Tampere University of Applied Sciences

Making a 2D Game with Godot 4 Engine

Son Nguyen

BACHERLOR'S THESIS
May 2024

Bachelor's Degree Program in Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Bachelor’s Degree Program in Software Engineering

Son Nguyen

Making a 2D Game with Godot 4 Engine

Bachelor's thesis 30 pages, appendices 0 pages
May 2024

This thesis goes into how a game is made using the Godot 4 engine, and open-source game engine. The author will focus on mostly theory behind how to make a game, decisions such as how to choose a theme for the game, what kind of tools to pick, and how to design it. Afterwards, a game will be made, keeping in mind with the choices above.

The goal of the thesis is to use Godot as a tool to implement aforementioned requirements to a game called “Shoot the Spy”, to apply some functions provided by Godot, and to study basic concepts of a game, such as movement, inputs, and response.

Key words: Godot, game development, 2D, game engine

CONTENTS

ABBREVIATIONS AND TERMS

HUD	Heads Up Display
2D	Two dimensions
3D	Three dimensions
UI	User Interface
FPS	First-person shooter
RPG	Role-playing game
PC	Personal computer
AI	Artificial Intelligence
Godot	An open-source game engine
Open-source	Software with freely accessible and modifiable code

1 INTRODUCTION

Clash of Clans, Arknights, Legend of Zelda, Bloodborne, Devil May Cry, League of Legends, Fortnites, Minecrafts are all too familiar names to gamers, yet they have nothing in common. The diversity of genres, platforms and engines that they run on are testament to the wild success of video games. This has no doubt, affect and inspired a lot of players to try analysing and bisecting what is so great about their favourite games like an art piece that can evoke emotions deep inside. And with that, players will become developers, attempt to capture and create their own wonders and share it back to the gaming community.

Realising the potential of these new aspiring developers, companies and communities made and release their own engines for graphic, sound, and art. To name a few, those are Unity, Godot, RPGMaker, CryEngine, UnrealEngine. While all of them are amazing and have their own unique strengths, Godot stands out as being an open-source software instead of propriety like the others. This make developers who are starting out an ease of access. Furthermore, the engine is continuously being improved under the efforts of other likeminded developers.

With the choice of engine now being Godot, this thesis goal is to shed lights on game usability, design choices, what tools are there to create your game, Godot features, and the implementation of “Shoot the Spy” project.

Hopefully with this, the author can channel some of the passions to developers that would someday publish their games.

2 Consideration of game development

2.1 What is a video game?

A game can be universally defined as a form of entertainment, an interactive kind, which mean it engages with players activity using rules that will imposes challenges, with the goal of achieving something rewarding. Game requires strategic thinking, competition, skills, and chance. There are genres of games, such as boardgame and card game. The genre is often defined by the medium of which the game is being played. A video game is just a digital construct of a game. What make it different from kinds is that it can produce sound, video, and sometimes even smell and kinetic feedback to player's action.

While a game is often made to have fun, it can also be educational, or enhance cognitive and motor skills. Video games, with the advantages that they have, can further boost the effectiveness of such purposes. The motivation to make game is often enjoyed fuelled by creating more creative experience above.

2.2 Picking a demographic

As with everything, a product needs to be recognized by its customers, else it will serve no purpose but to exist. Video games have thrived for multiple years, which, the gaming community have come up with many rules and conventions that should be followed due to optimisation and improvement to make a game better. Of course, many players have picked up on these habits of game developers and got used to it, to the points that everyone knows what to expect when it comes to certain genres, they don't even need tutorials to know the rules, or how to control the playable characters, according to Katherine and Noah (2008, 145). Examples are, FPS gamers can flick their mouse with speed and accuracy even in a new title, or RPG gamers can optimize a playstyle with just a few minutes looking at a new game skill tree, a mechanic that allow players to become stronger as they play the game. These players can be considered hard-core players.

And we also have the other end of the spectrum, the casual gamers. These are players who don't invest a lot of time to hone their skills in video games. They rarely play games, or they just don't think a lot about video games. They will pick up a new title, finish the game in multiple short sessions. They take their time to enjoy a game. They also often prefer playing game on mobile, as it is easier to set up and play compared to playing on PC. This notion is backed up by John Welch during a Game Developers Conference in 2008. It is safe to assume that most players are casual.

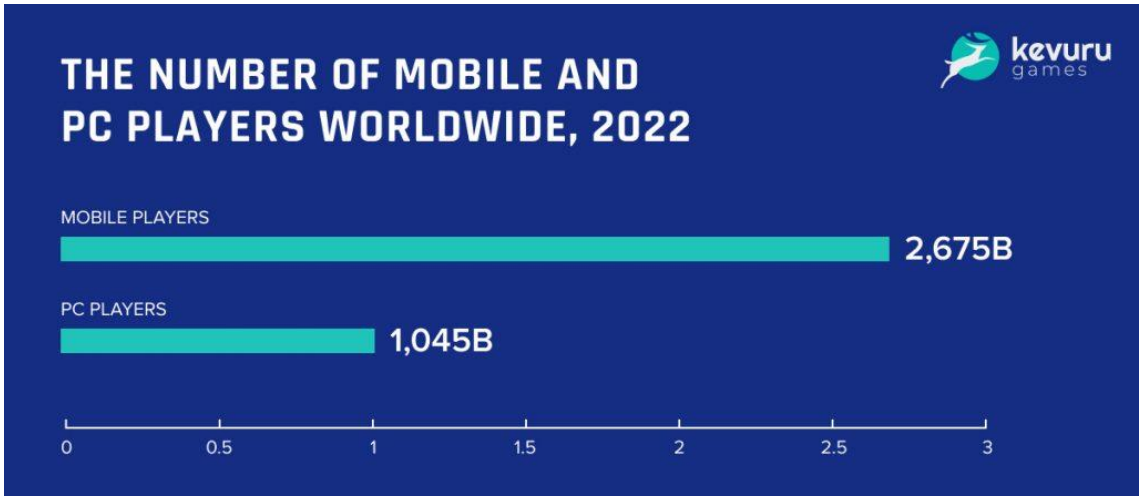

Figure 1: Number of players on each platform, 2022.

2.3 Design the game feel for each demographic.

According to Katherin and Noah (2008,273), game feel is how the gameplay and design of a game make a player feel. To further explain this in detail, they established a concept called gameplay garden. This consist of:

- Input: The activities that the player provides.
- Response: How the game responds to the input.
- Context: The context of the activities.
- Polish: The little details of the game.
- Metaphor: Provide emotional meaning and familiarity to the game to mitigate learning frustration.
- Rules: Application and/or tweaking of arbitrary variables to give challenges to player.

With this concept set up, now it can be applied to each demographic. Here are just some examples provided to guide the developers to make a better game for each demographic, not a set-in stone rule to be followed, creativity is really important.

2.3.1 Hardcore gamers.

As established above, hardcore gamers are quick to learn, they like to optimize, to trade information, making guide, keeping reputations with other players in the community. In essence, they want to be challenged (Dan, 2009). And so, rules of the game can be tweaked to make the game harder, such as making the enemy health higher, increase the speed of enemies, lower player health, add more enemies. It is also important to provide context and metaphor to the player, such as make the player have to wait less after each failure, so they can try again and perfect the action.

2.3.2 Casual gamers.

Casual players would have less time and patient to reach the failure state again and again like the hardcore do. Different parameters can be set, such as having a more exaggerated response to the player input to communicate clearly the consequences of each input. Polishing the game can have great benefits for the casuals, as they can be easily impressed, increasing their retention in doing so.

2.4 Designing the fun zone

People play games to have fun, that goes without saying, this is, after all, a form of entertainment. But how exactly do players have fun? People have different preferences, and there are multiple ways to enjoy something. Take ice cream for example, there are multiple flavours, vanilla, chocolate, with even more ways to present them, chocolate wrapped in vanilla, cone, stick… In order to solve this question, Nicole Lazzaro has devised a design called the four fun keys (2008, 317). These reasons of how games are played will help developers lock on the design flow that they want.

The first key is called hard fun. Essentially, it focuses on the players who loves mastery, challenges, and strategies. Perhaps players that enjoy this category can resonate with time limit, high scores, and puzzle.

The second key is called easy fun. Some players just prefer looking for details, iconic stories, satisfy their curiosity and ambiguity of the game. Game with cute art style, having dialogues or interactions between characters might be the design that could be aimed for these players.

The third key is called serious fun. Rhythm, practices and stimulations are what drives these players. A shortcut button for a quick level reset, rhythm game, simple controls could be effective designs.

And the last key is called people fun. Just like in real life, players want to socialize and be amused by other players. They could be competing with other players for a goal, being cooperative and help each other get through a challenge, maybe just a forum to share a result is the way to do this.

3 Choosing the tools for developments

3.1 Game engine.

At the start of this thesis, it has been mentioned that Godot would be the engine of choice for this project, that is just an example and what would fit the author's and the project's requirements. In fact, choosing the game engine first before thinking of the game that should be made is a bad idea. Game engines have limitations, and their strength respectively, it should be avoided to have a situation where a game engine is not suitable for the game that should be developed. Example would be making a 3D game with RPGMaker or making a 2D dating game with Unreal Engine. Now, the author will go over some of the game engines, with their strengths and weaknesses to help the developers choose their ideal engine for the game.

3.1.1 Godot

Being a free and open-source software, Godot is accessible to all developers of all levels. It is also lightweight and easy to pick up. For the type of game that it is most suitable for, it would be 2D games while 3D games are also supported for flexibility. However, large and complex game like a 3D game would be slowed down significantly due to limitation, and it does not have the best asset store available.

3.1.2 Unity

Unity is one of the most popular game engines, that make it worthwhile to pay and learn Unity. Not only that, wide adaptation means it has a huge array of assets available. Another strength that it has over other engine is multiplatform support. But, it can be costly to maintain a Unity license and the complexity of Unity is also something to be considered.

3.1.3 Unreal Engine

Being one of the best engines, if not the best, at supporting high-fidelity graphics, Unreal Engine has a specialty in supporting 3D games. But that would also mean requirement to run and play Unreal Engine games are high, and the learning curve is also high.

3.2 Graphical software.

If the developers are not interested in assets store, or if there were nothing of interest, they would need to create the assets themselves. To do that, graphical software is required. This thesis won't go in depth on how to create assets, as that does not concern the usage of the game engine, so some recommendations will be mentioned here quickly.

First off, Krita is a free and open-source painting software that has the UI of old Microsoft Paint with some improvement and modernization. Many old school artists and new artists can quickly pick this software up and learn it. Secondly, Blender is the go-to choice for creating 3D assets, due to wide adaptation, frequent update, and great length of documentation.

3.3 Sound engine.

Having soundtrack and sound effect are almost a requirement to making a video game. As with graphical assets, developers can find non copyright sound and sound assets store to use for their game. Creating sound assets would also be out of scope of this thesis. Some recommendations for sound engines are below.

Fmod is one of the top choices from independent game developers due to its integration with game engine such as Unity and Unreal Engine while also supporting wide range of gaming platforms. One of the other top choices would be Wwise, with ability to do real-time audio profiling and highly scalable.

4 Application to real project.

4.1 Design choice.

“Shoot the Spy” aim to be widely appealing, simple, but can be fun to master. So, the target audience would be the casual gamers, as casual game can be made for people of all ages and genders (Katherine and Noah, 2008, 143). This means the design of the game would try to be simple, in art style, control scheme, and gameplay. “Simplicity and clarity is the way of casual game” – Katherine and Noah (2008, 154).

4.1.1 Case study of other casual games.

We become what we behold by Nicky Case is a great example of a casual game. Its stickman art style is easy to comprehend what is happening at all time in the game, minimal HUD communicating only the necessary information, and only using mouse as the control make the players learn the game quickly.

Shogun King: The Final Checkmate by PUNKCAKE is another great example of a casual game with depth. The art style is of a chess website. Its rule is based on chess, making it easy to learn for most people who have played or understand chess. The UI is clear, and the control is also just using mouse click.

4.1.2 Apply to the game.

As with the two examples from above, and many others, the author chose to use only the mouse as the control scheme for the game. Furthermore, the two examples above and Katherin and Noah (2008, 148) seem to agree that only using the left mouse button is superior to any other type of control for casual game. It is even said that using the right mouse button is a taboo for casual game. A great case for this is the game Zuma, the game has a right mouse button function, along with a main left mouse button one. Player can totally beat the game without

using the right mouse button, so many did not even know the function even existed.

With art style, the author chose to use the stickman way. It is easy to recognise, easy to figure out what is happening on the screen. Simple and clear.

About HUD, keeping with the law “simple and clear”, only necessary information will be conveyed.

4.1.3 Decide the type of game that will be made.

As a simple casual game, the rule should be clear from the beginning. As the name suggests, “Shoot the Spy”, players will have to find and shoot the spy. There will be a spy stickman blended in with the other non-related stickman. The HUD will only show the target player have to shoot, score counter, and a timer. Start and game over menu would only show the play button, and the achievement that the player has done.

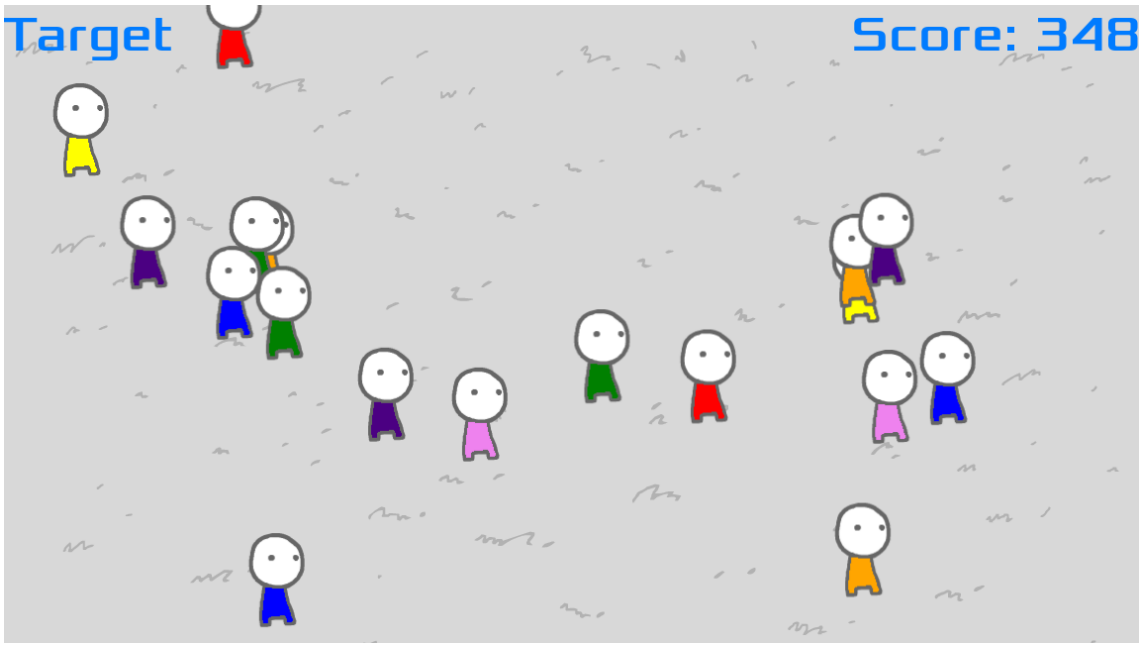

Figure 2: A screenshot of the style.

The specific rule of the game will be as follows: There will always be a target on the screen. There will be multiple other non-target characters that will look different than the target. The target will blend in and try to escape through the screen

border. The player will try to shoot the target using only the left mouse button. The faster the player can shoot the target, the higher the score. The goal of the game is to get the highest score possible. The fail state of the game is when the player shoot the wrong target, or the target managed to escape.

4.2 Implementation.

With the type of game defined, the thesis will now go into detail of how to use Godot to develop the game. The engine has an intuitive UI that allow 2D and 3D design, its own compiler and development environment that support coding with C# and integrated scripting language called GDScript. In this project, 2D UI of the engine will be utilized and GDScript is chosen for coding, as it is said by the developers to be made specifically for the Godot engine, so it will be easier to work with, as well as faster to run.

For sound and graphic design, the author used non copyright sound for audio, and used Krita, as well as free assets to create and modify the graphics for the game.

4.2.1 Make the character.

First, to make an asset, it has been pulled from a store or drawn. The author used Krita to draw the character sprite, using Nickey Case's sprites as inspiration. Since the game require picking the target from a crowd, using colour theory might help with the aesthetic and help picking out the characteristic of the spy faster. Brent (2005, 44) said colour wheel is a powerful tool, given that player can have the afterimage effect for looking at a colour for too long, and the game will have more soul for containing more colours. With that, the characters will now have all seven colours of rainbow, as they naturally complement each other. Next, the character needs to be created in Godot, animated, and given logic.

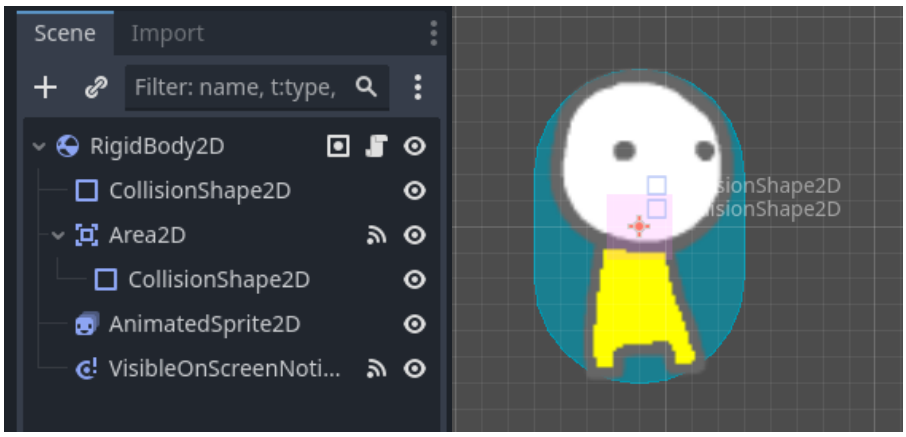

Figure 3: The Scene of the character.

In Godot, most objects are created using a Scene. To create a Scene, click on Scene and New Scene. Inside the Scene consists of Nodes. In all the Nodes that the engine support, RigidBody2D is chosen by the author for it could contain the drawing or animation of a character or object, boundary that can be set up and controlled or interacted with, handle logic applications. To do so, add the supplementary nodes in Figure 3 to the RigidBody3D node using the “+” shape.

Animations are usually created to make an object, a character, or a scene look more alive and interactive. Animations are a bunch of fast-moving pictures, move in a sequence in a short interval to create the illusion that something is moving. And so, to make something seem like moving, a bunch of succession pictures, or frames, must be drawn and aligned. The way to create 2D animations in Godot will be written in the paragraph below.

Firstly, add a AnimatedSprite2D node to the original node. Find the Sprite Frames property under the Animation tab in the Inspector and click "[empty]" -> "New SpriteFrames". The result should look like Figure 4. Click on the SpriteFrames you just created to start making the animation and a new UI tray would appear from the bottom of the application. Click on the “default” tab, the developer can now drag the images that would make an animation, in order, to the tray. A developer could also create multiple animations by simply pressing the “Add Animation” on the top left of the tray, or pressing Ctrl+N. The result should now look like Figure 5. Developer could also preview the animation using the buttons in the tray.

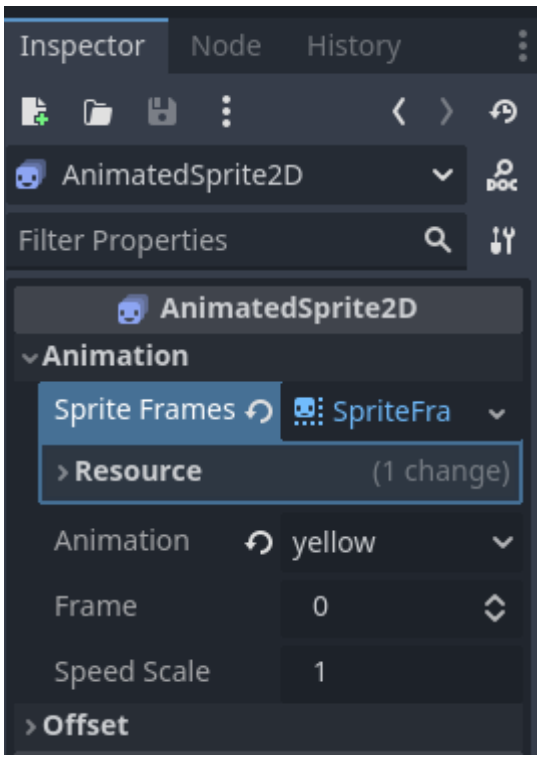

Figure 4: Adding sprite frames.

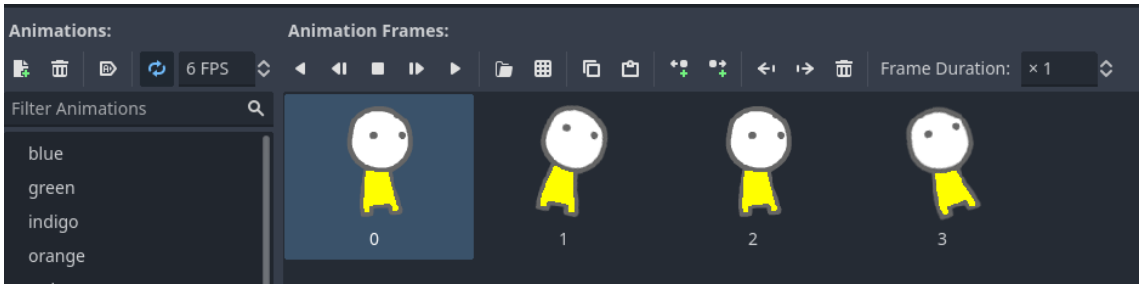

Figure 5: Creating the animation.

Next, to add logic for the character, as well as to run the animation, a script needs to be attached to the RigidBody2D node. Just right click on the node and press “Attach Script…”. With this, the character can now be alive.

To run the scripts in this project, first GDScript need to be understood. GDScript is a scripting language that has similar syntax to Python, making it relatively easy to learn for both beginner and experienced developers. Reference to the language documentation can be found in this report. Refer to Figure 6 to see an example of how AnimatedSprite2D can be called and played. In this function, the goal is to run a target animation. The author has created seven animations, one was chosen and is passed as the “animation” parameter. Variable “mob_type” is an array of names of the animations in Figure 5. Using the “$” symbol, the node

and its methods can be called. Finally, the animation is played, the target is marked for the player to shoot, and the entire object can be shown.

```gdscript
func run_target(animation):
    # Get the array of names of animations
    var mob_types = $AnimatedSprite2D.sprite_frames.get_animation_names()
    # Play the selected animation
    $AnimatedSprite2D.play(mob_types[animation])
    # This is the target that need to be shot
    killable = true
    # Show the target animation
    show()
```

Figure 6: Example of how to call other node inside the mother node and play the animation method.

And now to handle the clicked logic, the author used Area2D node. This node, along with RigidBody2D node, both require a CollisionShape2D node. What that do is to define the area that physics will take place on the body. Node in Godot could emit “Signal” based on the event that happened. Area2D can emit a signal when there is an input inside of its CollisionShape2D. Using that, the author can detect a click on the character.

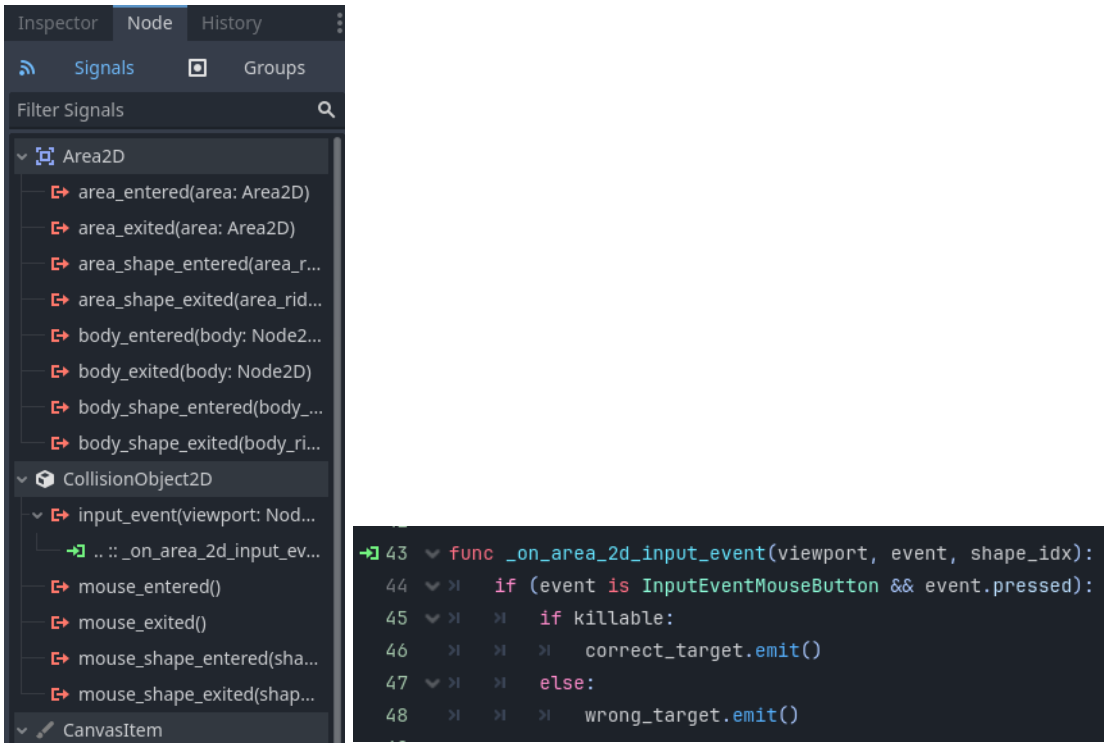

Figure 7: Example of Signal in Godot.

Same principle applies to the VisibleOnScreenNotifier2D, when the target escaped, the node will send a signal, despawn the object to free up space, and to announce that the player has lose.

4.2.2 Designing the HUD.

Brent Fox (2005, 69) said to keep the design simple, avoid clogging up the mouse, avoid providing too many options, so that it can be easy to scale and look pleasing. To do that, the author has decided to use the least amount of button possible, and to show very little to the player. He also mentioned using big composition on important element, such as make the “Play” button, the title screen and end game screen larger.

To make the HUD elements, the author used CavasLayer2D node, renamed “HUD” in Figure 8, to manage other UI elements with game logic. Add a script to the CanvasLayer2D node like when it is done to the character RigidBody2D node. Like Figure 8, add a Label node, a Button node, and a Timer node, these nodes will serve as the game start UI and recurring elements. Make 2 other CanvasLayer2D, with each having 2 Label like Figure 8, as the game need UI for in-game state, when the game is being played, and a game over state, when the player failed.

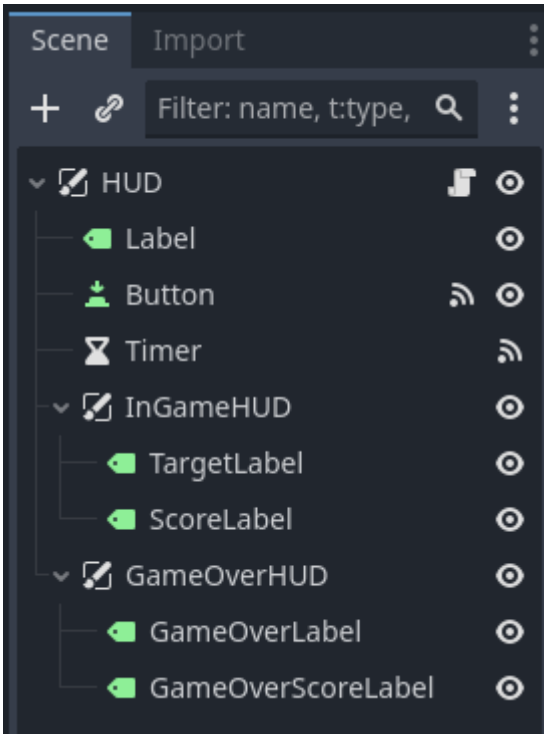

Figure 8: HUD Scene.

After attaching a script to HUD node to handle logic. To communicate that the button has been pressed, it will emit a _on_button_pressed() signal and will be caught by the scene that will handle the main game later. The timer is made so that every time the game reset, it gives the player some seconds to get ready. The time and property of the timer, loop or no loop, can be edited on the “Inspector” tab just like the AnimatedSprite2D example above.

In Figure 10 and 11, the game start scene and game over scene are shown. These elements can be dragged and dropped, sketched and edited using the 2D view of the Godot application on the top side, as well as the Inspector tab on the right side of the screen. To edit the font, colour, and size of the text, these options can be found in Control -> Theme Overrides as in Figure 12.

Figure 9: 2D view of all the HUD elements.

Figure 10: Start game UI.

Figure 11: Game over UI.

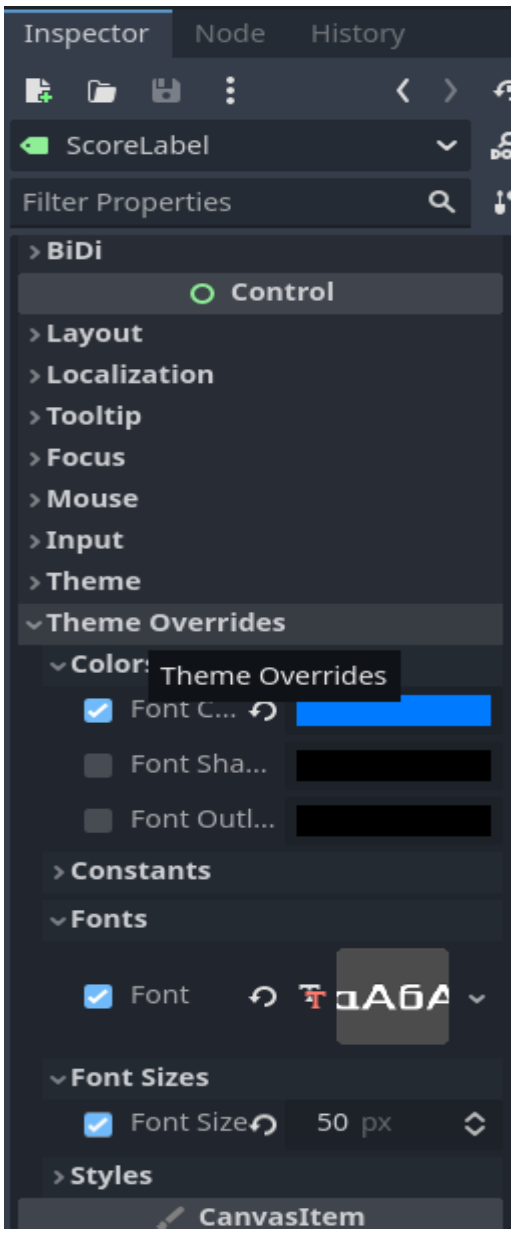

Figure 12: A Label Inspector tab, with text colour, font, and size settings.

4.2.3 Making the main game Scene.

Finally, the logic of the main game can finally be discussed. As Godot handle the game using Scenes, a new one is required to handle all the logic while running the game. A Node node, renamed “Main”, is created, since it can house every other node of the engine, this way, the author could add any element that may sprung up later in development. The other nodes are as follows:

- Path2D: Renamed “Spawn”, this node to mark the creation points of every characters.
- PathFollow2D: This is the path that the characters have to take.

- HUD: In the previous section where we created the “HUD” Scene, this can be applied by dragging and dropping the Scene from the work tray in the bottom left corner of the Godot application.
- Marker2D: Renamed “TargetHUDPosition”, this is where the image of the target is shown, so the player knows who to shoot.
- TextureRect: Renamed “Background”, this is an image of the background of the game.
- AudioStreamPlayer2D: Renamed "CorrectSound” and “FailedSound”, this is the audio feedback to the action of the player.

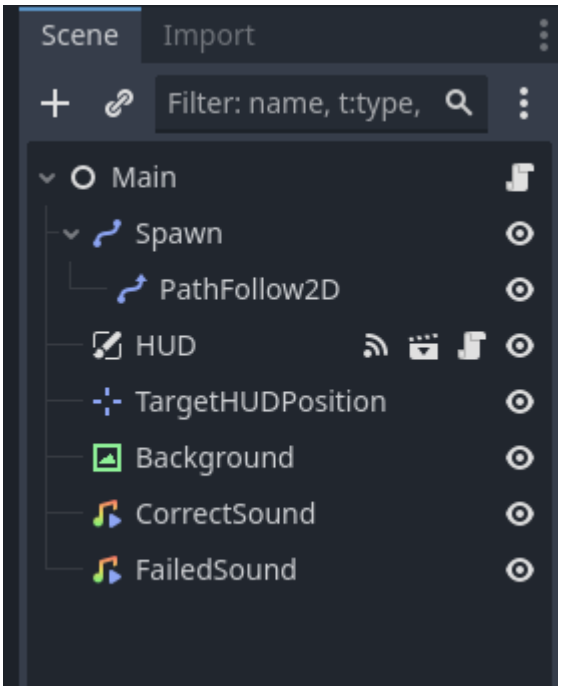

Figure 13: Main game Scene.

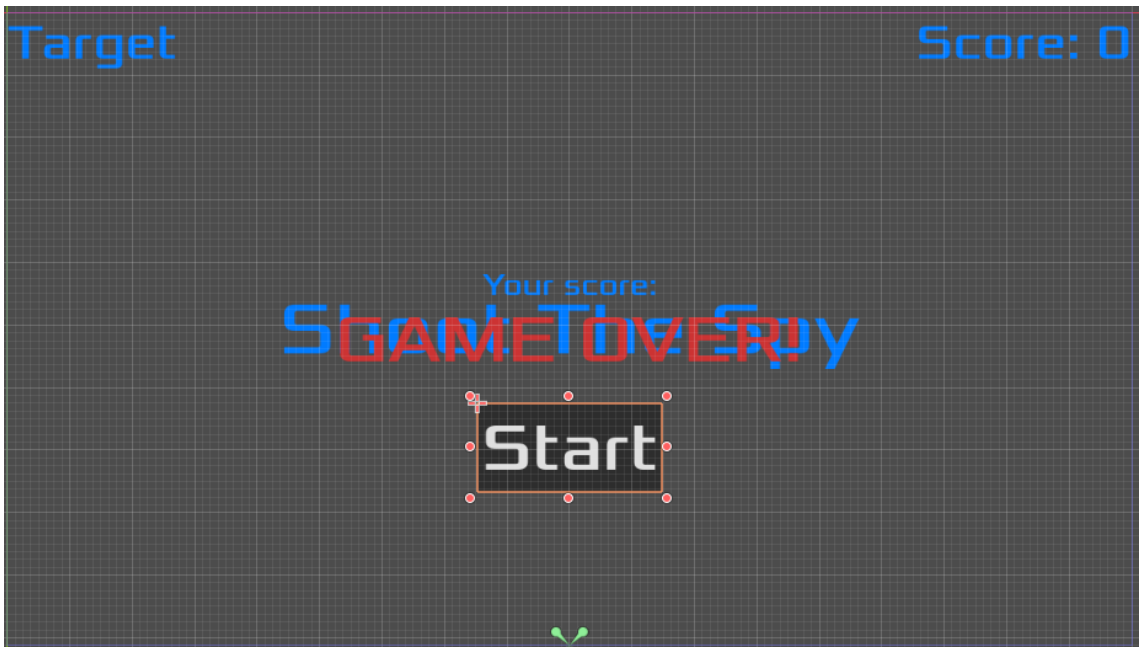

Figure 14: 2D view of the Main Scene.

To handle the main game, an empty node (renamed “Main”) is created and attached a script. This node will handle all the logic in the game. In Godot, a Scene can be a child of another Scene, just drag the HUD scene made from before under the Main Scene, and now all the properties of the HUD, as well as the signal emitted, can be caught by the Main node. Next, to the handling of characters. Path2D (renamed “Spawn”) and PathFollow2D is drawn on the map to indicate the cluster of spawn locations as well as the direction that the characters will move. Usually, the characters cannot leave the PathFollow2D drawn, but the function in Figure 15 has changed its direction, making the direction as well as movement speed more unpredictable. On that topic, the game also has a few variables that would make the game more interesting and harder as the player get more accurate. Those variables are “score” and “level”. Level would increase by 1 as the player get the correct answer, the amount unrelated targets would increase (handled in Figure 17) and the speed that the target would escape is also increased (handled in Figure 15). As for the score, the goal of the game, the faster the player pick up the target, the higher the score award to the player on that round (handled in Figure 16).

```
func spawn_mob(avoid_target, killable):
    # Instantiate the mob
    var mob = mob_scene.instantiate()
    # Get the set of locations the mob can spawn on
    var mob_spawn_location = $Spawn/PathFollow2D
    # Randomize the location
    mob_spawn_location.progress_ratio = randf()
    # Randomize the place the mob will go to
    var direction = mob_spawn_location.rotation + PI / 2
    # Place the mob in location
    mob.position = mob_spawn_location.position
    # Direct the mob
    direction += randf_range(-PI / 4, PI / 4)
    # Randomize the speed of the mob based on difficulty
    var min_speed = 100 + level * 10
    var max_speed = 200 + level * 10
    var velocity = Vector2(randf_range(min_speed, max_speed), 0.0)
    mob.linear_velocity = velocity.rotated(direction)
    # Spawn/Render the mob
    add_child(mob)
    # Handle the kill logic, whether or not the player hit the correct target
    if killable:
        mob.run_target(avoid_target)
        mob.connect("correct_target", _on_correct_target)
        mob.connect("escaped", _on_target_escaped)
    else:
        mob.run_civ(avoid_target)
        mob.connect("wrong_target", _on_wrong_target)
```

Figure 15: Handling of character logic.

```
# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
    # Stopwatch using delta
    # The faster the target is picked, the higher the score
    if newLevel:
        var thisRoundScore = 1/timeToShoot * 100
        score += int(thisRoundScore)
        timeToShoot = 0
        newLevel = false
        $HUD.update_score(score)
        new_round()
    timeToShoot += delta

# Signals from packed scene
func _on_correct_target():
    level += 1
    $CorrectSound.play()
    newLevel = true
```

Figure 16: Difficulty handling logic.

```gdscript
# Game logic
func new_round():
    # Despawn everything, fresh start
    get_tree().call_group("mobs", "queue_free")
    # Get the target
    target = randi() % 7
    print(target)
    # Spawn the target picture
    spawn_dummy(target)
    # Spawn target
    spawn_mob(target, true)
    # Amounts of civ to spawn
    var civs = level * 2 + (randi() % 5 + 1)
    # Spawn civs here
    for i in range(civs):
        spawn_mob(target, false)
```

Figure 17: New round logic.

Now that most of the logic is handled, the other nodes can now be addressed. The “Background” node is just an image that can be added using the Inspector tab while hovering on the node. The “TargetHUDPosition” is a 2D array that can be allocated using Inspector tab, or by drag and drop on 2D view. The position can be accessed and given to the target as seen in Figure 15. The HUD node can be handled however the developer feel most artistic, as it does not require complex methods or functions. The authored designed the HUD with 2D view, and using hide() and show() method as a simple method of making the UI elements to disappear and appear.

4.2.4 Reflection and refine the game.

Now that the game has been made, the author decided to reflect upon their own criteria. The target demographic is casual gamers, so the fun zone should be focused on easy fun, and potentially, lightly touched upon other zones. The project does not fully comply to the example shown in section 2.4 when describing the easy fun zone, however, that can be improved by swapping the graphics out with a better, more cute, more detailed art style. The game also took a bit from

the serious fun zone, with a simple rule and a quick button to restart the game and start the game again, make mastering the game simpler.

The game seems to run smoothly, but it felt dry, non-responsive. To fix this, the author considered using audio and graphic to respond to the mouse click. To add the audio, add two AudioStreamPlayer2D nodes like Figure 13, these nodes will play a sound when player click on the correct or wrong target. To attach the sound, use the Inspector tab, then the audio can be played using the play() method and stop using the stop() method in the script. As for the graphical aspect, the author decided against it, as it would clutter the screen and may block the target from the player view. Perhaps there are better solutions, but the author is not experienced in solving this question.

The background at the first iteration of the project was a grey wall, it makes the game look less appealing. But because of the choice of colours for the characters, using all 7 shades of rainbow, the only colour that would compliment them all is the grey colour. To improve this, the author decided to use white background, with grey shades as noise objects, such as grass, to make the background look better. The background is just an image drawn using Krita, and it was added to the TextureRect node.

While the control of the game is only using the left mouse button, following the principle of casual game, the author decided to add a shortcut button to make restarting the game somewhat faster. The reason for this is to encourage mastery of the game, the serious type of fun, as described in section 2.4.

First, go to Project -> Project Settings -> Input Map. In here, input that the game will pick up can be registered. Type “Start” in the “Add New Action” box and press the “Add” button. Press the plus sign and allocate the Spacebar to it. The result should look like Figure 18. Next, go to “HUD” Scene created previously, press the Button node, go to the Inspector tab and do as Figure 19, create a “Shortcut” under the Shortcut setting, create an “InputEventAction” in the Event tab of the newly created “Shortcut”, and then type “Start” in the Action tab. Now every time the player needs to restart, they can quickly press Spacebar to restart the game.

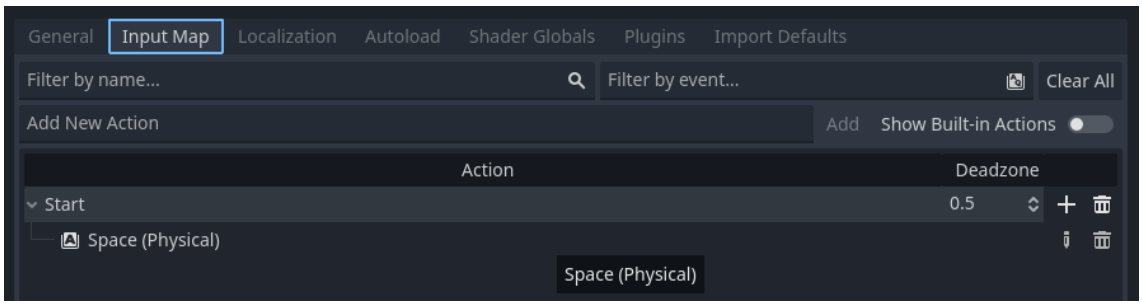

Figure 18: Adding the Spacebar as an input.

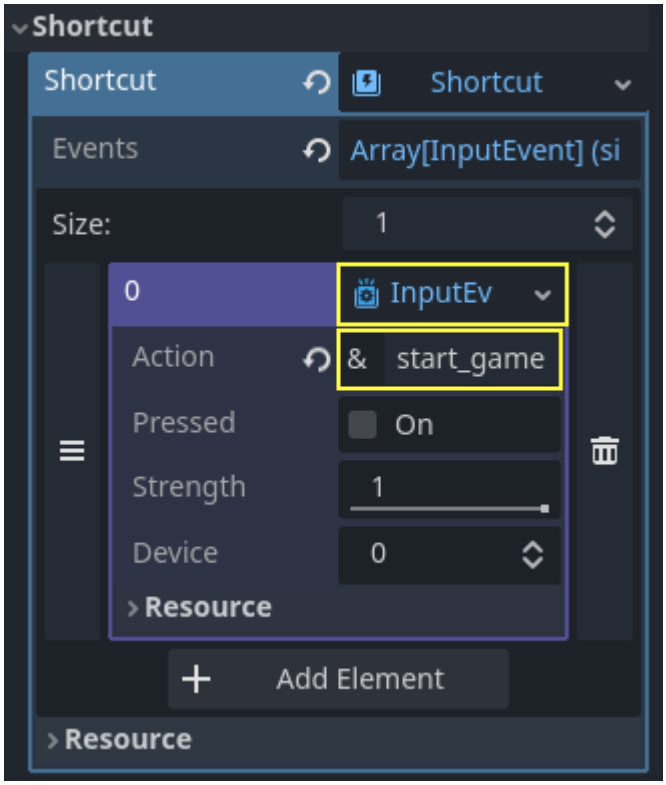

Figure 19: Connect Spacebar shortcut to start game button.

5 CONCLUSION

The thesis hopefully has reached some developers out there looking for an inspiration, a starting point, to go make game. The author had provided everything he had learnt while doing the project. The design process, directing where the game wants to go, choosing the tool for development, and actually develop the game itself.

Forming the game using the Godot envisioned to demonstrate the strength of the engine, being lightweight, cross-platform, quick, easily accessible. While the development could not show the entire strength of the engine, such as publishing in many operating system such as Windows and Linux, or the lightweight nature of the application, it did, however, exhibit the quick and easy access side, and display the potential that a developer could achieve using the Godot engine.

This thesis contributes to the growing documentation of Godot as a game engine and the body of knowledge of independent game development. More features could be study and build upon this project, such as converting to 3D animation, online leaderboard, multiplayer competitive.

A casual game was formed using the knowledge and research of the author. The goal was to make a casual 2D game and it was realised. The four fun keys principle felt like they had been somewhat reached, the control scheme is simple and quick, the rule of the game was easy to learn, so the author felt like they had done a good job. Although, there are things that could have been done better. The graphical design, as well as sound design can definitely be improved, the game could have a better way of explaining the difficulty, and the difficulty of the game itself should have been able to be tweaked by player.

The one thing missing from this thesis would be the step to publish the game. The game is only meant to be a demonstration of Godot engine anyway and are not meant to be published. The source code of the game is available in the Reference section and would serve as a great case study for people who want to follow along with this thesis report.

REFERENCES

GitHub Repository of the “Shoot the Spy” project.
https://github.com/NDSern/Shoot-the-Spy

Isbister, Katherine. & Schaffer, Noah. 2008. Game Usability: Advancing the player experience. Boca Raton: CRC Press.

Fox, Brent. 2005. Game Interface Design.

Emma Boyes. GamespotUK. Article: PlayFirst CEO John Welch thinks hardcore gamers will soon be in the minority as the masses embrace pick-up-and-play-title. Uploaded on 19.2.2008.
https://www.cnet.com/tech/gaming/gdc-08-are-casual-games-the-future/

Kevuru Games. Mobile Gaming vs PC Gaming: Overview, Prospects, and best PC and mobile games of 2024. Uploaded on 28.4.2023.
https://kevurugames.com/blog/mobile-gaming-vs-pc-gaming-overview-and-prospects/

Bress, Dan. Game Developer. Article: Player types: Casual and Hardcore. Uploaded on 2.11.2009.
https://www.gamedeveloper.com/design/player-types-casual-and-hardcore

Lazzaro, Nicole. Blog: The 4 keys 2 fun.
https://www.nicolelazzaro.com/the4-keys-to-fun/

Game: We become what we behold. 2019. Developer: Case, Nicky.
https://ncase.itch.io/wbwwb

Game: Shotgun King: The Final Checkmate. 2022. Developer: PUNKCAKE.
https://punkcake.itch.io/shotgun-king

Godot Official Documentation.
https://docs.godotengine.org/en/stable/index.html

GDScript Official Documentation.
https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html