1. Variables, Conditionals, Loops, and Collections.

I put a lot of my regular variables for sizes into setting.py, but I have variable in used all over my game.

I use if, else, and elif conditional statements throughout my game.

I use for and while loops.

I use dictionaries for player images for my character selection menu and lists are used for animations

A screen shot of a computer program

Description automatically generated A screenshot of a computer

Description automatically generated

1. code organization (formatting, identifiers, placement of definitions)

I tried to format my code so my formatting was well done, comments are all in the same spot. I tried to be descriptive in my identifiers for variables and definitions of functions and methods.

A screen shot of a computer program

Description automatically generated

1. code decomposition (functions, classes, methods, and modules),

I have a debug function and run\_game function, but most of my code is using classes and then the methods used within those. I have 9 modules, which is the group of related classes/methods/and functions that I have put into files.

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

1. **an understanding of design (including hierarchy of aggregate objects and an inheritance tree)**

**Hierarchy of Aggregate Objects:**

I organized the game into several files and classes that represent different aspects.

A screenshot of a computer

Description automatically generated**Game Class:** Manages nearly the entire game.

**Player Class:** This is the player character, deal with animations, movement, collisions, inherits from pygame.sprite.Sprite, which is built into the Pygame library

**Enemy Class:** Represents the enemy, similar to Player class, but has automatic movement.

**Attack Class:** Attacks made by the player.

**Environment:** Represent all the ground, rocks, and trees. Player interacts with these.

**Button Class:** This is a clickable button created for buttons on menus

**HighscoreManager Class:** Manages high scores

By creating different files and classes, it makes it easier to read and edit what I need. It also make adding new designs much easier, and using tools for maintenance.

**Inheritance Tree:**

Inheritance is used to create the hierarchy of classes.

**pygame.sprite.Sprite:** Player, Enemy, Attack, Block, Tree, Ground, Tilemap, Button inherit from the Pygame provided pygame.sprite.Sprite

**Game:** Manages instances of various classes, player, enemy, and orchestrates the game loop.

**Spritesheet:** Handles the sprite sheets, used by environment, player, enemy, and attacks.

**HighscoreManager:**  Manages high scores.

1. **an understanding of testing (test your methods and attributes, maybe have a whole automated example!)**

I have a debug function that allows me to display debug information on the pygame window. It’s convenient when I need to debug by playing the game.

I have a testgame class which allows me to check if the number of enemies increases to see if they were spawning correctly.

I have a test that verifies the expected behavior of methods.

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

1. **user IO, file IO, and input validation**

I have players enter initials upon getting a high score (user i/o). This is then saved using file i/o. I have it so that I have a 10 name list that is ordered by highest score, which means I must write and read the file created. I have input validation that ensures that only letters can be used and 3 must be entered to move on to the next screen.

A screen shot of a computer program

Description automatically generated

1. **recursion**

I have a recursive method for my countdown for my game over screen. I am calling up the countdown\_game\_over method within the method itself. unction recursively decrements the remaining time on a game over screen, introducing a one-second delay between steps, and displays either the high score or regular score based on the player's score.

A screenshot of a computer program

Description automatically generated

1. **GUI components and event driven programming.**

I have a lot of examples of GUI components with Pygame. In the pictures you can see where I render words, get their rectangle around them, and then use blit to put them on the screen. I also have players, enemies, and environmental objects that are animated.

For event driven programming, I have timers, mouse clicks, button clicks, collisions.

A computer screen shot of text

Description automatically generated

A computer screen shot of a program

Description automatically generated

1. **Exceptions**

I have a few exceptions in my code. I have one the tries to load the background music and in case of an error prints (‘cannot load sound’)

I also have one that is used to update the sprites in my game and in case of error raises a SystemExit(‘”Error”)

I also use one for my HighscoreManager for loading scores.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

1. **Inheritance**

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated pygame.sprite.Sprite is provided by the pygame library and I use it in quite a few classes so that I am able to use Pygame’s get\_rect. However, I feel like that’s not what you want.

My Block class is inherited from my Spritesheet class. This means that I can use the methods that are defined in there, like in this case get\_sprite.