

# **ALGORITHM AND PROGRAMMING FINAL REPORT**



**Name of lecturer: Jude Joseph Lamug Martinez, MCS**

**Made by:**

**STANISLAUS JUSTIN - 2702342741**

**BINUS INTERNATIONAL UNIVERSITY**

**2024**

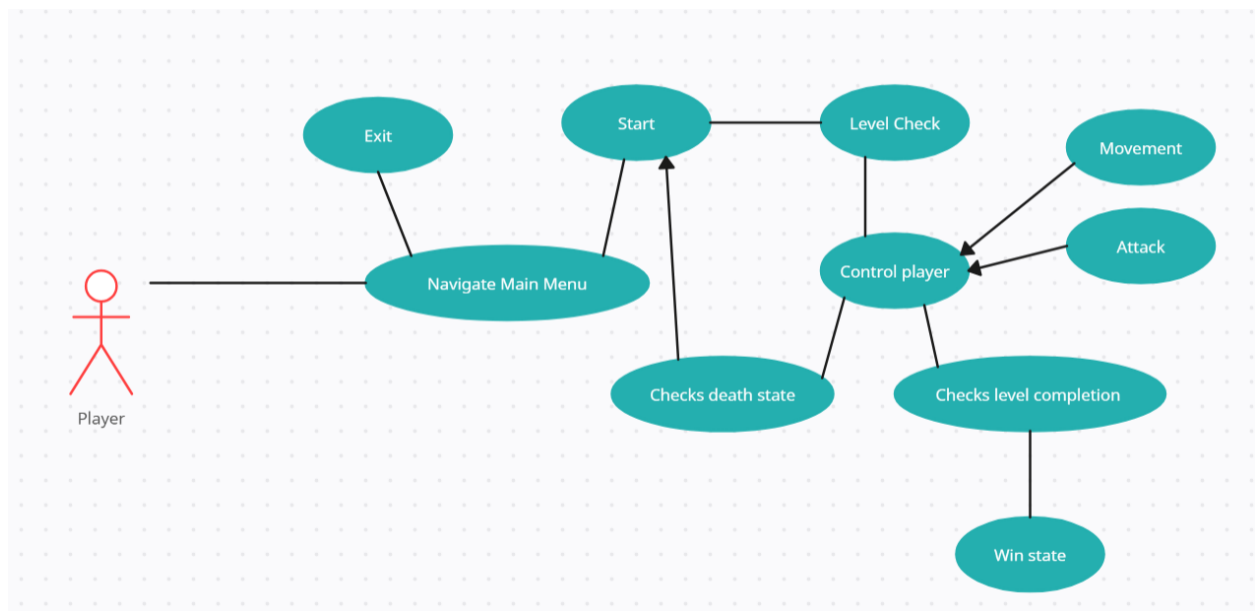
# TABLE OF CONTENT

<b>BRIEF DESCRIPTION</b>	<b>3</b>
<b>USE CASE DIAGRAM</b>	<b>3</b>
<b>ACTIVITY DIAGRAM</b>	<b>4</b>
<b>CLASS DIAGRAM</b>	<b>5</b>
<b>MODULES</b>	<b>8</b>
<b>ESSENTIAL ALGORITHM</b>	<b>9</b>
<b>GENERAL OVERWORLD ALGORITHM</b>	<b>10</b>
<b>SCREENSHOTS</b>	<b>11</b>
<b>LESSON LEARNED</b>	<b>18</b>

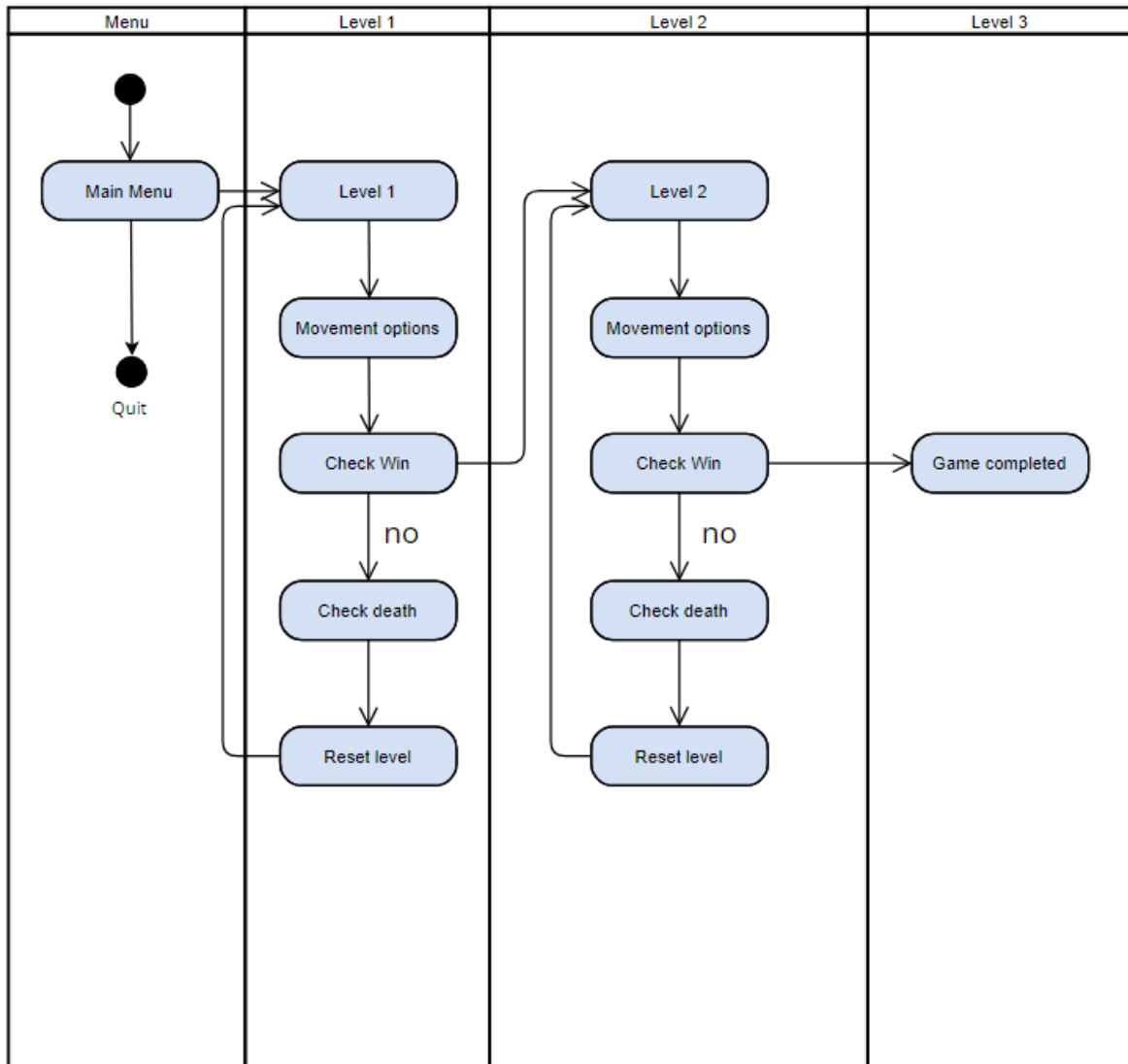
## BRIEF DESCRIPTION

For my final project, I made a side scroller shooting game, similar to the super mario bros games, but with guns and a lot of enemies shooting at you. It's a 2D platformer made through pygame with a few different added modules to aid in importing files. The game is able to run with a standard start or exit game menu. Once the user presses "START", they will be directed into the level 1 platform, where they will encounter and be required to kill multiple enemies, as well as do a series of difficult parkour by jumping, running, and falling. There are also power ups to get a boost in ammo, grenades, and even a recovery in health. After the player surpasses level 1, they will have to surpass level 2 to finish and win the game. If they fail to do the parkour or defeat the enemies and fall to their demise in game, they will be required to restart at the very beginning of the level they are currently at. The game also has a series of animations like idling, running, and jumping.

## USE CASE DIAGRAM



## ACTIVITY DIAGRAM



## CLASS DIAGRAM

Soldier
<ul style="list-style-type: none"> <li>- alive: bool</li> <li>- char_type: str</li> <li>- speed: int</li> <li>- ammo: int</li> <li>- start_ammo: int</li> <li>- shoot_cooldown: int</li> <li>- grenades: int</li> <li>- health: int</li> <li>- max_health: int</li> <li>- direction: int</li> <li>- vel_y: int</li> <li>- jump: bool</li> <li>- in_air: bool</li> <li>- flip: bool</li> <li>- animation_list: list[index]</li> <li>- frame_index: int</li> <li>- action: int</li> <li>- update_time: int</li> <li>- move_counter: int</li> <li>- vision: rect</li> <li>- idling: bool</li> <li>- idling_counter: int</li> <li>- image: list[index]</li> <li>- rect: rectangle</li> <li>- width: int</li> <li>- height: int</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ update()</li> <li>+ move()</li> <li>+ shoot()</li> <li>+ ai()</li> <li>+ update_animation()</li> <li>+ update_action()</li> <li>+ check_alive()</li> <li>+ draw()</li> </ul>

World
<ul style="list-style-type: none"> <li>- obstacle_list: list</li> <li>- level_length: int</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ process_data()</li> <li>+ draw()</li> </ul>

Decoration
<ul style="list-style-type: none"> <li>- image: pygame.Surface</li> <li>- rect: rectangle</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ update()</li> </ul>

Water
<ul style="list-style-type: none"> <li>- image: pygame.Surface</li> <li>- rect: rectangle</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ update()</li> </ul>

Exit
<ul style="list-style-type: none"> <li>- image: pygame.Surface</li> <li>- rect: rectangle</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ update()</li> </ul>

ItemBox
<ul style="list-style-type: none"> <li>- item_type: str</li> <li>- image: pygame.Surface</li> <li>- rect: rectangle</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ update()</li> </ul>

HealthBar
<ul style="list-style-type: none"> <li>- x: int</li> <li>- y: int</li> <li>- health: int</li> <li>- max_health: int</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ draw()</li> </ul>

Bullet
<ul style="list-style-type: none"> <li>- speed: int</li> <li>- image: pygame.Surface</li> <li>- rect: rectangle</li> <li>- direction: int</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ update()</li> </ul>

Grenade
<ul style="list-style-type: none"> <li>- timer: int</li> <li>- vel_y: int</li> <li>- speed: int</li> <li>- image: pygame.Surface</li> <li>- rect: rectangle</li> <li>- width: int</li> <li>- height: int</li> <li>- direction: int</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ update()</li> </ul>

Explosion
<ul style="list-style-type: none"> <li>- images: List[index]</li> <li>- frame_index: int</li> <li>- image: pygame.Surface</li> <li>- rect: rectangle</li> <li>- counter: int</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ update()</li> </ul>

ScreenFade
<ul style="list-style-type: none"> <li>- direction: int</li> <li>- color: tuple</li> <li>- speed: int</li> <li>- fade_counter: int</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ fade()</li> </ul>

Popup
<ul style="list-style-type: none"> <li>- font: pygame.font.Font</li> <li>- image: pygame.Surface</li> <li>- rect: rectangle</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ draw(screen)</li> </ul>

Button
<ul style="list-style-type: none"> <li>- image: pygame.Surface</li> <li>- rect: rectangle</li> <li>- clicked: bool</li> </ul>
<ul style="list-style-type: none"> <li>+ __init__()</li> <li>+ draw(surface)</li> </ul>

# MODULES

## Pygame Module

The fundamental building block of this game is Pygame, serving as the backbone that handles everything from the window display to loading various sprites. Pygame acts as a vast collection of libraries packed with functions crucial for crafting video games. Within this game, Pygame's pivotal roles include tasks like loading images, resizing them with transformations, and capturing input from keyboard presses.

## Sys Module

The `'sys'` module in the code is utilized for specific functionalities. It facilitates a smooth exit using `'sys.exit()'` when certain conditions arise, like when the player hits the escape key to quit the game. Additionally, the `'sys.clock.tick(FPS)'` function helps maintain a consistent frame rate, ensuring a seamless gaming experience.

## OS Module

The `'os'` module is employed within the code to manage file operations. It enables the game to interact with the underlying operating system. Particularly, the code uses it for file handling tasks like accessing and reading CSV files that store level data. This module streamlines the process of reading external resources, allowing the game to load and render level designs efficiently.

## CSV Module

The `'csv'` module plays a crucial role in handling comma-separated values (CSV) files within the game code. It facilitates the parsing and extraction of data from these files, specifically in loading level designs. The code utilizes the `'csv'` module to read and interpret level data stored in CSV format, enabling the game to generate and display intricate level layouts seamlessly. This module simplifies the process of accessing structured data, ensuring smooth level transitions and rendering. In order to make the tiles and map itself, I have another file called `level_editor.py` which allows me to edit the tiles however I want, as well as save and load them based on their levels.

## Random Module

The `'random'` module in the game code introduces unpredictability and randomness. It enables the generation of random numbers, which is utilized for various functionalities within the game, such as randomizing enemy behavior, spawning items or obstacles, or determining chance-based outcomes. By leveraging functions like `'random.randint()'` or `'random.choice()'`, it adds an element of chance and variation, contributing to a more dynamic and engaging gaming experience. This module injects



randomness into specific game components, making each playthrough unique and non-repetitive.


## **ESSENTIAL ALGORITHM**

1. Collision Detection Algorithm:
  - Determines when game entities intersect, facilitating interaction between the player, enemies, bullets, grenades, and obstacles based on bounding boxes or masks.
2. Enemy AI Algorithm:
  - Dictates enemy behavior, enabling them to navigate obstacles, follow the player, and make decisions using pathfinding algorithms or simple heuristic methods.
3. Player Movement Algorithm:
  - Manages player movement through keyboard presses.
4. Projectile Algorithm:
  - Governs the motion of bullets and grenades, incorporating ballistic physics principles to calculate trajectories, considering velocity, angle, gravity, and collision detection.
5. Level Generation Algorithm:
  - Generates game levels, obstacles, enemies, and item placements either procedurally or from predefined data sources like CSV files or maps.
6. Health/Damage Algorithm:
  - Tracks health points for entities, computes damage on collisions, updates health bars, and manages destruction of entities upon health depletion.
7. Screen Scrolling Algorithm:
  - Controls the movement of the game screen, ensuring the player remains centered while navigating through the level by updating the background/environment.
8. Win/Loss Condition Algorithm:
  - Monitors win conditions (such as defeating enemies or reaching an exit) and loss conditions (like player health hitting zero), triggering corresponding events for level completion or game over.

## GENERAL OVERWORLD ALGORITHM

1. **Jumping Algorithm:**
  - Manages player jumps by applying an initial vertical force and controlling gravity to simulate ascent and descent, incorporating variables like jump height, gravity, and ground detection to initiate and terminate jumps.
2. **Shooting Algorithm:**
  - Handles bullet creation, direction, and velocity, utilizing input triggers to spawn projectiles from the player's position while applying a velocity in the aimed direction.
3. **Grenade Throwing Algorithm:**
  - Controls the trajectory of grenades by determining the initial velocity and angle based on the player's position, applying gravity to simulate arc motion, and detecting collisions to initiate detonation or destruction.
4. **Pathfinding Algorithm (for enemies):**
  - Guides enemies towards the player by computing the shortest path, considering obstacles, employing algorithms like A\* or simple heuristic methods to navigate the game world effectively.
5. **Animation Rendering Algorithm:**
  - Manages the display of character animations, updating frames and rendering them onto the screen, coordinating different states (idle, running, jumping) based on player actions.
6. **Explosion Animation Algorithm:**
  - Renders explosion animations by cycling through a sequence of images at specific intervals, simulating the visual effects of an explosion.
7. **User Interface (UI) Updating Algorithm:**
  - Updates the graphical user interface (GUI) elements such as health bars, ammo count, and other indicators, reflecting changes in the player's status and game progression.
8. **Input Handling Algorithm:**
  - Processes user inputs from keyboard or mouse, interpreting them to trigger corresponding actions in the game, including movement, shooting, jumping, or throwing grenades.

## SCREENSHOTS

 Shooter

— □ ×

START

EXIT













RESTART



## LESSON LEARNED

I have learned a lot since the start of making this project – most importantly, time management. Having a 10 day break, I decided to go out of town and spend my break relaxing when I really should've started making this final project. I underestimated the amount of work that I needed to do in order to complete this game. It's true that I watched and followed quite a few youtube tutorials in order to complete this game, but I also learned so much from watching those videos. After watching countless videos and reading multiple websites about python, I am certain that my coding skills using python and the pygame module have significantly improved. I got to understand so much stuff like csv's and how to import those into the actual game, collisions and gravity, and even further my understanding of having AI in game.