Pirate adventure

#### By Charles Robert Milne

Contents

[By Charles Robert Milne 0](#_Toc188620764)

[**Analysis:** 3](#_Toc188620765)

[1.1 Statement of problem 3](#_Toc188620766)

[1.2 End users 3](#_Toc188620767)

[1.3 Initial research 3](#_Toc188620768)

[1.3.1 Existing similar programs 4](#_Toc188620769)

[1.3.2 First Interview 5](#_Toc188620770)

[1.4 Further research 5](#_Toc188620771)

[1.4.1 Prototype 5](#_Toc188620772)

[1.4.2 Second Interview 6](#_Toc188620773)

[1.5 Key requirements 7](#_Toc188620774)

[1.6 Objectives 7](#_Toc188620775)

[1.7 Modelling 9](#_Toc188620776)

[1.7.1 Potential data types 9](#_Toc188620777)

[1.7.2 Diagrams 9](#_Toc188620778)

[Formulae: 9](#_Toc188620779)

[Important variables: 10](#_Toc188620780)

[Class descriptions: 10](#_Toc188620781)

[Class hierarchy: 17](#_Toc188620782)

[State transition diagrams: 18](#_Toc188620783)

[User interface: 18](#_Toc188620784)

[Design: 19](#_Toc188620785)

[2.1 Programing language and libraries 19](#_Toc188620786)

[2.2 Cass descriptions and interactions 20](#_Toc188620787)

[Class interactions 21](#_Toc188620788)

[Class hierarchy 22](#_Toc188620789)

[2.5 UML class diagram and descriptions 23](#_Toc188620790)

[2.3 Algorithm design 25](#_Toc188620791)

[Algorithm 1 26](#_Toc188620792)

[Algorithm 2 27](#_Toc188620793)

[File reading algorithm 28](#_Toc188620794)

[2.4 User interface (GUI) 29](#_Toc188620795)

[Technical Solution 33](#_Toc188620796)

[3.1 Contents page 33](#_Toc188620797)

[3.2 Code 34](#_Toc188620798)

[Settings.py 34](#_Toc188620799)

[Data.py 35](#_Toc188620800)

[Groups.py 43](#_Toc188620801)

[Level.py 49](#_Toc188620802)

[main.py 59](#_Toc188620803)

[Menu.py 66](#_Toc188620804)

[Overworld.py 75](#_Toc188620805)

[Player.py 82](#_Toc188620806)

[Select\_save.py 93](#_Toc188620807)

[Sprites.py 95](#_Toc188620808)

[Timer.py 107](#_Toc188620809)

[Files.py 109](#_Toc188620810)

[Testing 111](#_Toc188620811)

[Evaluation 115](#_Toc188620812)

[5.1 Objectives 115](#_Toc188620813)

[5.2 User feedback 118](#_Toc188620814)

[5.3 System improvements 118](#_Toc188620815)

# **Analysis:**

## 1.1 Statement of problem

Since I was young I have played games, both physical and video games which have been some of my fondest memories, However I have only played intense multiplayer games and so I decided to create a different style of game that I could play, a 2-dimensional platformer game in the style of the iconic game super Mario bros, with extra features that I think would improve the game. My project, titled pirate adventure, starts on “overworld” island where you can choose between any levels you’ve unlocked by walking around the island. When you select the level, you control your character using arrow keys/ ‘wasd’ to move around and space to jump the “camera” then moves with you to uncover more of the level until you eventually find the goal and unlock the next level.

## 1.2 End users

My end user would be anyone who wants to play the game – this will most likely be students such as myself around the age of 15-18. The end user who will test and suggest improvements will be my close friend Charlie Owens who is in the same year as me and has shown interest in my game and others like it and so would be well suited to test the gameplay and find bugs along with fairly rating parts of the project which many need improvement.

## 1.3 Initial research

2D games were the first video games to every be created, starting in 1950 with Tic-Tak-Toe. This was played on a computer which was four meters tall and could only play tic-tac-toe on a lightbulb-backed display. Then games like pong in 1972 which had to be run on computers the size of refrigerators. 9 years after that the more complex Donkey Kong game released shortly followed by the first Super Mario Bros game. However, after evolution of 3D games these iconic games slowly lost popularity, and I want to create a game to revitalise 2D games by using modern library’s such as ‘pygame’ to create an aesthetically pleasing, relaxing yet fun modern 2D platformer.

Donkey Kong(1981): Cyberpunk 2077(2020):

<https://en.wikipedia.org/wiki/Donkey_Kong_(1981_video_game)> <https://en.wikipedia.org/wiki/Cyberpunk_2077>

### 1.3.1 Existing similar programs

The core gameplay of my project will be similar to Super Mario Bros where you move forward

(right) to reach the end flag and I like these concepts. However, I dislike the graphics, the monsters, how you acquire coins and the level selection along. I also like the idea of power ups but I would like to change the power ups themselves and how you get them.

Another similar game is Rayman legends which is also a single player 2d game where you explore forward/right to reach the end and advance to the next world. I really like the graphics of this game where the background seems 3D and I like the idea of bosses you have to fight. I also like how you

 can buy skins with your coins which I think would add something for the players to aim for and so I would also probably try to implement save files so that you can save you coins and skins when you close the program. Although I dislike how there are many different places you have to go to get skins and advance so I would make a simpler shop and a smaller world and a slower player but with everything more compact.

### 1.3.2 First Interview

I intend to interview a few people from my year to gauge a collective interest rather than just the interests of a certain person. These are the questions that I intend to ask and a summary of the collective answers:

* Have you played a platformer game before?
* If so what features did you like/dislike?
* What features of your own would you like to add?
* How do want the health/difficulty system to work?

**Interview Results:**

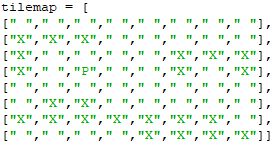
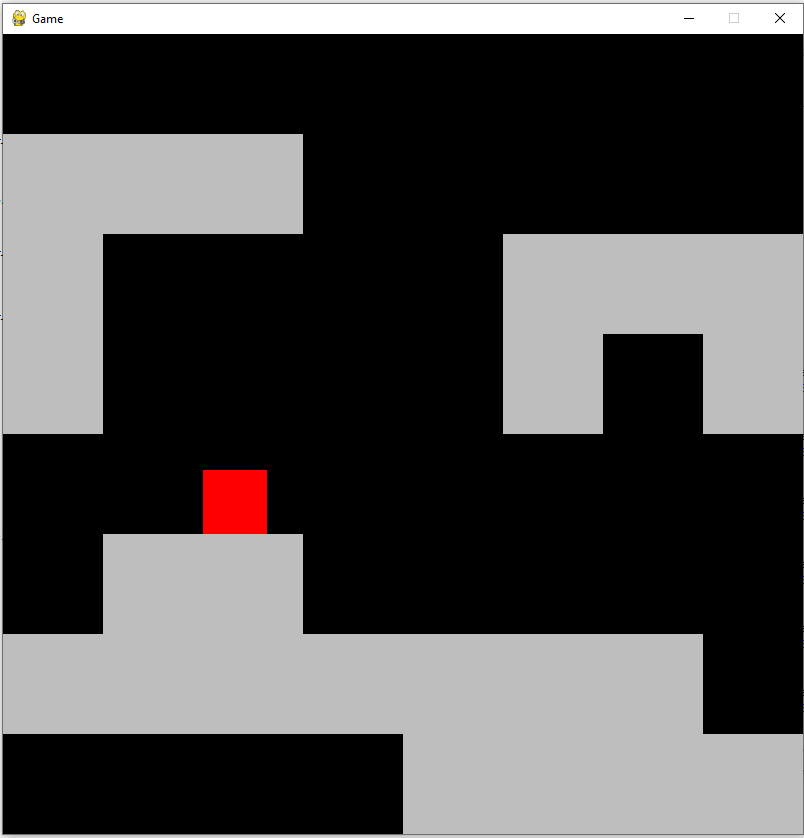
* Yes
* Exploring an interesting world and finding hidden secrets/ Easter eggs
* You can unlock skins/pets as you progress and earn coins/points
* You can regen health buy getting a certain amount of coins and you can select a difficulty which determines your start health and how many coins you need to gain an extra heart.

## 1.4 Further research

### 1.4.1 Prototype

My prototype aims to test the basic logic of the game and a simplified system to import a level layout. The prototype includes a class for the player which holds the movement and collision basics, theirs also a small class and method for the level and its setup as well as a class for the “building blocks” of the level. I also created a 2d array which from which the level is made from which proves to me that I can import the level designs and don’t have to hard code them into the program.

In order to learn the basics of pygame and 2d platformer logic I watched “Clear Codes” python and pygame tutorials, which can be found at: <https://www.youtube.com/@ClearCode>



Game interface (red dot is player) 2d array to design the level (“X” is a block and “P” is the with moving and jumping player spawn and a blank string is nothing)

The prototype had effective/reliable collision physics and good modularisation (using classes), however the players speed was related to the fps which seems to me like a waste so I would want to make my final project frame independent so it could run as many frames as possible without effecting the player speed. I have also realised that it would nearly be impossible to design/import a level using 2d arrays or csv files as their will likely be objects on top of one another in the final project, and so I will save the levels as a tmx file and then use a python library called “pytmx” to import the various layers, objects and their positions for the level.

### 1.4.2 Second Interview

To determine whether the variables in my prototype are satisfactory I intend to ask my end user the following questions:

* How do you find the player speed?(too slow/fast or just right)
* How do you find the gravity?
* How do you find the jumping mechanics?
* How long do you want each level to last/ take to complete?
* What types of enemies do you want?

**Second Interview results:**

* Good, liked how you can move while in the air
* Perfect
* Would like to add double jumping and wall jumping/sliding
* A few minutes, maybe 3-5
* Some that shoot things at you, some bosses that you have to fight and some that just walk back and forth like in Mario

## 1.5 Key requirements

The game needs to have multiple “saves” which are your previous games that you can continue, then the actual game needs to have movement including jumping and running there also needs to be items to collect and ways to die: traps and monsters. There also should be an end goal for each level that needs to be reached in order to unlock the next level.

## 1.6 Objectives

1. Updated graphics
   1. Pixel art for characters and level (from [https://github.com/clear-code-projects](https://github.com/clear-code-projects/Super-Pirate-World/tree/main/graphics))
   2. Moving background(moving clouds) that move for right to left then off the screen
   3. Animations for enemies, player and any other appropriate level assets (a few frames for each animation) , a run, jump, fall, death and idle animation
2. Multiple levels
   1. At least 5 levels
   2. Different layouts (same images can be used), each level has multiple monsters traps and items with a goal flag that you have to reach at the end
   3. Various length/ difficulties (time to complete = 3-5 minutes)
   4. Can choose which of your unlocked levels to play
3. Items
   1. Coins which regenerate health when you collect enough (amount depends on the difficulty around 100 – 300) which are everywhere in each level
   2. Various types of items that give coins (silver/gold coins and chests) that give different amounts of coins with the more common ones giving less
   3. Skulls, which are the shop currency. Only 1-3 hidden in each level
4. Movement
   1. You can double jump (jump (once) again when you in the air)
   2. You can (slowly)slide down a wall and jump off it.
   3. Can land on some objects e.g. moving platforms
   4. Can fall off the map (down holes/gaps) which kills you
5. Fighting
   1. You can swing your sword every half a second which will block the shoots of monsters and deflect the monsters that walk back and forth
   2. There will also be traps to avoid (like spikes in the floor)
   3. You can also find abilities like a shield to help you complete the level
6. Shop
   1. When the player presses escape the shop opens
   2. It displays what they can buy and haven’t brought already
   3. Displays your amount of skulls
   4. Allows you to purchase skins and other items like pets
   5. They can be equipped when the user closes the shop
7. Saves data
   1. After you run the program you select 1 of the 3 possible save files
   2. The save files are all individual text files
   3. When you close the game your progress gets saved. The progress being saved is the highest level unlocked, the current level you’re on, your skins/ pets, difficulty and your skulls(currency)

## 1.7 Modelling

### 1.7.1 Potential data types

For my various images and sounds, which I will have to use for the graphics, I will store them in a dictionary which may also contain other dictionaries or lists for some values to better organise the images into groups and for fast accessing. Each value for the dictionaries with be a pygame loaded image I will use convert\_alpha() for faster blitting. The levels will also be contained in a dictionary where the key is their order of being unlock(first level will be 0) the value will the by their respective tmx file. I will also use many 2-dimensional vectors, for the movement of various objects and additionally for the monsters attacks and movement this vector will then be added to the sprites positions each frame(loop) to give them the effect of moving.

### 1.7.2 Diagrams

#### Formulae:

X co-ordinate for a circle(centred at 0,0): radius \* cos(angle)

y co-ordinate for a circle(centred at 0,0): radius \* sin(angle)

To calculate acceleration and distance due to gravity:

speed += acceleration/2 \* delta time

position += speed \* delta time

speed += acceleration/2 \* delta time

(by “Manveti” on stack overflow)

Distance from a point: sqrt((delta x)^2 + (delta y)^2) (Pythagoras)

Splitting the screen into sections and finding the middle of each (for shop):

Space = screen width/ num of items

Middle = (space\*item number + space\*(item number+1))/2

#### Important variables:

These constant variables are referenced across the project for consistency and to make it easy to change any of these variables for every part of the project.

|  |  |  |
| --- | --- | --- |
| Variable | Data Type | Purpose |
| Screen width, Height | Integer | Variables for size of screen that can be changed to increase screen size |
| Tile size | integer | Size in pixels of each building block |
| orders | dictionary | Contains the order that each layer should be drawn in (values will be numbers, higher means drawn later) |

#### Class descriptions:

**Game:**

This is the first class to be updated I each frame and it subsequently calls the update method for every other class. This class therefore also controls what classes are run and therefore has a method for when you lose or swap stage. It also contains and imports all the images when first run and it also records the time between each frame

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access type | Field name | Field type | Initial value | description |
| Public | Display surface | Pygame.display | Display with the size of screen width/ height | Will be the screen that everything is drawn on |
| public | clock | Pygame.time.Clock | N/A | Will be used to find the time between each run(frame) so the game can run independent of frame rate |
| public | Current\_level | object | An instance of the level class of the current level | Will call it’s run function in order to run the logic and graphics of the level |
| public | overworld | object | An instance of the overworld class | Its run function will also contain all the logic ang graphics to run the overworld (level selection) |
| public | frames | Dictionary (2-d) | Contains all the images and frames for everything in the game | Will be created when the game is initially loaded and passed as parameters to multiple classes. Some values will be singular or could be a list or another dictionary. |
| Access type | Method name | Parameters | Return values | description |
| public | run | none | none | Calculates data time using clock.tick() then runs the current levels run function |
| public | Import assets | none | none | Sets self.fames equal to the dictionary of images |
| public | lost | none | none | Draws a game over screen with a restart button |
| public | Change stage | overworld | none | Swaps between the overworld and level |

**Level:**

This class creates all the objects/sprites for the level using the map file it also creates the player and runs its update function and other player related functions. This class also contains all the sprite groups and runs all their update methods

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access type | Field name | Field type | Initial value | description |
| public | All sprites | Pygame.sprite.group() | Empty group | Will contain every sprite in the game so that they can all be drawn with .draw() |
| public | Collision sprites | Pygame.sprite.group() | Empty group | Will contain all the sprites that the player can collide with. |
| public | Semi\_collision sprites | Pygame.sprite.group() | Empty group | Will contain all the sprites that the player can collide with and also phase through the bottom |
| public | Damage sprites | Pygame.sprite.group() | Empty group | Contains all the sprites that the play will take damage from if they collide |
| public | Item sprites | Pygame.sprite.group() | Empty group | Contains all of the item sprites |
| public | Moving platforms | Pygame.sprite.group() | Empty group | Contains all the moving platform sprites so you can tell when you’re on one |
| public | player | object | An instance of the player class | Will be the player that the user controls |
| public | Tmx\_map | Tmx file, loaded by pytmx | Is a parameter | Will contain all the info for the level layout |
| public | data | Object of the data class | Contains health and coins etc | Allows the health and other info to be edited when something happens in the level |
| Access type | Method name | Parameters | Return values | description |
| public | run | Delta time | none | Updates and draws all the sprites can calls the player run method and the deal damage method |
| public | Check player | none | none | See if the player has won or died |
| public | Item collision | Item sprites | none | Sees what if an items the player is colliding with and carries out the appropriate action |
| Public | Check camera | none | none | Make sure cameras not showing outside the map stop it if it’s about to (player near the edge) |
| public | setup | Tmx\_map, frames | none | Ran when level is initialised places all the sprites n the right place and adds each one to the correct group. |
| public | Draw abilities | none | none | Draws the abilities you have and which you’ve selected |

**Player:**

This class controls the player by checking for inputs and collisions and reacting accordingly it also animates the player.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access type | Field name | Field type | Initial value | description |
| public | gravity | integer | 1500 | Is the strength of gravity |
| public | image | Pygame.image | Player file | The picture representing the player |
| public | Hitbox\_rect | Pygame.frect | rect same size and location as image | Represents the players hitbox (used for collisions) |
| public | jump | boolean | false | Sees if the player has pressed the jump key |
| public | On surface | dictionary | False for all keys | Says what surface their on (wall, floor) |
| public | direction | 2d vector | (0,0) | Contains the direction the player should move |
| public | speed | int | ? | Speed for player |
| public | Collision sprites | Pygame.sprite.Group | Is a parameter | Contains all the sprites the player can collide with |
| public | state | string | idle | Contains what the player is doing which is used for animation |
| Access type | Method name | Parameters | Return values | description |
| public | update | Delta time | none | Executes all the methods in the class |
| public | move | Delta time | none | Moves player image and rect according to the direction |
| public | Check\_collisions | none | none | Check if the player is colliding with anything |
| public | animate | non | none | Animates the player |
| public | Check damage | none | none | Sees if the player has been hit and decreases health if they have |

**Sprite / Animated Sprite:**

These classes assign an image, rect and order to each object to allow them to be drawn and collided with. The animated sprite class also animates the object by updating its image.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access type | Field name | Field type | Initial value | description |
| public | order | int | Is a parameter | Contains relative order that its drawn in value is from the orders dictionary |
| public | image | Pygame.image | parameter | Image of the sprite |
| public | rect | Pygame.frect | Rectangle with the same size and location of the image | Used for collisions |
| public | frames | list | parameter | Contains the images that make up the sprites animations |
| public | Frame index | float | 0 | The index of the frames list with the animation is currently at |
| Access type | Method name | Parameters | Return values | description |
| public | update | Delta time | none | Runs all the classes methods |
| public | animate | Delta time | none | Animates the sprite |

**Clam:**

This class shoots a pearl if the player is in front of it, this pearl will then damage the player if it hits them. The clam will keep shooting as long as the player is still in front of it.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access type | Field name | Field type | Initial value | description |
| public | frames | list | List of images | Contains frames for animations |
| public | Collision sprites | Sprite.group | All the collidable sprites | Allows the projectile to see if it hit something |
| Access type | Method name | Parameters | Return values | description |
| public | Flip frames | none | frames | Flips each frame so that the clam is facing the other way |
| public | See player | none | none | Check if the player is in the clam's line of sight |
| public | update | Delta time | none | Animates the clam (inherited from animated sprite) and shoots if necessary |

**Tooth:**

This class will keep moving left and right along the platform it spawns on and will damage the player if they collide. It can also be deflected by the player

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access type | Field name | Field type | Initial value | description |
| public | frames | list | List of images | Contains frames for animations |
| public | Collision sprites | Sprite.group | All the collidable sprites | Allows the enemy to see if it hit something |
| Access type | Method name | Parameters | Return values | description |
| public | deflect | none | none | Reverses direction |
| public | move | none | none | Moves and turns around the enemy |
| public | update | Delta time | none | Animates the enemy (inherited from animated sprite) |

**Overworld:**

This class draws all the overworld objects and the unlocked levels using the map file. It is also responsible for the players movement and animation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access type | Field name | Field type | Initial value | description |
| public | Tmx\_map | Tmx file, loaded with pytmx | parameter | Contains the layout of the overworld |
| public | Current level | int | 0 | the level that the players currently on |
| Public | Unlocked level | Int | 0 | The highest level that the player has unlocked |
| Access type | Method name | Parameters | Return values | description |
| public | run | none | none | Runs all the classes methods |
| public | move | none | none | Moves the player along the path (either backwards or forwards) |
| public | animate | Delta time | none | Animates player |
| public | setup | map | none | Draws everything except locked nodes and paths |

**Data:**

This class contains valuable variables that change throughout the game, it also contains methods for setting certain variables when the player restarts or creates a new save file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access type | Field name | Field type | Initial value | description |
| private | hearts | int | Depends on difficulty | Number of hearts the player has when it reaches 0 the player dies |
| private | coins | int | 0 | Amount of coins the player has |
| private | skulls | int | 0 | Amount of skulls the player has (for the shop) |
| Public | Level unlocked | int | 0 | Max level that the player has unlocked |
| public | difficulty | int | 5 | How many hearts you have |
| Access type | Method name | Parameters | Return values | description |
| public | Create hearts | none | none | Draws the amount of hearts the user has |
| public | setup | data | File num | Loads saved data |
| public | Chose difficulty | none | none | Creates a screen where the user can select their difficulty |
| public | save | none | none | Writes current data to the previously selected save file |
| public | Flicker heart | none | none | Creates flicker effect so you can more easily see if you lost or gained hearts |

**Menu:**

This class draws the pause menu with its corresponding buttons and calls the appropriate functions when a button is clicked

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access type | Field name | Field type | Initial value | description |
| public | screen | pygame.display | The screen | Screen to draw on |
| public | frames | dictionary | parameter | All the images needed |
| Public | Resume/ shop/ quit buttons | frect | Frect generated from the button image | A rect representing each buttons hitbox |
| Access type | Method name | Parameters | Return values | description |
| public | fade | none | none | Darkens the screen |
| public | Draw menu | none | none | Draws all the buttons |
| public | Check mouse | none | none | Sees what button your mouse is over |
| public | run | Player, all sprites  (to redraw background) | none | Runs all the methods and will run the shop class if it’s clicked |

**Shop:**

This class draws the shop using various hard coded list for the items, their labels and their prices. It also allows the user to purchase the items

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access type | Field name | Field type | Initial value | description |
| public | items | list | List containing image of a dog | Images of the items |
| public | Clouds / sky etc | Pygame.surface | parameter | All the images needed for the background |
| Access type | Method name | Parameters | Return values | description |
| public | Draw items | none | none | Draws all the items available |
| public | Draw clouds | none | none | Draws background |
| public | Draw skulls | none | none | Shows how many skulls you have |
| public | click | none | none | Sees if you by something |
| public | run | none  (to redraw background) | none | Runs all the methods and will run the shop class if it’s clicked |

#### Class hierarchy:

Pygame.sprite.Sprite

Sprite

Animated sprite

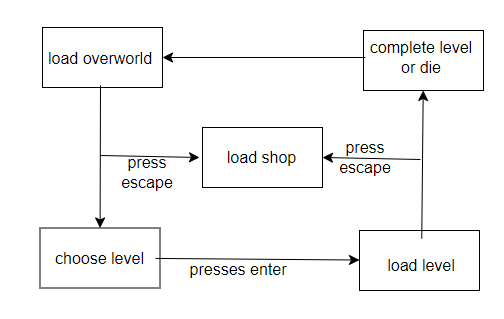
pearl / spiked ball

Particle effects / items

Enemies

path / icon

#### State transition diagrams:



#### User interface:

**Overworld:**

node

paths

player



Level:

100



health

coins

player

enemy

water

grass

# Design:

## 2.1 Programing language and libraries

This program is written in python as it is the language I am most familiar with and the language I am the most comfortable using. It is also the language with the most internal and external libraries for me to utilize and so was the ideal choice.

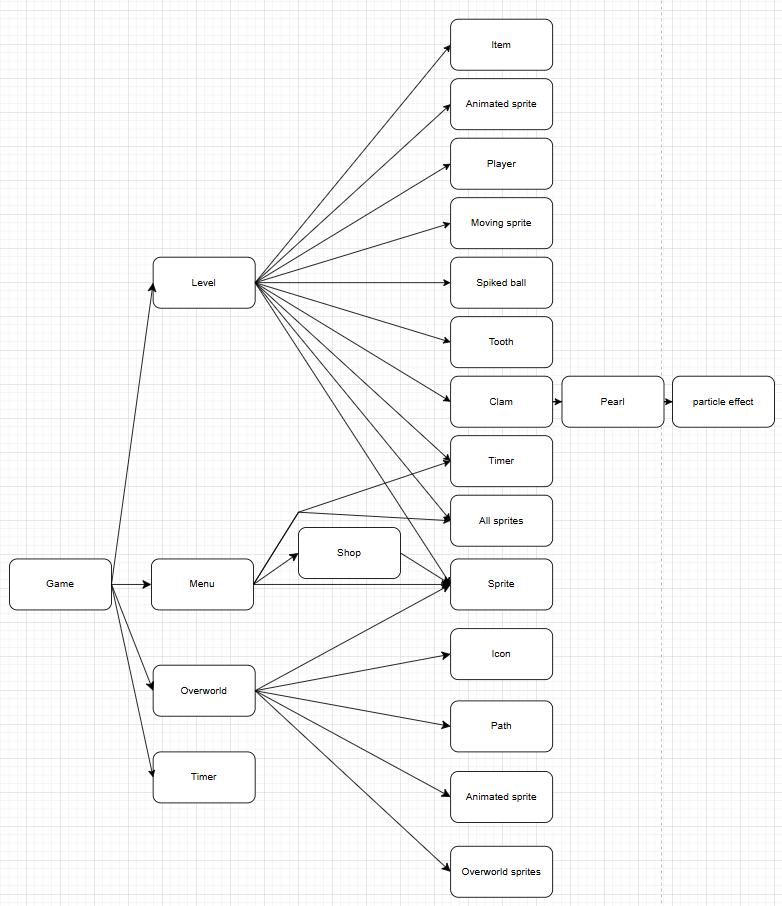
The following are the libraries that I used:

* Pygame-ce: This is a large game development library which is used for most of my project. I use it for the creation of my screen and for the drawing of all my objects onto the screen as well as getting the time between each frame. I chose to use the ‘-ce’ edition as it is the newest and so is the most efficient and has more functions than the normal edition.
* Random: I used random to generate random integers and decimals so that the objects in my project don’t animate at the same speed .
* Math: used for the trigonometric functions as well as for the power and square root function for various calulations.
* OS : A module which provides a portable way of using operating system dependent functionality. From the os module I used:
  + os.path.join which allows me to join path segments when referencing file location without having to use operating system specific Separators.
  + Os.walk which traverses a given directory tree returning a 3-tuple of (directory path, the names of the folders in that directory, and the names of all the files in the directory)
* Sys: I used sys.sxit to immediately end any processes and provide and exit message in the terminal when the player exits the game.
* Pytmx: In order to import and handle my map files I used ‘pytmx.util\_pygame.load\_pygame’ to get a referenceable object from my file map which is a ‘.tmx’ file as it is easier to make, edit and understand the maps.

## 2.2 Cass descriptions and interactions

The image below labeled class interactions is a diagram of the different classes and what other classes they use/ instantiate. Multiple arrows to the same class do not mean that they all reference the same object it just means they all create/use and object of that one class. When you switch into the overworld or level a new object of the class is created and when you leave them the objects are disregarded to save space and so that you don't have to create all the objects at the start. The next image labeled class hierarchy is a diagram containing all the parent and child classes and shows which class inherits from which other class. The descriptions of all my classes and data structures are included in [Class descriptions](#_Class_descriptions:)

#### Class interactions



#### Class hierarchy



## 2.5 UML class diagram and descriptions

Class descriptions:

|  |  |
| --- | --- |
| Class Name | Purpose |
| Data | Store valuable information's like coin count and health and the functions to draw and set them |
| Heart | Represents a life of the player and occasionally animates |
| Coin count | Updates and draws the coin count |
| Select save | Draws a screen where the user can select a save |
| Menu | Draws the buttons for the menu and handles any clicks |
| Shop | Draws a shop screen where the user can purchase items |
| Game | Is the base class which handles the game over screen and calls all the functions required for the normal running of the project |
| Level | Creates and updates all the items in the level (including the player) |
| Player | Handles all the movement and collisions of the player |
| Overworld | Creates an overworld screen and the player so that the user can select a level |
| All sprites | Draws all the sprites and the sky in the right order for the level |
| Overworld sprites | Draws all the sprites and background for the overworld |
| sprite | Creates and object with an image rect and order |
| Animated sprite | Creates a sprite with the animate function to loop through its frames |
| item | An animated sprite with a random animation speed and a name so that we know what to do when the player picks it up |
| Tooth | Creates an animated sprite that walks back and forth on the platform it was placed on |
| Moving sprite | An animated sprite that moves back and forth along is given path (line/rectangle) that can be deflected/turned around by the player |
| Clam | An animated sprite that will shoot a pearl at the player if it sees them |
| Pearl | An enemy sprite that will move always move forward until it hits something where it then dies |
| Particle effect | An animated sprite that will die when it completes one animation cycle |
| Spiked ball | A sprite which will traverse and arc and have chains joining it to its center or rotation |
| Path | A sprite with a name so we know what type of turn it is |
| Icon | A sprite that gets placed so that is feet are at the center of the given node |
| timer | A class that sets itself to active for a given amount of time once the activate procedure is run |

*UML diagram:*

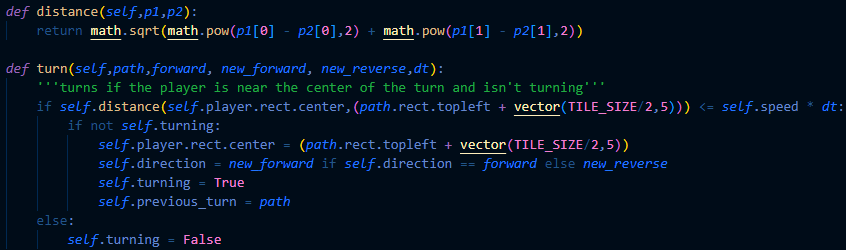
A diagram of a diagram

Description automatically generated with medium confidence

## 2.3 Algorithm design

In this section I will describe what I think to be the most complex algorithms as well as the algorithms I used for data handling in python accompanied by an explanation for each component of the algorithm.

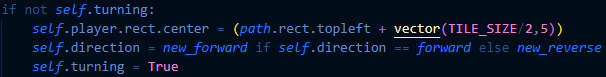
Algorithm 1: This first algorithm is for determining whether the player has reached the center of a turn and how to change their direction based on the name of the corner:





This small function returns the distance between the two inputted position vectors using and extension of Pythagorean theorem, I had to use math.pow instead of just ‘ ^2 ’ since I am dealing with floats which isn’t supported by the ‘ ^ ‘ function.

This if statement uses the distance function above to calculate the distance between the center of the turn and the players feet which is equivalent to the distance between the two position vectors passed into the distance function. It then sees if that distance is less than the distance that the player will travel this frame (speed \* time) meaning that it will see if they have passed the center of the turn or will pass it next frame.



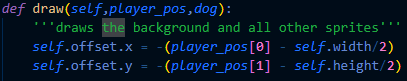
The turning variable is used to make sure that the player hasn’t already turned and therefore the if statement makes sure that the player doesn’t turn multiple times on one turn. The players' feet are then moved to the center of the turn which is the same as moving the players' center to the expression in the brackets (This would be unnoticeable from the users’ perspective as the distance they’ve been moved is less than the distance they would normally travel in the frame). The players’ direction is changed to what the direction should be if the player turned which is also dependent on which side of the turn the player comes from; these variables are passed to the functions as parameters and are determined by the name of the turn. “Self.turning” is then set to true so that the player does try and turn again.



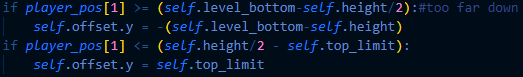
This else statement is only run when the player has passed the center of the turn so it is a reasonable time to set “self.turning” to False so that the player can do any other turns in the path.

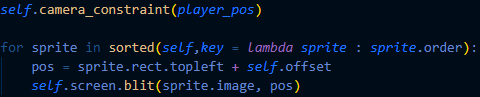
Algorithm 2: This algorithm is used when drawing all the sprites and keeps the player in the center of the screen as long as the player isn’t too close to the edge of the map.



These quick calculations set the 2-dimensional vector "self.offset" equal to how far away the player is from the center of the screen since the screens’ top left is (0,0) and "self.width" and "self.height” game screens’ width and height respectively. There is also a minus sign since objects should be moved in the opposite direction to the players movement

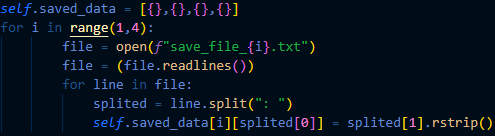
These first few lines of the camera constraint function make sure that the user can’t see too far left or right (out of the map). The first line sets the horizontal offset to 0 (meaning that objects with a x-coordinate of zero will be on the left edge of the screen) if the player is less than half the screen with away from the left of the map so that the user can’t see left of the map. The next if statement uses the same idea but makes sure that the user can’t see too far right. The offset value means things on the right edge of the map will be drawn on the right edge of the screen (”self.width") if the player is at the right end of the map.

 Similarly, these if statement make sure that the user can’t see too far down or up. ”self.level\_bottom” is the map height in pixels which is the same as the y-coordinate of the bottom of the map and ”top\_limit” is how high above the map the player is allowed to see (no minus sign is needed since being above 0 is equal to a negative y co-ordinates and ”self.top\_limit” is stored as a positive number and -\*- = +)



After the function explained above is called the program then loops through all the sprites and uses the “sorted” function and a lambda function as the key which allows the sprites to be sorted based on their variable called “order” which is defined when each sprite is created. The offset is the applied to each sprites co-ordinates before they are then drawn on the screen. This function draws sprites with higher orders after those with a lower order which means that higher order sprites will be on top.

File reading algorithm: This algorithm opens the text files and stores their data in a dictionary which is then stored in a list.





The variable “saved data” is where the data will be stored. The text files are named “save\_file\_x” where x is a number from 1-3 inclusive, therefore I used a for loop to iterate through the numbers 1-3. The list “saved data” also has an empty dictionary at index 0 which I've just used as a placeholder as I want to store the data at the same index as the number in their name.



The first line opens the text file in read mode (by default), the next line then calls the “readlines” function which returns a list where each line in a different element in the list.

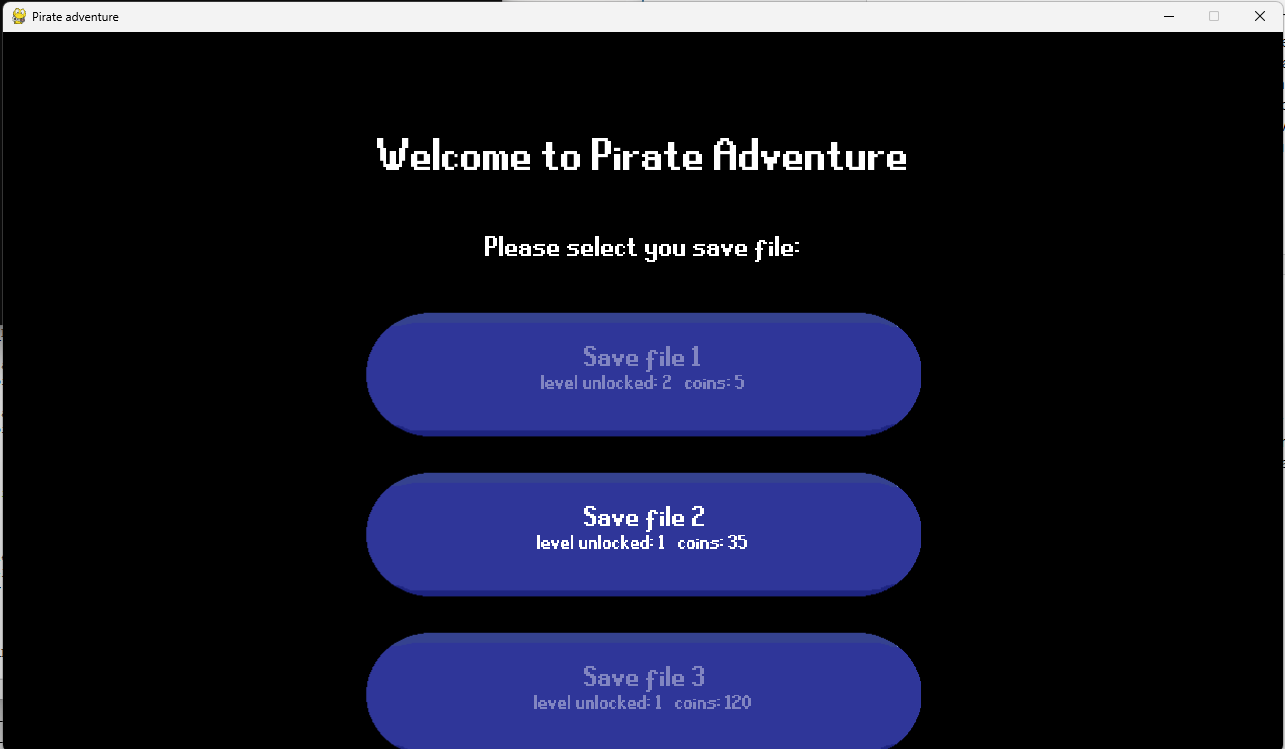


This loop iterates through each line in the file. The line is then split up into the variable and its corresponding value since each line of the text file is formatted like  where the character separating the variable and the value is ": "hence ". split(”:”)“. This split function returns a list where index 0 is the variable name and index 1 is the value. Then the dictionary in the "saved data" variable at the index “i“ (the same number as the file) is then edited to include the variable as a key and the value from the text file as its corresponding value. (The value is also removed of any extra unwanted spaces using ”.rstrip()”)

2.4 User interface (GUI):

As stated earlier, most of the images/animations for my project were taken from <https://www.youtube.com/@ClearCode> this creates the consistent high-quality look and feel which I was aiming for. Below are some examples of my GUI along with annotations:

**The level select screen:**

I choose the dark theme for this screen as it what is personally prefer the look of, it also complements the text that lights up when you hover over the button which creates an interactive screen and shows the user what they’re currently selecting. The font used

here is also the same font used throughout the entire project. The buttons also contain information on what they’ve done in each save so they know which is which and what they want to choose.

**The overworld:**

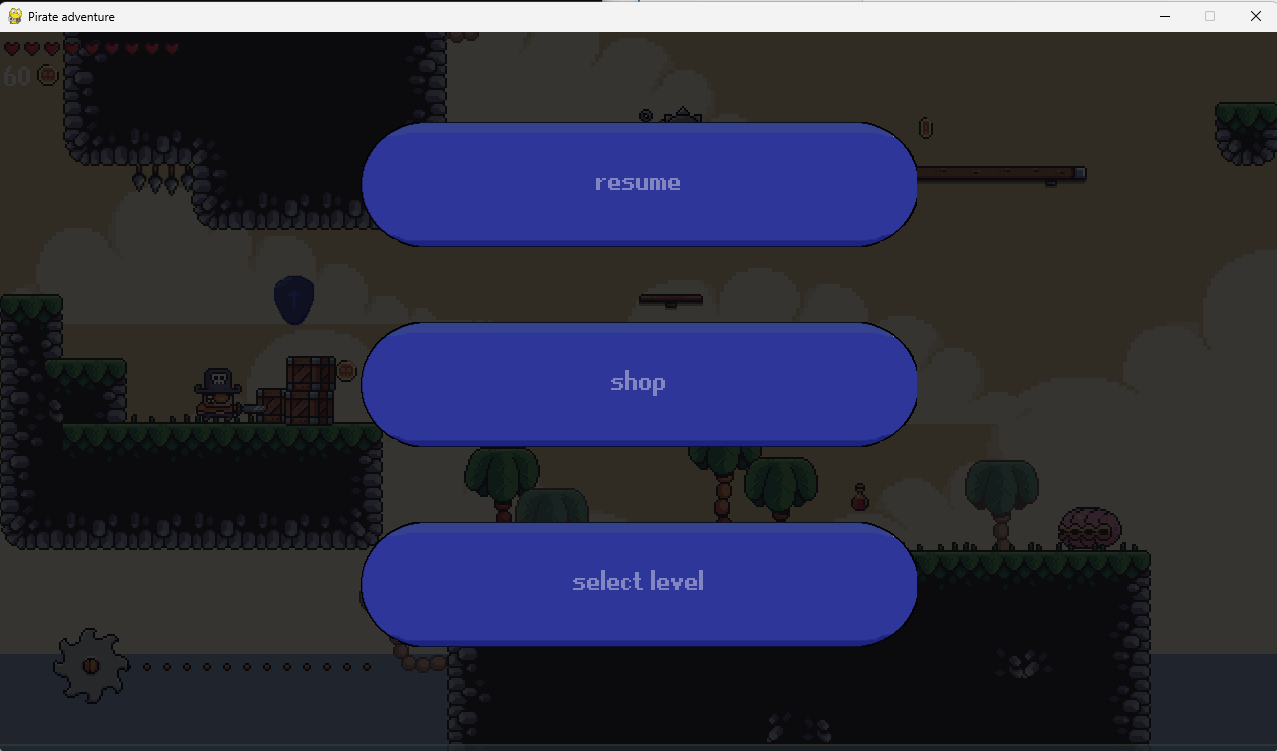
The player is in the middle of the screen like in the level so that the user knows where they are at all times. The paths also contrast the island, and the nodes are also clear which helps them stand out. This pixel style art is also consistent throughout the rest of the project for consistency. Furthermore, in the top left of the screen you can see the players hearts and coins which update in real time.

**Inside a level:**

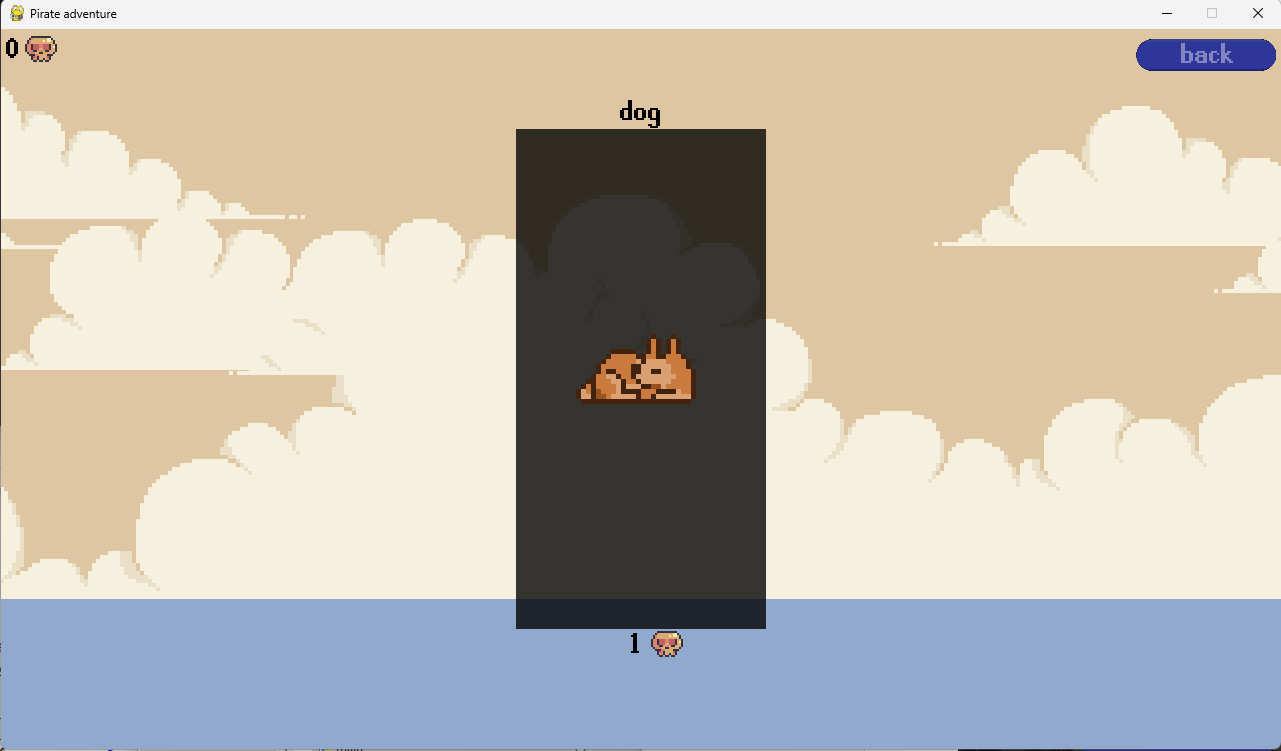


The players’ hearts and coins are still in the top left like in the level select screen, the hearts that you lose or gain also flicker to alert the player to what has happened, additionally the player will also flicker if they take damage. There are also abilities in the bottom right which appear when you pick the up in the level, the ability that you have selected is the image that isn’t shaded out (isn’t transparent). When the abilities are triggered, there is also a clear effect on the screen (like a blue bubble appearing around the player for the shield ability). The items are also clearly shown as floating and spinning objects such as the potion in the picture above, the end of the level is indicated by a flag with a skull and crossbones on it.

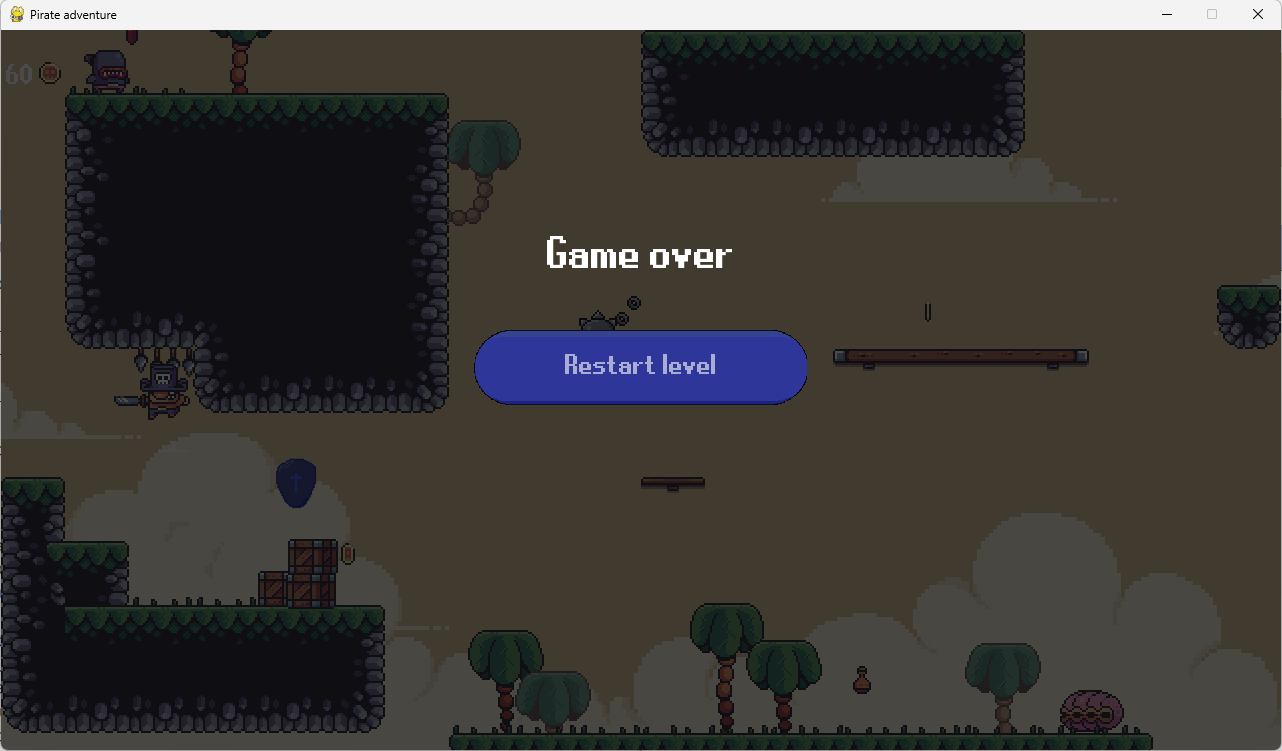
**Menu:**

When you enter the menu, the background is a faded image of the level before you entered the menu, The text lights up when you’ve selected it just like in the level select screen the text on the buttons also clearly conveys the effect of clicking on them. 

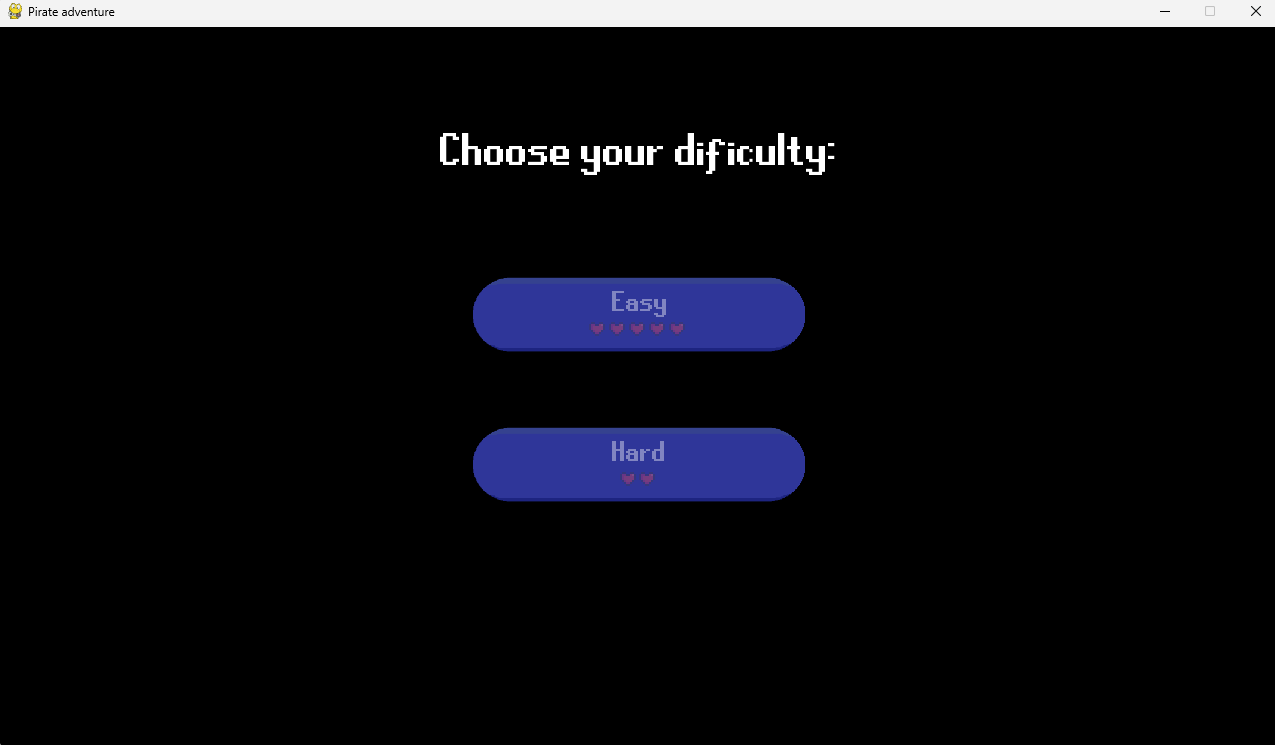
**Shop:**

There is a count of your skulls (the shop currency) in the top left and the price of each item is below the picture of it. The price is replaced with text saying ”equipped” once an item is purchased. The black box around the item also stops becoming transparent to indicate that you’re hovering over the item.

**Game over:**

Background is a faded image of the level when you died so that you can see how you died. Simplistic screens makes it easy for people to understand what happened and that all they can do is restart.

**Chose difficulty:**

The buttons are the same as in the earlier screens. The buttons this time also show the number of hearts you spawn with and helps the user to understand what they’re picking.

# Technical Solution

## 3.1 Contents page

|  |  |  |
| --- | --- | --- |
| Code file | Description | Page(s) |
| Settings.py | Contains crucial valuable variables and modules | [33](#_Settings.py) |
| Data.py | Contains Data class with various nested functions | [34](#_Data.py) |
| Groups.py | Contains All\_sprites and Overworld\_sprites classes  This is where algorithm 2 is implemented. | [42](#_Groups.py) |
| Level.py | Contains Level class and is where all the sprites are created | [46](#_Level.py) |
| Main.py | Contains Game class and is where all the images are imported | [56](#_main.py) |
| Menu.py | Contains Menu and Shop classes | [64](#_Menu.py) |
| Overworld.py | Contains Overworld class  Which contains algorithm 1 | [73](#_Overworld.py) |
| Player.py | Contains Player class and all the collision detection algorithms | [80](#_Player.py) |
| Select\_save.py | Contains Select\_save class and the file reading algorithm | [91](#_Select_save.py) |
| Sprites.py | defines all sprite type classes with their respective algorithms,  this is where all the inheritance is. | [93](#_Sprites.py) |
| Timer.py | Contains the Timer class and simple algorithms using pygame.time | [105](#_Timer.py) |
| Files.py | Contains all the functions for importing images | [107](#_Files.py) |

## 3.2 Code

#### Settings.py

import pygame, random

from pygame.math import Vector2 as vector

SCREEN\_WIDTH, SCREEN\_HEIGHT = 1280,720

TILE\_SIZE = 64

ANIMATION\_SPEED = 5

ORDERS = {

"sky": 0,

"bg": 1,

"main": 2,

"player":3,

"dog":4,

"fg": 5,

"water": 6

}

#### Data.py

from settings import\*

import sys

import random

from timer import Timer

from math import sin

class Data():

def \_\_init\_\_(self,health,frames):

'''creates variables and sprite groups'''

self.\_\_health = health

self.\_\_coins = 0

self.\_\_skulls = 0

self.current\_level = 0

self.level\_unlocked = 0

self.dog = False

self.frames = frames

self.font = frames["font"]

self.screen = pygame.display.get\_surface()

self.heart\_sprites = pygame.sprite.Group()

self.flicker\_timer = Timer(500)

self.coin\_count = Coin\_count((5,30),frames["coin"],self.font)

def setup(self,data,file):

'''loads the data from the save file'''

self.file\_num = file

if data != {}:

self.\_\_health = int(data['health'])

self.\_\_coins = int(data['coins'])

self.\_\_skulls = int(data['skulls'])

self.current\_level = int(data['current level'])

self.level\_unlocked = int(data['level unlocked'])

self.dificulty = int(data["dificulty"])

self.dog = True if data['dog'] == "True" else False

self.draw\_hearts()

else:

self.restart()

def chose\_dificulty(self):

'''asks the user to chose their difficulty easy or hard'''

self.screen.fill("black")

self.dificulty = None

text = self.frames["big\_font"].render("Choose your dificulty:",True,"white")

self.screen.blit(text,((SCREEN\_WIDTH-text.size[0])/2,100))

button = pygame.transform.scale\_by(self.frames["button"],0.6)

easy\_rect = button.get\_frect(topleft = (473.5,250))

hard\_rect = button.get\_frect(topleft = (473.5,400))

rects = [easy\_rect, hard\_rect]

texts = [self.frames["font"].render("Easy",True,"white"),self.frames["font"].render("Hard",True,"white")]

heart = self.frames["heart"][0]

difficulties = [5,2]

clicked = False

while self.dificulty == None:

mouse\_pos = pygame.mouse.get\_pos()

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit("Thank you for playing")

elif event.type == pygame.MOUSEBUTTONUP:

clicked = True

for i in range(2):

self.screen.blit(button,(473.5,250 + 150\*i))

if not rects[i].collidepoint(mouse\_pos):

texts[i].set\_alpha(100)

heart.set\_alpha(100)

else:

texts[i].set\_alpha(255)

heart.set\_alpha(255)

if clicked:

self.dificulty = difficulties[i]

self.screen.blit(texts[i],(613,260 + 150\*i))

for j in range(difficulties[i]):

width = heart.width \* difficulties[i] +5\*(difficulties[i]-1)

left = (SCREEN\_WIDTH-width)/2

self.screen.blit(heart,(left + width\*j/difficulties[i],295 + 150\*i))

pygame.display.update()

heart.set\_alpha(255)

def restart(self):

'''asks user to chose their dificultty and sets variables to 0'''

self.chose\_dificulty()

self.\_\_health = self.dificulty

self.\_\_coins = 0

self.\_\_skulls = 0

self.current\_level = 0

self.level\_unlocked = 0

self.dog = False

self.draw\_hearts()

def save(self):

'''re-writes the files with the current variables(info)'''

file = open(f"save\_file\_{self.file\_num}.txt", "w")

file.write(f'''current level: {self.current\_level}

level unlocked: {self.level\_unlocked}

coins: {self.\_\_coins}

skulls: {self.\_\_skulls}

health: {self.\_\_health}

dificulty: {self.dificulty}

dog: {self.dog}''')

def draw\_hearts(self):

'''creates the right amount of heart sprites'''

for sprite in self.heart\_sprites:

sprite.kill()

for i in range(self.\_\_health):

Heart((5 + 20\*i,10),self.frames["heart"],self.heart\_sprites)

def lose\_health(self):

'''decreases health and re-draws hearts'''

self.\_\_health -= 1

for sprite in self.heart\_sprites:

sprite.kill()

self.draw\_hearts()

self.flicker = self.\_\_health

self.flicker\_timer.activate()

def lose\_skulls(self,amount):

self.\_\_skulls -=amount

def gain\_health(self, amount = 1):

'''increase health by given amount and re-draw hearts'''

self.\_\_health += amount

self.draw\_hearts()

self.flicker = self.\_\_health-1

self.flicker\_timer.activate()

def get\_health(self):

return self.\_\_health

def get\_skulls(self):

return self.\_\_skulls

def gain\_coins(self,amount):

'''increases coins by amount and sees if they have enough coins to gain a heart'''

self.\_\_coins += amount

if self.\_\_coins >= 500//self.dificulty:#difficulty

self.\_\_coins -= 100

self.gain\_health()

def gain\_skull(self):

self.\_\_skulls +=1

def flicker\_heart(self):

'''creates a flicker effect on the heart you just gained/lost'''

if self.flicker\_timer.active and sin(pygame.time.get\_ticks() \* 50) >= 0:

white\_mask = pygame.mask.from\_surface(self.frames["heart"][0])

white\_surf = white\_mask.to\_surface()

white\_surf.set\_colorkey('black')

self.screen.blit(white\_surf,(5 + 20\*self.flicker,10))

self.flicker\_timer.update()

def draw(self,dt):

'''draws hearts and coins'''

self.heart\_sprites.update(dt, self.screen)

self.coin\_count.update(self.\_\_coins, self.screen)

self.flicker\_heart()

class Heart(pygame.sprite.Sprite):

def \_\_init\_\_(self,pos,frames,sprites):

'''creates necessary variables '''

super().\_\_init\_\_(sprites)

self.frames = frames

self.frame\_index = 0

self.image = frames[0]

self.rect = self.image.get\_frect(topleft = pos)

self.active = True

self.animation\_speed = 5

def animate(self,dt):

'''plays a heart animation for each on average once every 2000 frames '''

if self.active:

self.frame\_index += self.animation\_speed \* dt

if int(self.frame\_index) >= len(self.frames):

self.active = False

self.frame\_index = 0

self.image = self.frames[int(self.frame\_index)]

else:

self.active = True if random.randint(1,2000) == 1 else False # each heart animates on average once every 2000 frames

def update(self, dt, screen):

'''updates and draws itseld'''

self.animate(dt)

screen.blit(self.image, self.rect.topleft)

class Coin\_count(pygame.sprite.Sprite):

def \_\_init\_\_(self,pos,image,font):

'''sets the necessary variables'''

super().\_\_init\_\_()

self.pos = pos

self.image = image

self.rect = self.image.get\_frect()

self.font = font

def update(self, amount, screen):

'''draws the number of coins and a coin image'''

text = self.font.render(str(amount),True,"white")

screen.blit(text, self.pos)

width = self.font.size(str(amount))[0]

screen.blit(self.image, (self.pos[0]+width+5,self.pos[1]+2))

[back to contents](#_3.1_Contents_page)

#### Groups.py

from settings import \*

from sprites import Sprite

from math import ceil

from random import randint,uniform

class Overworld\_sprites(pygame.sprite.Group):

def \_\_init\_\_(self):

'''sets important variables'''

super().\_\_init\_\_()

self.width, self.height = SCREEN\_WIDTH, SCREEN\_HEIGHT

self.screen = pygame.display.get\_surface()

self.offset = vector()

def draw(self,player\_pos,dog):

'''draws all the overworld sprites centered arounf the players'''

self.offset.x = -(player\_pos[0] - self.width/2)

self.offset.y = -(player\_pos[1] - self.height/2)

for sprite in sorted(self,key = lambda sprite : sprite.order):

if sprite.order != ORDERS["main"] and not(sprite.order == ORDERS["dog"] and dog != True): # only draw the dog it they have it

pos = sprite.rect.topleft + self.offset

self.screen.blit(sprite.image, pos)

# in the main layer it draws images lower on the screen on top of things higher on the screen to give the illusion of being behind someting

for sprite in sorted(self,key = lambda sprite : sprite.rect.bottom):

if sprite.order == ORDERS["main"]:

pos = sprite.rect.topleft + self.offset

self.screen.blit(sprite.image, pos)

class All\_sprites(pygame.sprite.Group):

def \_\_init\_\_(self,level\_width, level\_bottom,level\_data,level\_frames):

'''creates important variables as well as all the sprite groups for the background'''

super().\_\_init\_\_()

self.screen = pygame.display.get\_surface()

self.offset = vector()

self.level\_width, self.level\_bottom = level\_width, level\_bottom

self.width, self.height = SCREEN\_WIDTH, SCREEN\_HEIGHT

self.level\_data = level\_data

self.top\_limit = self.level\_data["top\_limit"]

self.bg\_tiles = level\_frames["bg\_tiles"]

self.large\_cloud = level\_frames["large\_cloud"]

self.small\_clouds = level\_frames["small\_clouds"]

self.cloud\_sprites = pygame.sprite.Group()

self.small\_cloud\_sprites = pygame.sprite.Group()

self.num\_large\_clouds = ceil(self.level\_width/self.large\_cloud.get\_width())+1

self.num\_clouds = 10

if self.level\_data["bg"] != None:

self.draw\_bg\_tiles()

self.sky = False

else:

self.sky = True

def draw\_clouds(self):

'''draws clouds and moves them back to the right when they leave the screen'''

if len(self.cloud\_sprites.sprites()) == 0:

for x in range(0,self.num\_large\_clouds):#large clouds

Sprite((x\*self.large\_cloud.get\_width(),self.level\_data["horizon\_line"]-self.large\_cloud.get\_height()),self.large\_cloud,(self,self.cloud\_sprites),ORDERS["sky"])

for i in range(0,self.num\_clouds): # small clouds

index = randint(0,len(self.small\_clouds)-1)

x = i\*(self.level\_width/self.num\_clouds) + randint(-50,50)

y = random.randint(-self.top\_limit,self.level\_data["horizon\_line"] - self.small\_clouds[index].get\_height()-200)

Sprite((x,y),self.small\_clouds[index],(self.small\_cloud\_sprites,self),ORDERS["sky"])

else: #moves clouds left

for cloud in self.cloud\_sprites.sprites():

cloud.rect.x -=0.1

if cloud.rect.right<=0:

cloud.rect.left = cloud.rect.left + cloud.rect.width \* self.num\_large\_clouds

for small in self.small\_cloud\_sprites.sprites():

small.rect.x -= uniform(0.05,0.2)

if small.rect.right<=0:

small.kill()

index = randint(0,len(self.small\_clouds)-1)

x = self.level\_width + randint(0,50)

y = random.randint(-self.top\_limit,self.level\_data["horizon\_line"] - self.small\_clouds[index].get\_height()-200)

Sprite((x,y),self.small\_clouds[index],(self.small\_cloud\_sprites,self),ORDERS["sky"])

def draw\_bg\_tiles(self):

'''fills the background with the background tiles'''

for x in range(0,self.level\_width, TILE\_SIZE):

for y in range(-ceil(self.top\_limit/TILE\_SIZE)\*TILE\_SIZE,self.level\_bottom, TILE\_SIZE):

Sprite((x,y),self.bg\_tiles[self.level\_data["bg"]],self,ORDERS["sky"])

def draw\_sky(self):

'''draws the sky and sea'''

self.screen.fill('#ddc6a1')

sea\_rect = pygame.Rect((0,self.level\_data["horizon\_line"]+self.offset.y),(self.width,self.level\_bottom-self.level\_data["horizon\_line"]))

pygame.draw.rect(self.screen,"#92a9ce",sea\_rect)

def camera\_constraint(self,player\_pos):

'''makes sure the camera doesn't show outside the map'''

self.offset.x = 0 if player\_pos[0] <= self.width/2 else self.offset.x # too far left

if player\_pos[0] >= (self.level\_width-self.width/2): # too far right

self.offset.x = -(self.level\_width - self.width)

if player\_pos[1] >= (self.level\_bottom-self.height/2):#too far down

self.offset.y = -(self.level\_bottom-self.height)

if player\_pos[1] <= (self.height/2 - self.top\_limit):

self.offset.y = self.top\_limit

def draw(self,player\_pos,dog):

'''draws the background and all other sprites'''

self.offset.x = -(player\_pos[0] - self.width/2)

self.offset.y = -(player\_pos[1] - self.height/2)

self.camera\_constraint(player\_pos)

#print(self.offset.x)

if self.sky:

self.draw\_sky()

self.draw\_clouds()

for sprite in sorted(self,key = lambda sprite : sprite.order):

if sprite.order == ORDERS["water"]:

self.offset.x = round(self.offset.x, 0)

if sprite.order ==ORDERS["dog"] and dog != True:

pass

else:

pos = sprite.rect.topleft + self.offset

self.screen.blit(sprite.image, pos)

[back to contents](#_3.1_Contents_page)

#### Level.py

from settings import \*

from sprites import \*

import random

from timer import Timer

from player import Player

from groups import All\_sprites

class Level():

def \_\_init\_\_(self,tmx\_map, level\_frames,shop\_frames, Data,change\_stage):

'''creates all the sprite groups and variables'''

self.screen = pygame.display.get\_surface()

self.width = tmx\_map.width \* TILE\_SIZE

self.bottom = tmx\_map.height \* TILE\_SIZE

self.shop\_frames = shop\_frames

self.item\_frames = level\_frames["items"]

self.abilities = []

self.swap = Timer(250)

self.ability\_timer = Timer(200)

self.current\_ability = None

self.slow\_motion = Timer(15000)

self.Data = Data

level\_data = tmx\_map.get\_layer\_by\_name("Data")[0].properties

self.death\_border = self.bottom - level\_data["death\_border"]

self.change\_stage = change\_stage

self.all\_sprites = All\_sprites(self.width,self.bottom,level\_data,level\_frames)

self.collision\_sprites = pygame.sprite.Group()

self.semi\_collidable = pygame.sprite.Group()

self.enemy\_sprites = pygame.sprite.Group()

self.moving\_platforms = pygame.sprite.Group()

self.item\_sprites = pygame.sprite.Group()

self.setup(tmx\_map, level\_frames)

def setup(self,tmx\_map, level\_frames):

'''creates sprites for all the elements in the map file'''

#tiles ---------------------------------------------------------------

for x, y, surf in tmx\_map.get\_layer\_by\_name("Terrain").tiles():

Sprite((x\*TILE\_SIZE,y\*TILE\_SIZE),surf,(self.all\_sprites,self.collision\_sprites))

for x,y, surf in tmx\_map.get\_layer\_by\_name("Platforms").tiles():

Sprite((x\*TILE\_SIZE,y\*TILE\_SIZE),surf,(self.all\_sprites,self.semi\_collidable))

for x,y, surf in tmx\_map.get\_layer\_by\_name("Bg").tiles():

Sprite((x\*TILE\_SIZE,y\*TILE\_SIZE),surf,self.all\_sprites,ORDERS["bg"])

for x,y, surf in tmx\_map.get\_layer\_by\_name("Fg").tiles():

Sprite((x\*TILE\_SIZE,y\*TILE\_SIZE),surf,self.all\_sprites,ORDERS["fg"])

#water ------------------------------------------------------------

for obj in tmx\_map.get\_layer\_by\_name("Water"):

for i in range(0, int(obj.width / TILE\_SIZE)):

Animated\_sprite((obj.x + i\*TILE\_SIZE, obj.y),level\_frames["water\_surface"],self.all\_sprites,7,ORDERS["water"])

for j in range(1, int(obj.height / TILE\_SIZE)):

Sprite((obj.x + i\*TILE\_SIZE, obj.y + (j\*TILE\_SIZE)),level\_frames["water\_body"],self.all\_sprites,ORDERS["water"])

#objects ----------------------------------------------------------

for obj in tmx\_map.get\_layer\_by\_name("Bg details"):

if obj.name == "static":

Sprite((obj.x,obj.y),obj.image,self.all\_sprites,ORDERS["bg"])

else:

Animated\_sprite((obj.x,obj.y),level\_frames[obj.name],self.all\_sprites,8,ORDERS["main"])

if obj.name == "candle":

Animated\_sprite(((obj.x,obj.y)-vector(30,30)),level\_frames["candle\_light"],self.all\_sprites,5,ORDERS["main"])

for obj in tmx\_map.get\_layer\_by\_name("Objects"):

if obj.name == "flag":

self.end\_flag = Animated\_sprite((obj.x,obj.y),level\_frames["flag"],self.all\_sprites,5).rect

Sprite((obj.x,obj.y)+vector(0,125),self.shop\_frames["dog"],self.all\_sprites,ORDERS["dog"])

elif obj.name == "Player":

self.player = Player(

pos = (obj.x,obj.y),

frames = level\_frames["player"],

jump\_frames = level\_frames["jump"],

groups = self.all\_sprites,

sprites =(self.collision\_sprites,self.semi\_collidable,self.moving\_platforms,self.enemy\_sprites),

Data = self.Data)

elif obj.name in ("barrel","crate"):

Sprite((obj.x,obj.y),obj.image,(self.all\_sprites,self.collision\_sprites))

elif "palm" in obj.name:

if "bg" in obj.name:

Animated\_sprite((obj.x,obj.y),level\_frames["palms"][obj.name],self.all\_sprites,random.uniform(4,7),ORDERS["bg"])

else:

Animated\_sprite((obj.x,obj.y),level\_frames["palms"][obj.name],(self.all\_sprites,self.semi\_collidable),random.uniform(4,7),ORDERS["bg"])

else:

if obj.name == "floor\_spike":

frames = level\_frames["floor\_spike"]

y = obj.y

if obj.properties['inverted']:

frames = [pygame.transform.flip(frame,False,True) for frame in frames]

y -= 32

Animated\_sprite((obj.x,y+32),frames,(self.all\_sprites,self.enemy\_sprites),5)

else:

Animated\_sprite((obj.x,obj.y),level\_frames[obj.name],self.all\_sprites,5)

#moving objects -------------------------------------------------------------

for obj in tmx\_map.get\_layer\_by\_name("Moving objects"):

if obj.name == "moving\_platform":

Moving\_sprite((obj.x,obj.y),obj.width,obj.height,level\_frames["moving\_platform"],(self.all\_sprites,self.semi\_collidable,self.moving\_platforms))

elif obj.name == "saw":

if obj.properties["speed"] >= 0:

Moving\_sprite((obj.x,obj.y),obj.width,obj.height,level\_frames["saw"],(self.all\_sprites,self.enemy\_sprites),obj.properties["speed"],10)

if obj.width>obj.height:

for x in range(int(obj.x),int(obj.x + obj.width),20):

Sprite((x,obj.y+9),level\_frames["saw\_chain"],self.all\_sprites,ORDERS["bg"])

else:

Animated\_sprite((obj.x,obj.y),level\_frames["saw"],(self.all\_sprites,self.enemy\_sprites),20)

elif obj.name == "spiked\_ball":

Spiked\_ball((obj.x,obj.y),obj.image,level\_frames["spiked\_chain"],obj.properties["start\_angle"],obj.properties["end\_angle"],obj.properties["radius"],obj.properties["speed"],(self.all\_sprites,self.enemy\_sprites))

elif obj.name == "boat":

Moving\_sprite((obj.x,obj.y),obj.width,obj.height,level\_frames["boat"],(self.all\_sprites,self.collision\_sprites,self.moving\_platforms),obj.properties["speed"],0,ORDERS["main"],True)

#enemies ------------------------------------------------------------------

for obj in tmx\_map.get\_layer\_by\_name("Enemies"):

if obj.name == "Tooth":

Tooth((obj.x,obj.y),level\_frames["Tooth"],(self.all\_sprites,self.enemy\_sprites),self.collision\_sprites)

elif obj.name == "clam":

Clam((obj.x,obj.y),level\_frames["Clam"],level\_frames["Pearl"],(self.all\_sprites,self.collision\_sprites),self.enemy\_sprites,self.player.hitbox\_rect,obj.properties["inverted"],level\_frames["particle"])

#Items and abilities-----------------------------------------------------------------------

for obj in tmx\_map.get\_layer\_by\_name("Items"):

Item((obj.x,obj.y),level\_frames["items"][obj.name],(self.all\_sprites,self.item\_sprites),obj.name)

def check\_border(self):

'''makes sure player stays within the map'''

if self.player.hitbox\_rect.left <= 0:

self.player.hitbox\_rect.left = 0

elif self.player.hitbox\_rect.right >= self.width:

self.player.hitbox\_rect.right = self.width

def check\_items(self):

'''sees if the player picks up an item'''

for item in self.item\_sprites:

if self.player.hitbox\_rect.colliderect(item.rect):

match item.name:

case "diamond":

self.Data.gain\_coins(50)

case "gold":

self.Data.gain\_coins(20)

case"silver":

self.Data.gain\_coins(5)

case "skull":

self.Data.gain\_skull()

case "potion":

self.Data.gain\_health()

case "clock":

self.abilities.append("clock")

self.current\_ability = "clock"

case "shield":

self.abilities.append("shield")

self.current\_ability = "shield"

item.kill()

def draw\_abilities(self):

'''draws the abilites you have in the bottom right'''

self.swap.update()

self.ability\_timer.update()

for i in range(len(self.abilities)):

image = self.item\_frames[self.abilities[i]][0]

if not self.current\_ability == self.abilities[i]:

image.set\_alpha(100)

else:

image.set\_alpha(255)

self.screen.blit(image,(SCREEN\_WIDTH-64\*(i+1),SCREEN\_HEIGHT-64))

if len(self.abilities) != 0:

index = self.abilities.index(self.current\_ability)

keys = pygame.key.get\_pressed()

self.use\_ability = True if keys[pygame.K\_e] else False

if keys[pygame.K\_RIGHT] and not self.swap.active:

index = (index+1)%len(self.abilities)

self.current\_ability = self.abilities[index]

self.swap.activate()

elif keys[pygame.K\_LEFT] and not self.swap.active:

index = (index-1)%len(self.abilities)

self.current\_ability = self.abilities[index]

self.swap.activate()

if self.current\_ability == "clock" and self.use\_ability and not self.ability\_timer.active:

self.slow\_motion.activate()

self.ability\_timer.activate()

self.abilities.remove("clock")

self.current\_ability = None if len(self.abilities) == 0 else self.abilities[(index-1)%len(self.abilities)]

elif self.current\_ability == "shield" and self.use\_ability and not self.ability\_timer.active:

self.player.timers["shield"].activate()

self.ability\_timer.activate()

self.abilities.remove("shield")

self.current\_ability = None if len(self.abilities) == 0 else self.abilities[(index-1)%len(self.abilities)]

#use abilities

if self.slow\_motion.active:

fade = pygame.Surface((SCREEN\_WIDTH,SCREEN\_HEIGHT))

fade.fill((0,0,0))

fade.set\_alpha(100)#0 is fully see through 255 is not see-through

self.screen.blit(fade,(0,0))

if self.player.timers["shield"].active:

#creates a surface that can accept alpha values suggested by a user on a reddit forum in order to draw a transparent curcle

surface = pygame.Surface((SCREEN\_WIDTH,SCREEN\_HEIGHT), pygame.SRCALPHA)

pygame.draw.aacircle(surface,'#5751bd50',self.player.rect.center + self.all\_sprites.offset,50) # last 2 hex digits are the alpha value

self.screen.blit(surface, (0,0))

def check\_state(self):

'''see if play has reached the end or died'''

if self.player.hitbox\_rect.colliderect(self.end\_flag):

if self.Data.current\_level == self.Data.level\_unlocked:

self.Data.level\_unlocked +=1

self.change\_stage(True)

if self.player.hitbox\_rect.top >= self.death\_border or self.Data.get\_health() <=0:

self.Data.lose\_health()

self.change\_stage(True)

def run(self, dt):

'''runs all the methods needed'''

self.slow\_motion.update()

if self.slow\_motion.active:

dt = dt/3

self.all\_sprites.update(dt)

self.check\_border()

self.check\_items()

self.check\_state()

self.all\_sprites.draw(self.player.rect.center,self.Data.dog)

self.draw\_abilities()

[back to contents](#_3.1_Contents_page)

#### main.py

import sys

from os.path import join

from settings import \*

from Data import Data

from level import Level

from pytmx.util\_pygame import load\_pygame

from files import\*

from Menu import Menu

from timer import Timer

from overworld import Overworld

from select\_save import Select\_save

class Game():

def \_\_init\_\_(self):

'''load everything needed'''

pygame.init()

self.screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

pygame.display.set\_caption("Pirate adventure")

self.import\_assets()

self.level\_maps ={0:load\_pygame(join("data","levels","TEST.tmx")),

1:load\_pygame(join("data","levels","1.tmx")),

2:load\_pygame(join("data","levels","2.tmx")),

3:load\_pygame(join("data","levels","3.tmx")),

4:load\_pygame(join("data","levels","TEST.tmx")),

5:load\_pygame(join("data","levels","TEST.tmx"))

}

self.Data = Data(5,self.ui\_frames)

#self.current\_level = Level(load\_pygame(join("data","levels","TEST.tmx")),self.level\_frames,self.Data)

self.overworld\_map = load\_pygame(join("data","overworld","overworld.tmx"))

self.menu = Menu(self.ui\_frames,self.shop\_frames,self.change\_stage,self.Data)

self.clock = pygame.time.Clock()

self.select\_save = Select\_save(self.Data,self.ui\_frames)

self.paused = False

self.timers = {"pause": Timer(400)}

def change\_stage(self,overworld):

'''used to swap between levels and level select'''

if overworld:

self.current\_level = self.current\_level = Overworld(self.overworld\_map,self.overworld\_frames,self.shop\_frames,self.Data,self.change\_stage)

else:

self.current\_level = Level(self.level\_maps[self.Data.current\_level],self.level\_frames,self.shop\_frames,self.Data,self.change\_stage)

def import\_assets(self):

'''imports all the images'''

self.level\_frames = {

"palms" : import\_sub\_folders("graphics","level","palms"),

"flag" : import\_folder("graphics","level","flag"),

"boat" : import\_folder("graphics","objects","boat"),

"player" : import\_sub\_folders("graphics","player"),

"floor\_spike": import\_folder("graphics","enemies","floor\_spikes"),

"big\_chain": import\_folder("graphics","level","big\_chains"),

"candle": import\_folder("graphics","level","candle"),

"candle\_light": import\_folder("graphics","level","candle\_light"),

"large\_cloud": import\_image("graphics","level","clouds","large\_cloud"),

"small\_clouds": import\_folder("graphics","level","clouds","small"),

"moving\_platform": import\_folder("graphics","level","helicopter"),

"small\_chain": import\_folder("graphics","level","small\_chains"),

"water\_body": import\_image("graphics","level","water","body"),

"water\_surface": import\_folder("graphics","level","water","top"),

"window": import\_folder("graphics","level","window"),

"Tooth": import\_folder("graphics","enemies","tooth"),

"Clam": import\_sub\_folders("graphics","enemies","shell"),

"Pearl": import\_image("graphics","enemies","bullets","pearl"),

"particle": import\_folder("graphics","effects","particle"),

"jump":import\_folder("graphics","effects","jump"),

"items": import\_sub\_folders("graphics","items"),

"saw": import\_folder("graphics","enemies","saw","animation"),

"saw\_chain": import\_image("graphics","enemies","saw","saw\_chain"),

"spiked\_chain":import\_image("graphics","enemies","spike\_ball","spiked\_chain"),

"bg\_tiles": import\_folder\_dict("graphics","level","bg","tiles")

}

self.ui\_frames = {

"heart": import\_folder("graphics","ui","heart"),

"coin": import\_image("graphics","ui","coin"),

"skull":import\_image("graphics","ui","skull"),

"button": import\_image("graphics","ui","button"),

"font": pygame.font.Font(join("graphics","ui","runescape\_uf.ttf"),32),

"big\_font": pygame.font.Font(join("graphics","ui","runescape\_uf.ttf"),50)

}

self.shop\_frames = {

"dog": pygame.transform.scale(import\_image("graphics","shop","dog\_laying"), (100,100)),

"dog\_sitting": pygame.transform.scale(import\_image("graphics","shop","dog\_sitting"), (100,100)),

"large\_cloud":self.level\_frames["large\_cloud"],

"small\_clouds": self.level\_frames["small\_clouds"]

}

self.overworld\_frames = {

"icon": import\_sub\_folders("graphics","overworld","icon"),

"ojects": import\_folder\_dict("graphics","overworld","objects"),

"palm": import\_folder("graphics","overworld","palm"),

"path": import\_folder\_dict("graphics","overworld","path"),

"water": import\_folder("graphics","overworld","water"),

"dog" : import\_sub\_folders("graphics","overworld","dog")

}

def update\_timers(self):

'''updates all the timers'''

for timer in self.timers:

self.timers[timer].update()

def lost(self):

'''creates a game over screen'''

fade = pygame.Surface((SCREEN\_WIDTH,SCREEN\_HEIGHT))

fade.fill((0,0,0))

fade.set\_alpha(180)#0 is fully see through 255 is not see-through

self.screen.blit(fade,(0,0))

button = pygame.transform.scale\_by(self.ui\_frames["button"],0.6)

restart = self.ui\_frames["font"].render("Restart level",True,"white")

restart\_rect = button.get\_frect(topleft = (473.5,300))

self.text = self.ui\_frames["big\_font"].render("Game over",True,"white")

self.screen.blit(self.text,(546,200))

while True:

mouse\_pos = pygame.mouse.get\_pos()

if not restart\_rect.collidepoint(mouse\_pos):

restart.set\_alpha(150)

else:

restart.set\_alpha(255)

for event in pygame.event.get():

if event.type == pygame.QUIT:

self.Data.save()

pygame.quit()

sys.exit("Thank you for playing")

if event.type == pygame.MOUSEBUTTONUP:

if restart\_rect.collidepoint(mouse\_pos):

self.Data.restart()

self.run()

self.screen.blit(button,(473.5,300))

self.screen.blit(restart,(564,320))

pygame.display.update()

def run(self):

'''the main run loop, checks inputs and upadates display'''

self.overworld = Overworld(self.overworld\_map,self.overworld\_frames,self.shop\_frames,self.Data,self.change\_stage)

self.current\_level = self.overworld

running = True

while running:

dt = self.clock.tick() /1000

if self.paused:

dt = 0

self.paused = False

for event in pygame.event.get():

if event.type == pygame.QUIT:

self.Data.save()

pygame.quit()

sys.exit("Thank you for playing")

elif pygame.key.get\_pressed()[pygame.K\_ESCAPE] and not self.timers["pause"].active:

self.paused = True

self.menu.run(self.current\_level.player.rect.center,self.current\_level.all\_sprites)

self.timers["pause"].activate()

dt = 0 if dt>0.5 else dt

self.screen.fill("black")

self.update\_timers()

self.current\_level.run(dt)

self.Data.draw(dt)

pygame.display.update()

if self.Data.get\_health() <= 0:

running = False

# lose the game

self.lost()

if \_\_name\_\_ == "\_\_main\_\_":

game = Game()

game.select\_save.run()

game.run()

[back to contents](#_3.1_Contents_page)

#### Menu.py

from settings import \*

from timer import Timer

from sprites import Sprite

from random import randint, uniform

from Data import Data

import sys

class Menu():

def \_\_init\_\_(self,ui\_frames,shop\_frames,change\_stage,data):

'''creates the rects for each of the buttons (and other variables)'''

self.screen = pygame.display.get\_surface()

self.timer = Timer(400)

self.Data = data

self.state = "pause"

self.ui\_frames = ui\_frames

self.button = ui\_frames["button"]

self.font = ui\_frames["font"]

self.change\_stage = change\_stage

self.resume\_button = self.button.get\_frect(topleft = (362,90))

self.shop\_button = self.button.get\_frect(topleft = (362,290))

self.overworld\_button = self.button.get\_frect(topleft = (362,490))

self.shop\_frames = shop\_frames

def fade(self):

''' makes the screen darker; by making a slightly transparent black screen and covering the previous screen'''

fade = pygame.Surface((SCREEN\_WIDTH,SCREEN\_HEIGHT))

fade.fill((0,0,0))

fade.set\_alpha(200)#0 is fully see through 255 is not see-through

self.screen.blit(fade,(0,0))

def draw\_menu(self):

'''draws all the button and creates rects'''

text\_options = ["resume","shop","select level"]

selected = self.check\_mouse()

for i in range(3):

self.screen.blit(self.button,(362,90 +200\*i))

text = self.font.render(text\_options[i],True,"white")

if selected != i:

text.set\_alpha(100)

pos = ((600-self.font.size(text\_options[i])[0])/2 +340, (250 - self.font.size(text\_options[i])[1])/2 + 25+200\*i)

self.screen.blit(text,pos)

def check\_mouse(self):

'''checks if the mouse is over a button'''

mouse\_pos = pygame.mouse.get\_pos()

if self.resume\_button.collidepoint((mouse\_pos)):

return 0

elif self.shop\_button.collidepoint((mouse\_pos)):

return 1

if self.overworld\_button.collidepoint((mouse\_pos)):

return 2

def run(self,player\_center,all\_sprites):

'''fades the screen check for inputs and runs the draw funcs'''

self.timer.activate()

self.fade()

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

self.Data.save()

pygame.quit()

sys.exit("Thank you for playing")

elif pygame.key.get\_pressed()[pygame.K\_ESCAPE] and not self.timer.active:

return 0 # break

elif event.type == pygame.MOUSEBUTTONUP:

button = self.check\_mouse()

if button == 0:

return 0 # break

elif button == 2:

self.change\_stage(True)

return 0

elif button == 1: # creates and the runs and instance of the shop class

shop(self.screen,self.ui\_frames,self.shop\_frames,self.Data,self.button).run()

self.screen.fill("black")# now re-draws the menu and the background

all\_sprites.draw(player\_center,self.Data.dog)

self.fade()

self.timer.activate()

self.draw\_menu()

self.timer.update()

pygame.display.update()

class shop():

def \_\_init\_\_(self,screen,ui\_frames,images,data,button):

'''sets the items in the shop and other variables'''

self.items = [images["dog"]]

self.labels = ["dog"]

self.prices = [1]

self.screen = screen

self.Data = data

self.button = pygame.transform.scale(button,(140,32))

self.num\_large\_clouds = 6

self.num\_clouds = 6

self.large\_cloud = images["large\_cloud"]

self.small\_clouds = images["small\_clouds"]

self.cloud\_sprites = pygame.sprite.Group()

self.small\_cloud\_sprites = pygame.sprite.Group()

self.horizon\_line = SCREEN\_HEIGHT - 150

self.all\_sprites = pygame.sprite.Group()

self.font = ui\_frames["font"]

self.skull = ui\_frames["skull"]

if self.Data.dog == True:

self.prices[self.labels.index("dog")] =0

def draw\_items(self):

'''draws the items and creates their rects'''

self.rects = []

for i in range(len(self.items)):

space = SCREEN\_WIDTH/len(self.items)

mid = (space\*i + space\*(i+1))/2 # places the items in the middle depending on the num of items

item\_rect = pygame.FRect((mid - (250/2),100),(250,500))

self.rects.append(item\_rect)

item\_surface = pygame.Surface((250,500))

item\_surface.fill("black")

mouse\_pos = pygame.mouse.get\_pos()

if not item\_rect.collidepoint((mouse\_pos)):

item\_surface.set\_alpha(200)

item\_image = self.items[i]

item\_image = pygame.transform.scale\_by(item\_image,250/item\_image.width)

if self.labels[i] == "dog" and self.Data.dog == True:

price = "equipped"

else:

price = self.prices[i]

price\_width,price\_height = self.font.size(str(price))

price\_width +=10

price\_label = self.font.render(str(price),True,"black")

skull\_height,skull\_width = self.skull.height,self.skull.width

name = self.labels[i]

label = self.font.render(name,True,"black")

label\_width,label\_height = self.font.size(name)

self.screen.blit(price\_label,(mid - price\_width/2,600))

self.screen.blit(self.skull,(mid -price\_width/2 + price\_width,600))

self.screen.blit(label,(mid -label\_width/2,99-label\_height))# 100 - (250-width)/2 = (450-width)/2

self.screen.blit(item\_surface,item\_rect)

self.screen.blit(item\_image,(mid- (250/2),(700-item\_image.width)/2))

def draw\_clouds(self):

'''draws clouds and moves them back to the right when they leave the screen'''

if len(self.cloud\_sprites.sprites()) == 0:

for x in range(0,self.num\_large\_clouds):#large clouds

Sprite((x\*self.large\_cloud.get\_width(),self.horizon\_line-self.large\_cloud.get\_height()),self.large\_cloud,(self.cloud\_sprites,self.all\_sprites),ORDERS["sky"])

for i in range(0,self.num\_clouds): # small clouds

index = randint(0,len(self.small\_clouds)-1)

x = i\*(SCREEN\_WIDTH/self.num\_clouds) + randint(-50,50)

y = random.randint(0,self.horizon\_line - self.small\_clouds[index].get\_height()-200)

Sprite((x,y),self.small\_clouds[index],(self.small\_cloud\_sprites,self.all\_sprites),ORDERS["sky"])

else: #moves clouds left

for cloud in self.cloud\_sprites.sprites():

cloud.rect.x -=0.1

if cloud.rect.right<=0:

cloud.rect.left = cloud.rect.left + cloud.rect.width \* self.num\_large\_clouds

for small in self.small\_cloud\_sprites.sprites():

small.rect.x -= uniform(0.05,0.2)

if small.rect.right<=0:

small.kill()

index = randint(0,len(self.small\_clouds)-1)

x = SCREEN\_WIDTH + randint(0,50)

y = random.randint(0,self.horizon\_line - self.small\_clouds[index].get\_height()-200)

Sprite((x,y),self.small\_clouds[index],(self.small\_cloud\_sprites,self.all\_sprites),ORDERS["sky"])

def draw\_sky(self):

'''draws background'''

self.screen.fill('#ddc6a1')

sea\_rect = pygame.Rect((0,self.horizon\_line),(SCREEN\_WIDTH,self.horizon\_line))

pygame.draw.rect(self.screen,"#92a9ce",sea\_rect)

def draw\_skulls(self):

'''draws how many skulls you have'''

amount = str(self.Data.get\_skulls())

text = self.font.render(amount,True,"black")

self.screen.blit(text,(5,5))

self.screen.blit(self.skull,(10+self.font.size(amount)[0],5))

text = self.font.render("back",True,"white")

self.back\_rect = self.button.get\_frect(topright = (SCREEN\_WIDTH-5,10))

if not self.back\_rect.collidepoint(pygame.mouse.get\_pos()):

text.set\_alpha(100)

self.screen.blit(self.button, self.back\_rect)

self.screen.blit(text,(SCREEN\_WIDTH-100,10))

def click(self):

'''purchases the item you hovering over (only called when you've clicked)'''

mouse\_pos = pygame.mouse.get\_pos()

for i in range(len(self.rects)):

if self.rects[i].collidepoint((mouse\_pos)):

if self.Data.get\_skulls() >= self.prices[i]:

self.Data.lose\_skulls(self.prices[i])

self.prices[i] = 0

if self.labels[i] == "dog":

self.Data.dog = True

#buy item

if self.back\_rect.collidepoint((mouse\_pos)):

self.running = False

def draw\_everything(self):

'''runs all the draw methods'''

self.draw\_clouds()

self.draw\_sky()

self.all\_sprites.draw(self.screen)

self.draw\_skulls()

self.draw\_items()

def run(self):

'''updates display and checks inputs'''

self.running = True

while self.running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

self.Data.save()

pygame.quit()

sys.exit("Thank you for playing")

if event.type == pygame.MOUSEBUTTONUP:

self.click()

if pygame.key.get\_pressed()[pygame.K\_ESCAPE]:

self.running = False

self.all\_sprites.update()

self.draw\_everything()

pygame.display.update()

[back to contents](#_3.1_Contents_page)

#### Overworld.py

from settings import \*

from sprites import Sprite, Animated\_sprite, Icon, Path

from groups import Overworld\_sprites

import math

class Overworld():

def \_\_init\_\_(self,tmx\_map,overworld\_frames,shop\_frames,Data,change\_stage):

'''creates important variables and sets the speed'''

#self.tmx\_map = tmx\_map

self.overworld\_frames = overworld\_frames

self.shop\_frames = shop\_frames

self.Data = Data

self.change\_stage = change\_stage

self.nodes = pygame.sprite.Group()

self.node\_properties = []

self.all\_sprites = Overworld\_sprites()

self.path\_sprites = pygame.sprite.Group()

self.setup(tmx\_map)

self.moving = False

self.Turning = False

self.direction = ()

self.speed = 200

self.previous\_turn = None

self.state = "idle"

self.frame\_index = 0

def setup(self,tmx\_map):

'''creates all the sprites from the map file'''

# water ----------------------------------------------------------------------------------------

for x in range(tmx\_map.width):

for y in range(tmx\_map.height):

Animated\_sprite((x\*TILE\_SIZE,y\*TILE\_SIZE),self.overworld\_frames["water"],(self.all\_sprites),5,ORDERS["sky"])

# tiles-------------------------------------------------------------------------------------

for x,y,surf in tmx\_map.get\_layer\_by\_name("main").tiles():

Sprite((x\*TILE\_SIZE,y\*TILE\_SIZE),surf,(self.all\_sprites),ORDERS["bg"])

for x,y,surf in tmx\_map.get\_layer\_by\_name("top").tiles():

Sprite((x\*TILE\_SIZE,y\*TILE\_SIZE),surf,(self.all\_sprites),ORDERS["bg"])

# nodes and paths ------------------------------------------------------------------------------

for node in sorted(tmx\_map.get\_layer\_by\_name("Nodes"),key = lambda node : node.properties["stage"]):

if node.properties["stage"] == self.Data.current\_level:

self.player = Icon((node.x,node.y),self.overworld\_frames["icon"],self.all\_sprites)

pos = (node.x-75,node.y-28)

self.dog = Sprite(pos,self.shop\_frames["dog\_sitting"],self.all\_sprites,ORDERS["dog"]) # draw player and dog

if node.properties["stage"] <= self.Data.level\_unlocked:

self.node\_properties.append(node.properties)

pos = (node.x-TILE\_SIZE/2,node.y-TILE\_SIZE/2)

Sprite(pos,self.overworld\_frames["path"]["node"],(self.all\_sprites,self.nodes),ORDERS["bg"])

for path in tmx\_map.get\_layer\_by\_name("path"):

if path.properties["path\_id"] <= self.Data.level\_unlocked:

Path((path.x,path.y),self.overworld\_frames["path"][path.name],(self.all\_sprites,self.path\_sprites),path.name)

# objects -----------------------------------------------------------------------------------

for obj in tmx\_map.get\_layer\_by\_name("Objects"):

if obj.name == "palm":

Animated\_sprite((obj.x,obj.y),self.overworld\_frames["palm"],self.all\_sprites,random.uniform(4,5))

else:

Sprite((obj.x,obj.y),obj.image,self.all\_sprites,ORDERS["bg"])

def check\_input(self):

'''gets inputs and sees if the can move that way'''

keys = pygame.key.get\_pressed()

if keys[pygame.K\_w]:

direction = "up"

self.direction = vector(0,-1)

elif keys[pygame.K\_a]:

direction = "left"

self.direction = vector(-1,0)

elif keys[pygame.K\_d]:

direction = "right"

self.direction = vector(1,0)

elif keys[pygame.K\_s]:

direction = "down"

self.direction = vector(0,1)

elif keys[pygame.K\_SPACE]:

self.change\_stage(False)

try:

if int(self.node\_properties[self.Data.current\_level][direction]) <= self.Data.level\_unlocked:

self.moving = True

self.destination = int(self.node\_properties[self.Data.current\_level][direction])

else:

self.direction = vector()

except:

self.direction = vector()

def move(self,dt):

'''moves or stops the player and calles turn method'''

for node in self.nodes:# sees if you're at the correct node

if self.player.rect.colliderect(node.rect):

if (self.nodes.sprites()[self.destination] == node and

self.distance((node.rect.topleft + vector(TILE\_SIZE,TILE\_SIZE/2 +5)),(int(self.player.rect.centerx),int(self.player.rect.centery))) <=self.speed \* dt \*3):

self.moving = False

self.previous\_turn = None

self.Data.current\_level = self.destination

self.direction = vector()

self.player.rect.center = (node.rect.topleft + vector(TILE\_SIZE,TILE\_SIZE/2 +5))

for path in self.path\_sprites:

if self.player.rect.colliderect(path.rect) and path != self.previous\_turn:

if path.name == "tr":

self.turn(path,vector(0,1),vector(1,0),vector(0,-1),dt)

elif path.name == "tl":

self.turn(path,vector(1,0),vector(0,-1),vector(-1,0),dt)

elif path.name == "bl":

self.turn(path,vector(1,0),vector(0,1),vector(-1,0),dt)

elif path.name == "br":

self.turn(path,vector(0,-1),vector(1,0),vector(0,1),dt)

self.player.rect.topleft += self.direction \* self.speed \* dt

self.dog.rect.topleft += self.direction \* self.speed \* dt

def distance(self,p1,p2):

return math.sqrt(math.pow(p1[0] - p2[0],2) + math.pow(p1[1] - p2[1],2))

def turn(self,path,forward, new\_forward, new\_reverse,dt):

'''turns if the player is near the center of the turn and isn't turning'''

if self.distance(self.player.rect.center,(path.rect.topleft + vector(TILE\_SIZE/2,5))) <= self.speed \* dt:

if not self.turning:

self.player.rect.center = (path.rect.topleft + vector(TILE\_SIZE/2,5))

self.direction = new\_forward if self.direction == forward else new\_reverse

self.turning = True

self.previous\_turn = path

else:

self.turning = False

def animate(self,dt):

'''determines what the player is doing and animates accordingly'''

state = self.state

if self.direction == vector(1,0):

self.state = "right"

elif self.direction == vector(-1,0):

self.state = "left"

elif self.direction == vector(0,-1):

self.state = "up"

elif self.direction == vector(0,1):

self.state = "down"

else:

self.state = "idle"

if self.state != state or int(self.frame\_index) >= len(self.overworld\_frames["icon"][self.state]):

self.frame\_index = 0

self.player.image = self.overworld\_frames["icon"][self.state][int(self.frame\_index)]

self.dog.image = self.overworld\_frames["dog"][self.state][int(self.frame\_index)]

self.dog.image = pygame.transform.scale(self.dog.image, (100,100))

self.frame\_index += self.speed/20 \* dt

def run(self,dt):

'''checks inupt is the player isnt moving and calls all the draw methods'''

if not self.moving:

self.check\_input()

else:

self.move(dt)

#print(self.destination)

self.animate(dt)

self.all\_sprites.update(dt)

self.all\_sprites.draw(self.player.rect.center,self.Data.dog)

[back to contents](#_3.1_Contents_page)

#### Player.py

from settings import \*

from pygame.math import Vector2 as vector

from timer import Timer

from sprites import Particle\_effect

from math import sin

class Player(pygame.sprite.Sprite):

def \_\_init\_\_(self,pos, frames,jump\_frames, groups, sprites,Data):

'''sets the player rects and other important variables'''

super().\_\_init\_\_(groups)

#setup

self.image = frames["idle"][0]

self.rect = self.image.get\_frect(topleft = pos)

self.hitbox\_rect = self.rect.inflate(-76,-36)

self.all\_sprites = groups

self.order = ORDERS["player"]

self.collision\_sprites = sprites[0]

self.semi\_collision\_sprites = sprites[1]

self.moving\_platforms = sprites[2]

self.enemy\_sprites = sprites[3]

self.Data = Data

#movement

self.direction = vector()

self.gravity = 1500

self.speed = 400

self.jump = False

self.jump\_strength = 800

self.double\_jumped= False

self.jump\_frames = jump\_frames

self.use\_ability = False

#animation

self.state = "idle"

self.facing\_right = True

self.attacking = False

self.frames,self.frame\_index = frames,0

#collisions

self.touching = {"floor":False,"wall\_left":False,"wall\_right":False}

self.platform = None

self.timers = {

"jump": Timer(250),

"wall jump": Timer(200),

"platform fall":Timer(250),

"hit": Timer(1000),

"attack": Timer(500),

"shield": Timer(100000)

}

self.timers["jump"].activate() # so they don't jump as they enter

def input(self):

'''checks for any inputs'''

keys = pygame.key.get\_pressed()

input\_vector = vector()

if not self.timers["wall jump"].active:

if keys[pygame.K\_d]:

input\_vector.x +=1

self.facing\_right = True

if keys[pygame.K\_a]:

input\_vector.x -=1

self.facing\_right = False

if keys[pygame.K\_s]:

self.timers["platform fall"].activate()

self.direction.x = input\_vector.normalize().x if input\_vector else 0

if pygame.mouse.get\_pressed()[0]:

if not self.timers["attack"].active:

self.attacking = True

self.timers["attack"].activate()

if keys[pygame.K\_SPACE]:

self.jump = True if not self.timers["jump"].active else False

def move(self, dt):

'''applies gravity and moves the player in the right direction'''

self.hitbox\_rect.x += self.direction.x \* self.speed \* dt

self.collision("x")

if any((self.touching["wall\_left"], self.touching["wall\_right"])) and not self.touching["floor"] and not self.timers["jump"].active:

self.direction.y = 0 if not self.timers["wall jump"].active else self.direction.y

self.hitbox\_rect.y += self.gravity/10 \* dt

elif self.platform and self.direction.y >= 0 and not self.timers["platform fall"].active:

self.direction.y = 0

else:

self.direction.y += self.gravity/2 \*dt

self.hitbox\_rect.y += self.direction.y \*dt

self.direction.y += self.gravity/2 \* dt

if (self.touching["wall\_left"] or self.touching["wall\_right"] or self.touching["floor"]):

self.double\_jumped = False

if self.jump:

if self.touching["floor"]:

self.timers["jump"].activate()

self.direction.y = -self.jump\_strength

self.hitbox\_rect.bottom -=1

Particle\_effect(self.hitbox\_rect.bottomleft -vector(0,7), self.jump\_frames, self.all\_sprites,12)

elif not self.touching["floor"] and (self.touching["wall\_left"] or self.touching["wall\_right"]) and not (self.timers["wall jump"].active or self.timers["jump"].active):

self.timers["wall jump"].activate()

self.direction.x = 1 if self.touching["wall\_left"] else -1

self.direction.y = -self.jump\_strength

elif not self.double\_jumped and not self.timers["wall jump"].active and not self.timers["jump"].active:

self.direction.y = -self.jump\_strength

self.double\_jumped = True

Particle\_effect(self.hitbox\_rect.bottomleft -vector(0,7), self.jump\_frames, self.all\_sprites,12)

self.jump = False

self.collision("y")

self.semi\_collision()

self.rect.center = self.hitbox\_rect.center

def check\_contact(self):

'''sees if the player is touching anything and changes the required variables'''

floor\_rect = pygame.FRect((self.hitbox\_rect.bottomleft),(self.hitbox\_rect.width,2))

right\_rect = pygame.FRect((self.hitbox\_rect.topright + vector(0,self.hitbox\_rect.height/4)),(2,self.hitbox\_rect.height/2))

left\_rect = pygame.FRect((self.hitbox\_rect.topleft+vector(-2,self.hitbox\_rect.height/4)),(2,self.hitbox\_rect.height/2))

collide\_rects = [sprite.rect for sprite in self.collision\_sprites]

semi\_collide\_rects = [sprite.rect for sprite in self.semi\_collision\_sprites]

platforms = [sprite.rect for sprite in self.moving\_platforms]

self.touching["floor"] = True if floor\_rect.collidelist(collide\_rects+semi\_collide\_rects) !=-1 else False

self.touching["wall\_left"] = True if left\_rect.collidelist(collide\_rects) !=-1 else False

self.touching["wall\_right"] = True if right\_rect.collidelist(collide\_rects) !=-1 else False

index = floor\_rect.collidelist(platforms)

platform = self.platform

self.platform = self.moving\_platforms.sprites()[index] if index != -1 else None

#screen = pygame.display.get\_surface()

#pygame.draw.rect(screen,"yellow",right\_rect)

def collision(self, axis):

'''sees if the player is colliding with something and makes sure it doesnt go through it'''

for sprite in self.collision\_sprites:

if sprite.rect.colliderect(self.hitbox\_rect):

if axis == "x":

#left

if self.hitbox\_rect.left <= sprite.rect.right and int(self.old\_rect.left) >= int(sprite.rect.right):

self.hitbox\_rect.left = sprite.rect.right

#right

if self.hitbox\_rect.right >= sprite.rect.left and int(self.old\_rect.right) <= int(sprite.rect.left):

self.hitbox\_rect.right = sprite.rect.left

elif axis == "y":

#top

if self.hitbox\_rect.top <= sprite.rect.bottom and int(self.old\_rect.top) >= int(sprite.rect.bottom):

self.hitbox\_rect.top = sprite.rect.bottom

self.direction.y = 0

#bottom

if self.hitbox\_rect.bottom >= sprite.rect.top and int(self.old\_rect.bottom) <= int(sprite.rect.top):

self.hitbox\_rect.bottom = sprite.rect.top

self.direction.y = 0

def semi\_collision(self):

'''checks for collitions with semi-collidible and allows you to fall through them'''

if not self.timers["platform fall"].active:

for sprite in self.semi\_collision\_sprites:

if self.hitbox\_rect.colliderect(sprite.rect):

if self.hitbox\_rect.bottom >= sprite.rect.top and int(self.old\_rect.bottom) <= int(sprite.rect.top) and self.direction.y >=0:

self.hitbox\_rect.bottom = sprite.rect.top

self.direction.y = 0

def platform\_collision(self, dt):

'''moves you if your on a moving platform'''

if self.platform:

if self.platform.direction == "x":

self.hitbox\_rect.x += self.platform.speed \* dt

else:

self.hitbox\_rect.y += self.platform.speed \* dt

def check\_damage(self):

'''sees if you should take damage (or have blocked it)'''

if self.state == "attack":

if self.facing\_right:

sword\_hitbox = pygame.Rect((self.hitbox\_rect.topright),(25,60))

else:

sword\_hitbox = pygame.Rect((self.hitbox\_rect.topleft)-vector(25,0),(25,60))

elif self.state == "air\_attack":

sword\_hitbox = pygame.Rect((self.hitbox\_rect.bottomleft),(34,30))

else:

sword\_hitbox = pygame.Rect((0,0),(0,0))

for sprite in self.enemy\_sprites:

if sword\_hitbox.colliderect(sprite.rect) and hasattr(sprite, "destructible"):

sprite.die()

elif sword\_hitbox.colliderect(sprite.rect) and hasattr(sprite, "deflectable"):

sprite.deflect()

elif self.hitbox\_rect.colliderect(sprite.rect):

self.get\_damage()

if hasattr(sprite, "destructible"):

sprite.die()

def get\_damage(self):

'''decreases players health it they haven't been hit recently'''

if self.timers["shield"].active:

self.timers["shield"].deactivate()

self.timers["hit"].activate()

elif not self.timers["hit"].active:

self.Data.lose\_health()

self.timers["hit"].activate()

def flicker(self):

'''makes the player flash white to so that they've taken damage and are nw invulnerable'''

if self.timers['hit'].active and sin(self.timers['hit'].time) >= 0:

white\_mask = pygame.mask.from\_surface(self.image)

white\_surf = white\_mask.to\_surface()

white\_surf.set\_colorkey('black')

self.image = white\_surf

def get\_state(self):

'''finds out what state the player is in e.g falling'''

if self.touching["floor"] and self.direction.y>=0:

self.state = "idle" if self.direction.x == 0 else "run"

if self.attacking:

self.state = "attack"

else:

if any((self.touching["wall\_right"],self.touching["wall\_left"])):

self.state = "wall"

else:

self.state = "jump" if self.direction.y <0 else "fall"

if self.attacking:

self.state = "air\_attack"

def animate(self,dt):

'''draws the animations for the player'''

state = self.state

self.get\_state()

if state != self.state and not(self.state[-1] == "k" and state[-1] == "k"):

#dont set index = 0 if it's changing type of attack

self.frame\_index = 0

elif int(self.frame\_index)>= len(self.frames[self.state]):

self.frame\_index = 0

if self.attacking:

self.attacking = False

self.get\_state()

self.image = self.frames[self.state][int(self.frame\_index)]

if not self.facing\_right:

self.image = pygame.transform.flip(self.image,True,False)

self.frame\_index += dt \* ANIMATION\_SPEED

def update(self,dt):

'''calls all the methods and updates the timers'''

self.old\_rect = self.hitbox\_rect.copy()

self.input()

self.platform\_collision(dt)

self.move(dt)

self.check\_contact()

self.animate(dt)

self.flicker()

self.check\_damage()

for timer in self.timers.values():

timer.update\_dt(dt)

[back to contents](#_3.1_Contents_page)

#### Select\_save.py

from settings import \*

import sys

class Select\_save():

def \_\_init\_\_(self,data,ui\_frames):

'''loades in the stored data form each save file'''

self.Data = data

self.ui\_frames = ui\_frames

self.screen = pygame.display.get\_surface()

self.big\_font = ui\_frames["big\_font"]

self.font = ui\_frames["font"]

self.button = ui\_frames["button"]

self.saved\_data = [{},{},{},{}]

for i in range(1,4):

file = open(f"save\_file\_{i}.txt")

file = (file.readlines())

for line in file:

splited = line.split(": ")

self.saved\_data[i][splited[0]] = splited[1].rstrip()

def run(self):

'''blits all the buttons and text '''

self.screen.fill("Black")

self.text = self.big\_font.render("Welcome to Pirate Adventure",True,"white")

self.screen.blit(self.text,(374,100))

self.text = self.font.render("Please select you save file:",True,"white")

self.screen.blit(self.text,(482,200))

#print((SCREEN\_WIDTH-self.button.width)/2)

selected = None

while not selected:

clicked = False

mouse\_pos = pygame.mouse.get\_pos()

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit("Thank you for playing")

if event.type == pygame.MOUSEBUTTONUP:

clicked = True

for i in range(1,4):

self.screen.blit(self.button,(362,120 +160\*i))

rect = self.button.get\_frect(topleft = (362,120 +155\*i))

self.text = self.font.render(f"Save file {i}",True,"white")

if self.saved\_data[i] != {}:

self.info = self.font.render(f"level unlocked: {self.saved\_data[i]['level unlocked']} coins: {self.saved\_data[i]['coins']}",True,"white")

else:

self.info = self.font.render("Empty",True,"white")

self.info = pygame.transform.scale\_by(self.info, 0.7)

if not rect.collidepoint(mouse\_pos):

self.text.set\_alpha(100)

self.info.set\_alpha(100)

else:

if clicked:

self.Data.setup(self.saved\_data[i],i)

return i

self.screen.blit(self.text,(581,150 +160\*i))

self.screen.blit(self.info,((SCREEN\_WIDTH-self.info.width)/2,180 +160\*i))

pygame.display.update()

[back to contents](#_3.1_Contents_page)

#### Sprites.py

from settings import \*

from random import randint

from timer import Timer

import math

class Sprite(pygame.sprite.Sprite):

def \_\_init\_\_(self, pos, surf, groups, order = ORDERS["main"]):

'''joins the sprite group and sets a rect and image'''

super().\_\_init\_\_(groups)

self.image = surf

self.rect = self.image.get\_frect(topleft = pos)

self.order = order

class Path(Sprite):

def \_\_init\_\_(self, pos, surf, groups, name):

'''runs sprite.init and sets a name'''

super().\_\_init\_\_(pos, surf, groups, ORDERS["bg"])

self.name = name

class Animated\_sprite(Sprite):

def \_\_init\_\_(self, pos, frames, groups,animation\_speed, order = ORDERS["main"]):

'''sets animation related variables'''

super().\_\_init\_\_(pos,frames[0],groups,order)

self.frames = frames

self.frame\_index = 0

self.animation\_speed = animation\_speed

def animate(self,dt):

'''cycles through the frames'''

self.frame\_index += self.animation\_speed \* dt

if int(self.frame\_index) >= len(self.frames):

self.frame\_index = 0

self.image = self.frames[int(self.frame\_index)]

def update(self,dt):

'''calls animate method'''

self.animate(dt)

class Moving\_sprite(Animated\_sprite):

def \_\_init\_\_(self,pos,width,height,frames,groups,speed = 50,animation\_speed = 6, order = ORDERS["main"],flip = False):

'''calculates start pos and creates sets other important attributes'''

super().\_\_init\_\_(pos,frames,groups,animation\_speed, order)

self.flip = flip

self.start\_pos = pos

self.width = width

self.height = height

if self.width > self.height:

self.start\_pos += vector(0,self.height/2)

self.rect.center = self.start\_pos

else:

self.start\_pos += vector(self.width/2,0)

self.rect.center = self.start\_pos

self.speed = speed

self.direction = "x" if width>height else "y"

def flip\_frames(self,x=False,y=False):

'''flips the frames about the y axis'''

if self.flip:

self.frames = [pygame.transform.flip(frame,x,y) for frame in self.frames]

def update(self,dt):

'''moves the sprite and flips its direction if needed'''

#moving

if self.direction == "x":

self.rect.x += self.speed \* dt

#check to see if it should turn around

if self.rect.centerx < self.start\_pos[0]:

self.rect.center = self.start\_pos

self.speed \*=-1

self.flip\_frames(True)

elif self.rect.centerx > (self.start\_pos[0] + self.width):

self.rect.center = self.start\_pos + vector(self.width,0)

self.speed \*=-1

self.flip\_frames(True)

else:

self.rect.y += self.speed \* dt

if self.rect.centery < self.start\_pos[1]:

self.rect.center = self.start\_pos

self.speed \*= -1

self.flip\_frames(False,True)

elif self.rect.centery > (self.start\_pos[1] + self.height):

self.rect.center = self.start\_pos + vector(0,self.height)

self.speed \*= -1

self.flip\_frames(False,True)

#Animate

self.animate(dt)

class Spiked\_ball(Sprite):

def \_\_init\_\_(self,pos,ball\_image,chain,start\_angle,end\_angle,radius,speed,groups,order = ORDERS["main"]):

'''set start point and other attributes'''

super().\_\_init\_\_(pos,ball\_image,groups,order)

self.center = pos + vector(self.rect.width/2,self.rect.height/2)

self.angle = start\_angle

self.start\_angle = start\_angle

self.end\_angle = end\_angle

self.radius = radius

self.speed = speed

self.chain\_image = chain

self.all\_sprites = groups[0]

self.rect.centerx = self.center[0] + self.radius \* math.cos(math.radians(self.angle))

self.rect.centery = self.center[1] - self.radius \* math.sin(math.radians(self.angle))

self.chain = pygame.sprite.Group()

self.draw\_chain()

def draw\_chain(self):

'''draws chains every 20 pixel from the center'''

for chain in self.chain:

chain.kill()

for radius in range(10,self.radius-20,20):

x = self.center[0] + radius \* math.cos(math.radians(self.angle))

y = self.center[1] + radius \* math.sin(math.radians(self.angle))

Sprite((x,y),self.chain\_image,(self.all\_sprites,self.chain),order = ORDERS["bg"])

def update(self,dt):

'''uses trig to draw the ball traversing an arc'''

if self.angle >= self.end\_angle:

self.speed \*=-1

self.angle = self.end\_angle

elif self.angle <= self.start\_angle:

self.speed \*=-1

self.angle = self.start\_angle

self.angle += dt \* self.speed

self.rect.centerx = self.center[0] + self.radius \* math.cos(math.radians(self.angle))

self.rect.centery = self.center[1] + self.radius \* math.sin(math.radians(self.angle))

self.draw\_chain()

class Tooth(Animated\_sprite):

def \_\_init\_\_(self,pos, frames, groups,collision\_sprites, animation\_speed = random.uniform(4,6), order = ORDERS["main"]):

'''inherits from the animated sprite class and sets other attributes'''

super().\_\_init\_\_(pos,frames,groups,animation\_speed,order)

self.pos = pos

self.frames\_right = frames

self.frames\_left = [pygame.transform.flip(frame,True,False) for frame in frames]

self.speed = 60

self.direction = 1

self.collision\_sprites = collision\_sprites

self.deflectable = True

self.deflected = Timer(500)

def move(self,dt):

'''moves them and turns them around if needed'''

right\_rect = pygame.FRect((self.rect.topright),(5,self.rect.height-5))

left\_rect = pygame.FRect((self.rect.topleft-vector(5,0)),(5,self.rect.height-5))

bottom\_right = pygame.FRect((self.rect.bottomright),(5,5))

bottom\_left = pygame.FRect((self.rect.bottomleft-vector(5,0)),(5,5))

collisions = [sprite.rect for sprite in self.collision\_sprites]

if right\_rect.collidelist(collisions) != -1 or bottom\_right.collidelist(collisions) == -1:

self.direction = -1 # turn them around

if left\_rect.collidelist(collisions) != -1 or bottom\_left.collidelist(collisions) == -1:

self.direction = 1

self.rect.x += self.direction \* dt \* self.speed

def deflect(self):

if not self.deflected.active:

self.direction \*= -1

self.deflected.activate()

def update(self,dt):

self.deflected.update()

self.move(dt)

if self.direction ==1:

self.frames = self.frames\_right

else:

self.frames = self.frames\_left

self.animate(dt)

class Clam(Animated\_sprite):

def \_\_init\_\_(self,pos,frames,pearl\_image,groups,enemies,player\_rect,inverted,partice\_frames,order = ORDERS["main"]):

super().\_\_init\_\_(pos,frames["idle"],groups,6,order)

self.pos = pos

self.frames = frames

self.player = player\_rect

self.inverted = inverted

if self.inverted:

self.frames = self.flip\_frames()

self.shooting = False

self.wait\_timer = Timer(3000)

self.pearl\_image = pearl\_image

self.enemy\_sprites = enemies # enemy sprite group to the pearl can also be in the group

self.collision\_sprites = groups[1]

self.all\_sprites = groups[0]

self.particle\_frames = partice\_frames

def see\_player(self):

'''sees if the player is in the clam's view'''

if self.inverted:

if self.player.right >= self.pos[0] -500 and self.player.right < self.pos[0]:

if (self.player.center[1]-self.pos[1])\*\*2 <= (self.player.height/2)\*\*2:

return True

else:

if self.player.left <= self.pos[0] +576 and self.player.left > self.pos[0]+76:#76 is pixel width of the clam

if (self.player.center[1]-self.pos[1])\*\*2 <= (self.player.height/2)\*\*2:

return True

return False

def flip\_frames(self):

'''flips the frames along the vertical axis if the clam is facing the other way'''

frames = {}

frames["fire"] = [pygame.transform.flip(frame,True,False) for frame in self.frames["fire"]]

frames["idle"] = [pygame.transform.flip(frame,True,False) for frame in self.frames["idle"]]

return frames

def update(self,dt):

'''updates the Clam : shoots a pearl and animates when neccessary'''

if self.see\_player() and not self.shooting and not self.wait\_timer.active:

self.shooting = True

self.wait\_timer.activate()

direction = -1 if self.inverted ==True else 1

Pearl(self.pos,direction,self.pearl\_image,(self.enemy\_sprites,self.all\_sprites),self.collision\_sprites,self.player,self.rect,self.particle\_frames)#creates and shoots a pearl

if self.shooting:

self.frame\_index += dt \* self.animation\_speed

if int(self.frame\_index) >= len(self.frames["fire"]):

self.frame\_index = 0

self.shooting = False

else:

self.image = self.frames["fire"][int(self.frame\_index)]

else:

self.image = self.frames["idle"][0]

self.wait\_timer.update()

class Pearl(Sprite):

def \_\_init\_\_(self,pos,direction,image,groups,collision\_sprites,player,clam,particle\_frames):

'''creates a pearl at the mouth of the clam and sets other imortant attributes'''

self.clam\_rect = clam

self.direction = direction

self.all\_sprites = groups[1]

if self.direction == 1:

self.pos = pos + vector(20,22)

else:

self.pos = pos + vector(40,22)

super().\_\_init\_\_(self.pos,image,groups, order = ORDERS["bg"])

self.particle\_frames = particle\_frames

self.collision\_sprites = collision\_sprites

self.player = player

self.speed = 50

self.timer = Timer(350)

self.timer.activate()

self.destructible = True # so it dies when it hits something

def update(self,dt):

'''moves the pearl in the right diection and kills it it it hits something'''

if self.rect.collidelist([sprite.rect for sprite in self.collision\_sprites]) != -1:

if not self.rect.colliderect(self.clam\_rect):

return self.die() # returns it so that class update method ends (saves on unnecessary computation)

if not self.timer.active:

self.speed = 150

else:

self.timer.update()

self.rect.x += dt \* self.speed \* self.direction

def die(self):

'''kills the sprite and plays a particle effect'''

self.kill()

Particle\_effect(self.rect.topleft,self.particle\_frames,self.all\_sprites)

class Particle\_effect(Sprite):

def \_\_init\_\_(self, pos, frames, groups,animation\_speed = 7):

super().\_\_init\_\_(pos, frames[0], groups, order = ORDERS["main"])

self.frames = frames

self.frame\_index = 0

self.animation\_speed = animation\_speed

def update(self,dt):

'''animates the image and kills the sprite when the animation finishes'''

self.frame\_index += self.animation\_speed \* dt

if int(self.frame\_index)>= len(self.frames):

self.kill()# ends update method

return None

self.image = self.frames[int(self.frame\_index)]

class Item(Animated\_sprite):

def \_\_init\_\_(self,pos,frames,groups,name):

'''sets a name and random animation speed'''

animation\_speed = random.uniform(6,9)

super().\_\_init\_\_(pos,frames,groups,animation\_speed)

self.name = name

class Icon(pygame.sprite.Sprite):

def \_\_init\_\_(self,pos,frames,groups):

'''creates a basic class for the player that stores an image and rect'''

super().\_\_init\_\_(groups)

self.order = ORDERS["main"]

self.image = frames["idle"][0] # image is changed in the overworld file

pos = pos + vector(TILE\_SIZE/2,5)

self.rect = self.image.get\_frect(center = pos)

[back to contents](#_3.1_Contents_page)

#### Timer.py

from pygame.time import get\_ticks

class Timer():

def \_\_init\_\_(self, duration, func = None, repeat = None):

self.duration = duration

self.func = func

self.repeat = repeat

self.start\_time = 0

self.active = False

self.time =0

def activate(self):

self.active = True

self.start\_time = get\_ticks()

self.time = 0

def deactivate(self):

self.active = False

self.start\_time = 0

self.time = 0

if self.repeat:

self.activate()

def update(self):

current\_time = get\_ticks()

if current\_time - self.start\_time >= self.duration:

if self.func and self.start\_time != 0:

self.func()

self.deactivate()

def update\_dt(self, dt):

self.time += dt\*1000

if self.time>= self.duration:

if self.func and self.start\_time != 0:

self.func()

self.deactivate()

[back to contents](#_3.1_Contents_page)

#### Files.py

from settings import \*

from os import walk

from os.path import join

def import\_image(\*path, alpha = True, format = 'png'):

'''loads and image and converts its pixel format for fast blitting'''

full\_path = join(\*path) + f'.{format}'#use os.join so the it works of different operating systemes

# convert alpha allows for alpha pixels(transparency)

return pygame.image.load(full\_path).convert\_alpha() if alpha else pygame.image.load(full\_path).convert()

def import\_folder(\*path):

'''loads a folder of images and stores the in a list in the order their names'''

frames = []

for folder\_path, subfolders, image\_names in walk(join(\*path)):

for image\_name in sorted(image\_names, key = lambda name: int(name.split('.')[0])):

full\_path = join(folder\_path, image\_name)

frames.append(pygame.image.load(full\_path).convert\_alpha())

return frames

def import\_folder\_dict(\*path):

'''loads a folder of images as a dictionary with the key being the image name'''

frames = {}

for folder\_path, subfolders, image\_names in walk(join(\*path)):

for image\_name in image\_names:

index = image\_name.split(".")[0]

frames[index] = import\_image(folder\_path,index).convert\_alpha()

return frames

def import\_sub\_folders(\*path):

'''loads sub folders as lists within a dictionary with the keys bieng the sub folders name'''

folders = {}

for folder\_path, subfolders, image\_names in walk(join(\*path)):

if subfolders:

for sub\_folder in subfolders:

folders[sub\_folder] = import\_folder(\*path,sub\_folder)

return folders

[back to contents](#_3.1_Contents_page)

# Testing

This section is my post-development testing and so doesn’t include the bugs and fixes that were found and corrected earlier on during the development process. The test listed in the table below are demonstrated in “test\_video.mp4”. The tests in the table are not listed in order of when they occur but instead grouped with similar tests.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test objective: description | Expected outcome | Pass/ fail | evidence |
| 1 | 1: button graphics work properly | Does the text on the buttons light up when you hover over it | Pass | Video – 0:10 |
| 2 | 7: save files are clearly shown | Save file info shown correctly | Pass | Video – 0:25 |
| 3 | 7: can open save files | Can open empty save file | Pass | Video – 0:45 |
|  | 7: you can create a new save from an empty save file | after selecting the difficulty the right number of hearts are shown | Pass | Video – 1:00 |
| 4 | 7: save info can be loaded correctly | Can open a save file and load the right number of coins | Pass | Video –1:30 |
| 5 | 7: save info can be loaded correctly | Can open a save file and load the right number of hearts | Pass | Video-1:45 |
| 6 | 7: save info can be loaded correctly | Can open a save file and load the right number of skulls | Pass | Video- 6:45 |
| 7 | 7: save info can be loaded correctly | Can open a save file and load the right current level | Pass | Video –1:30 |
| 8 | 7: save info can be loaded correctly | Can open a save file and load the right max level | Pass | Video-1:30 |
| 9 | 7: save info can be loaded correctly | Can open a save file and load whether they have the dog | Pass | Video-1:45 |
| 10 | 2: The player can change levels | The player can move along path on the overworld | Pass | Video-2:00 |
| 11 | 2: can the player go somewhere they shouldn’t | Pressing random buttons doesn’t have any effect | Pass | Video-1:50 |
| 12 | 2: can the player go somewhere they shouldn’t | Player can’t leave the unlocked paths (w,a,s,and d are the buttons to go to the path above, left, down and right respectively) | Pass | Video- 2:00 |
| 13 | 2: pressing “space” enters the level | Player can enter a level | Pass | Video-2:15 |
| 14 | 6/1: game can be paused | Menu opens when you press escape | Pass | Video-6:15 |
| 15 | 6/1: game can be un-paused | You can exit the menu using escape and the resume button | Pass | Video-6:25 |
| 16 | 1/2: player can exit the level | The select level button takes you to the overworld | Pass | Video-7:00 |
| 17 | 2: player can enter multiple levels | Do the different nodes take you to their level (I only had 4 different levels 0,1,2,3 so level 4 and 5 were the same as level 0) | Pass | Video-7:15 |
| 18 | 6: shop can be opened | The shop button takes you to the shop | Pass | Video-6:35 |
| 19 | 6: shop info shown correctly | Shop shows the right number of skulls in the top left | Pass | Video-6:35 |
| 20 | 6: shop works correctly | Item can't be purchased when you don't have enough | Pass | Video-8:50 |
| 21 | 6: shop works correctly | Item can be purchased when you have click on it | Pass | Video-9:00 |
| 22 | 6: shop works correctly | The item is listed as equipped when you have purchased it | Pass | Video-6:50 |
| 23 | 6: shop works correctly | You lose the right number of skulls when you purchase something | Pass | Video-9:00 |
| 24 | 6: shop can be closed | You can exit the shop by clicking back or escape | Pass | Video-6:50 |
| 25 | 4: player movement works correctly | Player can move left and right | Pass | Video-2:20 |
| 26 | 4: player movement works correctly | The player can jump and double jump | Pass | Video-2:30 |
| 27 | 4: player movement works correctly | The player accelerates down when in the air | Pass | Video- 2:35 |
| 28 | 4: player can collide | The player can collide with the walls, ceilings and floors | Pass | Video-2:20 |
| 29 | 1/4: camera displays level properly | The camera doesn't show outside the level | Pass | Video-2:20 |
| 30 | 1/4: camera displays level properly | The camera follows the player | Pass | Video-3:00 |
| 31 | 5: traps deal damage to the player | The player loses hearts when they hit the spikes | Pass | Video-2:40 |
| 32 | 5/1: player can lose health and its shown clearly | The hearts and player image flickers when damage is taken | Pass | Video-2:40 |
| 33 | 3: coins can be collected | The coin count increases when you collect coins (it increases by different amounts depending on the item type e.g. ruby = 50) | Pass | Video-3:05 |
| 34 | 5: player can be damaged | You lose a heart when hitting the spiked ball | Pass | Video-4:20 |
| 35 | 4: player can wall slide | You wall slide when you touch the wall and jump of it | Pass | Video-3:15 |
| 36 | 3: items work correctly | You regain a heart when you touch the health potion | Pass | Video-4:50 |
| 37 | 3: you can heal using coins | You regain a heart when you collect a certain number of coins (250 for hard difficulty 100 for easy) | Pass | Video-5:30  Video-8:10 |
| 38 | 5/3: abilities can be picked up | When you pick up the shied it appears in the bottom left | Pass | Video-2:55 |
| 39 | 5/3: abilities can be picked up | When you pick up the clock it also appears in the bottom left | Pass | Video-3:20 |
| 40 | 5/3: You can choose what ability to activate | You can press the arrows to change ability (change which ability is highlighted) | Pass | Video-3:20 |
| 41 | 5/3: abilities can be used | When you press “e” with the shield you won't lose hearts when you first get hit | Pass | Video-4:00 |
| 42 | 5/3: abilities can be used | When you press “e” with the clock everything slows down | Pass | Video-3:35 |
| 43 | 5: enemies behave correctly | The tooth enemy turns around when it reaches the edge | Pass | Video-5:15 |
| 44 | 5: enemies can be interacted with | The tooth enemy can be deflected with an attack | Pass | Video-5:25 |
| 45 | 5: enemies deal damage | You lose a heart when the tooth enemy hits you | Pass | Video-5:25 |
| 46 | 5: enemies behave correctly | The clam shoots a pearl only when it sees you | Pass | Video-4:50 |
| 47 | 5: enemies deal damage | You lose a heart when a pearl hits you | Pass | Video-5:00 |
| 48 | 5: enemies can be interacted with | You can destroy a pearl with an attack | Pass | Video-4:50 |
| 49 | 4: you can collide properly with “platforms” | You can land on trees | Pass | Video-4:35 |
| 50 | 4: “platforms” can be interacted with | You can fall through trees by pressing “s” | Pass | Video-4:35 |
| 51 | 4: platforms can be interacted with | When you land on a moving platform you move with it | Pass | Video-5:55 |
| 52 | 4: platforms can be interacted with | You can fall through a moving platform by pressing “s” | Pass | Video-6:00 |
| 53 | 2: you can complete a level | When you collide with the end flag you unlock the next level | Pass | Video-8:10 |
| 54 | 7: data can be saved | When you exit the game, it saves your data to the right save file | Pass | Video-9:30 |

I also completed further testing where I played 20 minutes of gameplay and tested every way to die including falling off the map, as well making sure both difficulties work correctly. I also made sure that your game data is saved no matter what stage of the game you close it at.

# Evaluation

After reflecting on my project with my end users, I feel that overall my project completes the objectives quite well and fits the relaxing and enjoyable style game that we were hoping for. However, I feel that the shop section is lacking as there is only a dog that can currently be purchase and the aren’t any skins or other items for the player which were initially intended to be added.

## 5.1 Objectives

1. Updated graphics
   1. Pixel art for characters and level (from [https://github.com/clear-code-projects](https://github.com/clear-code-projects/Super-Pirate-World/tree/main/graphics))
   2. Moving background(moving clouds) that move for right to left then off the screen
   3. Animations for enemies, player and any other appropriate level assets (a few frames for each animation), a run, jump, fall, and idle animation

I feel that this objective has been meet as the graphics provide the aesthetic and comfy feel that I wanted. The animations also feel great, however when implementing them there was a problem with deciding what animation to use when the player starts attacking in the air and touches the ground during the animation, but I feel my solution was adequate. Additionally, to improve on this section I could also add a death animation to show more clearly how and when you die.

1. Multiple levels
   1. At least 5 levels
   2. Different layouts (same images can be used), each level has multiple monsters traps and items with a goal flag that you have to reach at the end
   3. Various length/ difficulties (time to complete = 3-5 minutes)
   4. Can choose which of your unlocked levels to play

I also feel that this objective has been meet as there are 5 levels to unlock by completing the previous and you can easily select which level to play. Even though there are only 4 level designs (2 of which are shown in the video) the implementation (code) is already there so all you would need to do is edit the tiled map file however you want and it would then be in the game

1. Items
   1. Coins which regenerate health when you collect enough (amount depends on the difficulty around 100 – 300) which are everywhere in each level
   2. Various types of items that give coins (silver/gold coins and chests) that give different amounts of coins with the more common ones giving less
   3. Skulls, which are the shop currency. Only 1-3 hidden in each level

This objective has also been met as each of the items have their own function and value and the skulls can also be used in the shop

1. Movement
   1. You can double jump (jump (once) again when you in the air)
   2. You can (slowly)slide down a wall and jump off it.
   3. Can land on some objects e.g moving platforms
   4. Can fall off the map (down holes/gaps) which kills you

I also think that this objective has been fully meet as I have implemented all the movement aspects that I originally wanted such as double jumping and wall sliding. The moving platform movement is also smooth and easy to understand

1. Fighting
   1. You can swing your sword every half a second which will block the shoots of monsters and deflect the monsters that walk back and forth
   2. There will also be traps to avoid (like spikes in the floor)
   3. You can also find abilities like a shield to help you complete the level

This objective is also meet as the player is able to destroy any pearls shot at themselves and they can also deflect some enemies using their sword with quite accurate collisions, the abilities also work well, and the images display their general effect.

1. Shop
   1. When the player presses escape the shop opens
   2. It displays what they can buy and haven’t brought already
   3. Displays your amount of skulls
   4. Allows you to purchase skins and other items like pets
   5. They can the be equipped when the user closes the shop

The shop is also displayed nicely with clear images and prices, and I also think that the addition of the menu was a good idea despite initially not being an objective. However, like I stated earlier there aren’t enough items in the shop and so with more time I would’ve liked to have been able to design and implement different skins for the player

1. Saves data
   1. After you run the program you select 1 of the 3 possible save files
   2. The save files are all individual text files
   3. When you close the game your progress gets saved. The progress being saved is the highest level unlocked, the current level you’re on, your skins/ pets, difficulty and your skulls(currency)

This objective has also been fully met as all data is stored in one of 3 text files. The data is also stored in an easy-to-read format. I also think that the start screen also nicely handles the problem of deciding which save file to load. The saving also happens without the user having to press a save button which means that the user can’t forget to save.

## 5.2 User feedback

Overall, the feedback from my end user was positive with the movement mechanics, such as the double jump and wall slide, being his favourite part. They also liked the fighting as it was rather intuitive and fun to use. However, There were also some parts that he thought could be improved which were: that it isn’t that clear that when you reach a certain number of coins that you gain a heart and it also isn’t clear which items do what, and that the levels could also be a bit harder to complete.

## 5.3 System improvements

Below is a list of improvements I would make if I were to redo the project or had more time:

To make every aspect of the game easier to understand for a first-time user I would add a tutorial which only runs when you open an empty save file which shows you all the buttons, items, enemies and the other fundamentals of the game.

To increase the difficulty, I could just change the map files to make them longer or maybe add more traps and less items, which wouldn't require any further coding. I could also add another enemy which acts as a boss which you have to fight and defeat in order to progress who could attack the player with spells or other attacks. Although the enemy couldn’t be too hard as the game would still need to be relaxing.

Additionally, I would probably remake all the collisions to use pygame masks instead of rects based of the image as using rects means that all my objects’ hitboxes are rectangular which may not reflect the object which caused the player to sometimes be hit by, or stand somewhere they shouldn’t. However, if I used mask I would be able to have pixel perfect collisions although they might be a bit CPU intensive (since pygame runs off the CPU).