Algorithm - ODD 2025

The Written Report Final Project

Roogey Boogey Documentation

Made By:
Louis Christian Dharma Panna Wibowo
L1BC | 2902731990

Table of Contents

CHAPTER 1
Project Specifications

I. Introductions

My final project is a dynamic 2D platformer game titled **Roogey Boogey,** developed primarily using the ***PyGame*** module. This project/platformer game features a controllable character for the user to utilize and effectively navigate throughout the designed levels filled with obstacles and difficult jumps. Furthermore, by capitalizing on the libraries of ***PyGame***, the game implements core platforming physics, including gravity, collision, and character movement. Thus, it provides a classic and engaging experience for the user.

Additionally, to enhance the user's immersion in the **Roogey Boogey** game, this project utilizes a dynamic camera system that tracks the player’s movement in real-time. This ensures that the character remains the focus point on the user’s screen while navigating through the expansive levels, and allows for a smooth scrolling effect as more of the map gets revealed as the user progresses. The world itself is designed using a 2D level editor known as ***Tiled,*** which allows for complex tile-based map design while further using assets obtained from ***itch.io***. By importing these maps into the project, ***PyGame*** can effectively render the environment using its layer-based approach.

This project represents my journey into understanding and learning the different core principles and mechanics of game architecture and design. By focusing on event-driven programming, OOP architecture, and asset management, I was able to alleviate my conceptual understanding of programming. Furthermore, by balancing the technical requirements of the different libraries utilized and the creative aspects of level design, I have gained the ability to translate user input and experience into a robust gaming experience. Lastly, **Roogey Boogey** was designed as a platformer game; however, having the OOP structure and approach to his project allows future implementations, such as enemy AI or even different mechanics to improve the gaming experience.

II. Project Links

The GitHub repository for the project can be accessed using the link provided below:

https://github.com/PuuPoo/RoogeyBoogey_FPANP

III. Modules

For this project, ***PyGame*** was used as the only module handling several critical components of the game's architecture seen below:

- **Input Handling:** Utilizing the `Pygame.event` and `Pygame.key` modules to capture real-time user input, allowing for responsive character movement and jump mechanics
- **Window & Display Management:** Utilizing the `Pygame.display` to initialize graphical interfaces and manage the screen refresh rate to ensure a smooth and constant gaming experience
- **Collision:** Utilizing `Pygame.Rect` and the `colliderect` method to manage complex interactions between the player sprite and the environment surrounding the player sprite.
- **Rendering & Animation:** Utilizing `Pygame.Surface` and the `blit` function to draw tiled maps and animate the player sprite, thus creating a smooth, fluid 2D experience

IV. The "Main Loop"

1. Initialization

   The entire project/process begins in the "RoogeyBoogey.py" file. This file serves as the entry point into the game and calls the `menu()` function. Inside "Menu.py", `pygame.init()` starts the engine, and the screen is set to the resolution of 928 x 793 pixels. Crucially, the system initializes a `pygame.tick.Clock()` to cap the frame rate at 60 FPS, ensuring consistent gaming speed across the entire project. During this phase, the user can select to play the game, in which the following level will be called, and the TMX map data, character sprite, SFX, and BGM will be called and loaded.

2. The Main Loop

   The main loop is a while gameRunning: block found in all of the levels, like "Tutorial.py" and "Level1.py." Each iteration of this loop represents one frame of the game and performs the following tasks:

   - **Tick Management:** `clock.tick(FPS)` to maintain the FPS of the game
   - **Event Handling:** Checks for user input using `pygame.event.get()`
   - **Update and Render Logic:** Triggers the `Knight.update()` method, which processes all internal player physics and animations for that given frame, as well as updating the map/environment around the knight character.

3. Camera System

The camera system implemented in the final project is implemented as a custom `pygame.sprite.Group` in “Camera.py”. Rather than moving the player sprite across the screen, the camera calculates an offset vector based on the player’s position relative to the center of the display. Thus, the principles below are used to implement the camera system:

- **Clamping:** The system ensures the camera doesn't scroll beyond the map designed in ***Tiled*** by checking the `minX` and `minY` limits along with the `maxX` and `maxY` limits.
- **Offset Rendering:** This principle is done utilizing the `draw()` method. When the `draw()` method is called, the camera applies the offset value to every sprite in the group, effectively shifting the entire world in the opposite direction of the player’s direction. This gives the illusion that the world is moving based on the player’s perspective.

4. Collision Detection

Collision logic for the player and its environment is handled within the `Player.update()` method in “Player.py”. This ensures that collision with the environment stays consistent as follows:

- **Horizontal and Vertical Splitting:** Movement is applied to the X-axis first, checked against the `collisionTiles`, and then the same is done with the Y-axis. All these checks ensure the player is prevented from letting the character clip through the corners.
- **Standard Collisions:** The standard collision is done by drawing a `Rect` over each sprite and checking for intersections between them. The `collisionTiles` list is filled by identifying each tile under the “Level” layer in ***Tiled***. When the character collides with these tiles, their delta movement is removed, and their position is reset to the edge of the block to prevent any clipping.
- **One-way Platforms:** This type of collision implements a “pass-through” logic for specific environmental tiles. By comparing the player’s position and current vertical velocity, the application allows for only upward traversal through the bottom of the platform.
- **Hazardous & Goal Conditions:** This type of collision is implemented by using `colliderect` on specific tiles. Tiles under the “Liquid” layer in ***Tiled*** will immediately trigger a “death: state upon the player resetting the player’s progress within the stage. Tiles under the “FinishedBlock” in ***Tiled*** will immediately act as a gateway for level transition, signalling the program to stop the current loop and start the next loop and environment.

5. Animation

The animation system in the application is powered by the animationSheet class, which extracts the frames from the individual sprite sheets.

- **Sprite Sheet Extraction:** The application utilizes a specialized `loadAnimation` method that handles the precise slicing of the individual frames in the sprite sheet into a list of usable frames. This method utilizes a lot of parameters to accommodate different sheet layouts, including:
    - **Coordinates & Dimensions:** `startX` and `startY` will define the initial anchor point for the slicing, while `frameWidth` and `frameHeight` establish the dimensions of each frame window.
    - **Spacing & Count:** `frameGap` accounts for the pixel gap between frames, and `numFrames` determines how many interactions the loop performs to populate the list with the number of frames in the sheet.
    - **Scaling:** A scale parameter is applied during the extraction of the individual frames, this is to ensure that the character size stays consistent with the game world's resolution and size.
- **State Control:** The player state (stored as `self.action`) acts as an index to determine which list of frames is currently active and displaying (**idle (0), walk (1)**)
- **Dynamic Sequencing:** A `frameTimer` increments based on the `animationSpeed` of 0.1 per loop. When the timer reaches a value of 1, the `currentFrame` index advances, thus controlling the speed of an individual frame displayed. Lastly, pygame.transform.flip is applied to the image if the player is moving to the left, thus flipping the entire sprite.

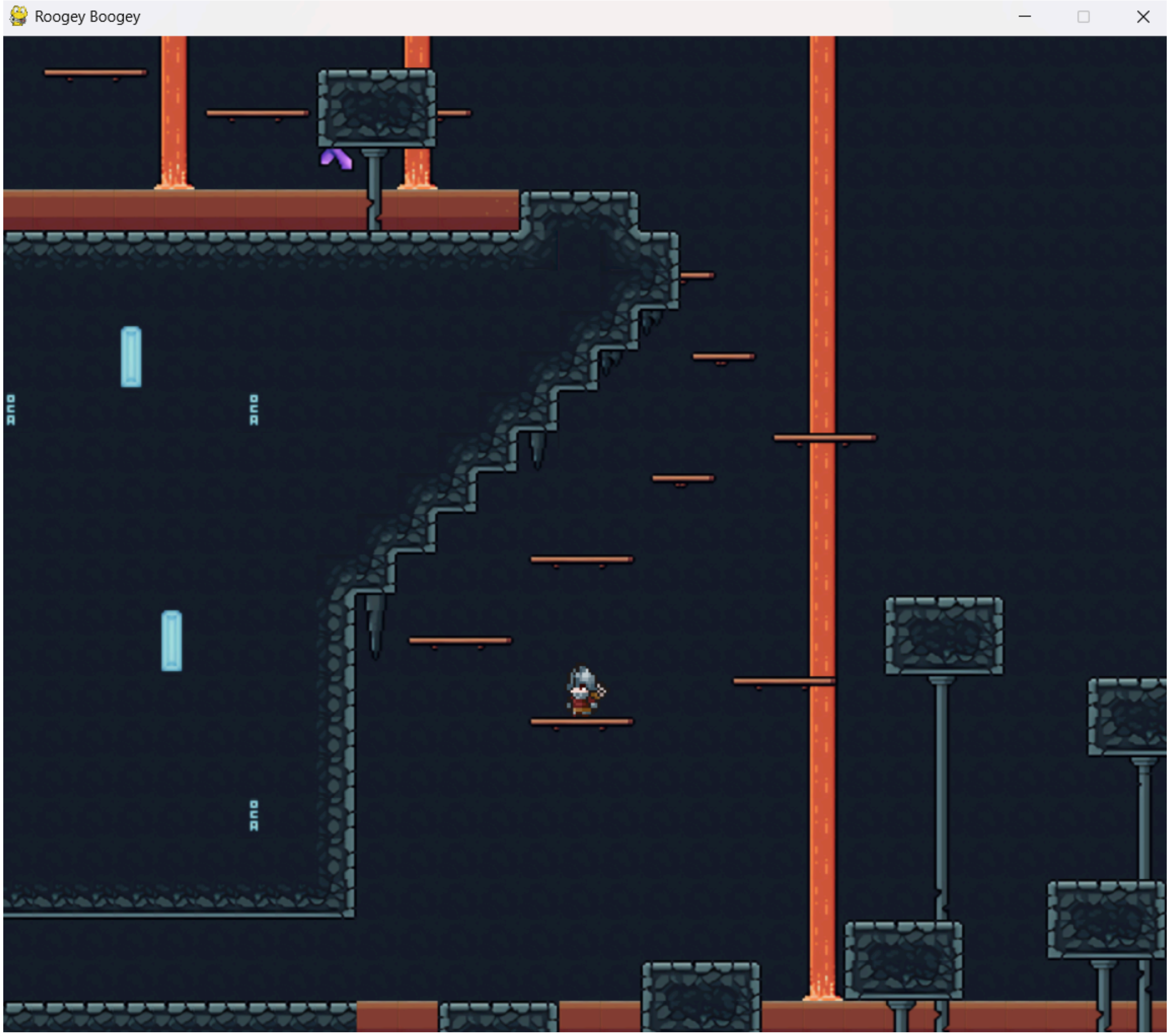
6. Rendering

The rendering process follows a strict "Painter's Algorithm", or in other words, back to front, to ensure correct layering.

- **Clearing:** The display for each level is displayed with a sky-blue color (RGB 47, 203, 255) to start each frame refresh.
- **Layer Order:** Following the algorithm (back to front), the layers are drawn with the following order: background > liquids > decorations > finishedBlock > platforms > levels.
- **Player & UI:** The playerGroup and UI elements are drawn last to ensure they always remain visible against other environmental sprites.

7. Conclusion

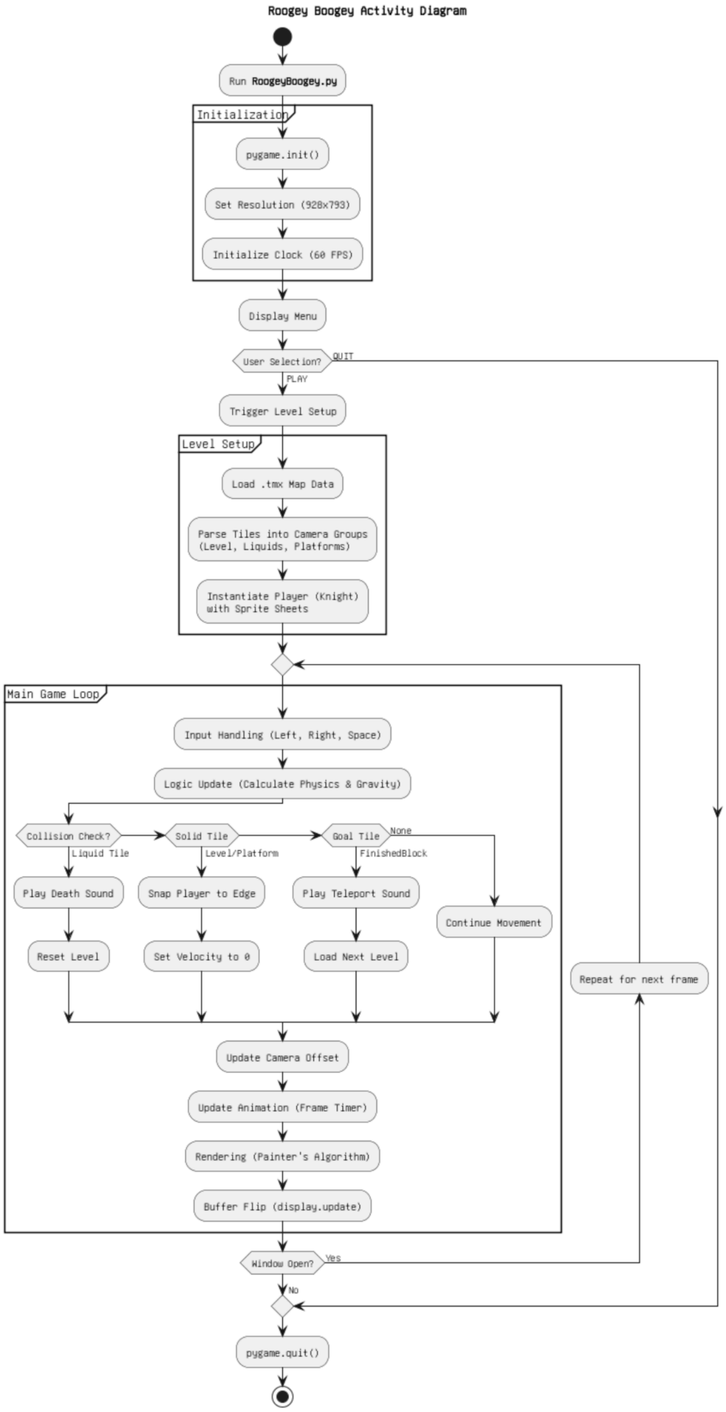
The result of the application after implementing the “main game loop” is an application/project that generates a 2D environment by integrating various layers in ***Tiled*** into a unified bit world of **Roogey Boogey**. Once the level data and assets are fully loaded, the program passes all the necessary components needed for the ***PyGame*** engine to render into the user’s display. This rendered world becomes fully interactive with a camera and physics system, along with a steady tick rate clock of 60 FPS. This ensures the user can effectively navigate and explore the different levels with precision. Finally, it allows the combination of responsive player input, state-based animations, and real-time collision detection to provide an excellent platforming and gaming experience.



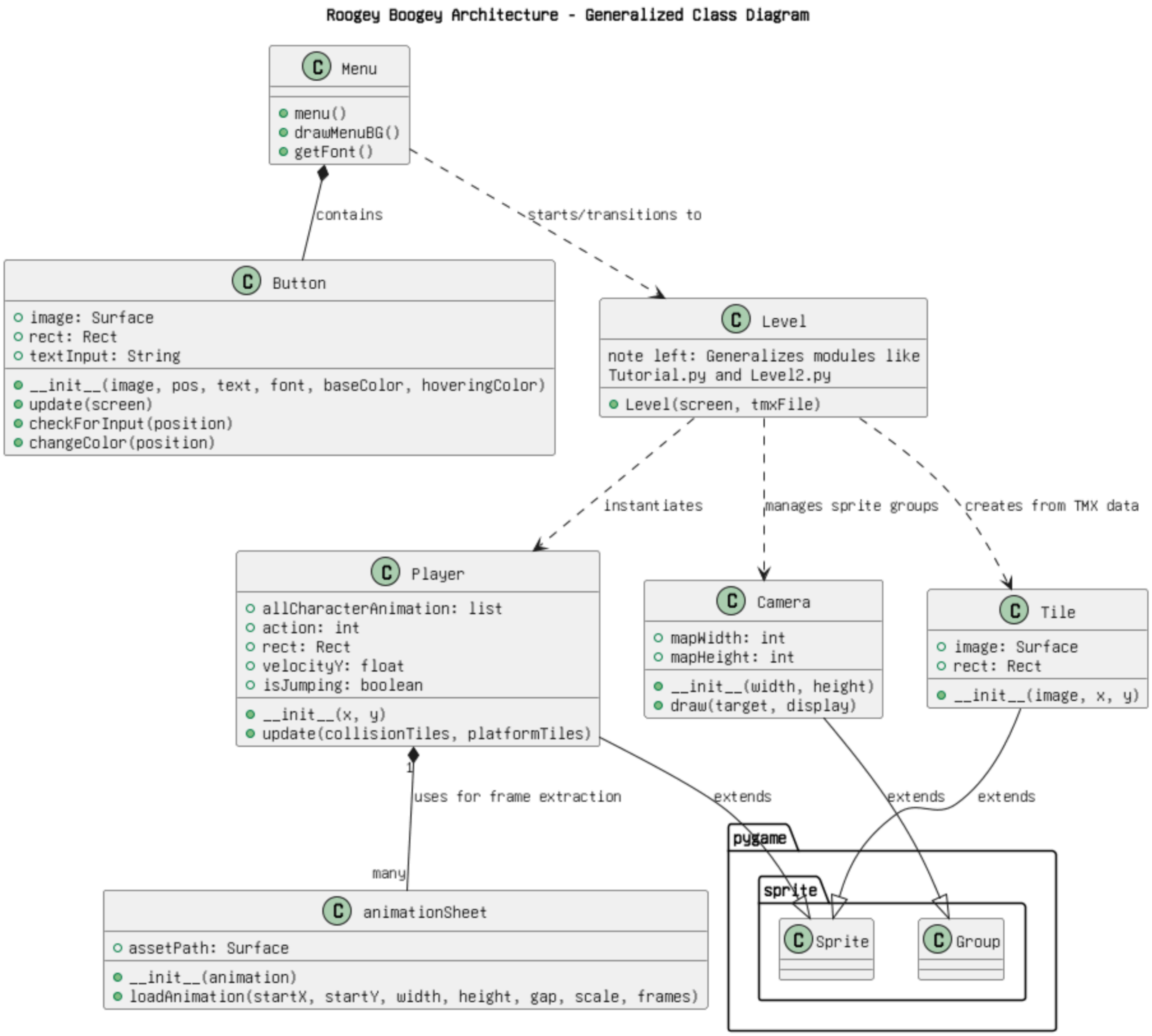
(Preview of the Application/program: **Roogey Boogey**)

CHAPTER 2
Project Diagrams

I. Activity Diagram



II. Class Diagram



CHAPTER 3

Project Documentation

I. Setup and Installation

To ensure the setup and installation process runs smoothly for the perfect testing and gameplay of the application, the following is recommended:

1. **Language:** Python 3.10 or higher
2. **Library:** Pygame Module (Can be installed in CMD (`pip install pygame`))
3. **Assets:** All images, sprites, and audio assets must remain in their respective folder relative to the "RoogeyBoogey.py" file

This will ensure the application will run smoothly, following its native 60 FPS and physics calculations

II. User controls

The control input for the **Roogey Boogey** comprises 2 types of inputs: Keyboard and mouse. The mouse inputs will be used for menu navigation, and the keyboard for real-time character movement.

Keyboard Mapping		
Action	Keys	Alternative
Move Right	D	–
Move Left	A	–
Jump	W	Spacebar

Menu Interface
The main menu serves as the primary hub. It will be the initial state before the user can start the game. This menu interface utilizes a hover-detection system where the buttons change color when the cursor is overlapped. This visual feedback confirms the interactivity of the button and ensures navigation to the user before transitioning to the gameplay.

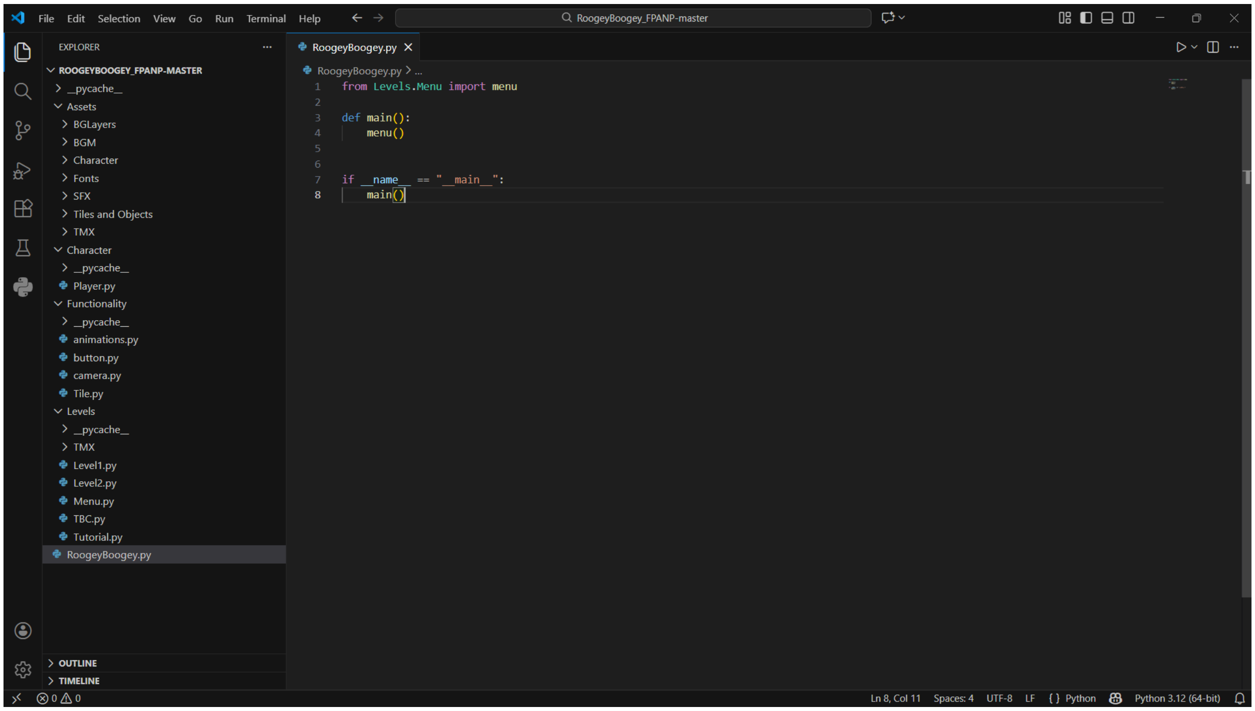
III. Gameplay Mechanics

The gameplay mechanic behind **Roogey Boogey** is to navigate through a series of increasingly difficult platforms to reach the teleport block or “FinishedBlock.” This will trigger the next stage of platforming challenges.

1. **Exploration:** The player must use vertical and horizontal movement to explore the different stages and make progress across obstacles or gaps.
2. **Hazard Avoidance:** Throughout the different stages, the player will encounter hazardous material (Liquids). This will result in immediate reset or death upon collision or contact. This enforces the “trial and error” of platformers upon the player.
3. **One-Way Traversal:** Players can jump through and navigate various obstacles to progress. However, there are cases where certain choices make it impossible to traverse back. This is to reinforce the feeling of risk upon the player.
4. **Goal Progression:** Upon the player character colliding with the designated goal of the level, the current main loop of that level will be suspended, and the next level’s loop will start.

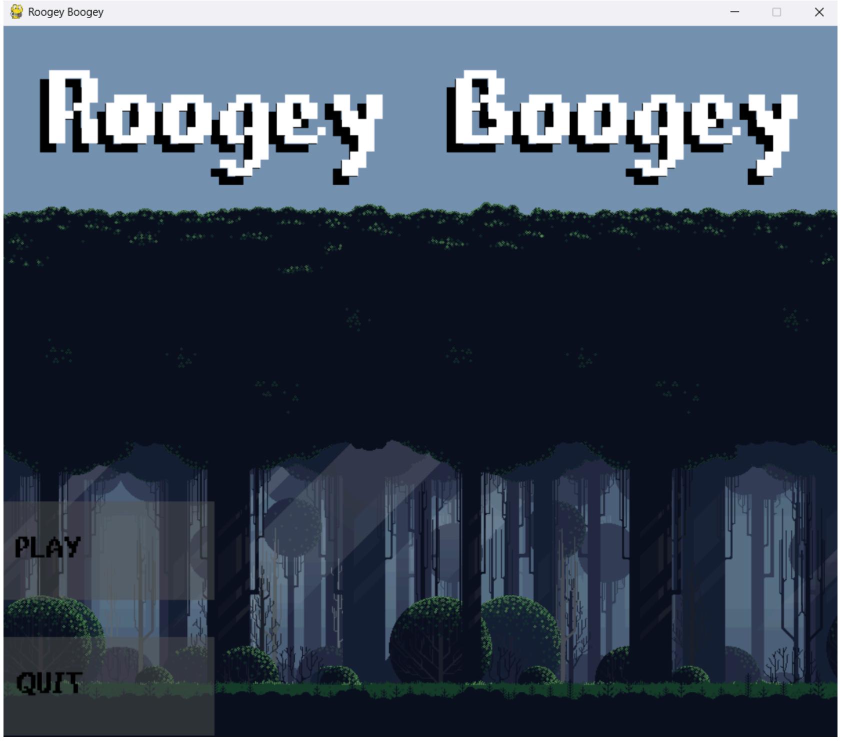
IV. Screenshots

Screenshots below will be a quick preview of the gameplay, visual appearance, and the coding structure of the **Roogey Boogey** game/application.

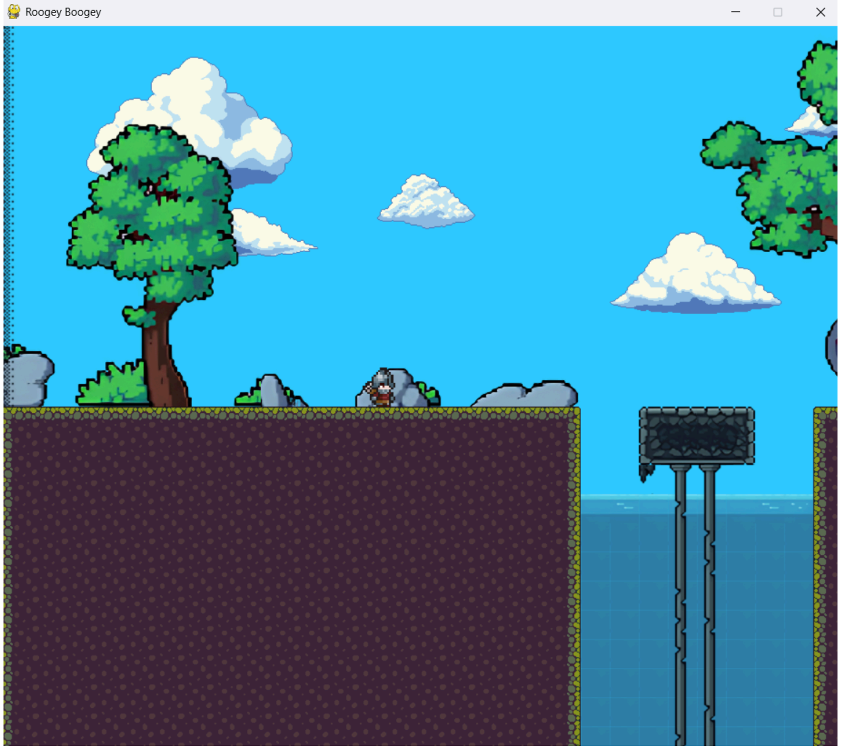


```python
from Levels.Menu import menu

def main():
    menu()


if __name__ == "__main__":
    main()
```

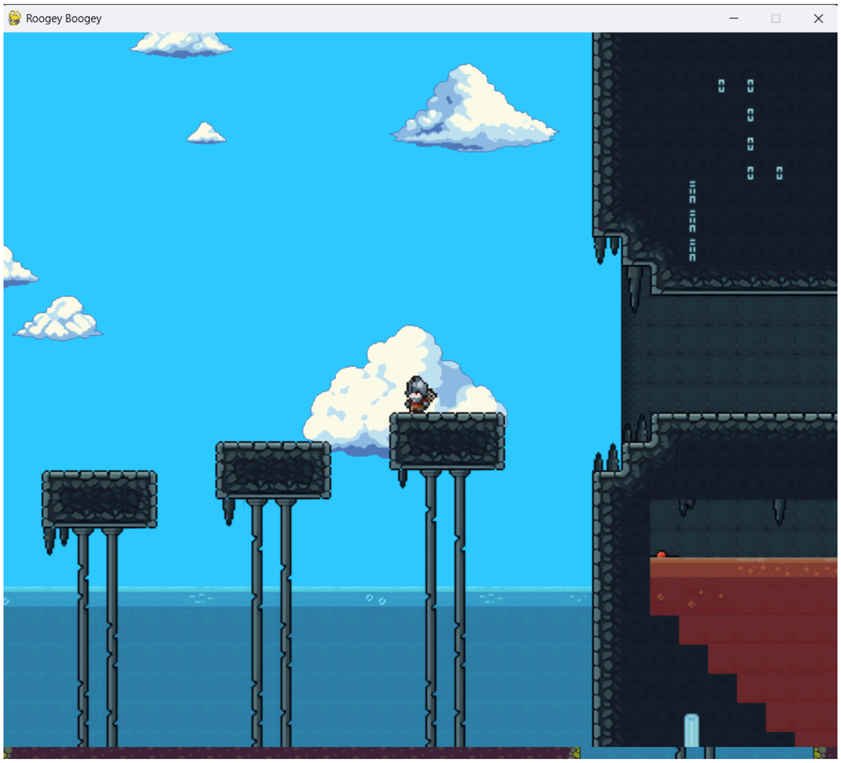
(Code Structure and Heirarchy)



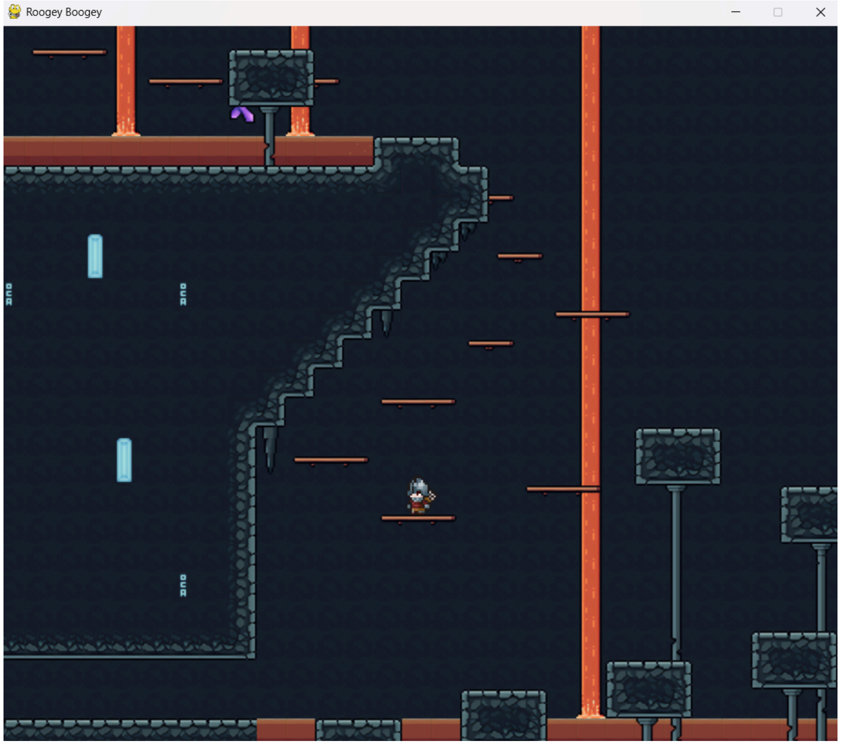
(**Roogey Boogey** Gameplay Menu)



(**Roogey Boogey** Gameplay Screen)



(**Roogey Boogey** Gameplay Screen 2)



(**Roogey Boogey** Gameplay Screen 3)

CHAPTER 4

Reflection and Learning Experience

I. Reflection and Learning Experience

Creating **Roogey Boogey** has been an eye-opening experience for me because it has taught me that programming isn't just about writing code; it is also about designing and creating something with a purpose. In this case, the purpose was to develop a game that allows different players to engage with a virtual environment structured as a platformer game in real-time.

Throughout the process of developing **Roogey Boogey**, I learned many new things about programming, including how to build a whole new project using Object-Oriented Programming (OOP). Before starting this project, I had experience with OOP; however, that experience was under the Java language. By trying to do this project with the OOP concept in an entirely new language, I have learned how OOP is a paradigm that can be utilized in most projects and languages. Furthermore, it solidified my understanding of OOP as a paradigm and enlightened me to the connections that OOP can bring to projects that I previously didn’t know.

The most valuable lessons I learned when developing **Roogey Boogey** were the importance of encapsulation and modularity, as well as how to separate animation logic from physics. When I created the Knight class, I implemented an animationSheet system that separated my animations from the rest of my game code, which allowed me to diagnose and fix movement issues without affecting the rendering of the sprites. This experience taught me how important it is for all software systems to follow the practice of good architectural design.

Additionally, the process of resolving the Collision Detection challenges taught me logic and spatial reasoning. My moment was when I began handling horizontal and vertical collisions independently of each other, which is a major contributor to the character "clipping". What I learned from this is that large, complex problems can be broken down into small, sequentially completed parts.

Furthermore, using a third-party tool, such as Tiled, opened up my mind to how modern game developers use multiple software applications, in conjunction with one another, to create their final product.

The entire project has helped me develop a deep appreciation for Event-Driven Programming. When I see my code translate to a player making their character jump, or my code facilitating the transition of one level to another, it brings a sense of accomplishment that is not present through simply writing text-based code. Moreover, the project has helped me sharpen my thinking and motivated me to learn more about game structure and software development.

II. Future Improvements

Roogey Boogey, with its modular design, can be further improved to become more fluid and immersive for the user. Some future improvements for the game can be in the form of the list below:

1. **Enemies and their algorithm**: One implementation is to implement a set of character sprites represented by their logic in states. This would change the challenge associated with 'static hazard' enemies into developing dynamic threats that require players to adjust their gameplay in response to the enemy's movement patterns.

2. **Save/Progression System**: As of right now, the game does not allow you to save progress before exiting the game. A future improvement would be to create a Save System using either JSON or Text Files that allows players to save their position within the game so that they can play through the game across multiple sessions without restarting.

CHAPTER 5

Statement on the Use of Artificial Intelligence

I. Artificial Intelligence Utilization

Throughout the creation of **Roogey Boogey**, AI was used at different parts of the sections to help provide ideas and understanding to the project. The AI used, and its usage, are listed below:

1. **Gemini 3 Flash**: Google's AI Gemini was used throughout the idea creation and debugging phase of the project. This tool was made to provide some fun, creative ideas to implement in my project, as well as to debug code if a persistent error occurred. Some examples of prompts include:
    - What are fun ideas to create for my Python project?
    - What types of games are challenging to create in Pygame?
    - From this block of code, is this part [CODE] causing the issue, and why?

    By having Gemini's output, it aided in efficiently solving problems or providing a unique perspective. However, I also evaluated its response to better suit my project, along with changing the solution to better fit my project.

    Finally, to verify it, I never blindly input Gemini's response into my project. If the response was about idea creation and perspectives, I make sure to check in with different people to see if the idea is viable and interesting. For coding responses, I make sure to test each block of code given and thoroughly understand it. Finally, integrate them manually to ensure everything works to order in **Roogey Boogey**.

2. **Leonardo.ai**: Leonardo AI was used during the presentation phase to generate the artwork poster for **Roogey Boogey**. This is done to create a high-quality poster that intrigues players to test the game. Some examples of prompts include:
    - Make me a hand-drawn A3 Poster for a game about a knight escaping
    - Center the knight, and the title is Roogey Boogey

    The output of this AI was heavily evaluated to ensure that the response matched the quality and relevance of the project **Roogey Boogey**. Furthermore, I ensure that the response matches the aesthetics as well as the message I was trying to convey.

    Finally, to verify it, I ensure to look deeply into the response and manually change any discrepancies and holes found. Thus, ensuring high quality and a smooth final integration into the documentation/presentation phase of this project.

Bibliography

CDcodes. (2020, October 8). *Pygame Tile Based Game Tutorial: Tilemaps*. YouTube. https://www.youtube.com/watch?v=37phHwLtaFg

Code, C. (2022). A guide to level creation with Tiled [+ how to use it with pygame]. In *YouTube*. https://www.youtube.com/watch?v=N6xqCwblyiw

Coding With Russ. (2020, December 2). *PyGame Platformer Game Beginner Tutorial in Python - PART 2 | Creating the Player*. YouTube. https://www.youtube.com/watch?v=W_JRd3ntyBg&list=PLjcN1EyupaQnHM1I9SmiXfbT6aG4ezUvu&index=2

Coding With Russ. (2021a, March 7). *PyGame Scrolling Shooter Game Beginner Tutorial in Python - PART 1 | Creating the Player*. YouTube. https://www.youtube.com/watch?v=DHgj5jhMJKg&list=PLjcN1EyupaQm20hlUE11y9y8EY2aXLpnv

Coding With Russ. (2021b, March 7). *PyGame Scrolling Shooter Game Beginner Tutorial in Python - PART 1 | Creating the Player*. YouTube. https://www.youtube.com/watch?v=DHgj5jhMJKg&list=PLjcN1EyupaQm20hlUE11y9y8EY2aXLpnv

Coding With Russ. (2021c, March 30). *PyGame Scrolling Shooter Game Beginner Tutorial in Python - PART 9 | Loading Levels*. YouTube. https://www.youtube.com/watch?v=5VSzZu5904c&list=PLjcN1EyupaQm20hlUE11y9y8EY2aXLpnv&index=9

Coding With Russ. (2021d, April 2). *PyGame Scrolling Shooter Game Beginner Tutorial in Python - PART 10 | Collision*. YouTube. https://www.youtube.com/watch?v=m7GnJo_oZUU&list=PLjcN1EyupaQm20hlUE11y9y8EY2aXLpnv&index=10

Coding With Russ. (2021e, May 1). *PyGame Beginner Tutorial in Python - Loading Spritesheets*. Www.youtube.com. https://www.youtube.com/watch?v=M6e3_8LHc7A

Coding With Russ. (2021f, May 14). *PyGame Beginner Tutorial in Python - Sprite Animation*. Www.youtube.com. https://www.youtube.com/watch?v=nXOVcOBqFwM

Coding With Russ. (2022, April 8). *PyGame Beginner Tutorial in Python - 3D Background Effect with Parallax Scrolling*. Www.youtube.com. https://www.youtube.com/watch?v=OAH8K5lVYOU

Pygame. (n.d.). *Pygame Front Page — pygame v2.0.0.dev15 documentation*. Www.pygame.org. https://www.pygame.org/docs/

Tiled. (n.d.). *Introduction — Tiled 1.8.4 documentation*. Doc.mapeditor.org. https://doc.mapeditor.org/en/stable/manual/introduction/