Developer documentation
Project: Space-run

Adilović Una

Table of contents

1. How to run the program?

1.1. Installation requirements

- Godot Engine v3.2.3.stable

(Note: this is **not** the latest available stable version)

1.2. Setup

1. From https://godotengine.org install Godot Engine v3.2.3.stable
2. From https://github.com/AdilovicUna/Space-run clone the repository
3. Open Godot and click the import button
4. Navigate to the "./Space-run/Game" and import “project.godot”.
5. Open the project

2. General information

2.1. Operating Systems and Programming Language

Godot itself is a multiplatform engine that supports the following operating systems: Windows, macOS, Linux, and *BSD. Consequently, this game can be compiled and run on any of those.

Similarly, an executable can be made for any system listed in Project -> Export -> Add.

However, it is worth mentioning that the project itself will not run on the latest stable version of Godot (3.4.3). In this environment, it shows unexpected behaviors which make the game inadequate for use.

Whilst Godot supports other languages, amongst which is C#, this game uses GDScript as the main and only programming language. The advantages of GDScript are mostly highlighted by the fact that it was made specifically for this engine. That being so, it is easy to learn and use for someone with not much experience with game development. In addition, it does not require an external editor, but rather, everything is on-hand for the user.

Of course, C# would provide the programmer with better performance. Nevertheless, the fact that there aren't as many learning resources as with GDScript, makes it much more difficult to use.

2.2. Performance

The average number of frames on a computer with an AMD Ryzen 7 processor is 35fps. Of course, this isn't a fixed number and it will variate during the gameplay.

With this speed, the game runs fairly smoothly and without any major delays.

To achieve better performance, the project was constructed in a way that everything will be either reused or dynamically created and later on deleted from the game.
To be specific, tunnels, ground, and the lights are moved forward every time the main character finishes passing through one of the levels, seemingly making the game infinite.
On the other hand, all of the objects inside the tunnels are created dynamically (objects for the 2nd tunnel are made while the player is in the 1st, …) and likewise deleted once they get out of the viewport of the camera.

2.3. Sound

All of the sounds and music used in the project were downloaded from the following websites:

- https://kenney.nl/assets
- https://mixkit.co/free-sound-effects
- https://www.free-stock-music.com
- https://www.chosic.com/free-music

2.4. Plugins

Since some of the more complex objects in the game required very precise collision shapes, the following plugin was used to generate them:

https://godotengine.org/asset-library/asset/563

This was true only for the Trap type objects (not all of them), and other obstacles such as Bugs or Viruses have the primitive collision shapes already available in Godot.

3. Project structure[1]

The repository consists of two main parts:

- BlenderCharacters
- Game

The BlenderCharacters folder consists of all the character blends used in the game (everything is already imported in project.godot).

The Game folder contains all of the components of the game (textures, meshes, scenes, etc.). The ones we want to focus on are:

- Scenes
- Scripts

3.1. Scenes

Like any other project in Godot engine, this one is also split into scenes. Every scene represents a certain object used in the game and everything is tied up in Main.tscn[2]. Furthermore, structure of a single scene is represented as a tree with a single root node.

3.1.1. Main scene

In the figure 3.1.0. we can see that the Main.tscn consists of root node Main with 6 children. So let us take a closer look into them:

- Green nodes contain the whole UI of the game (battery, score, all the layers of the Help menu showed at the beginning, final score and messages showed at the end, pause button, etc.)
- The ground node is simply a platform that Hans[3] walks on.
- Sound node contains the background music for the help menu and overall gameplay, as well as the sound that plays after the player has lost.
- Hans node contains an instance of scene Hans.tscn
- Tunnels node contains all 3 levels of the game each of which is a parent of a torus node. Those torus nodes are the physical representations of the tunnels (lights are accordingly attached to them).

[1] Before reading this, it is advised to get yourself familiar with the game either by playing it or by reading the Game_specifications.pdf.
[2] .tscn is a filename extension used to represent a scene
[3] The main character of the game

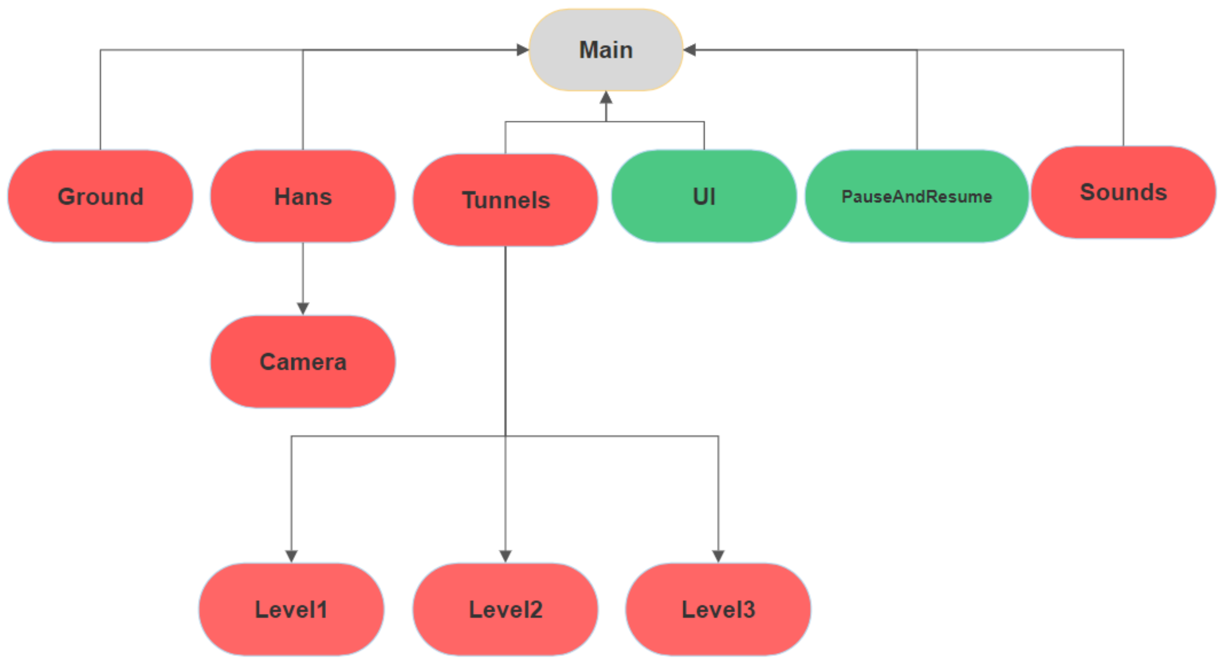

Figure 2.1.0.

Now that we understand the main structure of the game, we can take a look at other scenes.[4]

3.1.2. Character scenes

All of these scenes have a few things in common. They all contain:

- a kinematic body node
- an instances of a .glb file exported from Blender[5]
- animations needed to represent their movements
- appropriate collision shapes

Character scenes can be divided into 3 categories:

1. Hans:
    - The main character
    - In the whole game, only one instance of this scene exists
    - Contains sounds such as shooting, squishing a bug, killing something, etc.
    - Has a muzzle from which bullets come from
    - Has timer that counts down sickness time
    - Has a timer that controlls the shooting speed

[4] Note: all of the scenes that are not instanced in Main.tscn are (mostly) randomly generated and accordingly deleted as the game progresses.
[5] Version 2.92 of Blender was used to create them

2. Bugs
    - Flying and a walking ladybug, and a worm included
    - Are added as instances of level2 tunnel
    - Move with the tunnels but also have their individual movement in which they try to align with Hans
    - Can be shot down when collided with instances of Bullet scenes
    - Drain Hans's battery when in contact with him
3. Viruses
    - Bacteriophage and rotavirus included
    - Are added as instances of level3 tunnel
    - Move with the tunnels but also have their individual movement in which they try to align with Hans
    - Can be shot down when collided with instances of Bullet scenes
    - Bacteriophage instantly kills when in contact with Hans, rotavirus will make him sick for a while

3.1.3 Trap scenes

All of the trap scenes have a kinematic body as a root node, are made from CSG shapes, and have adequate collision shapes. Some of them are also animated.

These scenes are instances of all of the 3 children of the Tunnels node found in the Main scene.

3.1.4 Bullet scenes

Include:
- BulletZero.tscn
- BulletOne.tscn

These scenes are instanced when the space button is pressed. Their initial position is in the Muzzle node contained in Hans.tscn, but later on, they become children of level1, level2, or level3, depending in which tunnel Hans currently resides. Due to this, the bullets will behave independently of Hans after they have been shot.

Instances of these scenes will disappear after approximately 3 seconds and no more than 8 of them can be created with one press of the space key.

3.1.5 Token scenes

At the moment, only EnergyToken.tscn exists. This scene contains a coin-shaped object that will recharge Hans' battery when in collision with him.

3.2. Scripts

Scripts are pieces of code attached to a certain node in a scene.

In this project, there are a couple of scripts that contain the whole logic of the game, and there are some that simply contain exported variables.

Scripts with only exported variables can be found in the root and kinematic body node of each bug and each virus.

Now let us get to the scripts that combine this whole project together and make it function properly.

Similarly, as with the scenes, we will start with Main.gd[6] (attached to root node of Main.tscn)

3.2.1. Main.gd

Functions:

1. _ready()
2. disable_sound_loops()
3. _start()
4. _show_first_help_layer()
5. _show_help_layer()
6. _game_over()
7. _on_Resume_pressed()
8. _on_Pause_pressed()
9. _on_Continue_pressed()
10. _on_Skip_pressed()
11. _on_Start_pressed()
12. _on_Previous_pressed()

- #1 _ready function is where the whole game starts. It is called once at the beginning. It contains a call to the _show_first_help_layer function (#4).
- #2 Used to disable sound loops from all of the sounds used in different parts of the game
- #3 Starts up the game by showing/hiding the appropriate UI elements and loading all of the trap, bug, and viruses scenes. In there we also call create_first_level_traps() function from Tunnels.gd.
- Functions numbered with 4,5,9,10, 11 and 12 are used to jump between the help layers appropriately. When the start button is pressed (_on_Start_pressed() - #11), the _start() (#3) function is called.
- #6 Shows the appropriate UI after the player has lost the game.
- #7 and #8 are used to handle the pause option accordingly.

[6] .gd is a filename extension used to represent a file written in GDScript

3.2.2. Hans.tscn::1 (script)

Functions:

1. _physics_process(_delta)
2. check_collisions()
3. create_new_trap()
4. update_curr_tunnel()
5. show_level_label()
6. translate_tunnel_and_ground()
7. shoot()
8. get_current_tunnel()
9. switch_animation()
10. _on_shooting_timer_timeout()
11. _on_Sick_timer_timeout()

- #1 function that is constantly being executed. It handles Hans's movement and all of the functions #2 until #7 (inclusive) are called within this one, at appropriate times.
- #2 checks what Hans has collided with and handles the situation accordingly.
- #3 creates a new trap by calling create_one_obstacle() function from outside this script.
- #4 updates certain variables, shows the label of the current level (by calling function #5), and increases speed when necessary. It is called every time we transfer from one tunnel to another.
- #5 shows the level label when called.
- #6 translates the tunnel we just finished forward, thus making the game infinite.
- #7 handles shooting. Here, instances of Bullet scenes are made.
- #8 retrieves curr_tunnel variable.
- #9 switches the animation of Hans, between shooting and regular movement.
- #10 and #11 are signals caught by timers that exist in Hans.tscn.

3.2.3. Tunnels.gd

Functions:

1. _physics_process(_delta)
2. create_first_level_traps()
3. create_one_obstacle(level,x)
4. pick_scene(level,scene = trap_scenes)
5. pick_obstacle(scene)
6. rotate_obstacle(obstacle)
7. delete_obstacle_until_x(level,x)
8. bug_virus_movement(curr_tunnel)

- #1 function that is constantly being executed. It checks if any of the keys for the "right" or "left" movement have been pressed, and if so, performs the rotation of the tunnel Hans is currently going through.

- #2 creates traps for the first level.[7]
- #3 uses functions #4, #5 and #6 to create 1 obstacle given the level and translation x.
- #4 picks a scene given a level. Generally, 4 types of scenes exist (trap_scenes, bug_scenes, virus_scenes, token_scenes) and depending on which tunnel we are in, some subset of them are among possible candidates.[8]
- #5 given a scene, it randomly picks an obstacle.
- #6 rotates a given obstacle by n * (PI / 3) where n is a random number between 0 and 5.
- #7 deletes obstacles that Hans has already passed by (helps avoid lagging of the game)
- #8 called from Hans.tscn::1, performs the independent movement of bugs or viruses, depending on which tunnel we are in at the moment.

3.2.4. Score.gd

Functions:

1. _on_Meter_Passed()
2. _on_Shooting_Obstacle()
3. _display_Final_Score()

- #1 increases the score and updates its label. Called from _physics_process(_delta) function inside Hans.tscn::1 script.
- #2 is similar to the previous function, but the amount the score increases is different. Called when a bug/virus is shot down.
- #3 displays the score after the game has been lost. Mainly takes care of the positioning of the Score label.

3.2.5. BatteryProgress.gd

Functions:

1. _on_DropTimer_timeout()
2. decrease_value()
3. update_timer()

- #1 signals when the timer from the Battery node has counted down. Calls #2.
- #2 decreases the value of the battery, updates the label and, if necessary, calls game_over() from Main.gd
- #3 when the speed of Hans increases (after the player goes one full lap through tunnels), the battery will decrease a bit faster. This is accomplished in this function by decrementing the wait_time variable of the Battery nodes timer.

[7] This happens only for the first tunnel when the game starts. After that, traps (and all other obstacles) are created incrementally by calling create_new_trap() from _physics_process function in Hans.tscn::1 script.
[8] This process is random most of the time. However, if a token scene hasn't appeared among last 10 obstacles, it will be chosen.

3.2.6. BulletZero.gd and BulletOne.gd[9]

Functions:

1. _physics_process(delta)
2. _on_Area_body_entered(body)

- #1 gives the bullets speed and direction and is responsible for removing them from the game after a certain time.
- #2 called when the bullet collides with something. If 3 bullets hit a bug/virus, that bug/virus will be removed from the game. This function also removes bullets that come into contact with any obstacle.

3.2.7. End.gd

Function:

1. _unhandled_input(event) - resets the game if the user presses enter after losing the game. It does so by reloading the Main.tscn

3.2.8. Level.gd

Function:

1. update_level() - updates the Level label node

[9] These are two identical scripts

4. Conclusion

To sum it up, as a developer of this project, I am very happy with the tools I chose to make it. The Godot Engine as well as the GDScript had very accessible learning resources and were not too difficult to maneuver.

As mentioned above, to create the characters for the game, I used Blender (version 2.92) and glTF 2.0 to export them. One limitation I found, is that it is not possible to transfer an emission shader between Blender and Godot. I am not sure if the problem occurs while exporting or importing, but nevertheless, Godot's built-in emission shader sufficed to achieve similar results.

Of course, there are some details that have not been pointed out in this document, however, they are not critical for the core structure of the game.

Overall, this project was written with the intention that whoever wishes, could easily build upon it. Whether it is by adding more traps, tunnels, some other tokens, etc. The goal was to make a fairly well commented, understandable, and maintainable code (nevertheless, there is always room for improvement).