CM50109 Software Engineering

Coursework 2: A serious game

Turquoise Group

Installation Guide + User Manual + Maintenance Guide

Installation Guide:

<u>Installation guide for the player</u>

1. Copy the zip file to your computer and extract it.

2. Run the geogame.exe executable file to play the game.

<u>Installation guide for developers</u>

The source code for Geogame is available on GitHub here. After cloning the repository and installing Godot 4, the user simply needs to locate the "project.godot" file on the disk. Executing that file will load the project in the Godot 4 game engine which can then be modified.

We have used an additional package in Godot to handle the dialogues. This is the "Dialogue Manager" package, details of which can be found here. To sum up, developers can follow these steps to install and make changes to the game:

1. Download and install Godot 4.

2. Clone the Geogame sourcecode.

3. Checkout the "develop" branch. (can be done via git checkout)

4. Open the "project.godot" file in Godot 4.

5. Make changes to the code.

6. Create a pull request on the "develop" branch.

User Manual

Game Overview:

Embark on a thrilling adventure with "Geogame", an exhilarating puzzle and quiz-like experience that immerses players in the captivating narrative of a detective on a mission to apprehend the notorious international thief, The Phantom. Players take on the role of The Detective, who traverses the globe, following the footsteps of The Phantom, putting their knowledge to the test with the risk of losing the culprit's trail.

In the game, players will explore different countries in the form of rooms in which they need to escape. Hence, the theme of this game is escaping rooms where the player needs to answer country-specific questions tailored to the room(country) they are in, to score points and escape the room.

The objective of this game is for the player to learn about the different countries that their character, The Detective, travels to. Each country(room) has specific questions that need to be answered correctly to move on to the next stage. These questions hold the ability to educate the player, introducing curious facts about the country they are in.

A time limit is applied to each room, which shortens every time the player progresses through to the next room, increasing the difficulty of the game. A maximum number of wrong attempts are also included, which would cause the player to lose when reached, potentially reducing the action of players guessing answers for the questions.

System Requirements:

Minimum System Requirements:

Graphics:

- **Recommended:**

    Hardware compatible with Vulkan 1.0

- **Minimal:**

    Hardware compatible with OpenGL 3.3 / OpenGL ES 3.0

Starting The Game:

.exe file: Double click the ".exe" file and the game will start.

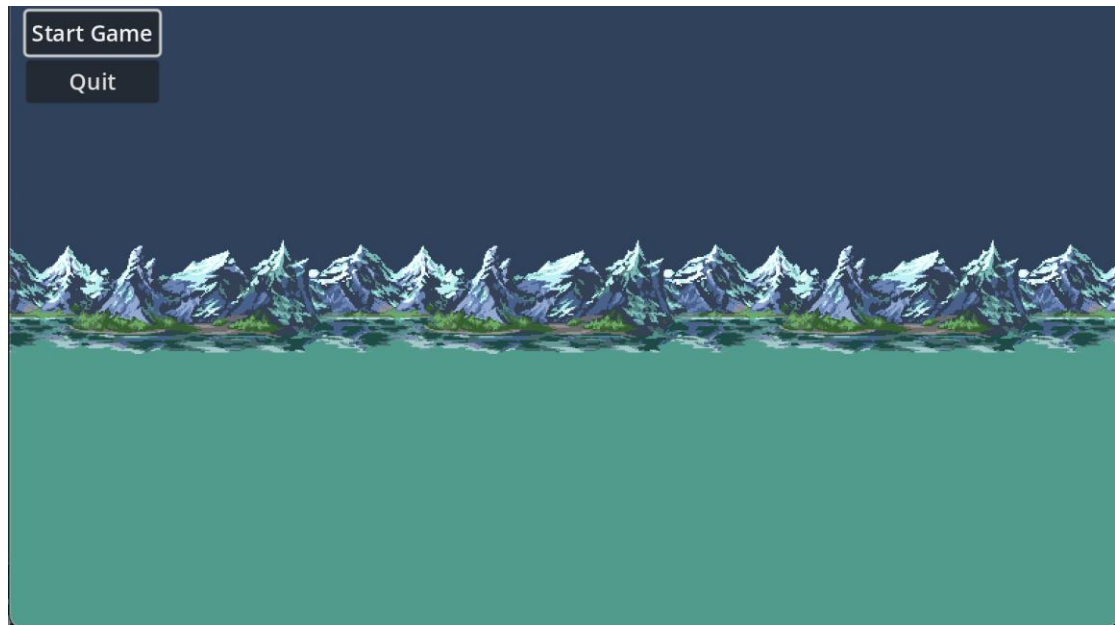This is the start screen that will display at the start of the game:

Figure 1 - Start Screen.

Start Game

Click on "Start Game" and the game will begin, on level 1.

Quit Game

Click on "Quit" and the application will close, quitting the game.

Keyboard Controls:

Movement:

To move around in the game, there are two sets of controls for movement.

WASD keys – where "W" = up, "A" = left, "S" = down, "D" = right.

Up down left right keys – Where the arrow keys on the keyboard move the character with respect to the direction.

Interaction:

Use the "E" key on the keyboard to activate interactions in the game with various curious looking objects.

Character Description:

The player enters the game and plays as "The Detective", tasked to follow the footsteps of the culprit. The player interacts with objects and solves questions about the country while they are in the shoes of The Detective.

Figure 2 – The Detective.

Dialogue Boxes:

In the game, there are several dialogue boxes that provide an integral narrative storyline for the game. There are also several portraits that are included in these dialogue boxes which have been generated using prompts for OpenAI's ChatGPT and DALL-E. These are AI software that has been developed by OpenAI that can be used to generate images. Here is an example image of a dialogue box with a portrait on the left that has been generated by the AI software:
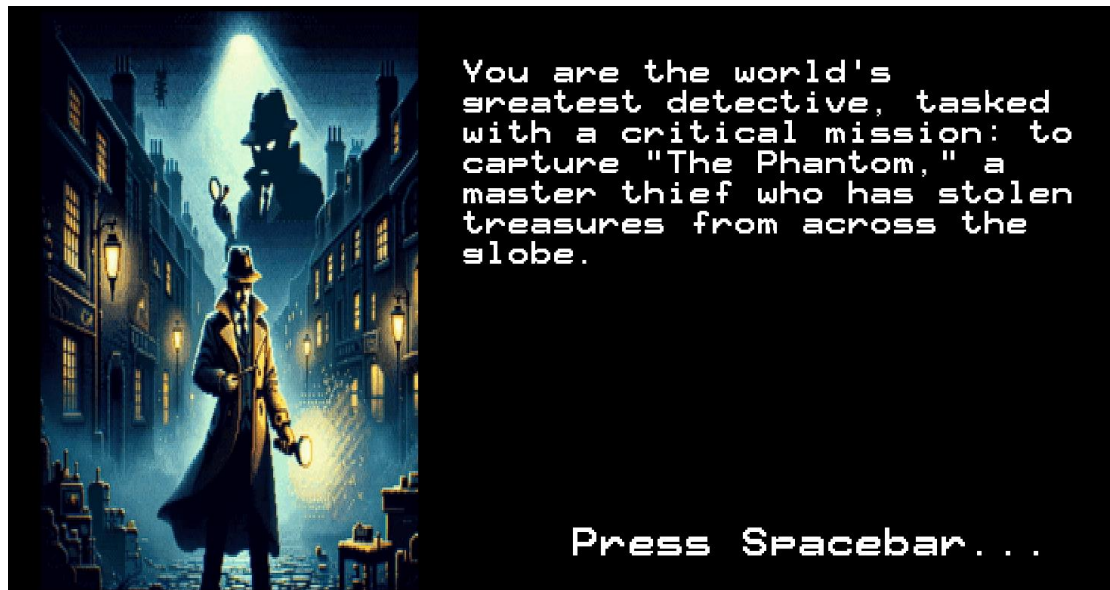

Figure 3 – Dialogue Box.

Oracle Description:

An oracle has been implemented to help players with answering the questions if the player is unable to answer the question.

After navigating to the pause menu, the player can contact the oracle by pressing on the "Get help" button. After that, a menu will appear where the player can choose the questions that they need help with.
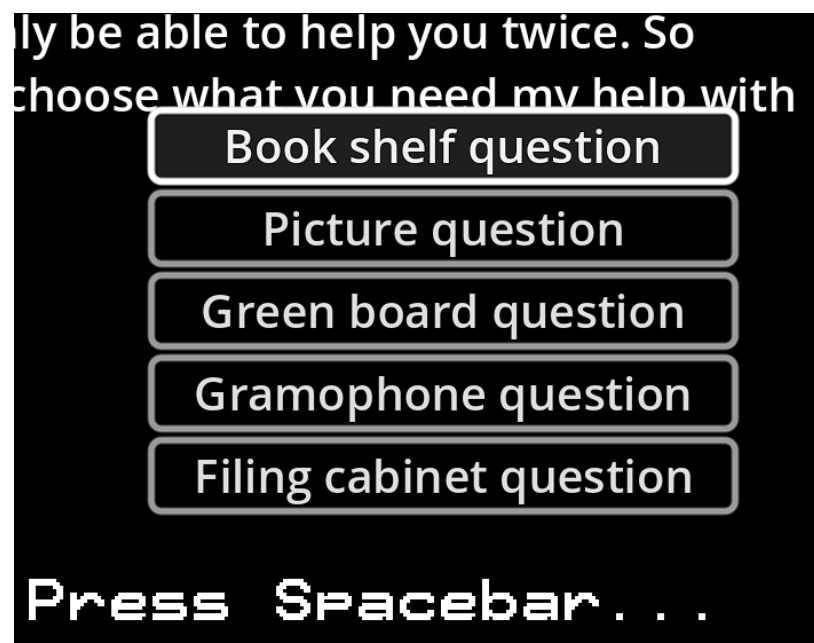

Figure 4 – Oracle example.

Scoring System:

When the player answers questions correctly, there will be a counter that displays on the top of the screen. This shows how many questions the player has answered correctly, and how many questions the player must answer to move onto the next stage.

Figure 5 – Counter for answer.

Time Limit:

A bar is included at the top of the screen informing the player about the time remaining to complete the level. If the time runs out where the bar reaches zero, and the player has still not answered all the questions in the level correctly, the player loses, and the game ends.
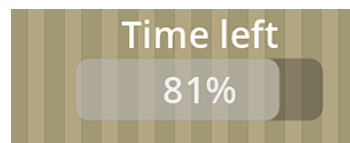

Figure 6 – Time remaining bar.

Escape button:

Once the player has answered all questions correctly, the player would be informed by the game that the next stage is available. To move to the next stage, the player would have to click on the "Escape" button located in the room.

Figure 7 – Escape button in level 1.

Figure 8 – Escape button in level 2.

Game Levels:

Level 1 – Japan:

The first level of the game represents Japan. There are five questions in this level, and by answering all of them, the player would be able to progress to the next stage. Additionally, there is also background music that has been chosen to fit this level. A traditional Japanese soundtrack was chosen to represent this level. The artist for this soundtrack is MOJI(MOJI, 2020).

Here are the objects that have questions linked to them:

From left to right, top to bottom, they are the "bookshelf", the "file cabinet", the "painting", the "gramophone", and the "board".
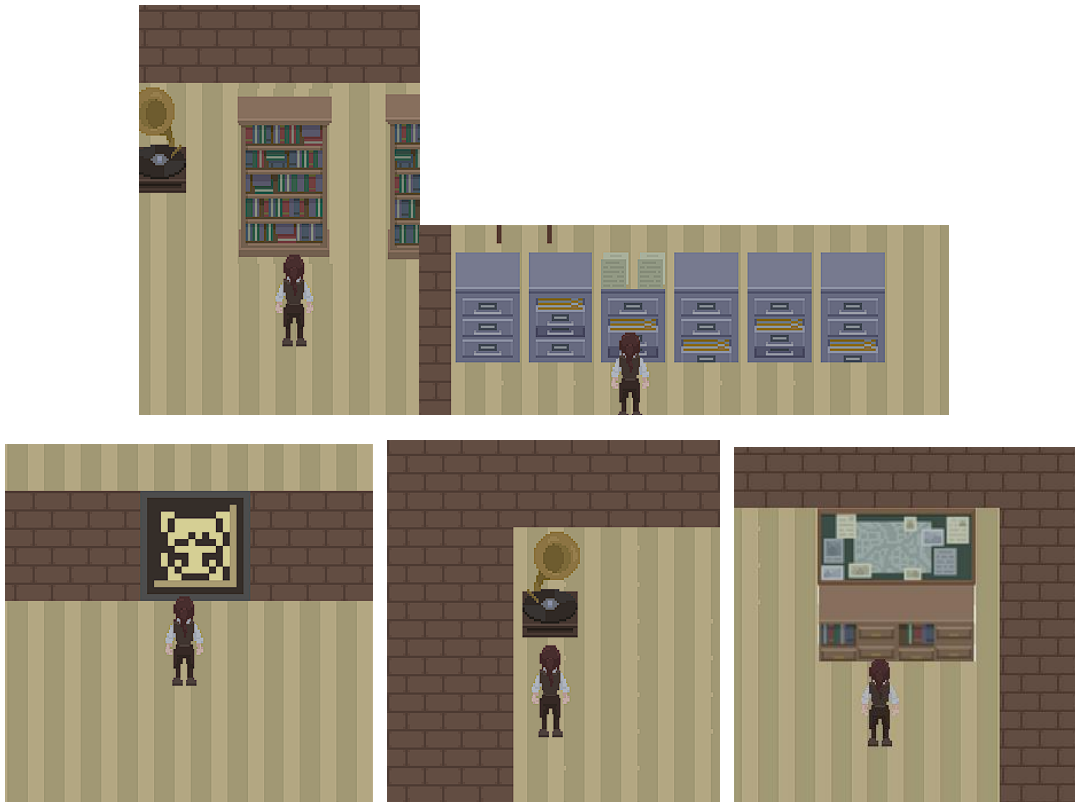

Figure 9 – Interactable objects in level 1.

Level 2 – Egypt

The second level of the game represents Egypt. There are also five questions in this level, and answering all of them would allow the player to "Escape" and move to a cut scene which represents the completion of the game. The soundtrack for this level was created by Angels of Venice (Angels of Venice, 1999).

For this level however, there are four objects that must be answered first before the player can interact with the final object, which is the pyramid. Attempting to interact with the pyramid without meeting these conditions would produce an error message.

Here are the four objects that needs to be interacting and successfully answered first:

From left to right, top to bottom, they are the "Sphinx", the "debris", the "oasis", and the "tomb".

Figure 10 – Four preliminary interactable objects for level 2.

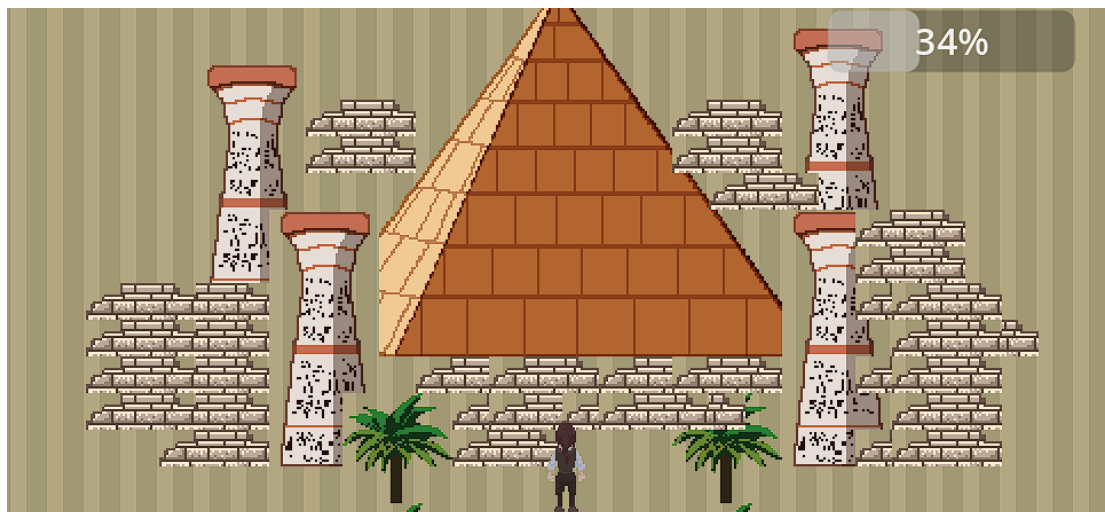The final interactable object would be the Pyramid:

Figure 11 – Final interactable object for level 2.

Troubleshooting:

Before you send us an e-mail to ask about problems encountered in the game, please make sure that your computer meets our minimum system requirements. It would be better if your computer meets our recommended system requirements.

Please note that sometimes uninstalling and reinstalling can resolve most issues you encounter with the game.

Technical Support E-mail: jh3984@bath.ac.uk, other people's emails...

Here are some self-examination questions that help most people solve their problems:

Installation Issues:

Solution: Please ensure that system meets the necessary requirements and ensure there is sufficient storage space within the system. Additionally, verify the integrity of the game files.

Game Not Starting/Loading:

Solution: Please try restarting the device and check if there are any necessary software updates.

Level Access Problems:

Solution: Please ensure that all questions have been completed on the level before clicking on the "Escape" button to move to the next stage.

Performance Issues:

Solution: Please try closing any background applications to lower RAM usage and potentially increase the smoothness of the game, or try restarting the device.

<u>Audio/Visual Issues</u>:

Solution: Please check system's audio settings and update any graphics drivers.

<u>Game Controls Not Responding</u>:

Solution: Please verify game control settings in the game and ensure hardware is properly connected and configured.

<u>Miscellaneous Glitches</u>:

Solution: Please try restarting the game and report any bugs to jh3984@bath.ac.uk.

Maintenance Guide

Game Engine Overview:

The game engine used to implement Geogame is Godot 4, an open-source game engine. Godot uses a very object-oriented approach to game development where the game is a tree of Nodes that you group into Scenes. Some of the Nodes communicate with each other using Signals.

The following is a brief definition of these key concepts in Godot:

1. Scene - A scene is a self-contained part of the game. A scene can be a character, a weapon, a menu, a user interface, or anything else you can think of. Scenes are also the container that holds the various nodes that serve different functions.

2. Node - Nodes are the building blocks in Godot that you arrange in trees to form the game. Godot provides an extensive library of nodes that can be combined to create new nodes for the game.

3. The scene tree - All the scenes in the game come together to form the scene tree. The versatility of scenes allows them to be nested or instantiated within each other.

4. Signal - Nodes emit signals when some event occurs. These signals can be connected to the scripts to implement the behaviour we want for that event.

Game Flow:

As mentioned in the user manual, the game starts with the Start Screen menu which is a scene named "Menu". Then the Start_story scene gives the context of the story to the user and transitions to the gameplay levels which are Level1 and Level2. After completing the gameplay levels, the user gets to the Final_scene which concludes the narrative of the game.

As mentioned in the user manual, the game starts off in the start screen menu, which is a scene called "menu.tscn" file. After clicking start, the start story scene gives players the context and narrative behind the game, in the "start_story.tscn" scene. The game then transitions to the level 1 scene where the player must solve the necessary questions before moving on to the level 2 scene. After reaching the necessary requirements to complete level 2, the player is then able to reach the final scene, which also represents a narrative that concludes the game, in the "final_scene.tscn" scene.

On the other hand, if it occurs that the player accumulates too many wrong attempts, the game is over. This would cause the scene to switch to the "lose_scene.tscn" scene, where it shows that the player has lost.

Navigating scenes and scripts:

Scenes in Godot are represented by a ".tscn" file in the codebase. And for most of the ".tscn" files, there will be a ".gd" file with the same name which represents the script or code respective to that scene.

Switching to different views of a scene is simple with the help of the Navigation panel located at the top of the game engine.

Figure 12 – Navigation Panel

The main levels in Geogame consist of the following scenes:

- Menu.tscn
- Start_story.tscn
- Level1.tscn
- Level2.tscn
- Final_scene.tscn
- Lose_scene.tscn

These scenes make up the main structure of the game, and they contain other scenes that implement the various functions of gameplay and game mechanics. Some of the scene views can be seen below:
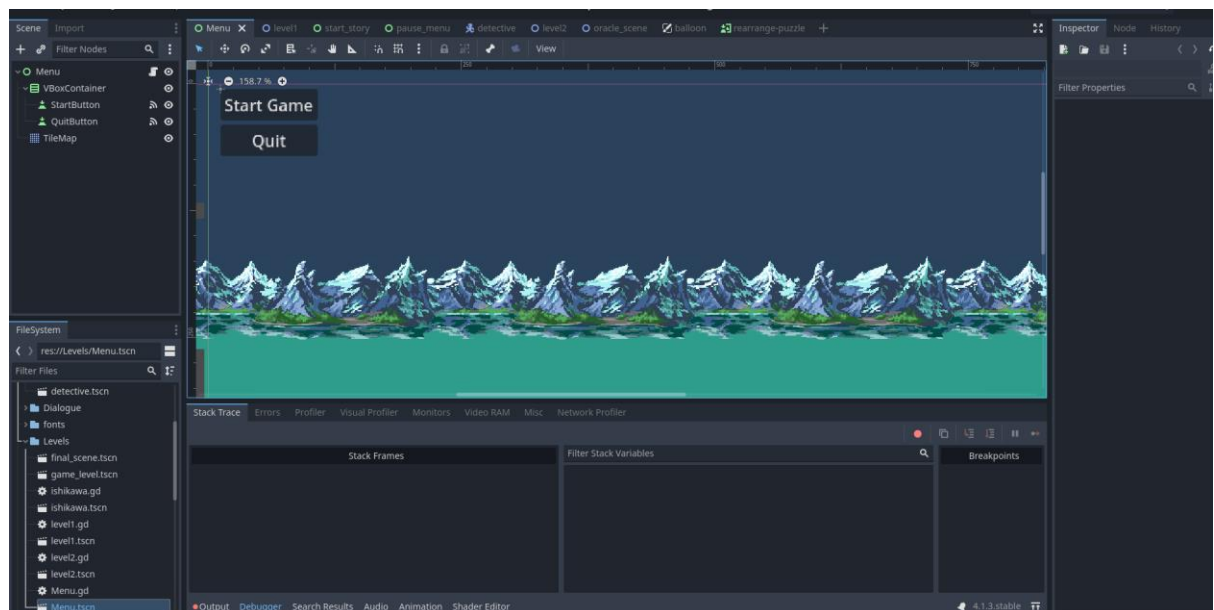
Figure 13 – 2D Godot view for the Menu scene

18
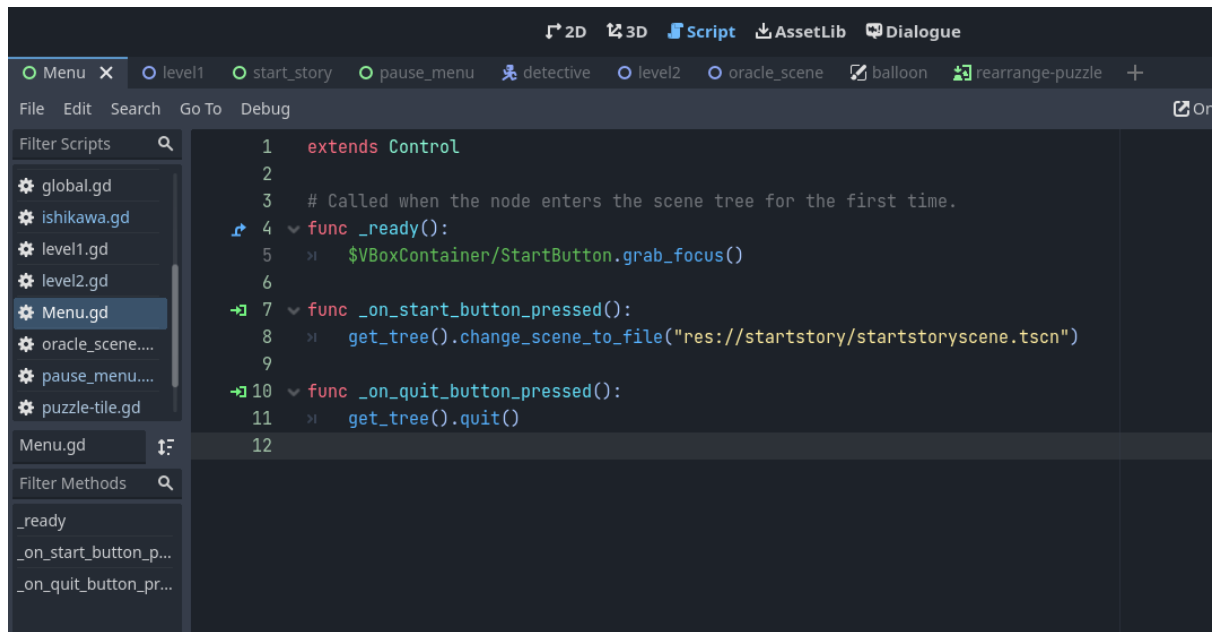
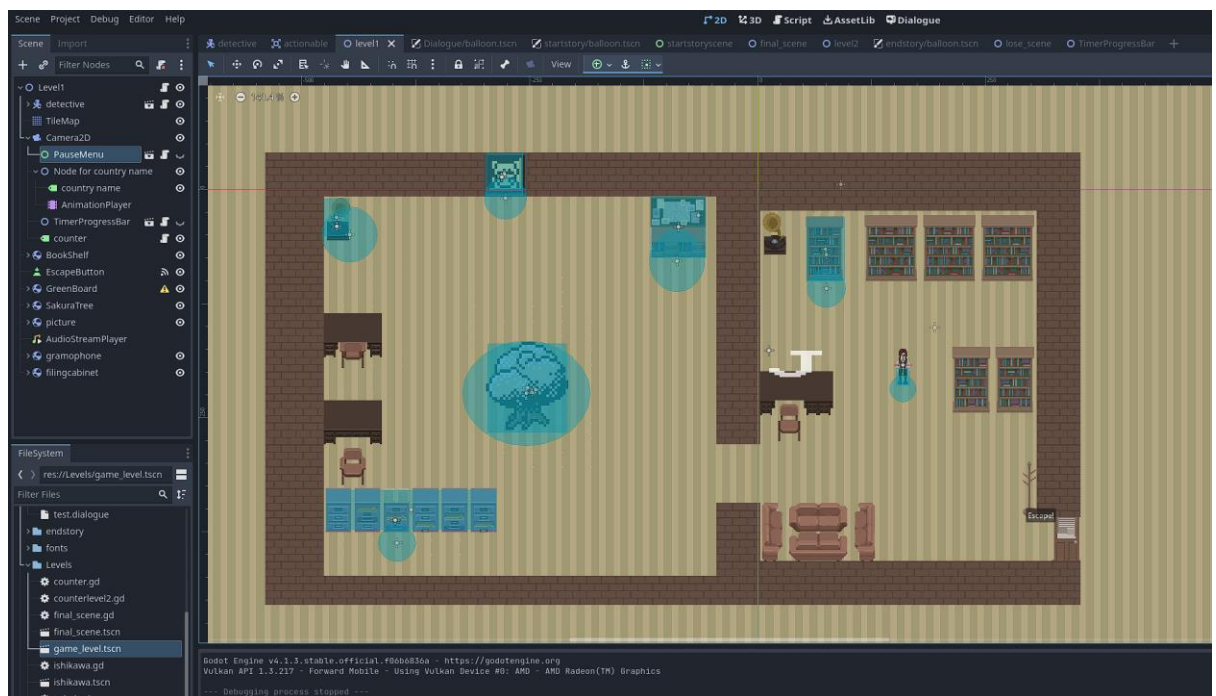Figure 14 – Script view for the Menu scene

Figure 15 – 2D Godot view for the Level1 scene containing various nodes.

Implementation of Game Mechanics:

<u>Character movement animation</u>

To create the movement animations for the character, the "AnimationPlayer" and "AnimationTree" nodes are added as child nodes to the detective scene. Using the "AnimationPlayer" node, we can edit the keyframes for the movement and add the frames from the sprite sheet for our detective character to define various animations.
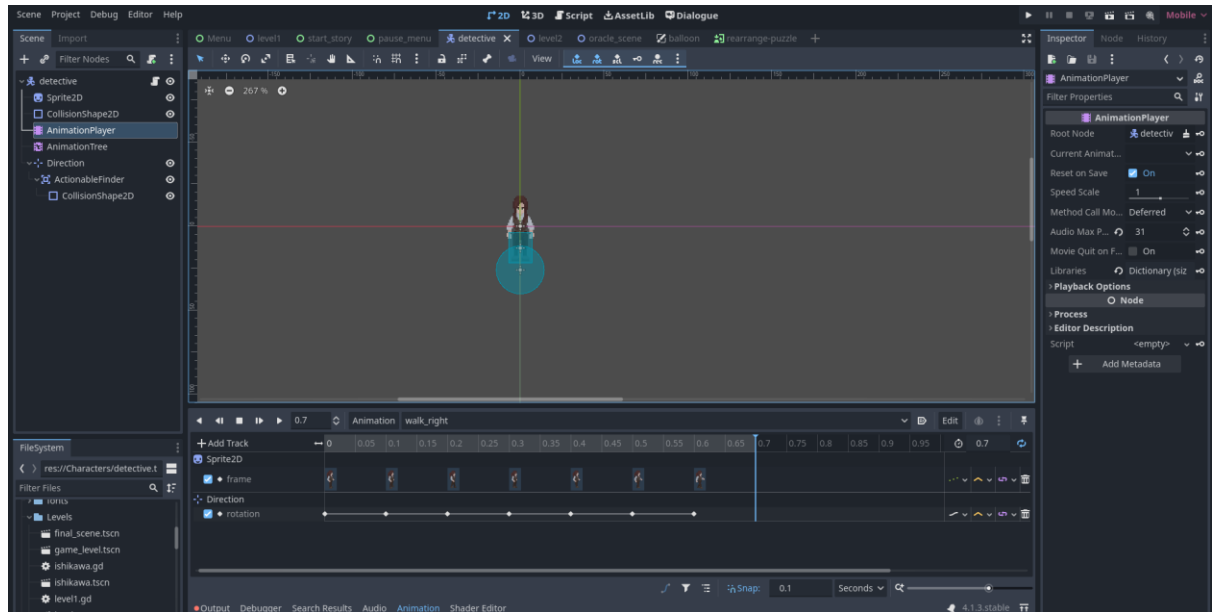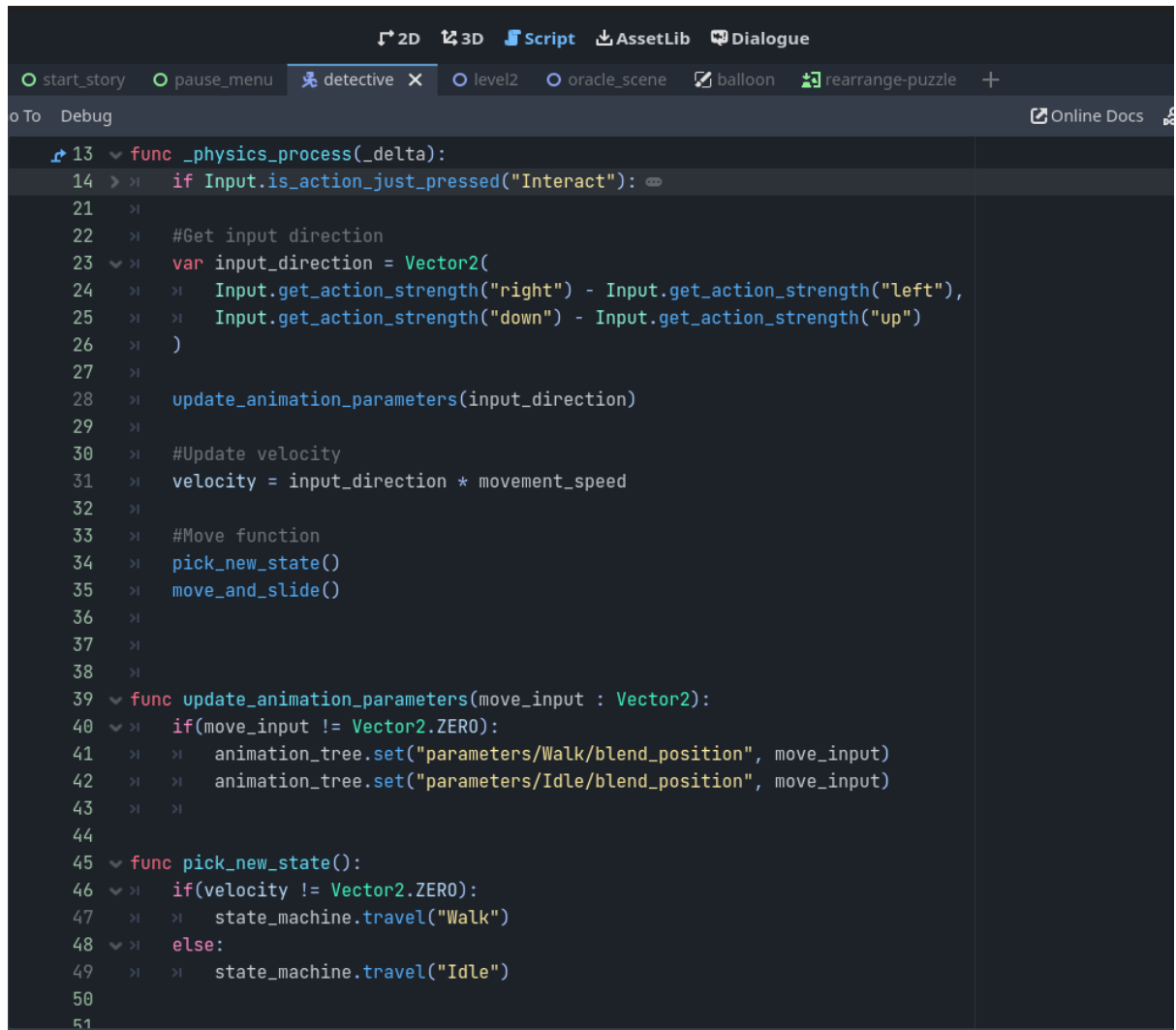

Figure 16 – 2D Godot view for detective scene

We can use the AnimationTree to use the animations defined with the AnimationPlayer. This is done by creating a state machine where the different animations are triggered based on the state of the object (which is determined by the movement input given by the user). The details for using AnimationPlayer and AnimationTree are given below.

○ start_story   ○ pause_menu   ⚹ detective ✕   ○ level2   ○ oracle_scene   ☑ balloon   ⬈ rearrange-puzzle   +

o To   Debug   ⬀ Online Docs ⚙

```
 13  ⌄ func _physics_process(_delta):
 14  ⌄  ⟩  if Input.is_action_just_pressed("Interact"): ⬚

 21  ⟩
 22  ⟩       #Get input direction
 23  ⌄ ⟩     var input_direction = Vector2(
 24  ⟩   ⟩       Input.get_action_strength("right") - Input.get_action_strength("left"),
 25  ⟩   ⟩       Input.get_action_strength("down") - Input.get_action_strength("up")
 26  ⟩   ⟩   )
 27  ⟩
 28  ⟩       update_animation_parameters(input_direction)
 29  ⟩
 30  ⟩       #Update velocity
 31  ⟩       velocity = input_direction * movement_speed
 32  ⟩
 33  ⟩       #Move function
 34  ⟩       pick_new_state()
 35  ⟩       move_and_slide()
 36  ⟩
 37  ⟩
 38  ⟩
 39  ⌄ func update_animation_parameters(move_input : Vector2):
 40  ⌄ ⟩  if(move_input != Vector2.ZERO):
 41  ⟩   ⟩     animation_tree.set("parameters/Walk/blend_position", move_input)
 42  ⟩   ⟩     animation_tree.set("parameters/Idle/blend_position", move_input)
 43  ⟩   ⟩
 44
 45  ⌄ func pick_new_state():
 46  ⌄ ⟩  if(velocity != Vector2.ZERO):
 47  ⟩   ⟩     state_machine.travel("Walk")
 48  ⌄ ⟩  else:
 49  ⟩   ⟩     state_machine.travel("Idle")
 50
 51
```

Figure 17 – Script view for the character movement implementation

21

AnimationPlayer

The AnimationPlayer class in Godot is used for general purpose playback of animations. In Geogame, we have used AnimationPlayer in the following way. We have defined eight animation states that will be animated. These are "idle_up", "idle_left", "idle_right", "idle_down", "walk_up", "walk_left", "walk_right" and "walk_down":
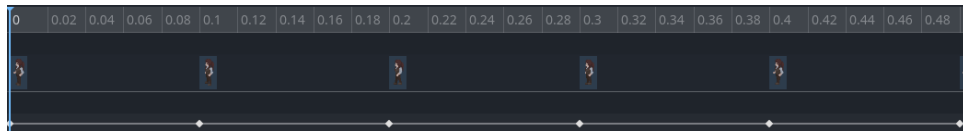

Figure 18 – The animation frames for walk_left animation

The animation frames can be added by following these steps:

1. When you click the AnimationPlayer node, there will be a window that appears at the bottom of the screen, click on animation at the extreme bottom, then at the top of the bottom window click on animation -> new.

2. Then depending on the frames of your sprite art, we need to justify which frames represent idle and walking, with respect to the directions. E.g.: frame 0 - 1 is idle_down, frame 2 - 3 is walk_down, frame 4 - 5 is idle_right and so on.

3. Click on add track -> property track -> sprite2D (the node you're targeting, not the parent node but the child node which is the node of the sprite) -> frame and click open.

4. With our animation bar open, when we click on our sprite's node, there will be 'keys' on the right column where our frames are.

5. Identify the frames you wish to use and then click on the 'key' symbol to insert into the animation bar.

6. After inserting the frames, you can drag it around to set apart how far you want the animations to be. There is a slider on the bottom right to expand the time zone.

7. On the right of the bar, you can set the animation duration and even make it loop.

8. You can then duplicate this to create the animations for the other frames.

22

AnimationTree

The animations defined in the AnimationPlayer are used in the AnimationTree node. The transitions between the various animations are handled in the AnimationTree. This is done by creating an AnimationNodeStateMachine and linking the states to the input provided by the user in the detective.gd script.

The AnimationTree can be created and linked to the AnimationPlayer by following these steps:

1. Add the AnimationTree and click on it. Then on the right column, in the "Anim player" resource, click AnimationPlayer. This is so we know where the animations come from.

2. Then, for the tree root, click New AnimationNodeStateMachine.

3. Right-click the space and add two BlendSpace2D and rename it to idle and walk, for your idle and walk animations.

4. Then click on the connector button to connect idle and walk. And have a two-way connection between them. And have a one-way connection from the start to idle.

5. Now, click on the edit pencil symbol next to idle.

6. Set the y-axis dimensions to 1.1 and -1.1 (so that left and right animations are prioritized).

7. To add animations, click on the pencil icon on the top left, click on the MIDDLETOP of the graph -> add animation and click the idle_down (y-axis inverted). And MIDDLEBOT for idle_up. Do this for idle_left and idle_right as well.

8. Then, edit walk following the same steps.

9. Change the blend option for both idle and walk to the dotted line (discrete).

10. Click on the 2 arrows between idle and walk and on the right-column, under "advance" change mode to disabled.

11. Click on AnimationTree and set it to active on the right column.

12. Finally, click on the transition-arrow between start and idle and on the right-column, set advance mode to auto. [This starts the idle animation automatically when game starts]

Now that the AnimationTree is created and has been linked with AnimationPlayer, we need to connect it to Detective script so that we can change the animation states based on user input. This is shown in the following screenshots.
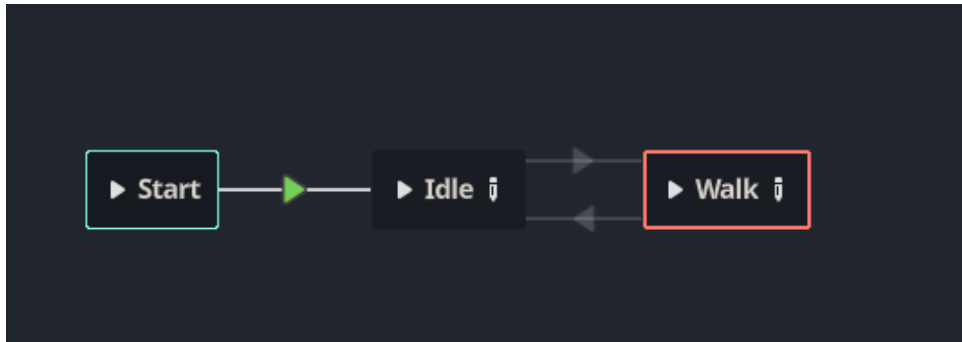
Figure 19 – The "AnimationNodeStateMachine" for movement animation

```gdscript
func _physics_process(_delta):
    #Get input direction
    var input_direction = Vector2(
        Input.get_action_strength("right") - Input.get_action_strength("left"),
        Input.get_action_strength("down") - Input.get_action_strength("up")
    )

    update_animation_parameters(input_direction)

    #Update velocity
    velocity = input_direction * movement_speed

    #Move function
    pick_new_state()
    move_and_slide()
```

```gdscript
func update_animation_parameters(move_input : Vector2):
    if(move_input != Vector2.ZERO):
        animation_tree.set("parameters/Walk/blend_position", move_input)
        animation_tree.set("parameters/Idle/blend_position", move_input)


func pick_new_state():
    if(velocity != Vector2.ZERO):
        state_machine.travel("Walk")
    else:
        state_machine.travel("Idle")
```

Figure 20 – Switching between animation states depending on the user input in "detective.gd"

Dialogue Manager and implementation of "actions":

Geogame uses a Godot package called Dialogue Manager to handle the branching dialogue which is triggered after the player interacts with an object. As shown in the figure above, each actionable item has an action attached to it and when the action function is called, their respective dialogues are triggered.

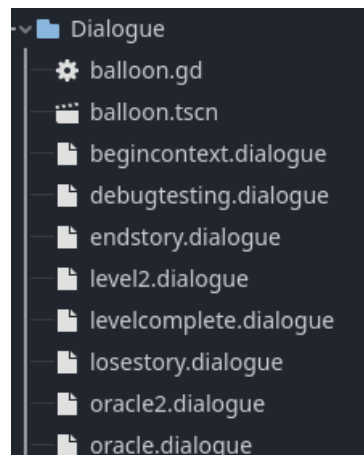Dialogues are stored in ".dialogue" files which are in the Dialogue directory:

Figure 21 – The ".dialogue" files

The dialogue in the respective dialogue files can be viewed by selecting Dialogue option in the navigation panel:
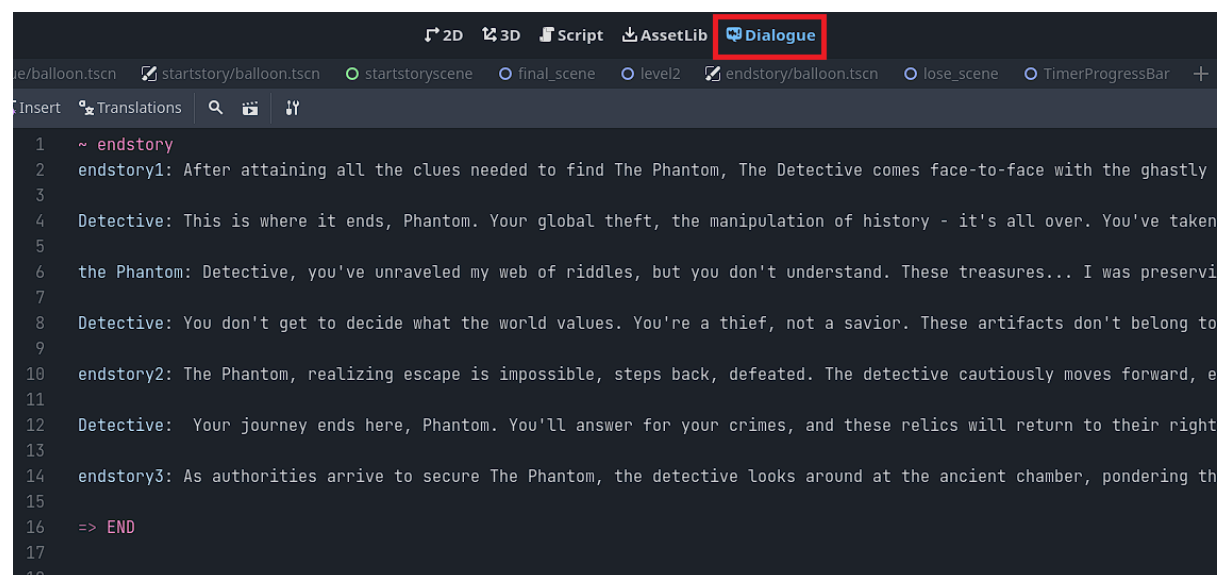

Figure 22 – Dialogue example.

<u>Creating a dialogue:</u>

To create and write out a dialogue, we need to have a node title to signify when the dialogue should be executed, character names to signify who is currently talking, and an end function.

Here is a simple example to code a dialogue:

Figure 23 – Dialogue example.

In the above figure:

- "~ start" – Corresponds to the node title.

- "Detective: Hello" – Corresponds to the Detective saying 'hello'.

- "=>END" – Corresponds to the end function to terminate the dialogue once finished.

Non-linear dialogues:

To create non-linear dialogues to include conditions and options for the player to choose, leading to different dialogue consequences, we just need to add a few more simple syntaxes.

A global variable can be created by creating a script that is an "autoload". An "autoload" script contains variables that can be accessed anywhere in the code. This is also stated in the section "Game State Variables" below.

Here is an example of a non-linear dialogue:

```
19    ~ start
20    Nathan: Hi Cat.
21    if apple_status == "has":
22        Cat: I have an apple.
23        - Give to Nathan
24            set apple_status = "gave"
25            Cat: Here you go
26            Nathan: Thanks
27        - Keep it
28            Cat: But its mine
29            Nathan: Fair enough
30    elif apple_status == "gave":
31        Nathan: Thanks again for the apple
32    else:
33        Nathan: I sure wish I had an apple, theres one over there
34    => END
35
```

Figure 24 – Non-linear dialogue example.

In the figure above, the global variable used is "apple_status". Hence, the "if-else" conditions are executed depending on the state of the "apple_status" variable. Additionally, lines 23 and 27 represent the options that the dialogue gives to the user. Hence, if the user chooses those options, the dialogue below them is executed.

Collision detection and Interactions

We have used a custom-defined 2Dnode which we renamed "Actionable" to handle the collisions and interactions for our game. Consider the following interactable Sakura tree as an example.
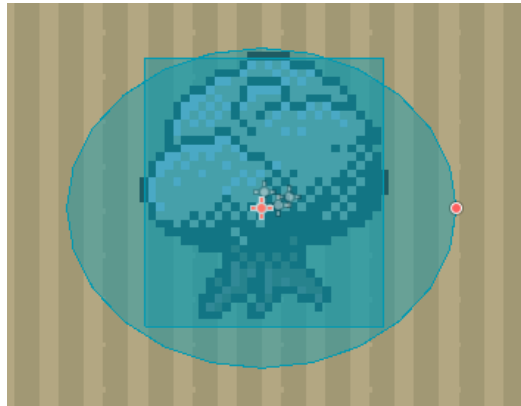

Figure 25 – Interactable Sakura Tree

The Sakura tree is represented as a "RigidBody" node in our scene, and we have attached the "Actionable" node as a child node of the Sakura Tree node.
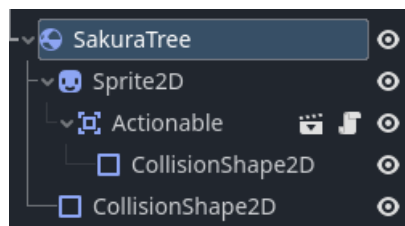

Figure 26 – Node structure for Sakura tree

The actionable node has a "CollisionShape2D" node as a child node which is visible in the above figure as the circular boundary around the tree. This node represents the area which will execute the interaction between objects when the character comes close to it.

The actionable node also has a dialogue resource attached to it, which represents the "action, and what will be executed for that node.
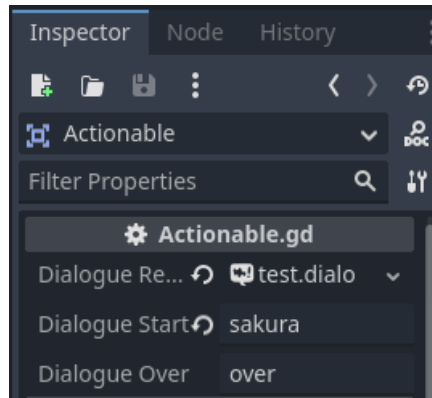
Figure 26 – A dialogue resource can be attached to the Actionable node

Our main character, the detective, also has a "CollisionShape2D" as a child node of a custom node which we renamed to "ActionableFinder". This is represented by the circular boundary as seen in the figure below. This node is used to find the actionable node area of objects and trigger the "action" for that node.
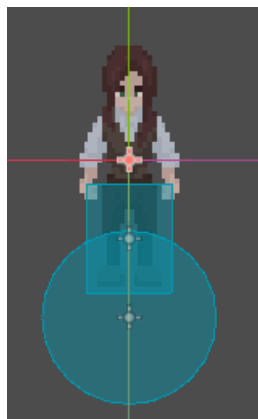

Figure 27 – The "ActionableFinder" shape in front of the Detective

To implement this "ActionableFinder" shape properly, we needed to add this shape to every direction the character is facing. Specifically, this means there was a need to edit the up, down, left, and right directional frames in the "AnimationPlayer" node to include the "ActionableFinder" shape.

In our code and script below, we have implemented a way to detect whenever the "Actionable" boundary for the interactable object and the "ActionableFinder" boundary of the Detective overlap. Whenever the areas bounded by these boundaries overlap and the key (keyboard key "E") to "Interact" is pressed, the action function of the "Actionable" node is executed.

```
func _physics_process(_delta):
    if Input.is_action_just_pressed("Interact"):
        var actionables = actionable_finder.get_overlapping_areas()
        if actionables.size() > 0:
            actionables[0].action()
            return
```

Figure 28 – Implementation for detecting the collisions in "detective.gd"

Making a dialogue appear:

To make a dialogue appear when the character interacts with an object, using the "ActionableFinder" and "Actionable" nodes, we include this into our script for our "Actionable" node.

```
1   extends Area2D
2
3   @export var dialogue_resource: DialogueResource
4   @export var dialogue_start: String = "start"
5
6
7   func action() -> void:
8       DialogueManager.show_example_dialogue_balloon(dialogue_resource, dialogue_start)
9
```

Figure – Actionable Node Script for character and object interaction.

This script means that whenever the character's "ActionableFinder" node overlaps with the "Actionable" node, and the "Interact" key is pressed, the respective dialogue resource will be executed. This is evident in Figure 28, where the "action" function is executed.

Adding portraits to dialogue boxes:

To add portraits to dialogue boxes, first clone the dialogue balloon by going to project – tools – create copy of dialogue example balloon:
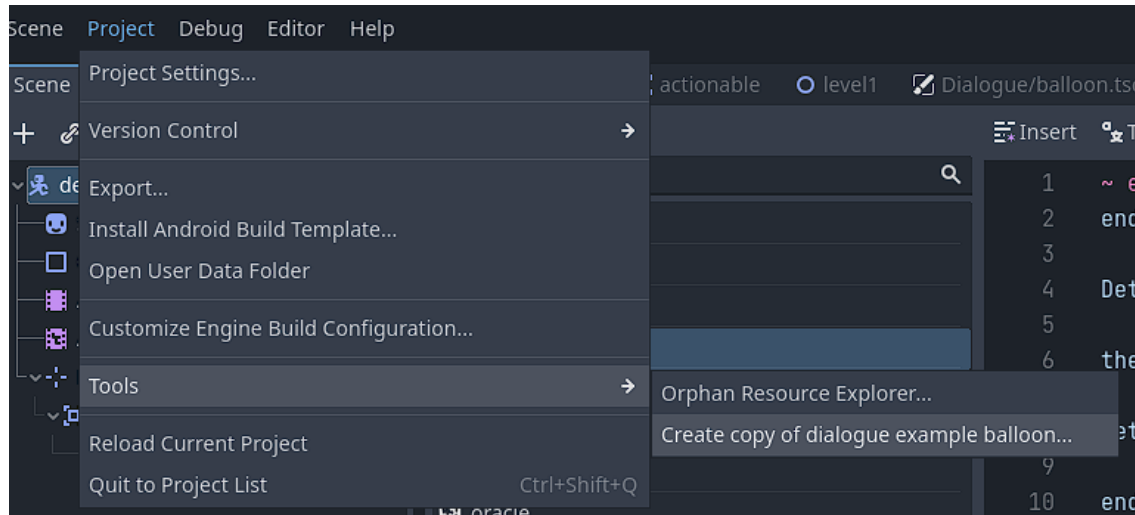

Figure 29 – create copy of dialogue balloon.

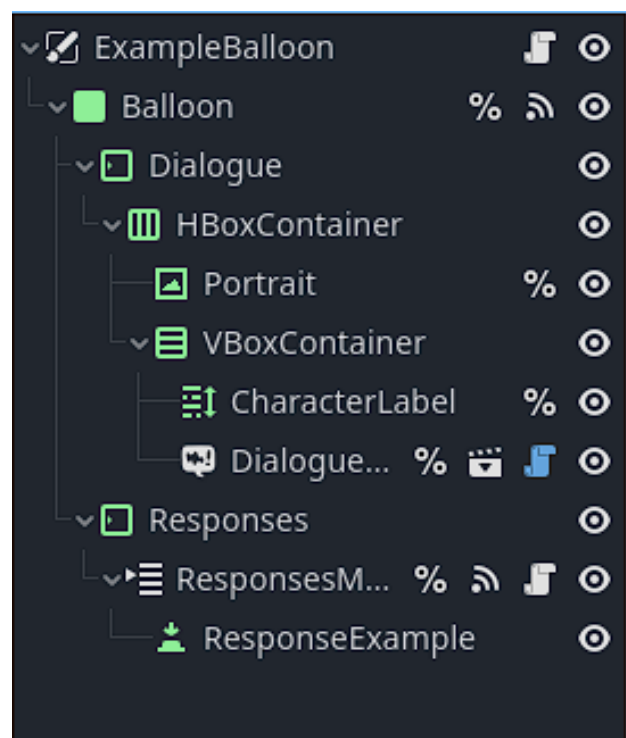After creating a clone of the balloon, modify and add nodes to the balloon in this manner:

Figure 30 – Manipulated nodes to add portrait to dialogue balloon.

In the figure above, the "Portrait" node is renamed from a "TextureRect" node. To adjust the settings to create space for the portrait in the balloon, enable "VBoxContainer's" horizontal container sizing to "expand" in its inspector tab. Then, manually adjusting the size of the portrait is now possible in the "Portrait's" inspector tab, under the layout tab.
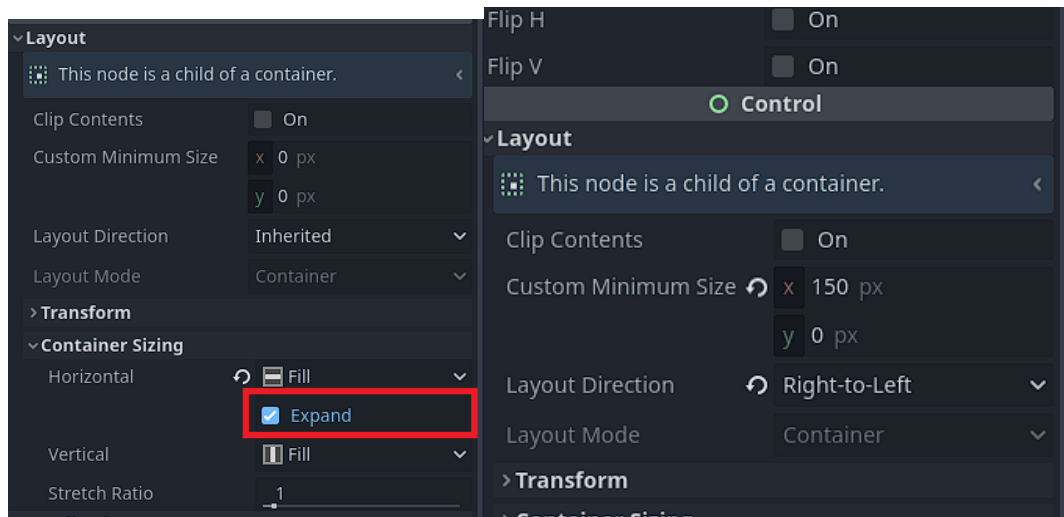

Figure 31 – Portrait and VBoxContainer settings.

To enable the appropriate pictures to appear in the dialogue boxes, include into the script of the balloon these lines of code:

- Include this line as a variable in the beginning of the code:

```
@onready var portrait = %Portrait
```

Figure 32 – Variable in the beginning of the code.

- Include these few lines of code to call the variable and insert the photo in the file.

```
34        character_label.visible = not dialogue_line.character.is_empty()
35        character_label.text = tr(dialogue_line.character, "dialogue")
36        var portrait_path: String = "res://portraits/%s.png" % dialogue_line.character.to_lower()
37        if FileAccess.file_exists(portrait_path):
38            portrait.texture = load(portrait_path)
39        else:
40            portrait.texture = null
41
```

Figure 33 – code to insert pictures as portraits.

In the code given above, it is necessary that the file where the pictures are stored must be called "portraits", as it is stated as the path. Additionally, the picture must be a ".png" and must be renamed to the name of the character that is speaking. For example, if the detective is speaking, the picture should be renamed to "detective.png".

32

Game State Variables

The game state variables are the variables that represent the state of the game at any moment. These variables are used for various decisions in the game flow such as recording the number of correct answers given, the number of incorrect attempts, if the user has cleared the level, etc.

These variables are stored in global.gd file and can be accessed anywhere in the code, as this class is an autoload class and implements the Singleton design pattern.

Figure 34 – The game state variables for level 1 in global.gd

Adding new fonts:

To add new fonts into the game, first download a ".ttf" and add it into the godot game file. Then, on the inspector tab of a node that uses text, drag the ".ttf" font file into the "theme overrides" section in the inspector tab.

Making the camera follow character:

We have implemented the camera to follow the character in the game. To do this, create a child node called "Camera2D" in every level scene of the game. Camera smoothing can be enabled in the inspector tab if you wish.

Then, add a child node for the character (detective), called "RemoteTransform2D". Assign this to "Camera2D" in its inspector tab.

Adding tiles:

"Tiles" make up the scenery of the scene, and it creates the background of the scene. To create tiles, add a child node for the scene of the game level and create a new tile set in its inspector tab. This will create a "TileSet" option at the bottom of the tileset window.

After dragging the desired tile art into the tileset window, clicking on TileMap will allow you to draw on the scene to create the tiles:
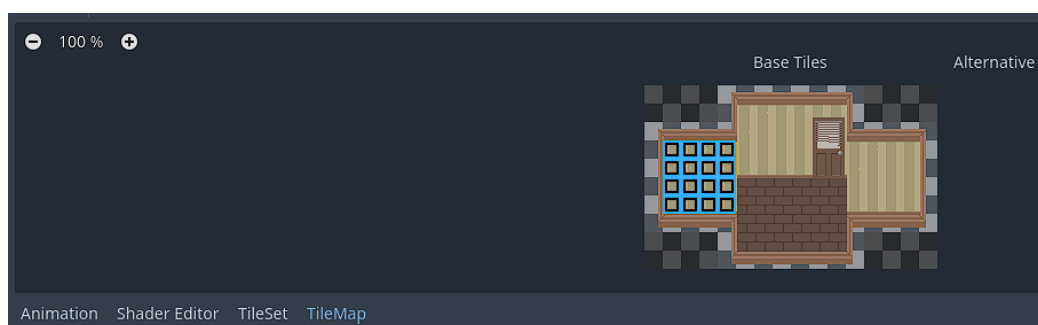

Figure 35 – TileMap

TileMap layers:

To layer tiles above the base, we can create multiple layers by clicking on the top right dropdown menu of the TileMap:

Figure 36 – TileMap layer menu

Credits:

This game includes many materials and resources sourced from various artists and entities. We acknowledge these creators:

Asset Packs:

1. *Private investigator office asset pack*: Created by KY.Pixel, https://kypixel.itch.io/private-investigator-office-asset-pack

2. *Egypt Environment – Assets pack:* Created by KimSsuki, https://kimssuki.itch.io/egypt-enviroment

Music:

1. *Chi Dori: 18ᵗʰ Century Kengyo Yoshizawa*: Composed by Shinici Yuize, https://open.spotify.com/track/56GFCPu8xiN3aCZS5Qmatq

2. *"Salah" – Traditional Egyptian Folk Song*: Composed by Michael Levy, https://open.spotify.com/track/3RiguC4RfuJ6PaK9n0RrMb

Fonts:

1. *Pixel 12x10 Font*: Created by Corne2Plum3, https://www.fontget.com/font/pixel-12x10/#

Godot Addons:

1. *Dialogue Manager*: Created By Nathan Hoad, https://github.com/nathanhoad/godot_dialogue_manager

Portraits used in Dialogue Boxes:

1. *Image 1 of beginning story*: OpenAI's ChatGPT and DALL-E
2. *Image 2 of beginning story*: OpenAI's ChatGPT and DALL-E
3. *Image 3 of beginning story*: OpenAI's ChatGPT and DALL-E
4. *Image 4 of beginning story*: OpenAI's ChatGPT and DALL-E
5. *Image 5 of beginning story*: OpenAI's ChatGPT and DALL-E
6. *Image 1 of ending story*: OpenAI's ChatGPT and DALL-E
7. *Image 2 of ending story*: OpenAI's ChatGPT and DALL-E
8. *Image 3 of ending story*: OpenAI's ChatGPT and DALL-E
9. *Sakura Tree:* OpenAI's ChatGPT and DALL-E
10. *The Detective*: OpenAI's ChatGPT and DALL-E

11. *The Oracle*: OpenAI's ChatGPT and DALL-E

12. *The Phantom:* OpenAI's ChatGPT and DALL-E

13. *Ishikawa Goemon:* Toyokuni Utagawa,
    https://www.mediastorehouse.co.uk/granger-art-on-demand/world-history/ishikawa-goemon-1558-1594-legendary-japanese-12418366.html

14. *Execution of Ishikawa Goemon:* Utagawa Kunisada,
    https://en.wikipedia.org/wiki/File:Excecution_of_Goemon_Ishikawa.jpg

We express gratitude to these creators and entities that have helped enhance our game with their amazing work.

References

Angels of Venice (1999). Queen of The Sun. On Angels of Venice [Audio file].
    Retrieved from https://open.spotify.com/track/4TM4XezSpqYzdeVA8B4lFd.

Corne2Plum3. (n.d.). Pixel 12x10 [Font]. Retrieved from
    https://www.fontget.com/font/pixel-12x10/

Hoad, N. (n.d.). godot_dialogue_manager [Software]. GitHub. Retrieved from
    https://github.com/nathanhoad/godot_dialogue_manager

Kimssuki, n.d. Egypt environment - assets pack by kimssuki. Available from:
    https://kimssuki.itch.io/egypt-enviroment.

Kunisada, U. (Mid 19th century). *Goemon Ishikawa and his son Goroichi* [Image].
    Retrieved from
    https://en.wikipedia.org/wiki/File:Excecution_of_Goemon_Ishikawa.jpg

KYPixel, n.d. Private investigator office asset pack by ky.pixel. Available from:
    https://kypixel.itch.io/private-investigator-office-asset-pack.

MOJI. (2020). 懐石. On 日本料亭で流れてそうな和風 BGM [Audio file]. Retrieved from
    https://open.spotify.com/track/3v4TtSlf7UM2gqNJBvvbpk

OpenAI. (2023). [Image 1 of opening dialogue]. Image generated upon request using
    OpenAI's ChatGPT with DALL-E.

OpenAI. (2023). [Image 2 of opening dialogue]. Image generated upon request using
    OpenAI's ChatGPT with DALL-E.

OpenAI. (2023). [Image 3 of opening dialogue]. Image generated upon request using
    OpenAI's ChatGPT with DALL-E.

OpenAI. (2023). [Image 4 of opening dialogue]. Image generated upon request using
    OpenAI's ChatGPT with DALL-E.

OpenAI. (2023). [Image 5 of opening dialogue]. Image generated upon request using
OpenAI's ChatGPT with DALL-E.

OpenAI. (2023). [Description of the image]. Image generated upon request using
OpenAI's ChatGPT with DALL-E.

OpenAI. (2023). [Strange Object]. Image generated upon request using OpenAI's
ChatGPT with DALL-E.

OpenAI. (2023). [Oracle]. Image generated upon request using OpenAI's ChatGPT with
DALL-E.

OpenAI. (2023). [The Detective]. Image generated upon request using OpenAI's
ChatGPT with DALL-E.

OpenAI. (2023). [The Phantom]. Image generated upon request using OpenAI's
ChatGPT with DALL-E.

OpenAI. (2023). [Image 1 of ending dialogue]. Image generated upon request using
OpenAI's ChatGPT with DALL-E.

OpenAI. (2023). [Image 2 of ending dialogue]. Image generated upon request using
OpenAI's ChatGPT with DALL-E.

OpenAI. (2023). [Image 3 of ending dialogue]. Image generated upon request using
OpenAI's ChatGPT with DALL-E.

OpenAI. (2023). [Sakura Tree]. Image generated upon request using OpenAI's ChatGPT
with DALL-E.

Utagawa, T. (circa 1827). *Legendary Japanese outlaw, as portrayed by an actor* [
Woodcut]. Media Storehouse. Retrieved from
https://www.mediastorehouse.co.uk/granger-art-on-demand/world-history/ishikawa-
goemon-1558-1594-legendary-japanese-12418366.html