release the currently selected item

get_items

```
func get_items() -> Array
```

Returns the current list of inventory items

has_item

```
func has_item(needle: InventoryItem) -> bool
```

Check, whether the player carries a specific item

**** Parameters ****

- needle: item searched for
- returns: true if the player is carrying the item, false if not.

toggle_inventory

```
func toggle_inventory()
```

Show or hide the inventory

Signals

- signal triggered_inventory_item(first_item, second_item): Emitted, when another inventory item was triggered
- signal released_inventory_item(item): Emitted when the player released an item

1.13.26 main_menu.gd

Extends: *CanvasLayer*

Description

The MDNA main menu

Constants Descriptions

AUDIO_MIN

```
const AUDIO_MIN: float = -60
```

Lowest Audio level

MINIMUM_SAMPLE_TIME

```
const MINIMUM_SAMPLE_TIME: float = 2
```

Minimum number of seconds a speech or background sample should be played

Property Descriptions

resumeable

```
var resumeable: bool = true
```

Whether the main menu can be hidden

saveable

```
var saveable: bool = true
```

Whether the a game can be saved

disabled

```
var disabled: bool = false
```

Whether the menu can be displayed at all

Method Descriptions

configure

```
func configure(configuration: GameConfiguration)
```

Configure the menu

**** Parameters ****

- configuration: The game configuration resource

toggle

```
func toggle()
```

Toggle the display of the menu and play the menu music

Signals

- signal new_game(): Emitted when the user wants to start a new game
- signal quit_game(): Emitted when the user wants to quit the game

1.13.27 map_notification.gd

Extends: *CanvasLayer*

Description

A simple sprite flashing a map

Method Descriptions

notify

```
func notify()
```

Run the notification

1.13.28 menu_grab.gd

Extends: *CanvasLayer*

Description

A layer that is supposed to grab right mouse button clicks to show the menu

Method Descriptions

set_top

```
func set_top(top: float)
```

Set the top margin to under the inventory bar

1.13.29 notepad.gd

Extends: *CanvasLayer*

Description

The EgoVenture Notepad hint system

Property Descriptions

goals

```
var goals: Array
```

The map of the goals. Each entry contains a dictionary of Goal objects

Method Descriptions

configure

```
func configure(configuration: GameConfiguration)
```

Configure the notepad and load the hints

finished_step

```
func finished_step(goal_id: int, step: int)
```

A step of a goal was finished, advance the hints and switch to the next goal until a goal with an unfinished step comes along

show

```
func show()
```

Show the notepad

1.13.30 notification.gd

Extends: *CanvasLayer*

Description

A generic notification

Method Descriptions

notify

```
func notify(icon: Texture, sound: AudioStream = null)
```

Show a generic notification with an icon and (optionally) sound

**** Parameters ****

- icon: A texture showing the icon of the notification
- sound: Sound that should be played

1.13.31 plugin.gd

Extends: *EditorPlugin*

Description

First person point and click adventure framework for Godot

1.13.32 waiting_screen.gd

Extends: *CanvasLayer*

Description

A waiting screen shown when the scene cache is updatede

Property Descriptions

is_skippable

```
var is_skippable: bool = false
```

Whether the loading is currently skippable

Method Descriptions

show

```
func show()
```

Show the waiting screen

hide

```
func hide()
```

Hide the waiting screen

is_visible

```
func is_visible()
```

Is the waiting screen visible currently?

set_progress

```
func set_progress(value: float)
```

Update the progress on the screen

**** Parameters ****

- value: The current progress as a percent number

set_image

```
func set_image(image: Texture)
```

Set the waiting image

**** Parameters ****

- image: The image to set

Signals

- signal skipped(): The screen was skipped

DEVELOPMENT

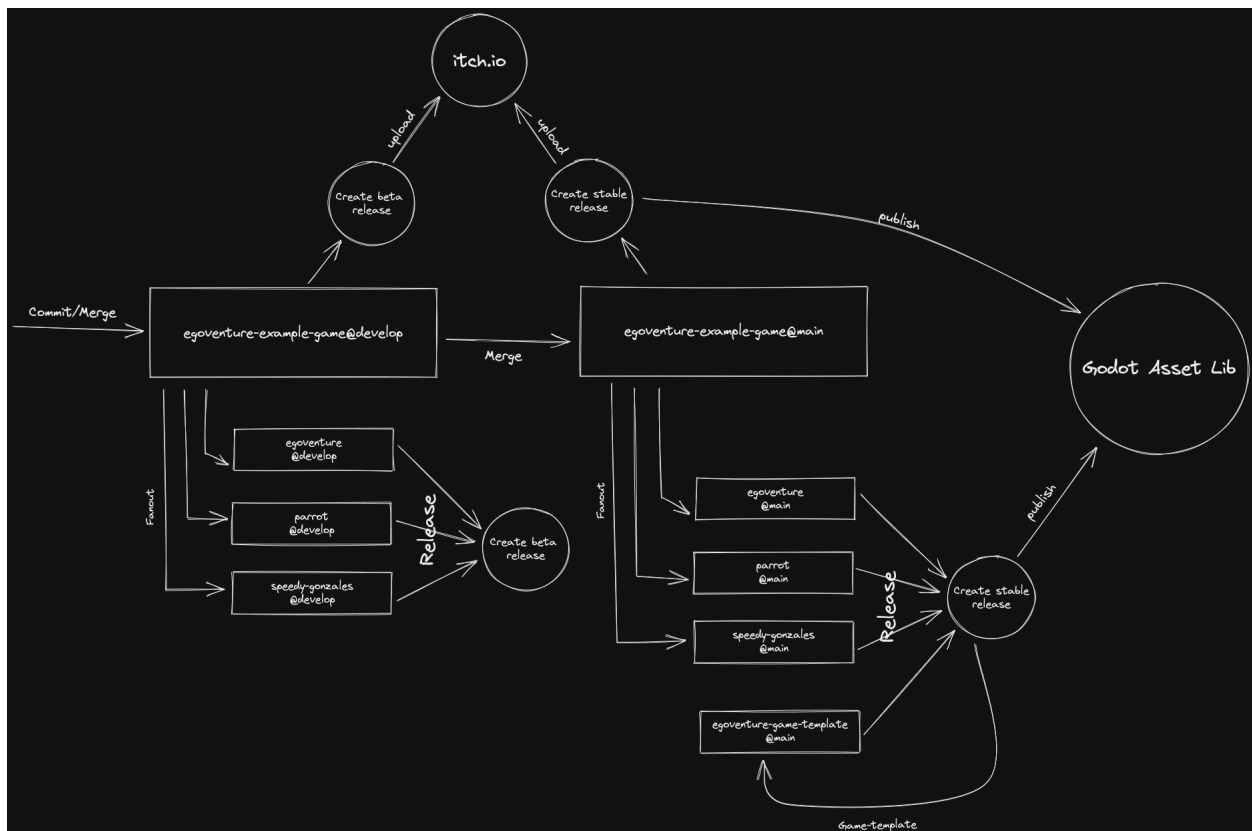
If you find bugs or need more features, please [file an issue within this repository](#).

As this framework is used internally at Mdna Games, we'll have to consider each new feature.

If a new feature doesn't match or contradicts our needs, we might refuse to accept it, but we welcome you to fork this repository and implement it yourself.

You're welcome to open pull requests about bugs or (confirmed) features any time and we'll review it and might ask you for modifications. Thank you for your work!

The release-process can be visualized like this:



All parts of the release process concentrate around the demo game. On commits on `develop` or `main`, the Github actions will fan out changes of the included addons to their own respective repositories. This will trigger a release on those repositories, which in turn will put the updated versions in the game template, which itself will trigger a release of the game template.

Releases in the `main` branch will also release new versions to the Godot Asset Lib.
The game itself will release versions on itch.io for `develop` and `main` changes.

**CHAPTER
THREE**

LOGO

Logo based on “Brown eye” by secretlondon on OpenClipart