Final Project Report

Top-Down Shooter Game “The Bunker”

Python Game Development

Made By:

Alvyn Ilham Kurniawan – 2902696281

Lecturer:

Jude Joseph Lamug Martinez, MCS

Algorithm and Programming – L1AC

January 2026

Chapter I
Project Specification

A. Introduction

"The Bunker" is the title of a top-down shooter game written in the Python Programming language using Pygame. This program serves as a submission for the final project of the Algorithm and Programming class taken in BINUS International University. The objective of this project is the creation of an enjoyable video game experience, while using Python and Pygame to incorporate elements such as procedural generation and top-down gameplay.

The goal of the game itself is to progress through sets of procedurally generated levels in order to reach and defeat the final boss. The gameplay consists of a player character moving through rooms, constantly fighting enemies using firearms, and finding "stairs" to progress to the next level, all the way until the boss. Should the player's health reach zero due to enemy attacks, the user will be notified of their defeat and promptly be sent back to the menu, where they can start over through a new set of rooms. Should the player win, they will be rewarded with a congratulatory message, before being returned to the menu and given the chance to start again from the beginning.

B. Project Scope

- An interactive python program displaying certain game states.
- Multiple game states, such as "menu", "gameplay", and "ending".
- A controllable player character for the gameplay content, with the ability to move and shoot.
- A set of procedurally generated levels for the gameplay content.
- Hostile entities such as enemies and bosses for the gameplay content, with the ability to shoot at the player character.
- Roguelike features, such as a progress and level generation reset after a game over or victory.
- Basic error handling, such as default textures incase assets fail to load

C. Import Used

The following are libraries utilized in the project:

- Python
- Pygame

Chapter II
Solution Design

A. Architecture Overview

The video game program “The Bunker” adheres to a modular and object-oriented architecture. Python features such as data structures are used to create a modular code, while classes are used for an object-oriented design. Meanwhile, Pygame is utilized for its display features, alongside sprites/textures, audio, collision, etc.

B. Requirements

- Modular Design
- Object-Oriented Design
- Readable
- Documented (In-code)

C. Major Algorithms

1. Procedural World Generation Algorithm

The world generation algorithm exists as a class, which relies on two other classes, mainly the class for wall objects, and for rooms. Walls are basic objects with dimensions and position, whereas rooms contain a list of walls, alongside things such as positions, dimensions, a list of doors, variation, etc.

The world generation algorithm itself is a simple grid-like pattern and is as follows:

a. Decide how many rooms need to be generated and their dimensions, usually obtained from the parameters
b. Create a container for all rooms alongside a container for the coordinates of all rooms.
c. Create a base room alongside its coordinates and insert them to their respective containers.

d. Choose a random room as a parent room to generate one new room from, unless all rooms have been generated.
e. Choose a random cardinal direction to generate a new room in.
f. Checks if there is a room in that direction, if not, return to step d, else move forward.
g. Creates the room object and stores it in the room container
h. Creates the coordinates for the new room and stores it in the coordinate container
i. Adds a door to the list of doors in the parent and new room, where the parent room will get a door in the direction of the new room, whereas the new room will get a door in the direction of the parent room.
j. Repeat until all rooms have been generated.
k. Generate the doors for all rooms, based on the list of doors contained in each room.
l. Find the furthest room from the original room.
m. Randomly decide on the variations each room will have, which will determine the contents of the room. Exception given to original and furthest room.
n. Return the list of room objects to the caller of this algorithm.

2. Custom Draw/Display and Camera Offset Logic Algorithm

Every frame, the program attempts to display every object in the current game state to the screen. But this surface or virtual map is larger than the screen itself. Thus, it could only display a portion at a time. In order to create the imagery of movement, the game's map is drawn relative to the player camera's current position. If the camera goes north, every texture is drawn to the south, to simulate the visual of going north.

3. Enemy/Hostile Entities Algorithm

Every enemy in this program is static, meaning that their position does not change, unless to a specified location. But they are able to perform behaviors such

as attacking the player character. To do that, they perform a scan action to see if the player is within range, then perform their respective attack behavior. When a player projectile collides with an enemy sprite, it calls a function in the enemy class which reduces health by one, and erases the object instance if health reaches zero.

D. Data Structures

1. Lists

Lists are used to store a myriad of things, such a list of objects, entities, rooms, projectiles, etc.

2. Tuples

Tuples are used to store certain things such as position/coordinates, and objects

3. Dictionary

Dictionaries are used to store texture and objects such as game state, levels, entities, etc.

4. Classes

Classes are used for all sorts of things, such as game states, player character, entities, projectiles, weapons, tiles, generation algorithms, etc.

Chapter III
Implementation

A. Implementation Details

1. Frames and Display

   Pygame is used as the primary tool for frames and display. Pygame's clock feature is used to track frames, using 60 as the chosen frames per second. Meanwhile, the display feature is used to create a game window, and everything is drawn onto the surface displayed on the game window.

2. Main Game Loop

   The main game loop consists of a while loop, which calls upon a "run" function of the active game state every frame. There consist multiple game states, such as "menu" and "gameplay", and they each have a "run" function that is called every frame, and these functions have the functionalities of their respective game state, acting as the primary controller of what is displayed at any given moment. Dictionaries are used to store game states.

3. Menu, Game Over, and Game Won game-states

   Each of these three game states are simple display states, in which images and texts are displayed. These screens are meant to be limited in their features, as they serve primarily as intermediary states. Nevertheless, each is a class that has a "run" function which is called every frame.

4. Gameplay game-state

   The gameplay state is a major state in the program, as it manages the gameplay features of the program itself. First, it calls upon two other classes, being the player class and level class, to create instances of these classes and store them. The primary purpose of this state is to manage active levels and handle drawing using a custom drawing class. It draws the player character in the center of the screen, and draws everything else based on an offset, which is obtained by

calculating the position of every other texture and drawing them relative to the player's current position, which is also drawn relative to itself to ensure it is on the center of the screen. It follows, the entire surface of the map is drawn every frame to the screen, but only a partial amount is visible in any given moment. Every texture or sprite has a position on the virtual map, and the game essentially takes their position, and draws them based on where the camera is in any given moment. If the camera goes north, every other thing is drawn an equal distance to the south to simulate the visuals of going north.

5. Player Character

This section of the program starts with an attempt to load all relevant textures, using a try-except block and using default textures if this process fails. There exists a "Player" class which uses Pygame's "Sprite" feature to manage this class object's textures and collision. The player character is the user's vessel to interact with the game's world. The player itself is always drawn to the center of the screen, with the ability to move and change its position, which the gameplay state uses to create the illusion of movement in the display. The player has a hitbox and health bar, with the ability to induce a game over state if their health reaches 0. Their sprite has the ability to rotate and aim toward the mouse cursor, and has the ability to shoot projectiles by storing an instance of the "gun" class.

6. Hostile Entities

This section of the program starts with an attempt to load all relevant textures, using a try-except block and using default textures if this process fails. Enemy AI in this program follows a simple stand still and attack the player if they are in range. Using a base enemy class that inherits from Pygame's "Sprite" class, while also being used to manage activities, while child classes are used to determine certain specific behaviors such as attack patterns. The base class itself has the ability to scan for player, attack, manage its health, and erase itself if that health reaches 0. It also has an optional calculate angle to player should their behavior need player location and aiming.

7. Projectiles

This section of the program starts with an attempt to load all relevant textures, using a try-except block and using default textures if this process fails. For the "projectiles" feature in this game, this section contains a base projectile class which manages base behavior, with child classes that inherits from this base class, with the possibility of new behavior.

The main "Projectile" class itself inherits Pygame's "Sprite" class in order to use its functions for display and collision. As for custom functions, this class contains a move function, which changes the projectile's position based on the angle that it spawned in. This class also contains a function to manage projectile's lifespan, which erases a given instance after a given time has passed. All of this is called by the update function, which can be called each frame.

8. Procedural World Generation

The procedural world generation logic in this program is one of its most ambitious and complex algorithms—or at least—complex relative to the rest of the program. The primary purpose of this algorithm is to generate a level map using algorithms alone, without the need for hand-crafting every pixel. Essentially, this algorithm is both a convenience, and an object a novelty. The main idea of this algorithm is that it is called, and it returns a world/map to the caller to be inserted into the caller's sprite groups. The map itself follows a grid-like generation pattern, with square rooms being generated randomly and connected to another.

When called, this generation accepts several parameters, primarily the dimensions of the rooms, and how many rooms is desired to be generated. After this, it creates a list of rooms to later be returned, and create a base room to start generation. On the side, it also creates a list of coordinates, starting with (0, 0) for the origin room. This is to ensure that no overlaps occur. It also creates some attributes to later find the furthest room geometrically from the origin room. As for the room objects themselves, they are created by calling another class called the "Room" class, which itself calls another class called the "Wall" class. Together, these three classes allow the random creation of any number of rooms,

whereas without the world generation logic, each room must be created and specified in the code.

What is called to start generation is a function in the world generation class that initiates generation and returns the list of rooms mentioned. The way this part of the algorithm works is that it essentially generates the desired amount of room using a while loop, and uses the random module to randomly choose a room to turn into the parent room, then it chooses a direction to generate a room in, check if that direction has a room already generated, then generates a room if there is no room in that direction. For the generation itself, it first appends a new room object into the list of rooms, then inserts the appropriate doors in the correct direction for both the new and parent room for later. But the doors are generated later, and this list that each room has for their doors simply are commands for later.

After each room has been generated, this algorithm moves to the door generation algorithm, which simply adds and deletes certain walls in certain directions in each room, based on the list of directions in the door list of every room. After all doors have been generated, it then chooses a random variation for each room out of the pool of pre-existing variation, determining what special "objects" and enemies each room will have. In this step, it also first finds the furthest room by using the absolute amount of the custom coordinates of each room, before determining the furthest room as a special variation. Finally, it returns a list of these room objects, each holding their own list of objects, and have it all be received by the caller.

9. Level Logic

The level logic consists of two separate parts, the parent level class, and the child level classes. The parent class handles the main algorithms, while the child class includes any special behaviors such as for the boss level. The first thing that the parent room class does take in its parameters of screen, player character, and world generation arguments. Then, should world generation be called, it will call the world generation class which returns room objects to be stored by level in a list. Then, the level class creates a dictionary of sprite groups using Pygame's

"Sprite" feature. Then, it proceeds to extract every wall and enemy object from the rooms in the list of rooms, and input them into their respective sprite groups.

This class also calculates the current offset and collision mechanics using Pygame's "spritecollide" feature. Each level also has their own run/update function, which updates/calls the offset, collision, and enemy behavior function. The child classes all follow similar logic, with the exception of the boss level child class, which alters directly certain elements of the parent level class.

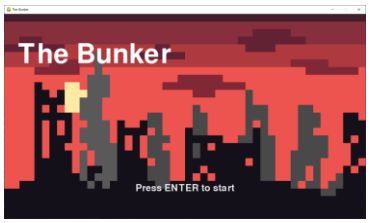
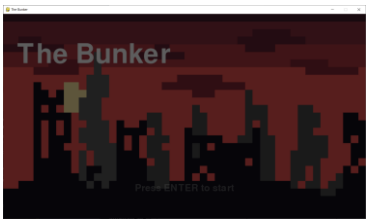
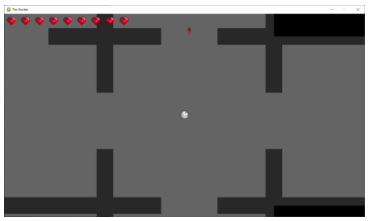
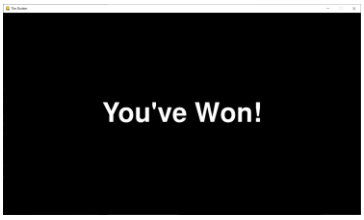
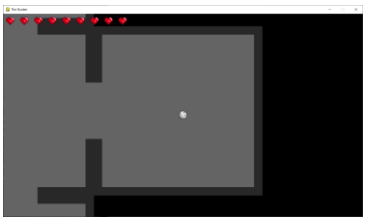
10. Error Handling

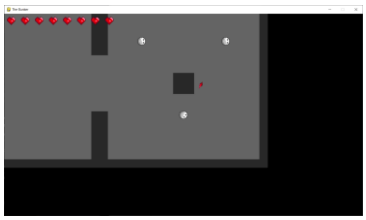
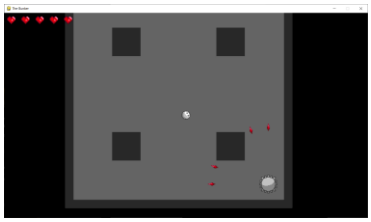
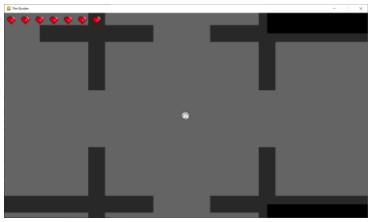
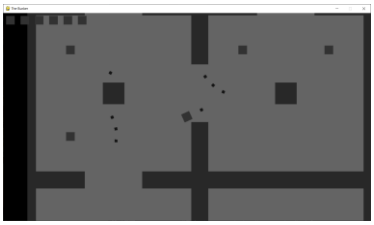
There are a few instances of error handling in this program, primarily surrounding game state changes and asset loading. Try-Except blocks are used in instances of asset loading to ensure that, should any asset fail to load due to any reason, the program will still be able to run. Missing assets are replaced with defaults or not included if it fails to load. Meanwhile, when changing game states, there is a check to see if the new game state is valid, and if it is not, the program stays in the current game state.

Chapter IV

Evidence

A. Screenshots

Num	System	Screenshot 1	Screenshot 2
1.	Menu state	(Idle menu) 	(Transition into gameplay) 
2.	Gameplay state	(Idle gameplay) 	
3.	Game over state	(Game over) 	
4.	Victory state	(Victory/Ending) 	
5.	Player character with movement and shooting	(Location 1 and idle) 	(Location 2 and shooting) 

6.	Hostile entities	(Normal enemies) 	(Final Boss) 
7.	Procedural Generation	(Generation 1) 	(Generation 2) 
8.	Asset Error Handling	(Textures failed to load) 	

Statement on the Use of Artificial Intelligence

AI Tools Used

- ChatGPT – Version 4.0

Purpose of AI Tools

- ChatGPT – Version 4.0
  This AI tool was used in the learning part of the project, in which I use this tool to help me learn about Pygame.

Prompt Examples

- "How do I use Pygame to play a piece of audio?"
- "What feature of Pygame can I use to detect collision, and how do they work?"

Critical Reflection

The primary purpose of AI usage in this project is to help learn more about Pygame and its features. AI would output suggestions on what features can be use, which I would then research myself by going to popular Pygame documentations online. If I have any confusions with the features I find, I will ask AI further on how these features work. But I do not integrate AI's output into my work directly, as AI only serves the role of helping me learn about Pygame.

Final Statement

Overall, I used AI tools such as ChatGPT 4.0 to help me learn more about Pygame and its features. I would take AI's output as suggestions, and then do my own research to confirm the output and further increase my understanding. AI did not make a single line of code, image, or text. I tried using AI for code, but I didn't understand it, and I don't like to write what I don't understand, so I cancelled that idea.

References

JCode. (2023, Mar 17). *Pygame top down shooter tutorial in python #1 – Setup & player movement* [Video]. Youtube. https://www.youtube.com/watch?v=OUOI6iCrmCk

JCode. (2023, Mar 22). *Pygame top down shooter tutorial in python #2 – Player rotation* [Video]. Youtube. https://www.youtube.com/watch?v=wu1cd1Qycz4

JCode. (2023, Mar 29). *Pygame top down shooter tutorial in python #3 – Player shooting* [Video]. Youtube. https://www.youtube.com/watch?v=paItMJuKwwM

JCode. (2023, April 18). *Pygame top down shooter tutorial in python #4 – Camera & creating the enemy* [Video]. Youtube. https://www.youtube.com/watch?v=IhJCpya_FW8

Pygame community. (2025). Pygame (Version 2.6.1) [Computer software]. https://www.pygame.org

Appendices

Repository Link: https://github.com/Bdarz/Algorithm-Programming---Final-Project---The-Bunker-

Poster: