**Group 5**

**Harrison Bawden, Matthew Carroll, Shane Fox,**

**Kenneth Gallo, Ben Leasure, Alex Scarboro**

## 1. Project Overview

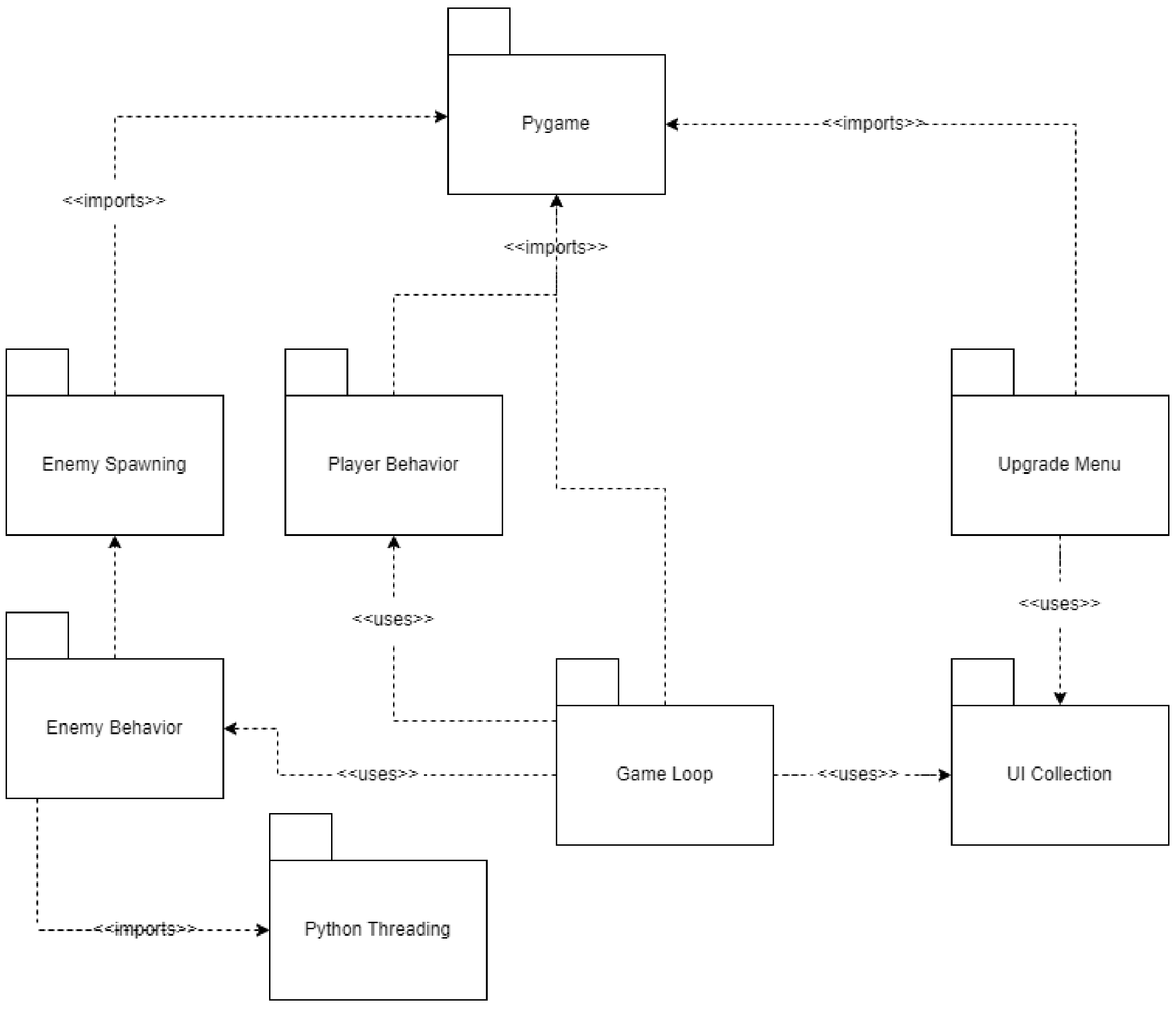
The project is a small game. The game has been implemented in the Python programming language, using the Pygame module for the core game engine. We also used the Pytmx module, which is used specifically for importing and processing the map and overworld assets that create the setting for the game. The map itself was created using Tiled, a mapping application for two-dimensional game maps. The core gameplay is a top-down shooter, with enemies coming in waves as the player defends by moving and attacks by shooting a projectile that damages the enemies. Generally, the gameplay itself is not overly complex, however there are some mechanics that add to the variability of the gameplay. There is an upgrade system in which the player earns “money” for defeating enemies. The upgrades include increasing health, damage and the speed of the player character.

Due to the nature of gaming, the stakeholders for our game are essentially the people who would play the game. In that realm, the game is targeted towards people who would like a challenging game that does not involve a large or complex scope, and can be played in snippets without large time or power commitments.

## 2. Architectural Overview

For the overall architecture of the system, we had initially discussed two possible options: Creating the game as a web application, implementing it using JavaScript for user input and manipulation, and the second option was implementing the game as a standalone application that could run locally. We decided against creating the game as a web application so that we could focus more on the core functionality of the game instead of involving or spending effort on implementing the web architecture. Generally, we believe that creating the game as a standalone application is a better solution and more inline with how video games are delivered to end users. Creating the game as a standalone application also gives us the opportunity to deliver the game as a deliverable executable file.

## 2.1. Subsystem Architecture



Note: Since we used Python, our project was not organized technically by packages, but instead by a general file structure. The term “package” will generally refer to a group of classes with a specific functionality.

- **Pygame:** The Pygame module itself. This package is imported by all functional packages in the system, and provides functions to create sprites, access a sprite’s rect and movement functions, display functions, and main game loop functionality.
- **UI Collection:** A package that contains the classes for text displays buttons, and other UI components that are reused. Both the upgrade menu and game loop use this frequently.
- **Upgrade Menu:** The package that handles the logic and display of the upgrade menu. This contains the data of the player’s current level in different areas of proficiency, and allows the user to upgrade their abilities. This communicates with the UI Collection package for display and the Pygame module for functionality.
- **Game Loop:** This package encompasses the main menu and the main game loop. The menus all access the UI Collection as previously mentioned for the buttons to start/quit the game, and also any titles or displays such as currency, high score, and current round. The Pygame Module is needed here in order to check for **events**, as anything a user clicks or presses is registered as an event. Functionality for these events may affect the player, enemies, or other, so the Game Loop also uses the Player Behavior and Enemy Behavior packages.
- **Player Behavior:** The Player Behavior package contains all logic for the player, including movement of the sprite, logic for health, death, and other attributes of the player. Some of this functionality requires the pygame module, so that is a dependency.

- **Enemy Spawning:** The logic for the enemy spawning is contained in the Enemy Spawning package. This includes all logic for the rounds and how many enemies to spawn and which ones should be spawned. Each round spawns enemies on a timer, so in order to avoid delaying the entire game loop, the Enemy Spawning package uses the Python Threading package. This allows every round to have its own thread which can handle the enemy spawn timing.
- **Enemy Behavior:** Similar to the Play Behavior package, the Enemy Behavior package involves all attributes and functions of the enemies, including following and attacking the player, changing health, checking enemy collision, and others. This is used by the enemy spawning class to create enemies, and thus the Enemy Behavior and Enemy Spawning packages have a dependency.
- **Python Threading:** This package is the built in python module that contains classes and functions for multithreading. This is used by the Enemy Spawning as previously mentioned, and for us it allows delaying certain game events without delaying the entire game loop.

### 2.2 Deployment Architecture

This software will run client-side on a single processor, therefore there is no need for hosting on an external server.

## 2.3 Persistent Data Storage

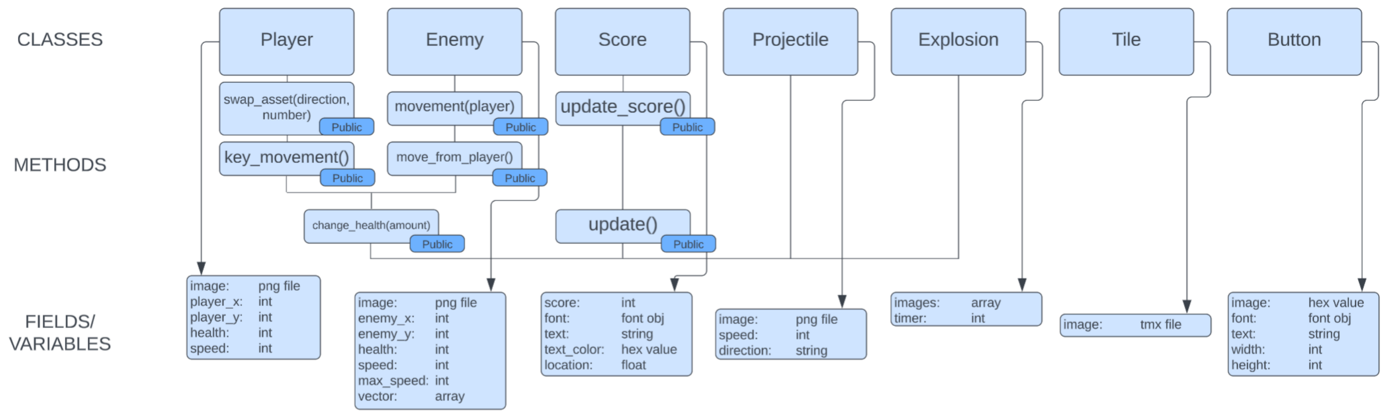
This software does not require persistent data storage, as the data is stored one instance at a time and is reset when the game is closed. The only data that would be stored is the high score, however this will be stored locally in a variable and not remain persistent.

## 2.4 Global Control Flow

The game is event-driven, with two while loops controlling two different states of the game, the first is the main menu loop, the second is the main game loop. The main menu loop runs on startup of the game and allows the user two options, to either “Start” or “Quit”. Clicking on start then continues into the gameplay or clicking on quit terminates the program. Once start is clicked, and before the main game loop begins, the game assets get loaded into the game engine, with these being the game map, camera, the necessary sprites groups, and the game’s song. The main game loop continues to run while the running flag remains true, and continues to remain true until the game is closed by the player. The main game loop contains all the logic for the event driven structure of the game, including things like firing the projectiles, moving the player, spawning the enemies, checking the player’s health, counting the money earned, and updating the game display with each frame. The player character moves and shoots the projectile based on user input while the enemies move towards the player's location.

## 3. Detailed System Design

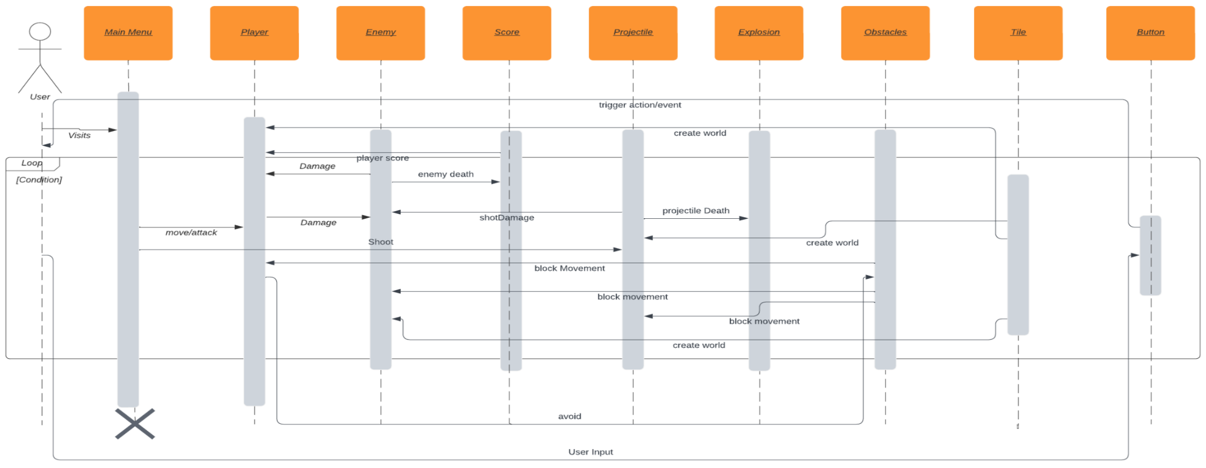
### 3.1 Static View



The classes are named to be self-explanatory, as they contain the attributes and methods that create and control the game element they represent ("Player", "Projectile", etc.). Each class that controls an entity, in this case Player and Enemy, have functions for movement. The player moves based on user keyboard input, while the enemy moves based on the player's location. Each of these classes share a health function, and every class that is dynamic in gameplay shares the update function (with exception of "Tile" and "Button," as these are not gameplay-dynamic). The health function contains values for health regeneration and health reduction when taking damage. The update function continuously draws up-to-date data to the game surface. The entity functions share variables such as image (models), x and y coordinate values, a health value, and speed. The enemy requires an additional field for vector, used to track and follow player movement. Furthermore, projectiles have speed and direction values, and explosions have a timer value for cooldown. Lastly, the tiles are their own 'tmx' file type, and the button class uses values for stylization and positioning on screen.

The game functionality was designed in this way so that each aspect being displayed on the game surface is independent of one another. This is a beneficial design approach for various reasons. For one, debugging is easier with this architecture because a single error can be pin-pointed to a specific class if an import is not functioning properly. Another benefit is that entity-specific attributes are well ordered and easily identifiable, which is important for when code gets complex.

## 3.2 Dynamic View



As for the Sequence Diagram, the game starts with the User at the Main Menu, where they can either choose to enter the game, or exit the game, by clicking the appropriate button. If they choose to enter the game, the world is created and the main body or loop of the program is executed. Simultaneously, the player and a few enemies spawn into the world. The player and enemies can move in 2 dimensions and can attack the other - the enemy attacks the player by physical contact and the player can attack the enemies by shooting a projectile from a gun. Damage to both the enemy and the player is tracked with a visual health bar indicator above the head of the player and enemy. Killing an enemy results in the player's score increasing, and the player can use their acquired points to purchase upgrades in the upgrade menu. Additionally, there are obstacles that are randomly positioned at the start of the game, and the player and enemies must navigate their way around the obstacles.