GOAT

Release latest

Sep 12, 2023

Getting started

1 Installation 3

2 Main concepts 5

3 Directory structure 9

4 Tutorials 11

5 Player 13

6 Interactive items 15

7 Interactive screens 19

8 Game mode 21

9 Game state 23

10 Interaction 25

11 Inventory 27

12 Context inventory 31

13 Voice 33

14 Settings 37

15 Screenshots 39

16 Credits and licenses 41

Godot Open Adventure Template (GOAT) is a tool for making 3D adventure games. It supports a simple inventory system, interacting with objects and screens, playing voice recordings with subtitles, and changing game settings.

Web demo is available on gotm.io.

CHAPTER 1

Installation

- install Godot Engine 3.2 stable on your machine
- clone GOAT repository and import it as a Godot project
- play the project (F5) to start a demo game: **"2 minute adventure"**
- once you are familiar with how the template works, remove demo game files and create your own adventure!

CHAPTER 2

Main concepts

GOAT is intended to help you make your own adventure games, but there are some assumptions that you need to keep in mind while working with it. This section provides a general overview of the template. More details can be found in other sections of this document.

GOAT supports only 3D, first person perspective, single player games. The player can explore a 3D environment and interact with objects in it. A special type of interaction allows the player to pick up an object and add it to the inventory. The inventory contains 3D items that can be rotated and interacted with. There is a general distinction between the environment (where the player can move) and the inventory (where picked up items are stored). However, both support practically the same types of interaction, e.g. you can press a button on a wall (environment) or on a remote control (inventory).

The template uses several global variables (AutoLoad). One of them is the mode of the game. Currently, 5 modes are implemented:

- `NONE` the game hasn't started yet (default)
- `EXPLORING` the player is moving and interacting with the 3D environment
- `INVENTORY` the player is browsing 3D items in the inventory
- `CONTEXT_INVENTORY` the player is trying to use an inventory item on the environment (e.g. open a door with a key)
- `SETTINGS` the settings screen is shown during the game

Note: the game doesn't pause in any of these modes.

GOAT is a Godot project. The assumption is that if you want to create your own game, you will clone this repository and replace the demo game files with your own. In order to create an interactive environment, you can use 3 main scenes provided by GOAT (all described in detail in other sections):

- `Player`
- `InteractiveItem`
- `InteractiveScreen`

You also need to provide the configuration for:

- inventory items
- voice recordings

Most GOAT resources (voice recordings, interactive objects, inventory items) are identified by unique names. In most cases, getter methods return those names, not the actual resources (e.g. audio files or 3D scenes).

GOAT provides many signals, that let you react to different situations in the game. For example, you can decide to play a voice recording after a button on a wall is pressed or remove an inventory item after the player uses it.

GOAT provides also default styles and layouts for the inventory, context inventory, subtitles, and settings. Usually, there is no reason to change anything in the template files, unless you want to modify the style or add new features (which you are encouraged to do, this is open source after all!).

CHAPTER 3

Directory structure

There are 2 directories in this repository:

- `goat`: contains all scenes and resources of the template; normally there is no need to change anything here, unless you want to implement features that the template doesn't support yet
- `demo`: contains actual game resources, such as sounds, voice recordings, 3D models, materials etc.

The structure of the `demo` folder looks like this:

```
demo
├── fonts              # All fonts used in the game
├── globals            # All global scripts, configured as Singletons (AutoLoad)
├── images             # All images, e.g. a custom cursor
├── inventory_items    # REQUIRED; configuration for inventory items
│   ├── icons          # Icons representing inventory items (one per item)
│   └── models         # 3D models representing inventory items (one per item)
├── materials          # Materials for 3D models
├── models             # 3D models (usually reused in more than one scene)
├── pickable_items     # 3D scenes representing environment objects that can be picked␣
↪up
├── scenes             # Most of the scenes used in the game
│   ├── main           # Scenes responsible for main game screens (main menu, credits,␣
↪settings, gameplay)
│   └── other          # All remaining scenes
├── sounds             # All sounds used in the game, except for voice recordings
└── voice              # REQUIRED; all voice recordings
```

When you start working on your project, you should replace the `demo` folder with the resources of your own game. You can modify the structure in the listing above, but you need to keep the folders marked as `REQUIRED`.

To use a custom game resources directory, you need to configure it first, probably in one of your game's global files:

```
goat.GAME_RESOURCES_DIRECTORY = "demo"
```

As mentioned before, GOAT provides a default settings screen. The "Exit" button on that screen exit the program by default. If you want it to redirect to e.g. the main menu of your game, you can change it like this:

```
goat.EXIT_SCENE = "res://demo/scenes/main/MainMenu.tscn"
```

CHAPTER 4

Tutorials

This talk from GodotCon 2020 in Brussels explains how to get started with GOAT:

More tutorials will be available soon.

CHAPTER 5

Player

GOAT features `Player.tscn` scene. Adding it to your project enables a 3D rotating camera (first person perspective) that can move (currently only on flat surfaces). Movement can be performed using arrow keys or WSAD keys. Player is surrounded with a collision shape, so it will interact properly with obstacles in your game. It also includes a raycast used for selecting 3D environment objects.

All GOAT interface elements are already attached to the player. That way you only need to instance `Player.tscn`, and you will automatically get an inventory, context inventory, subtitles for voice recordings, and a simple settings screen.

CHAPTER 6

Interactive items

Simple interaction can be added to your program by using `InteractiveItem.tscn` scene. It represents an object that can be activated, for example a switch on a wall or a button on a keyboard.

There are 3 types of iteractive items:

- `NORMAL`: can be activated multiple times, e.g. a door that opens and closes
- `SINGLE_USE`: can be activated only once, e.g. a rock pushed from a cliff
- `INVENTORY`: activating it removes the interactive item and adds an inventory item

Interactive item contains a collision shape that is detected by raycasts. A default collision shape is provided, but you can easily change it. You can also configure sounds that will be played randomly when the item is activated.

Note: interactive items don't have any visible elements by default. If you want to associate them with a 3D model, add a mesh as interactive item's child. You can also use models added elsewhere, but it will not work correctly with `INVENTORY` items (which should be removed after activation).

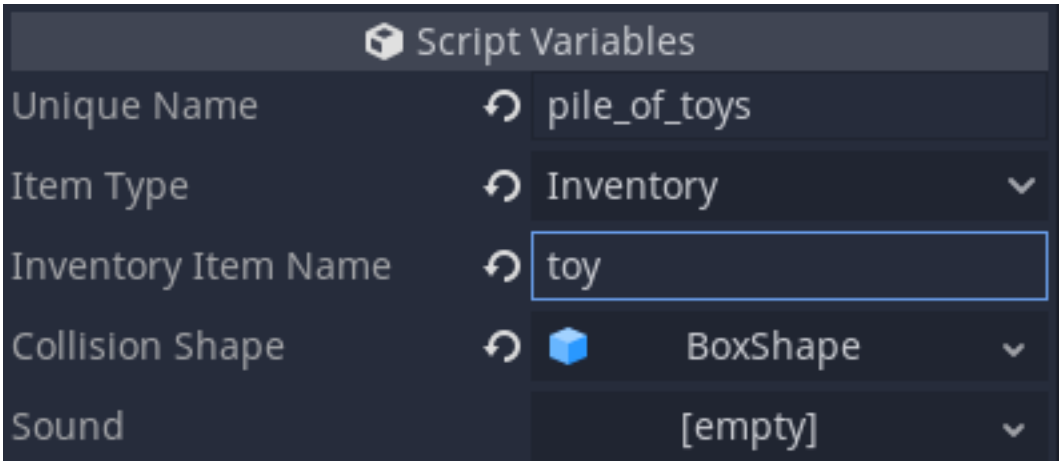
When an interactive item is in range (that is: close enough and in front of the camera or under the mouse cursor) it will be selected. This state is indicated by an interaction icon:

When an item is selected, LMB click will activate it. RMB click will call an alternative activation, which for `INVENTORY` type items works like normal activation (the item will be picked up), and for other items shows the context inventory screen.

Interactive items work both in the environment and in the inventory, however, in inventory mode alternative activation is not supported.

Each interactive item should have a unique name, which will be used to send signals (more about it later). Moreover,

for `INVENTORY` items you need to configure the name of the inventory item that will be added after interactive item is activated (inventory items configuration will be explained later).



CHAPTER 7

Interactive screens

A different type of interaction is offered by `InteractiveScreen.tscn` scene. It represents a flat surface that the player can click on, such as a touch screen of a smartphone. Any type of 2D content can be used with interactive screens, including movie players and minigames.

There are some things to keep in mind while working with interactive screens:

- each screen needs a child node called `Content`, which represents a 2D content displayed on the screen
- content size has to be specified
- the screen's surface is a square mesh, but it can be resized to achieve other aspect ratios

When a screen is in range, an iteraction icon will be shown, and it will follow the movement of the raycast (either the camera or the mouse cursor):

While doing that, the `Content` node will receive `InputEventMouseMotion` events. When the screen is selected, LMB click will activate it. As a result, the `Content` node will receive two `InputEventMouseButton` events, one for pressing the mouse button, and one for releasing it. RMB click will show the context inventory screen (but only for environment screens).

Similar to interactive items, interactive screens should have a unique name. They also work both in the environment and in the inventory. Additionally, you can set the emission energy, which can be useful for computer screens.

CHAPTER 8

Game mode

GOAT stores the current mode of the game in a global `goat.game_mode` variable. There are 5 modes available at the moment:

- `NONE`
- `EXPLORING`
- `INVENTORY`
- `CONTEXT_INVENTORY`
- `SETTINGS`

Each game mode has a corresponding screen, e.g. `CONTEXT_INVENTORY` shows a compact view of all available inventory items and allows the player to choose one of them and use it on the currently selected environment object. The `NONE` mode should be used for e.g. main menu of the game.

You can change the game mode at any time like this:

```
goat.game_mode = goat.GameMode.EXPLORING
```

This will send a signal: `goat.game_mode_changed` with the new value of the game mode. Each of the screens provided by GOAT is connected to this signal and when its own mode is chosen, the screen shows up (otherwise the screen is hidden). You can safely change the game mode's value in order to force showing a specific screen (e.g. when an important item is obtained, you might want to open the inventory to indicate that the player should interact with it).

Note: game mode is only used during gameplay, it should not affect e.g. the main menu of the game. Also, the game doesn't pause in any of these modes.

CHAPTER 9

Game state

GOAT offers a simple way of storing the state of the game. The state consists of variables, and each variable has a name and a value. To add a variable to the game state, use the `register_variable` method:

```
goat_state.register_variable(variable_name, initial_value)
```

This will create a new variable and set its initial value. A variable cannot be used if it was not registered first. The variable can be acccessed later like this:

```
goat_state.get_value(variable_name)
```

It is also possible to change its value:

```
goat_state.set_value(variable_name, value)
```

This will emit a signal: `changed (variable_name, from_value, to value)`, which can be used to react to game state changes in different scenes.

Before each new game, the state should be reset:

```
goat_state.reset()
```

This will set the initial values to all registered variables.

CHAPTER 10

Interaction

Interaction with objects is handled in the following way:

- when an object is in range, it is selected
- when an object is no longer in range, it is deselected
- when an object is selected and LMB is pressed, the object is activated
- when an object is selected and RMB is pressed, the object is activated alternatively

Every change of state emits a signal:

```
goat_interaction.object_selected (object_name, point)
goat_interaction.object_deselected (object_name)
goat_interaction.object_activated (object_name, point)
goat_interaction.object_activated_alternatively (object_name, point)
```

`object_name` represents the unique name associated with an interactive object that is currently being selected/deselected/activated (either an item or a screen). Additionally, `point` stores a global 3D location where an interaction took place (that usually doesn't matter for items, but is quite important for screens).

To make things easier, currently selected objects and their corresponding points are stored globally and can be accessed like this:

```
goat_interaction.get_selected_object(category)
goat_interaction.get_selected_point(category)
```

`category` represents the type of interaction object. Currently, only two values are used: `environment` and `inventory`. Each category can store only one item and one point at the same time. That means that if a new object is selected, the previous one will be deselected first.

`goat_inventory` contains methods for selecting/deselecting/activating interactive objects, but they are used internally by GOAT and usually don't need to be used directly by game developers.

CHAPTER 11

Inventory

GOAT offers an inventory system. It holds a maximum of 8 items, each represented by a 3D model that can be rotated and interacted with. Items can be added either by your script or automatically when an `INVENTORY` interactive item is activated. You can also use items on each other (drag and drop), on themselves ("Use" button), or on environment (using the custom inventory, explained in the next section).

You can open the inventory by pressing Tab while in `EXPLORING` mode. In inventory, you can rotate items by holding RMB and moving the mouse. You can also interact with items by clicking on them with LMB (that is, if they contain any interactive parts). Clicking on a button on the left side selects a different item.

To use the inventory, you need to register an inventory item first:

```
goat_inventory.register_item(item_name)
```

Inventory items must be stored in the `inventory_items` directory of your game resources folder. You need to provide both a 2D icon (shown in the inventory bar), and a 3D model. `item_name` must match both files (without extension, and following Godot's naming convention for scenes). For example, a `floppy_disk` item would need the following files:

```
demo/inventory_items/
├── icons
│   └── floppy_disk.png
└── models
    ├── FloppyDisk.gd # This is not obligatory
    └── FloppyDisk.tscn
```

And the registration would look like this:

```
goat_inventory.register_item("floppy_disk")
```

Icons have to be 64x64 PNG images, and should have a transparent background. Models can be any Spatial scenes, and can have additional logic included in a script. Models should be centered in (0, 0, 0) and should not be larger than a sphere with a 1 unit radius (otherwise they will not be displayed properly).

Item registration process stores the names of icons and models, but nothing more. At some point you might want to actually load those resources:

```
goat_inventory.get_item_icon(item_name)
goat_inventory.get_item_model(item_name)
```

Icons are returned as a `StreamTexture`, models as a `PackedScene` (which has to be instanced). Both of these methods are used internally by GOAT and usually don't have to be accessed directly by game developers.

You can add, remove, and replace items in the inventory using the following methods:

```
goat_inventory.add_item(item_name)
goat_inventory.remove_item(item_name)
goat_inventory.replace_item(replaced_item_name, replacing_item_name)
```

You can also use items, but that part is handled automatically by GOAT and there is usually no reason to use it directly:

```
goat_inventory.use_item(item_name, used_on_name=null)
```

`item_name` is an inventory item that you want to use on something else. `used_on_name` can represent another inventory item, an interactive item or screen (property `unique_name`) or it can be null, meaning that you use the item on itself or on the player (e.g. use a pizza slice = eat it, use a lamp = turn it on).

GOAT also allows you to select one of the inventory items. That item will be shown in the 3D inventory. If the inventory is empty, then no item is selected.

This feature is usually managed by GOAT, but sometimes you might want to select a specific item to bring more attention to it. You can do it like this:

```
goat_inventory.select_item(item_name)
```

`item_name` can be null, which means that you deselect the current item.

Each action mentioned above emits a signal:

```
goat_inventory.item_added (item_name)
goat_inventory.item_removed (item_name)
goat_inventory.item_replaced (replaced_item_name, replacing_item_name)
goat_inventory.item_used (item_name, used_on_name)
goat_inventory.item_selected (item_name)
```

Moreover, every time the content of the inventory changes, this signal will be emitted:

```
goat_inventory.items_changed (new_items)
```

`new_items` is the current content of the inventory (as `Array`), in order the items were added to it.

You can also access the list of items or the selected item at any time:

```
goat_inventory.get_items()
goat_inventory.get_selected_item()
```

If you want to clean the content of the inventory (but keep the inventory item configuration created by `register_item` method) you can do it like this:

```
goat_inventory.reset()
```

CHAPTER 12

Context inventory

Sometimes you might want to use an inventory item (e.g. a key) on an object in the 3D environment (e.g. a door). You can do it by using the context inventory. First, you need to approach an interactive item or screen (until you see the interaction icon). Then, you need to click RMB. A simple layout of all currently available inventory items will be shown in the middle of the screen:

Selecting an item will emit a signal:

```
goat_inventory.item_used(item_name, used_on_name)
```

Where `item_name` is the name of the selected inventory item and `used_on_name` is the `unique_name` property of the environment object.

Currently, there are no signals or methods associated with the context inventory.

CHAPTER 13

Voice

Very often in an adventure game you want your protagonist to say something, e.g. to comment a specific behavior or event. GOAT makes that process easier for you.

First thing you need to do is to register a voice sample:

```
goat_voice.register(audio_file_name, transcript, time = 0, audio_name = null)
```

All voice samples must be stored in the `voice` directory of your game resources folder. By default the name of the registered audio will be the same as the name of the audio file (`audio_file_name`) without the extension (e.g. `but_why.ogg` will be registered as `but_why`). The name can also be set manually by using the `audio_name` attribute. `transcript` is the text included in the voice recording (in this case, probably, "But why?"). It will be shown at the bottom of the screen when the recording is played. The transcript is fully translatable (you just need to add the translation to your Godot project).

There is an additional parameter to this method: `time`. If you specify a value greater than 0, the `register` method will not attempt to load the audio file. Instead, it will register the transcript and the duration it should be "played" (for the purpose of showing the subtitles). This feature is useful if you don't want to or cannot provide an audio sample, but still want to see the subtitles.

You can play a registered voice sample at any time:

```
goat_voice.play(audio_names)
```

`audio_names` can be either a single name or an array of correct audio names (in which case a random one will be played). When the audio is playing, input is blocked in `EXPLORING` and `INVENTORY` game modes. Pressing any of the `goat_dismiss` actions interrupts current audio.

To stop the current voice recording, call:

```
goat_voice.stop()
```

You can also play a sequence of sounds:

```
goat_voice.play_sequence(audio_names)
```

There are 2 signals associated with voice recordings:

```
goat_voice.started (audio_name)
goat_voice.finished (audio_name)
```

If a voice recording is already playing while a new one starts, the old one will be stopped first and `goat_voice.finished` signal will be emitted. Otherwise the signal will be emitted after the recording is played fully.

Transcript for any recording can be obtained this way:

```
goat_voice.get_transcript(audio_name)
```

13.1 Default voice recordings

In GOAT you can set a group of default voice recordings, that will be played when certain actions happen. For example, you can configure lines like "But why?" or "What for?" to be played when the player tries to use items in a way that doesn't make sense.

You can set default voice recordings like this (remember to register them first):

```
goat_voice.set_default_audio_names([audio_name1, audio_name2, ...])
```

You can also play a random default recording later:

```
goat_voice.play_default()
```

You might want to play a random recording every time a meaningless action happens, e.g. when the player uses an item that is not supposed to be used. You can automate this process by connecting default recordings to signals:

```
goat_voice.connect_default(object, signal_name)
```

This way a random recording will be played when a signal is emitted by an object. For example, to play a default recording every time the player activates an object, you can use this code:

```
goat_voice.connect_default(goat_interaction, "object_activated")
```

However, if the player performs a meaningful action, you probably want do play a specific voice recording and ignore the default ones. In that case, you can deactivate the default recordings (just for one action) like this:

```
goat_voice.prevent_default()
```

CHAPTER 14

Settings

GOAT provides a default settings screen, available when the player presses Esc while in `EXPLORING` mode. Currently, only the most important settings are supported. The settings file (INI-style) is saved in `user://settings.cfg`. When the program runs for the first time, the file is created with the default values.

Section	Key	Default value
graphics	fullscreen_enabled	true
graphics	glow_enabled	true
graphics	reflections_enabled	true
graphics	shadows_enabled	true
graphics	ao_enabled	true
sound	music_volume	1.0
sound	effects_volume	1.0
controls	mouse_sensitivity	0.3
gui	subtitles	true
gui	scope	true
gui	language	first matched

Note: the volume is a value between 0 (complete silence) and 1 (default bus volume), which is recalculated to a non-linear value between -80 and 0 dB (using a linear value causes the sound to almost vanish long before the volume slider reaches minimum). Mouse sensitivity is used for player's camera rotation and inventory item rotation.

The default value of the `language` setting is determined based on provided translations and currently selected locale. Please check the `goat_settings.find_matching_loaded_locale` method for more details about the process. The matching happens only once, when the game runs for the first time and the `language` is not yet set. After that, the value is saved with the rest of the settings and can be changed manually to any language with provided translations.

In order to react to changes in settings, you can connect to a signal:

```
goat_settings.value_changed (section, key)
```

Methods for reading and modifying the values are also available:

```
goat_settings.get_value(section, key)
goat_settings.set_value(section, key, value)
```

By default, any change will emit a signal and save the settings file. You can prevent the file saving:

```
goat_settings.autosave = false
```

CHAPTER 15

Screenshots

GOAT offers a simple feature of taking screenshots. The default key binding is `P`. Screenshots are stored in user's local directory (`user://`), in a subdirectory called `Screenshots`. The subdirectory's name can easily be changed:

```
goat.SCREENSHOT_DIRECTORY = "Your Custom Screenshot Directory"
```

CHAPTER 16

Credits and licenses

16.1 Game engine

This project uses Godot Engine version 3.2 stable, distributed under MIT license.

16.2 Plugins

GOAT uses an addon called Random Audio Stream Player, which can be downloaded from Godot Asset Library. It was created by Tim Krief and is licensed under MIT license.

16.3 Models

Pizza slice and plate models were created by Quaternius and are available in the Ultimate Food Pack distributed under CC0 1.0.

Floppy disk, remote, portal, and door models were created by Paweł Fertyk and are distributed under CC0 1.0.

The remaining models were created by Dalton5000 and can be found in this repository distributed under CC BY-NC 4.0.

16.4 Images

Item interaction icon was created using this image, distributed under CC0 1.0.

Screen interaction icon was created using this image, distributed under CC0 1.0.

Reset rotation icon was created using this image, distributed under CC0 1.0.

16.5 Voice

All voice samples were recorded by VoiceOverAngela and are now in public domain.

16.6 Sounds

All sounds used in the project are in public domain and can be downloaded from Freesound:

- battery_on_remote.ogg
- click01.ogg
- click02.ogg
- click03.ogg
- click04.ogg
- generator.ogg
- pick_up.ogg
- shutter.ogg
- the_other_side.ogg
- tray.ogg

16.7 Fonts

The `Tuffy.ttf` font was downloaded from Public Domain Files and is in public domain.