Name: Jevgenij Ivanov
Student Number: 20748055
Project Title: Room Generator Extension for Godot 4.1
Supervisor: Ralf Bierig
ECTS Credits: 15

1) Overall Project Objectives.

The primary objective of “Room Generator Extension” project is to streamline the process of creating 3D environments within the Godot 4.1 engine, enhancing the speed and efficiency of building such spaces from scratch. By developing an intuitive and user-friendly plugin, the aim is to empower designers and developers to quickly assemble scenes using a library of prefabricated rooms and various components.

The plugin will be installed effortlessly by adding the plugin directory to “addons” folder in any project. This addon is then activated within the project settings, and users will immediately see a new tab appear in the bottom right section of the Godot editor interface, with all the tools available to be used.

This tool will not only be used to accelerate the manual creation of expansive and complex structures, but also to ensure that these constructions adhere to high-level design principles. These principles are planned to include features such as avoidance of redundant elements, for example double walls or overlaps, that can potentially hinder the aesthetics and performance in games and other 3D projects.

As an aspiring video game developer, my main focus is to create an addon/plugin that would not only assist me in my development journey, but also resonate with the broader game development community. As the inaugural user of this add-on, my personal investment in its development goes beyond mere functionality; I am deeply committed to crafting an add-on that is not only enjoyable to use but also free from bugs.

2) Description of work completed.

With no previous experience in Godot engine, it was essential that I start with basics and spend a lot of time researching and learning about the engine and its scripting language “GDScript”. I started this learning process by developing a basic game, which served as a practical exercise in mastering scenes and nodes, studying their mechanics and interplay.

After having created a playable game with fully controllable characters, I not only learned a new coding language, but I also created a fun game that can now serve as the testing ground for my plugin. I then began the plugin development. Initially, the plugin was just a tab in the upper-right section of the editor, and it had a couple of buttons that spawn boxes, walls and rooms. After some experimenting, I made a decision to move the plugin to the bottom-right section and have a standalone window. Inside the window, I added 3 different tabs, “Models”, “Layouts” and “Preview”. The “Models” tab has a “Dungeon Menu” button that reveals four textures which can be added to the current scene. These textures are able to create a dungeon environment or dungeon room/cave with floor, roof, normal and corner walls. I have tested the plugin extensively and was able to create a small dungeon environment, which I then added to the “Layouts” tab in my plugin for future use and testing.

One of the first major issues I encountered during the plugin development, was that the fundamental CTRL+Z and CTRL+Y commands (Undo, Redo) did not work as intended. One of the bugs, for example, when adding a box through the plugin, and then pressing CTRL+Z to undo the box, Godot instead would undo the last action performed in the Godot editor, prior to any actions in the plugin. In some cases, the undo operation would undo 2 actions at the same time, the Godot editor action and the Plugin action simultaneously. This was quite an irritating behaviour, so it was the first issue I decided to solve and develop the correct functionality between plugin and Godot editor.

Drawing on my past experience with implementing Memento and Command design patterns in C# to facilitate undo/redo functionality, I was pleasantly surprised to find that Godot's scripting language, GDScript, offered a more straightforward solution. The built-in "get_undo_redo()" object provided a suite of methods that perfectly suited my needs. By leveraging methods like ".create_action" and ".add_undo_method", I was able to seamlessly integrate the undo/redo capabilities within the plugin, ensuring a smooth and intuitive experience that mirrors the native behavior of the Godot editor.

2.1) Evidence of work completed.

The plugin itself operates with a script that has around 300 lines of GDScript code, which include functions that spawn meshes, walls, dungeon textures for environments, and various pre-made layouts. The Godot project repository that I am working in has more than 20 folders, excluding the addons and asset folders. There are around 25 scripts that I wrote and attached to various nodes during my Godot familiarisation phase, game development, plugin development, as well as learning and testing during the whole process.

I use GitHub as my main version control platform. There are around 250 commits that I made since the start of my Godot journey on the 2nd of October. All of these commits are exclusive for the Godot development (plugin, my personal game, testing and playground scenes, GDScript code testing and more). There are currently 22 branches in the Godot project repository, 3 of which are related to plugin and the rest were used previously for learning, testing, development of a game, levels, dungeon rooms and environments. Screenshots from my personal GitHub account "Ladnopoka", as well as screenshots of the plugin and its code, can be found in the appendix below. Access to the private repository that contains the plugin, game, assets and all the other components is available on request.

3) Outline of future work.

The current plugin will continue to expand and become more useful with each new feature implemented. The plan is to focus more on developing a plugin that can generate fantasy style environments, such as dungeons, caves, dungeon cellars, forest huts, bandit hideouts and many more! The next step is the implementation of Godot's 3D Grid Map node features into the plugin, which will enable the plugin to place textures and meshes on a grid interactively using tiles. Another feature that would be extremely useful, is the implementation of logic that can smartly create connections between rooms, for example, a tunnel that connects 2 parts of a dungeon. This feature will enable the plugin to create large and more interesting layouts, as another example, a cave entrance with a tunnel to a dungeon, connected by a hole that goes downwards into another cave. Implementing functionality for the "Preview" tab, which will display the potential connections between rooms and dungeons.

APPENDIX

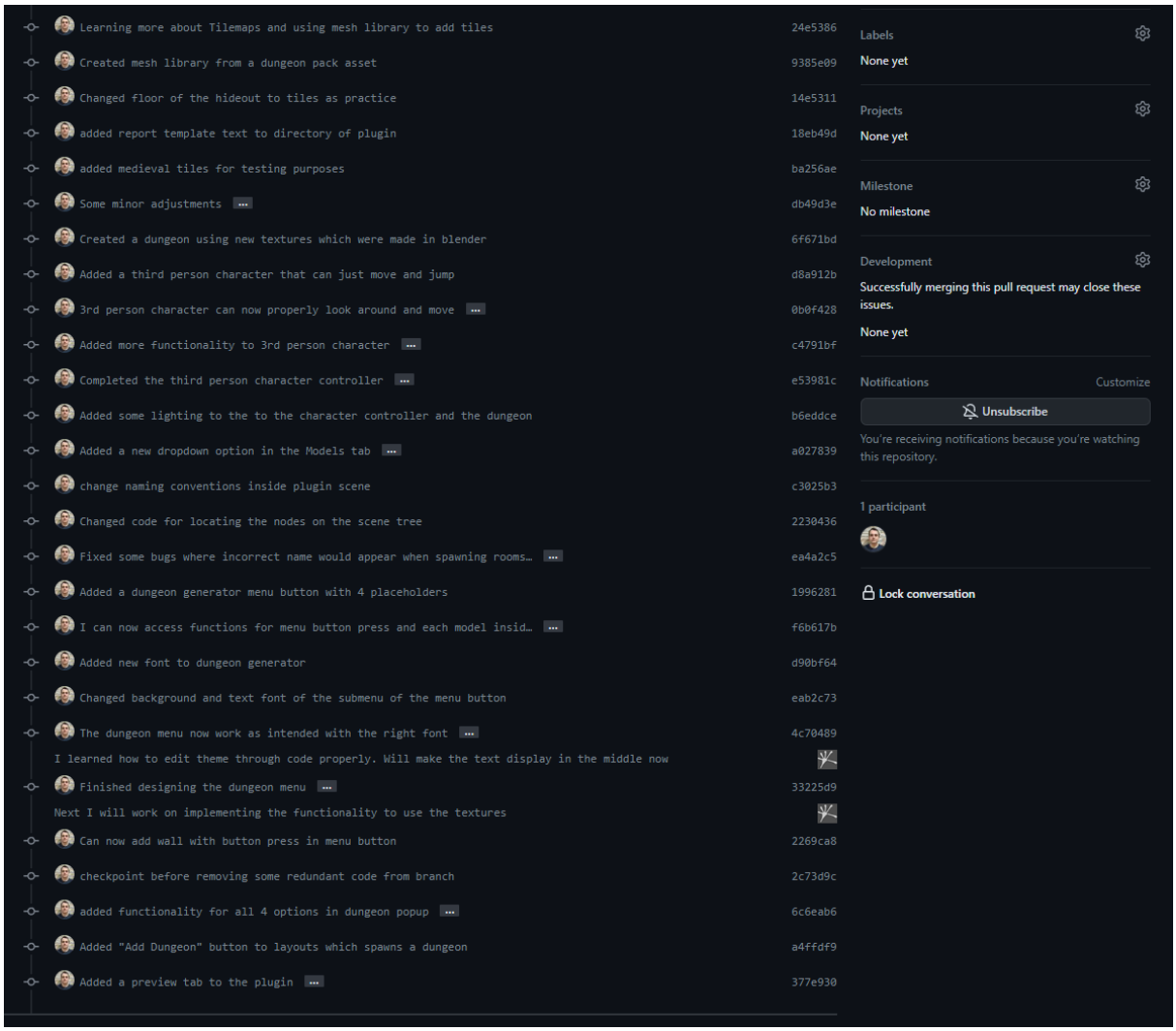

Fig 1. One page of commit history for the Plugin branch of the project repository.
Note: I use descriptions and comments in my commits in a form of diary entries, explaining and showing what I did and what I'm planning to do. Such example descriptions can be seen expanded in the GitHub screenshot above.

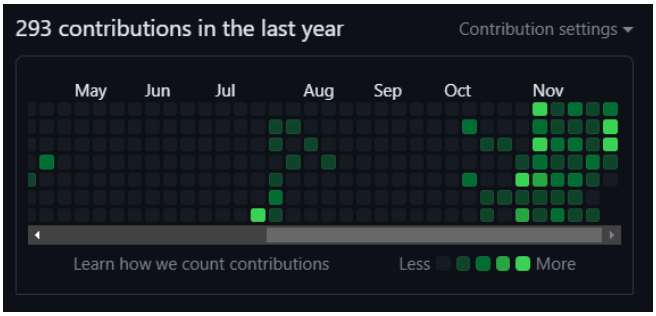

Fig 2. Contributions map (Godot starts early October).

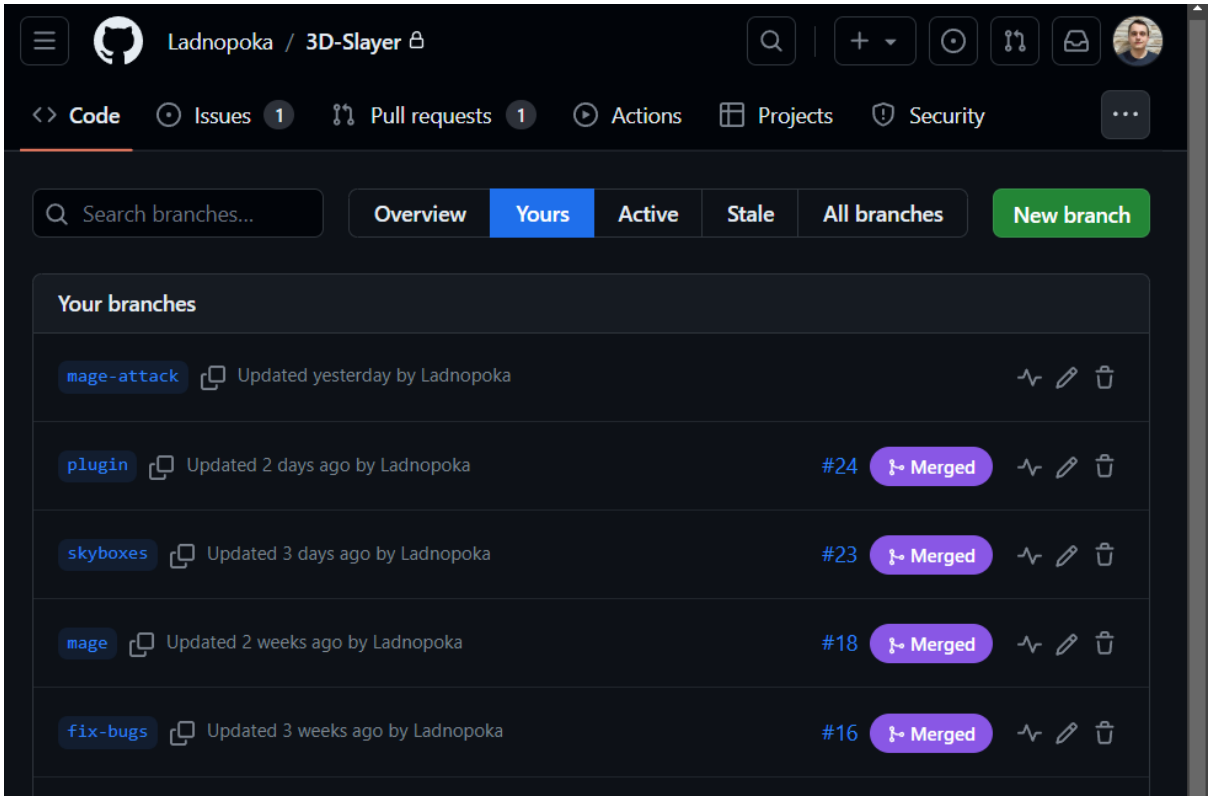

Fig 3. Screenshot of the last 5 branches used, which include plugin, skyboxes (including environments) and bug-fix branches.

```
29  func _enter_tree():
41
42  func setup_button_connections():
43      # Connect the toggle button signal
44      wall_button = dockedScene.get_node("TabContainer/Models/Wall")
45      button2 = dockedScene.get_node("TabContainer/Models/Cube")
46      button3 = dockedScene.get_node("TabContainer/Layouts/Room")
47      hideout_button = dockedScene.get_node("TabContainer/Layouts/Hideout")
48      menu_button = dockedScene.get_node("TabContainer/Models/DungeonGeneratorMenu")
49      dungeon_layout_button = dockedScene.get_node("TabContainer/Layouts/Dungeon")
50
51      wall_button.connect("pressed", create_wall)
52      button2.connect("pressed", create_box)
53      button3.connect("pressed", create_room)
54      hideout_button.connect("pressed", create_hideout)
55      menu_button.connect("pressed", menu_button_pressed)
56      dungeon_layout_button.connect("pressed", dungeon_layout_button_pressed)
57
58  func setup_menu_button():
78
```

Fig 4. A function that handles all of the button connections in the plugin through signals. Doing it this way is much cleaner, and does not require enabling signals through editor.

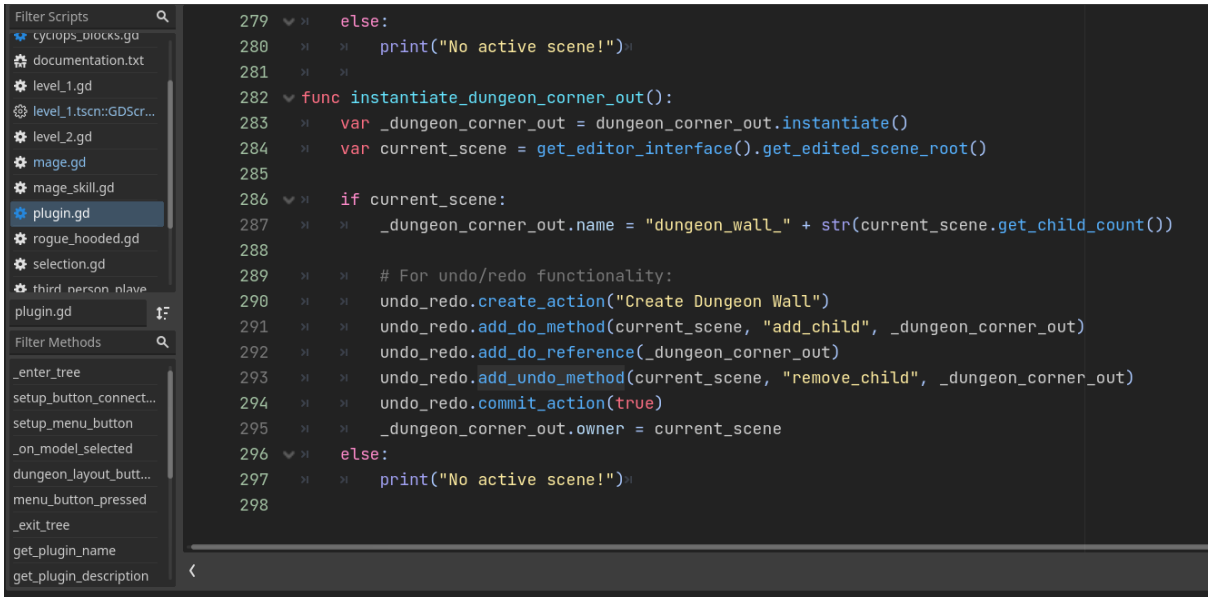

Fig 5. One of the last functions created inside the plugin script, which instantiates a dungeon corner texture and spawns it in a scene (with undo/redo functionality).

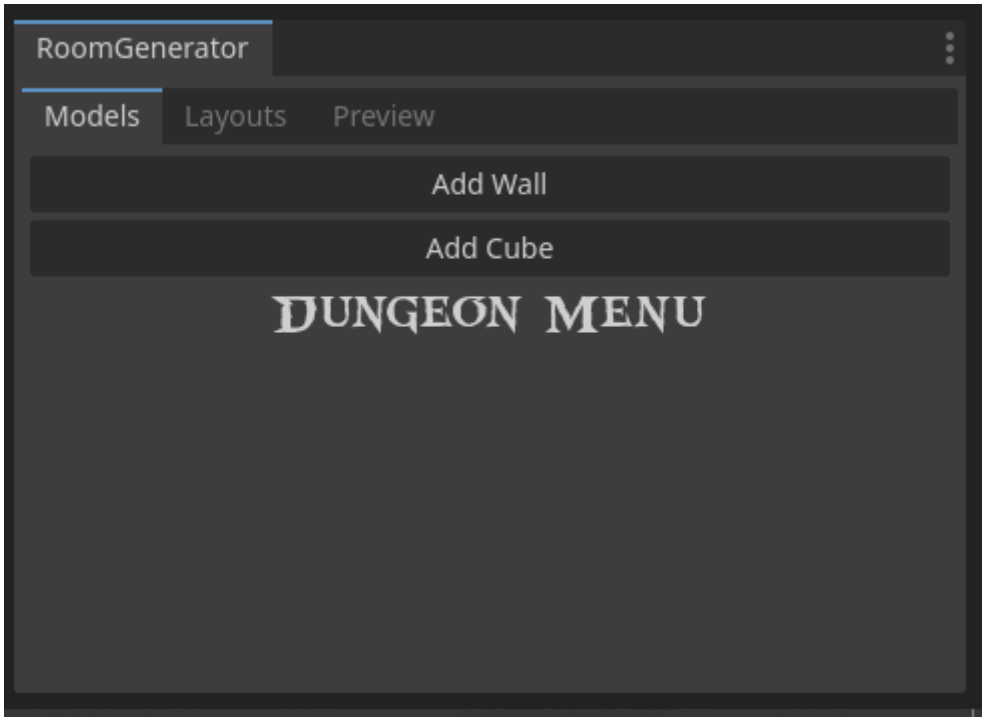

Fig 6. Plugin screenshot 1.

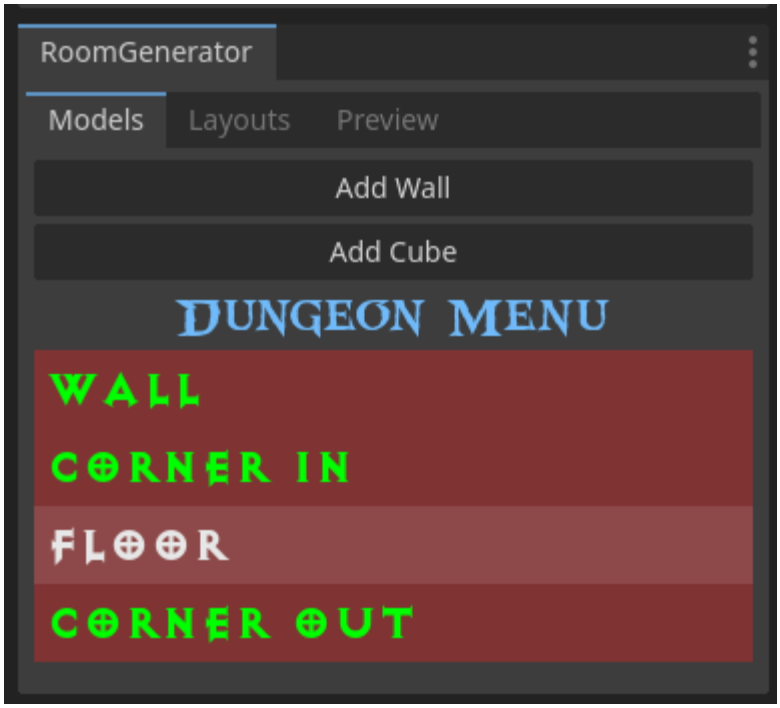

Fig 7. Plugin screenshot 2 (The text font and colours are experimental and were used for learning purposes).

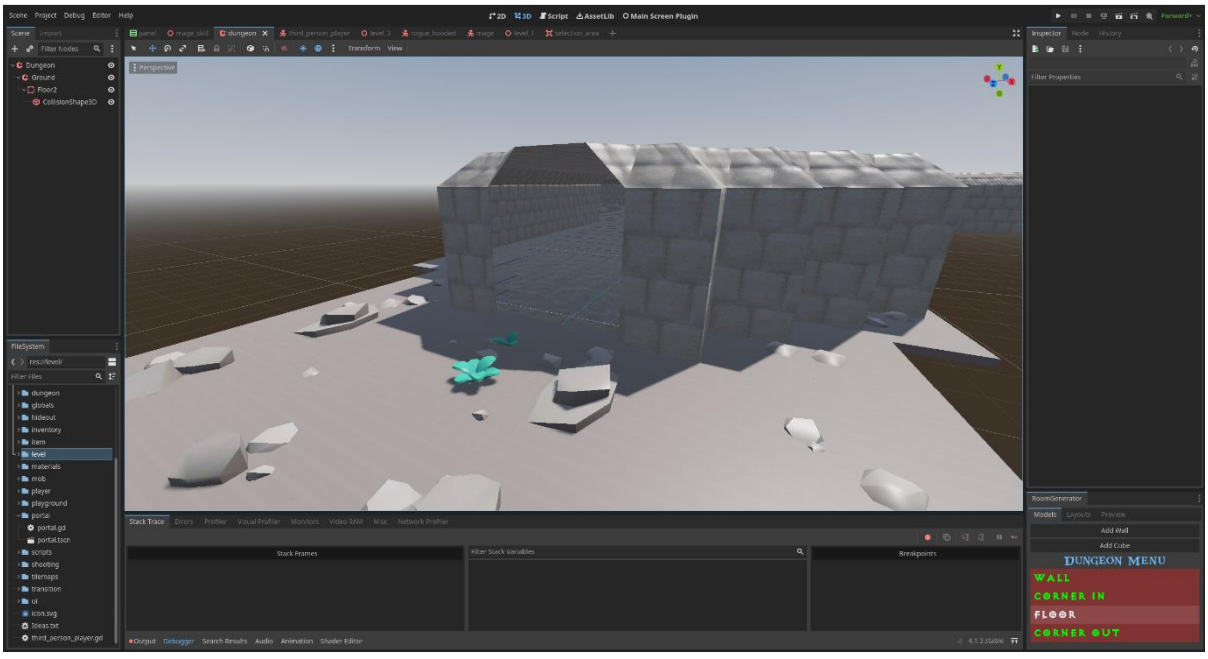

Fig 8. Plugin in action screenshot

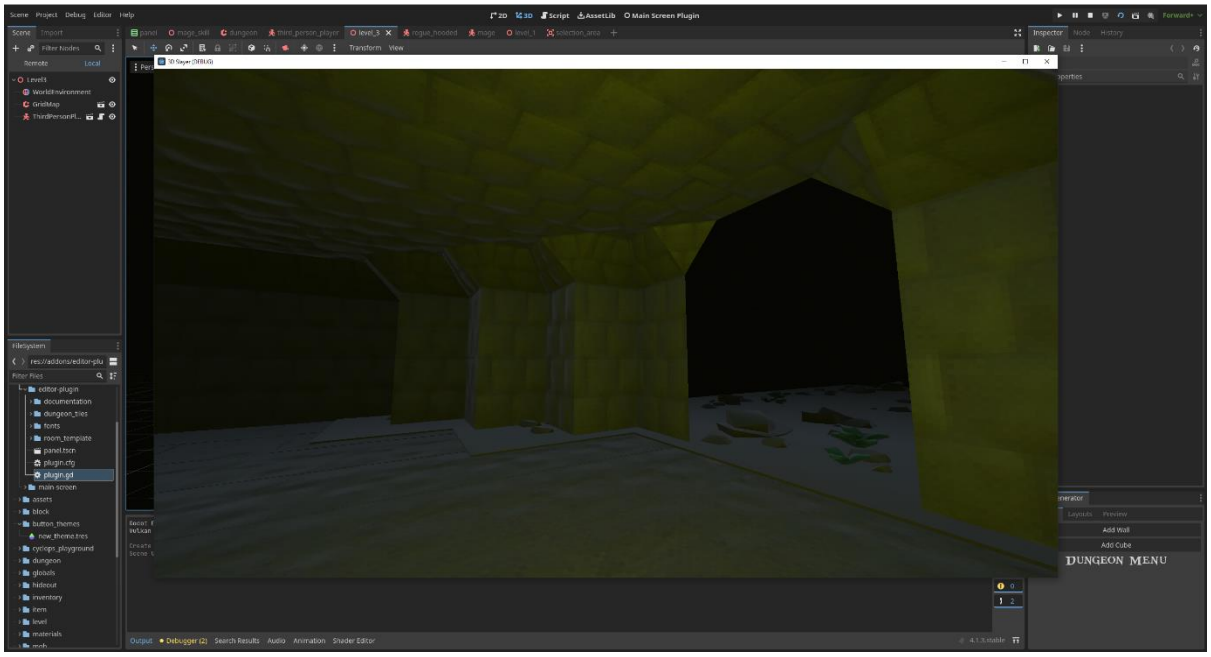

Fig 9. Using a simple First Person Character controller to test and observe the dungeon room created with the help of the plugin.