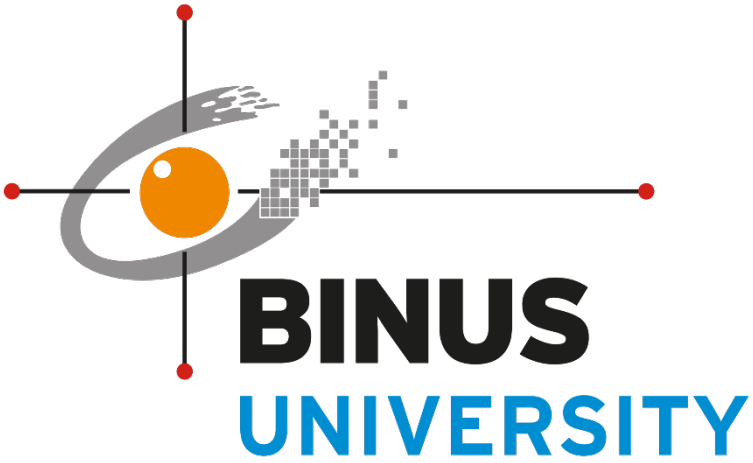
Odd Semester (2023)



BINUS UNIVERSITY

BINUS INTERNATIONAL

Algorithm and Programming

Final Project

Student Information:

Surname : Rahyang
Given Name : Fadhillah Haidar Rahyang
Student ID : 2702337211

Course Code : COMP6047001 **Course Name** : Algorithm and Programming

Class : L1AC **Lecturer** : Jude Joseph Lamug Martinez, MCS

Type of Assignment : Final Project Report

Submission Pattern:

Due Date : 12 January 2023

Submission Date : 12 January 2023

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.

2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.

3. The above information is complete and legible.

4. Compiled pages are firmly stapled.

5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.
6.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in sever penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been notified and accepted in advance

Signature of Student:

Fadhillah Haidar Rahyang

Table of Contents

Table of Contents

A. Introduction

1. Background

Students are expected to make a program to solve a relatively small, but interesting problem. The goal of this project is to apply all the knowledge of python programming language gained from the past lectures and attempt an execution by making a program that also include materials outside of class.

After thinking about it for some time and brainstorming ideas, I have come to the decision to make a tower defence game. The game would involve a tmj file for the enemies' path in which I have never heard of that file type before.

2. Problem Identification

According to the latest available data people spend on social media on average for about two hours and 24 minutes. That's approximately eight years of their life if they were to have that continuous habit for their lifetimes. Compared to the previous generations, the current young generations of ages of less than twelve are scrolling through Instagram reels, YouTube shorts and TikTok. The more they indulge the more addicted, especially when we are in an era where social media becomes more popular as the day goes by there are higher chances that the youngsters are exposed to more useless content.

The problem with this is that when they lay down and scroll, not much is happening in their brain. They do not have much to think about and the way that TikTok and its variants give content to its user decreases their attention span. The videos are short, but catchy as each new content gives an element of surprise. This does keep people entertained; however, it's doing its job a little too well. Thus, I have come up with the solution to provide an interactive tower defence game that will be enjoyed by the youngsters, not excluding teenagers and adults.

With an interactive game, the younger generations would actually be using their brains to process a method, a strategy to achieve a certain goal and have a

feeling of purpose in life. Not only does this prevent reduced attention spans, but it also ensures that people won't become dumb as they continuously think.

B. Project Specification

1. **Game Name**
   - EnTD
   - Pun intended from the word entity and in short for entity TD (tower defense)
2. **Game Concept**
   - "EnTD" is a tower defence game where you have to defend a portal that leads to Earth. Entities from an unknown region have constructed a path that leads to that portal and wishes to colonise planets and Earth is on their list.
   - There are multiple objects in this game which is divided into two subdivisions, entities and turrets. The entities will approach and pass through their constructed path in an attempt to reach the portal. The player then has to stop these entities from passing by buying turrets on the sidebar and placing them in space platforms.
   - To give the player more of a challenge, there will be multiple waves of entities that will pass through the same path and each wave will have more enemies, exponentially increasing the difficulty. They also have a limited amount of health.
3. **Game Flow Summary**
   - The player has to kill all of the upcoming entities with the provided turrets that is available for defending. When the player wants to buy a turret, they have to place them in designated areas for these turrets, namely the space platforms. As there is a limited number of platforms, the player can sell placed turrets to buy and place better ones. The game will increase difficulty exponentially hence, they would need to devise a strategy for turret placements. The player wins if all waves of entities have passed and are killed with the condition that their health is more than 0. The player loses if their health reaches 0.
4. **Game Objective**

- Defending the portal as long as possible until all of the entity waves are finished.

5. **Game Display**
   - It is a two-dimensional game that uses pixel art as the art style. The background created is a space map with stardust clouds as the path leading to a cyan-coloured oval, the portal. At the left side of the background is the GUI for buying turrets. Here, certain buttons will appear according to what the player wants to do, which includes buy, upgrade and sell turrets. The foreground consists of moving entities with their respective speeds and health with health bars on top and animated turrets. They will rotate according to the direction they are headed. The turrets will instantly rotate towards their selected target and produce a subtle spark when they shoot that entity.
6. **Game Mechanics**
   - When the player clicks on one of the turret buttons, their cursor will have an image of that turret indicating that currently they are buying and placing that selected turret.
7. **Game Physics**
   - The way that the entities are spawned is by accessing a dictionary from a python file. After accessing that dictionary, the entities are shuffled hence, their appearance into the map is randomised every time. The way the entities move through the stardust path is by following waypoints, which are pre-determined beforehand. Once they reach the destination, which is the portal, they disappear. For the buttons on the sidebar, it checks for a collide point with the mouse cursor. If the mouse position is on that collide point and is clicked, turret placing is enabled for that specific turret.
8. **Game Input**
   - Mouse left button/trackpad – to click the start wave, buy turret, upgrade turret, sell turret, fast forward and the X button on the top right of the window to close the game. It is also used to click on the space platforms to place the turrets.
9. **Game Output**

- Entity images will appear at the beginning of the path into the map in a random sequence and move through the pre – determined waypoints of the map
- Turret images will appear on the cursor after clicking on the buy turret button with the condition that the mouse position is in the map area, not the turret panel.
- The background image, which is the map (Space_Map.png)
- The player's health
- Each entity health
- The total money
- Background music (Background_Music.mp3)
- Basic turret shot sound effect (Turret_Basic_Shot_SFX.mp3)
- Sniper turret shot sound effect (Turret_Sniper_Shot_SFX.mp3)
- Machine gun shot sound effect (Turret_MachineGun_Shot_SFX.mp3)
- Win sound effect (win_SFX.mp3)
- Lose sound effect (lose_SFX.mp3)

10. Game Libraries/Modules

- Pygame – a collection of different modules in a single python package. Used to cast a window screen, respond to player input, draw images and producing sound.
- random – a built-in library and defines a series of functions for generating random integers. Used to randomise the entity spawn for every wave.
- Vector2 – a class provided from the module 'Pygame.math' for 2D vectors. Used to represent position of entities and the movement vectors.
- math – a built-in library that provides mathematical functions and constants. Used to calculate the angle in degrees that is used to rotate entity image which determines their direction.
- Json – a built-in library that provides methods to encode and decode JSON data in python. Used to access data of Space_Map.tmj file

11. Game Files

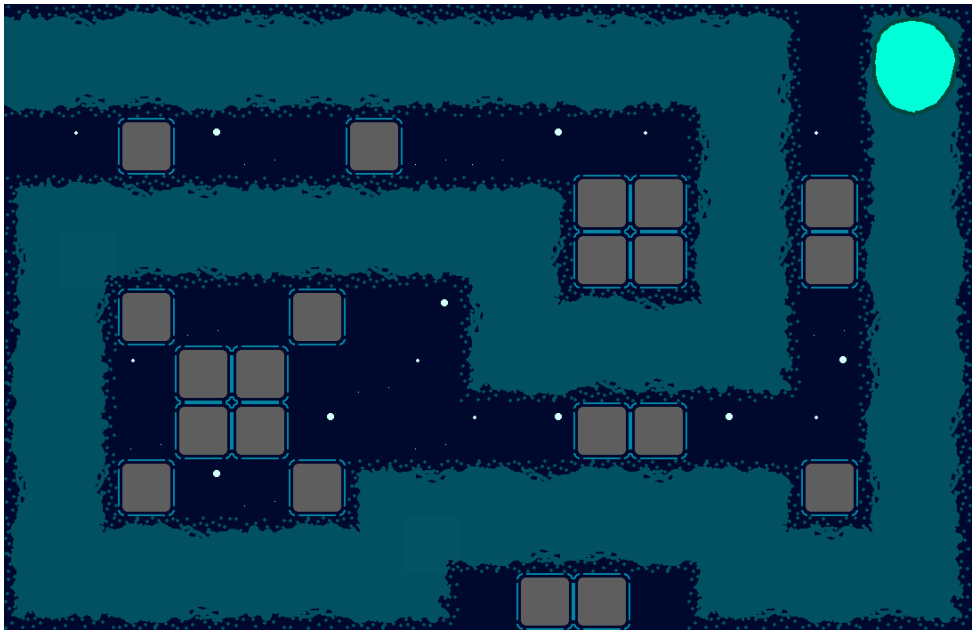
- 'Documents' folder – contains the report pdf file, game screenshots and the diagram images.
- 'audio' folder – contains all the audio used in the game

- ‘entityImages’ folder – contains all the entity images, the png files with the snake_case naming are the finalised ones.
- ‘GUIImages’ folder – contains all the images for the buttons and the coin and heart image.
- ‘map’ folder – contains the png file of the background game and the tmj file of the background game used to get the waypoints’ coordinates.
- ‘turretImages’ folder – contains all the turret images as well as their respective sprite sheets for animations
- button.py – contains the ‘Button’ class to be used to create every clickable button.
- entities.py – contains the ‘Entity’ class inheriting from the ‘P.sprite.Sprite’ class and the “HealthBar” class. They are responsible for the entities’ functionalities and their health bars respectively.
- main.py – the main file to be run; all necessary functions and class are called into this file.
- map.py – contains the ‘Map’ class which handles the waves of entities.
- turret.py – contains the ‘Turret’ class inheriting from the ‘P.sprite.Sprite’ class and a ‘loadTurretData’ function. They are responsible for
- settings.py – contains the ‘Settings’ class that has all necessary constants to be used in the python files above, except button.py

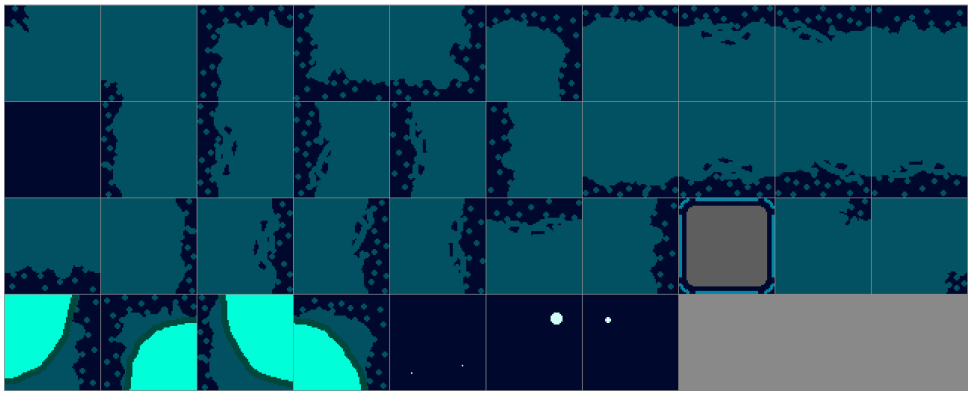
12. Game Images

All of the images were made by me.

- Background image

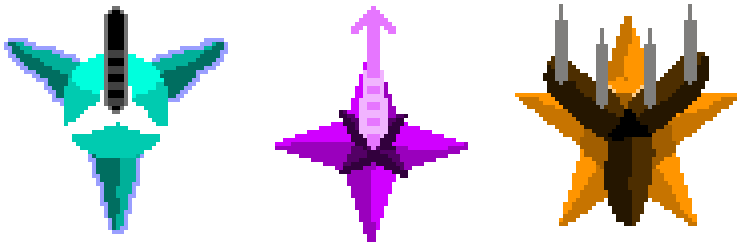


The tile set used to create the map



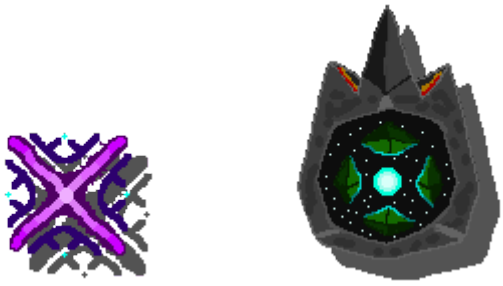
- Button images

- Turret images

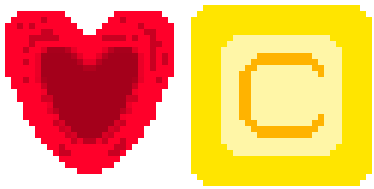


Their respective sprite sheets

- Entity images



- Other images (heart and coin)



13. Game Sounds

All the audios used were not made by me

- Background music
  https://youtu.be/gR4iFDXmpAw?si=4mkDtIFcMeWgNgeT

- Basic turret shot SFX
  https://youtu.be/PRjqL_6HC50?si=8eTDZczlNswG_Ion&t=18

- Sniper turret shot SFX
  https://youtu.be/jQ-RVSje1sw?si=OMBkRwkEDqR-T1zW&t=6

- Machine gun turret shot SFX
  https://youtu.be/PRjqL_6HC50?si=aNTmQ1xDpVXIYl_i&t=31

- Win SFX
  https://youtu.be/unSh4c_GtEY?si=iWR3ZVbH6pQnQcWw

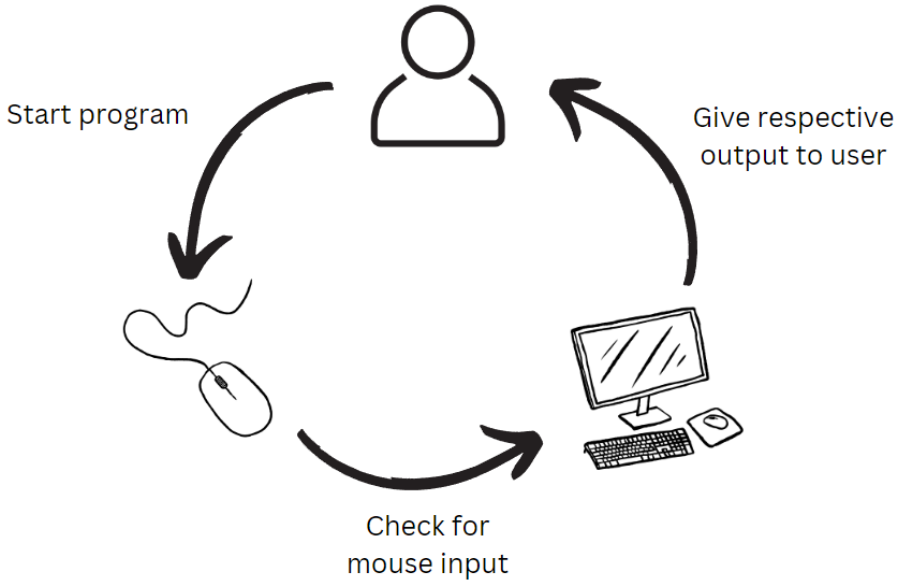
- Lose SFX
  https://youtu.be/SV2AsMDgRNI?si=mkJlK8mbXtZ9zMrG&t=5

14. Game Fonts

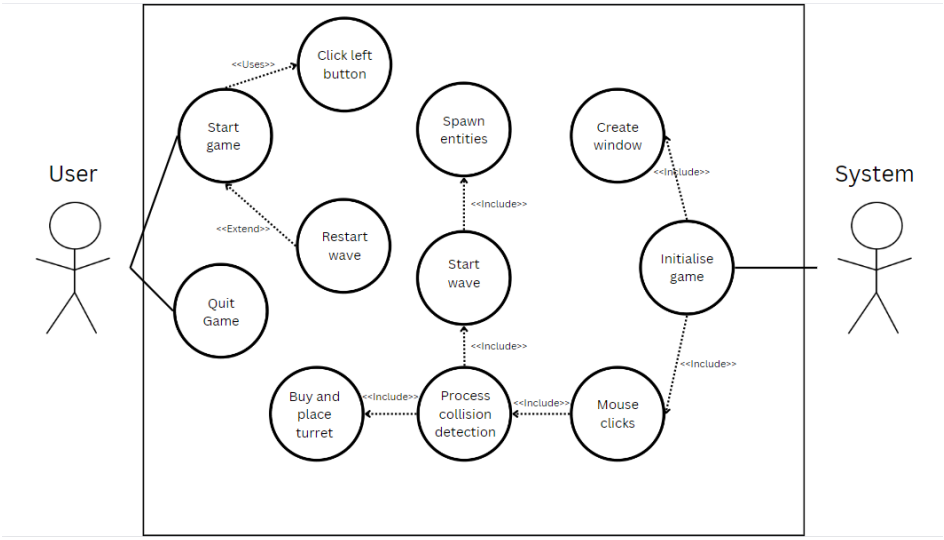
- Consolas font used for every text
  https://www.dafontfree.io/consolas-font/

C. Solution Design

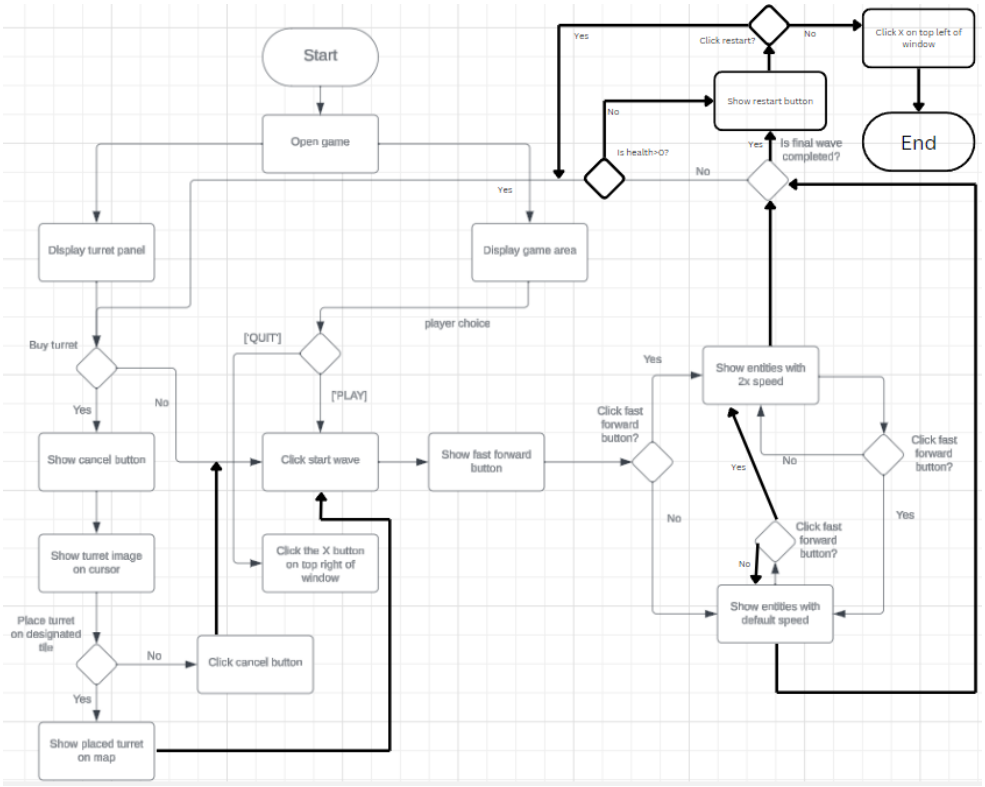
1. System Architecture



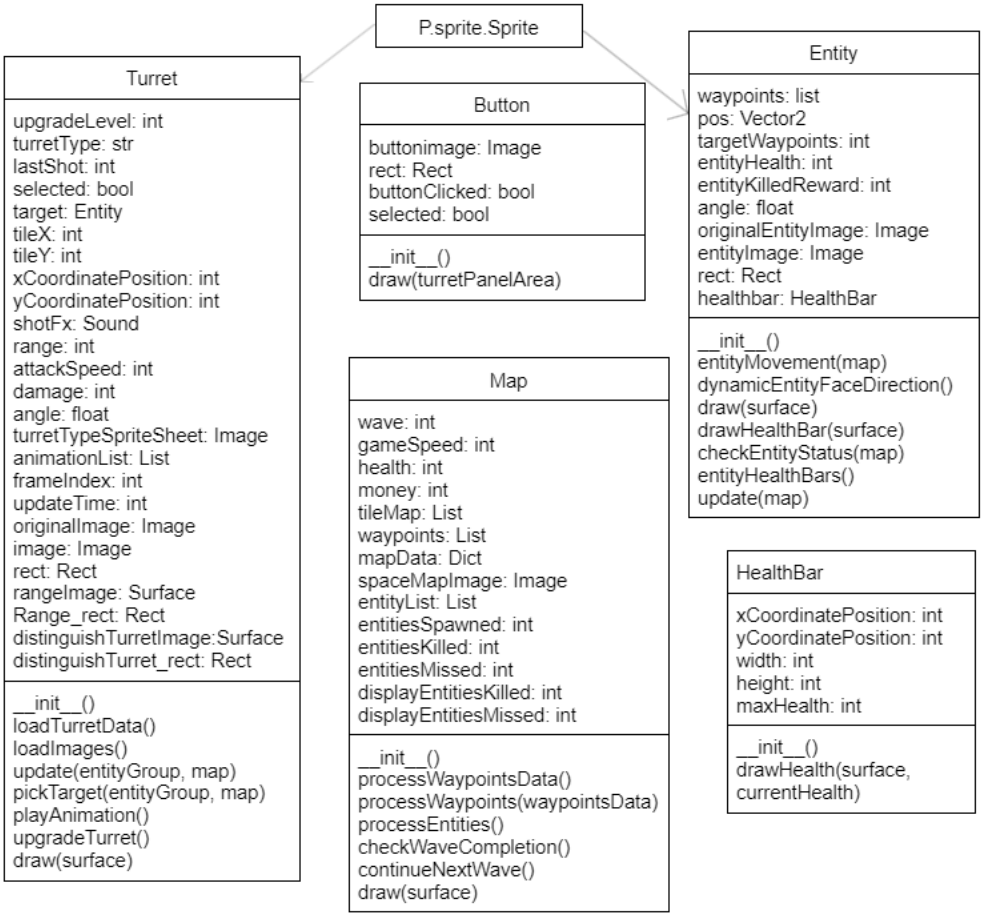
2. Use Case Diagram

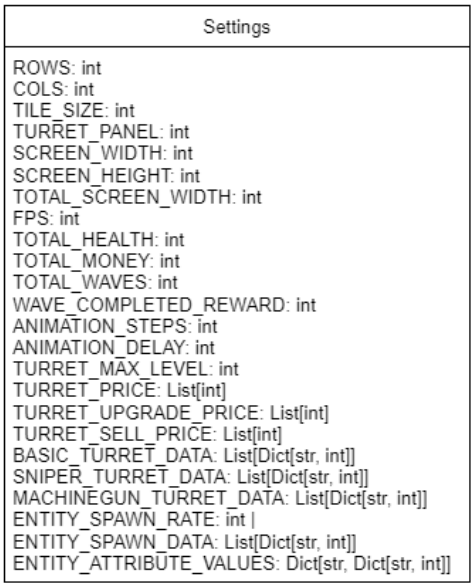


3. Activity Diagram



4. Class Diagram





D. Essential Algorithms

1. settings.py

This file contains the necessary constants that will be used in the game. It has one class, the Setttings class, and is subcategorised into four parts, namely screen settings, gameplay settings, turret settings, and lastly entity settings.

a) Screen Settings

In screen settings there are constants for the screen sizing and the screen frame rate. The screen sizing is responsible for defining a suitable screen size for the pygame window and makes the screen a tile-based map. ROWS is for the height of the map, COLS is for the width of the map, TILE_SIZE is to define a size for 1 tile, and TURRET_PANNEL provides a separate screen area to buy turrets. The frame rate will determine how fast the game moves and also limits the frame rate regardless of the computer. It is set to a value of 60.

b) Gameplay Settings

For the gameplay settings, there are constants used to track the player's health and money, the total waves available, and a money reward every time the player completes a wave. TOTAL_WAVES will also be used to determine whether the game is finished or not.

c) Turret Settings

Next is the turret settings and here lies all the constants for turrets, in this file. First are the turret animations and it will be utilising sprite sheets and each sprite sheet has 3 individual images combined which will be the ANIMATION_STEPS. When the animation is played it will cycle through the sprite sheet with a delay defined by ANIMATION_DELAY in milliseconds. Then is the turret's max level which is set to 4. After that are the values for turret modifier prices which include purchase, upgrade and sell. All these variables are a list of 3 integers which corresponds to basic, sniper and machine gun turret respectively. Lastly, the turret constants responsible for its attributes and its upgraded attributes. The attributes involved are the range, speed

and attack speed of the corresponding turret and are in a list of dictionaries.

d) Entity Settings

Then it's the entity settings which will include all constants for entities, in this file. ENTITY_SPAWN_RATE is used to determine how fast entities each wave will spawn. ENTITY_SPAWN_DATA is defined to set the entities that will spawn on each wave. Finally, the ENTITY_ATTRIBUTE_VALUES will be used to set each entity type their corresponding attributes. The attributes involved are, the entity's health, their speed and their money drop when they are killed.

2. map.py

This file is used to create the map instance and handle entity path as well as keeping track of the entities on it. It involves an imported module called 'random' which will be used to randomise entity spawn and variables from the Settings class. This file has 1 class, the Map class.

a) Constructor

First it initialises the necessary variables that will be used for the functions defined in the Map class. Variables involved are;

self.wave, to track the current wave; set to 1 as first wave,
self.gameSpeed, to determine game speed; set to 1 as default speed,
self.health, to keep track of player health; set to 100 as the start health,
self.money, to keep track of player money; set to 600 initially,
self.tileMap, empty list for integers to represent tiles of the map,
self.waypoints, empty list for 2-tuples for waypoint coordinates,
self.mapData, a parameter to pass the json data file of the map,
self.spaceMapImage, a parameter to pass the png file of the map,
self.entityList, empty list for entities to be added,
self.entitiesSpawned, to keep track of entities spawned,
self.entitiesKilled, to keep track of entities killed,
self.entitiesMissed, to keep track of entities missed,
self.displayEntityKilled, to display entities killed when game over,
self.displayEntityMissed, to display entities missed when game over.

b) Processing Waypoints Data

First it uses a for loop to go through every ‘property’ in the layers’ list of the tmj file of the map. Then using conditional statements if the name of the property is ‘map’ it will take the ‘data’ list and use that list for the self.tileMap list. This will be used to filter turret placement. The other statement checks if the property name is ‘waypoints’ and if it is, another for loop is used to go through every ‘object’ in the ‘objects’ list. Then the polyline ‘object’ is stored in waypointsData as a list of dictionaries to then be individually processed in the next function.

c) Processing Each Waypoint

In this function, utilising the for loop to get each waypoint in waypointsData (each dictionary in the list of dictionaries), get the values from the keys ‘x’ and ‘y’ and store them in respective variables. Finally, these two variables are added in the form of a 2-tuple into the self.waypoints list. This is all essentially to convert a list of dictionaries into a list of 2-tuples. The 2-tuples will represent waypoint coordinates.

d) Processing Entities

It starts by taking the first dictionary from ENTITY_SPAWN_DATA list of the Settings class and is stored in the variable ‘entities’. Then for each entityType key in that entities dictionary store the key value in ‘entitiesToSpawn’. This will determine how many times that specific entity will spawn on that wave. Then another for loop is executed to append that ‘entityType’ for ‘entitiesToSpawn’ number of times. For instances in wave 1, basic entity spawn for 15 times. Lastly, using the random module the list is shuffled to randomise the list order and apply randomised entity spawn.

e) Checking Wave Completion

This function uses the sum of self.entitiesKilled and self.entitiesMissed to be equal to the length of the entityList. If it was equal then return true else false.

f) Continuing Next Wave

When the next wave starts, certain variables need to be reset. These include self.entityList, which makes a new empty list to be used for the next wave of entities to spawn, self.entitiesSpawned, self.entitiesKilled, and self.entitiesMissed are all reset to 0 to keep track of the next wave

g) The Draw Method
Has a ‘surface’ parameter which will be used to pass the png file of the map and ‘blit’ it on the top-left corner of the screen essentially filling the game area.

3. button.py

This file contains a class that is responsible for creating all the buttons on the turret panel.

a) Constructor
A few variables are initialised to make a clickable button functional. These include;

self.buttonImage, a parameter applied to pass the button’s image,
self.rect, to get the rectangle area of that image,
self.rect.topleft, to draw the button from the top left of the area,
self.buttonClicked, to check if button is clicked; set to false,
self.selected, to check if a specific button is selected; set to false.

b) The Draw Method (including button functionality)
The buttonResponse Boolean value determines when the buttons will provide the specific action. First the position of the mouse is stored in a variable. Then a conditional statement combining with the collidepoint function from pygame checks if the mouse is hovering over the area of the button. Then it checks for a left click from that mouse and if it is clicked, all the Boolean values become true. The self.buttonClicked Boolean is to ensure that the button is clicked once even when the player holds the left click button. To prevent the button from being in a locked state of ‘already clicked’, a conditional statement to check if it

is not clicked is executed making the Boolean value false again. The place for the buttons to be drawn is the screen, specifically the turret panel area. Then returns true if the button was clicked.

4. entities.py

This file contains two classes, first is the Entity class inheriting the P.sprite.Sprite class, and the HealthBar class. The Entity class handles the functionalities of an entity and the HealthBar class handles the visibility of entity health. A Vector2 module is utilised from the pygame.math library to calculate entity movement. The math library is utilised for calculating angle to be used for entity rotation. Settings imported to access necessary entity constants.

1. Class Entity

a) Constructor

First the parent class, in this case the Sprite class of pygame is initialised to be able to use functions from that class. Then the variables are initialised which are;

self.waypoints, a parameter to accept the waypoints list,
self.pos, to get position of entity,
self.targetWaypoint, to keep track of subsequent waypoint,
self.entityHealth, to get entity health attribute,
self.entitySpeed, to get entity speed attribute,
self.entityKilledReward, to get entity money drop attribute,
self.angle, to determine entity face direction, set to 0 initially,
self.originalEntityImage, used to reset face direction,
self.entityImage, responsible for rotating the entity image,
self.rect, gets the rectangle area of the rotated image,
self.rect.center, centralises the rectangle area,
self.healthBar, creating an instance of the HealthBar class.

b) Entity Movement

First it checks whether the entity has reached the portal by utilising the changing value of self.targetWaypoint and the length of self.waypoints

list. If self.targetWaypoint is less, then define the next waypoint as the target waypoint. Calculate the distance for entity movement by subtracting the target waypoint coordinates and the entity position coordinates. If self.targetWaypoint is more however, entities have reached portal and self.kill() is a function from the Sprite class used to remove that entity from the game. The player's health is reduced based on the remaining health from an entity. The counter for self.entitiesMissed and self.displayEntitiesMissed increases by 1.

Then it calculates the entity path distance using the length function on self.movement to get a single numerical value instead of a tuple. If this distance was still more than the entity's speed, which is 1 by default, self.pos will be continuously added by self.movement.normalise() multiplied by corresponding entity speeds which is a unit vector that will increase the 2-tuple of the entity's position. An error handler is also added in case the remaining distance is less than the entity speed or when self.movement becomes zero and using normalise function will not work. For these case, simply add an if statement to check that the remaining distance is not 0 and then if it isn't, self.pos is added by self.movement.normalise() multiplied by the remaining distance instead of the entity speed. Lastly, when the remaining distance is zero, self.targetWaypoint is added by 1 to move onto the next waypoint.

c) Dynamic Entity Face Direction

First calculate the distance between entity and its next target waypoint. Then use that value to calculate the angle by inputting the y-axis and x-axis values of entityPathDistance in math.atan2(y, x). The y axis is negative because in python the y-axis starts from the top of the screen. The function outside it, math.degrees ensures that the default radian calculation is converted into degrees. The way atan2 function works is that for when the x value is zero, if the value of y is positive the result should be undefined, but atan2 still calculates angle based on the signs hence, the angle would be pi/2 or 90 degrees anticlockwise. Here is the reference;

I. When x is 0:
    i. If y is positive, angle is pi/2 or 90 degrees anticlockwise
    ii. If y is negative, angle is -pi/2 or 90 degrees clockwise
II. When y is 0:
    i. If x is positive, angle is 0
    ii. If x is negative, angle is -pi or 180 anticlockwise

Then using the rotate function from pygame, input the original image when the image has the default angle and self.angle as the angle to rotate the image. Get the rectangle area of that rotated image and centralise it from its position.

d) Draw Method for Entity
Accepts surface as parameter, which would be screen. Then draw the rotated entity image on the corresponding rectangle area.

e) Draw Method for Entity Health Bar
This function draws the created HealthBar instance on the screen with the entity health as its current health.

f) Checking Entity Status
This function checks whether the entity is still alive. If it wasn't the counter for map.entitiesKilled and map.displayEntitiesKilled is added by 1 and the player's money is added depending on the enemy type killed. Lastly, the entity is removed from the map.

g) Setting Health Bar Location
The created HealthBar instance is now repositioned and this function will be placed in the update function to continuously follow the corresponding entities.

h) Update Function
Updates entity movement, entity face direction, entity status, and entity health bars. To be placed in the main file, main.py.

2. Class HealthBar

a) Constructor

Has five parameters and all the attributes are those parameters. The attributes in question;
self.xCoordinatePosition, to define an x coordinate for health bar,
self.yCoordinatePosition, to define a y coordinate for health bar,
self.width, to give the width of the health bar,
self.height, to give the height of the health bar,
self.maxHealth, used for ratio purposes.

b) Draw Health Ratio

Here it accepts two parameters, one is the surface and the other is the current health. This current health will be the health of the entity when it starts taking damage. To do this, the ratio between the current health and the maximum health is utilised. First the red rectangle is drawn on the screen. Then a green rectangle of the same area is drawn over the red rectangle. However, for the green rectangle the width is multiplied with the ratio in which the ratio will always be equal or less than 1. The ratio goes down when currentHealth decreases. Essentially a green rectangle with varying widths, exposing the red rectangle overtime.

5. turret.py

This file has one class. The Turret class inherits the Sprite class of pygame. The math library is also used here for angle calculation

a) Constructor

It has 5 parameters, turretTypeSpriteSheet, tileX, tileY, shotFx, turretType. The same with the Entity class, the Sprite class needs to be initialised first in order to use those functions. Then the attributes which are as follows;

self.upgradeLevel, keep track of turret level; set to 1 initially,
self.turretType, parameter to filter turret selection,
loadTurretData, function called to load turret specificity,
self.lastShot, used to get last turret shot as current time,

self.selected, determines if a turret is selected,
self.target, used for turrets to select entity as target; set to none at start,
self.tileX, turret on tile x coordinate position,
self.tileY, turret on tile y coordinate position,
self.xCoordinatePosition, calculates center x coordinates of a tile,
self.yCoordinatePosition, calculates center y coordinates of a tile,
self.shotFx, parameter for turret shooting sound,
self.turretTypeSpriteSheet, parameter to load respective sprite sheets,
self.animationList, loads the images for turret animation,
self.frameIndex, determines frame number of each sprite sheet,
self.angle, for turret direction; set to 90 initially hence, upwards,
self.originalImage, takes the first image from the sprite sheet
self.image, responsible for rotating the turret image,
self.rect, gets the rectangle area of the rotated image,
self.rect.center, centralises the rectangle area on the tile,
self.rangeImage, first a square surface is created, then the area is filled with a colour, then that colour is set to be used for modifying the opacity, then that range image is drawn as a circle with light grey as the colour, then set alpha is used to determine the opacity,
self.range_rect, to get the rectangle area of the range image,
self.range_rect.center, centralise that range image,
self.distinguishTurret, a smaller circle to distinguish selected turret,
The rest is a repeat of the previous circle, but with the colour red.

b) Loading Turret Data

Here is where self.Turretype comes to play. It accepts string as its parameter and three specific strings will be used. When turretType is equal to string “Basic”, it will store BASIC_TURRET_DATA from the Settings class in turretData. It will also store the upgrade price and sell price from the corresponding lists with an index of 0. The same applies to the string “Sniper” and “MachineGun”. Then when the turretData is loaded, the list of dictionaries for each turret upgrades will be stored in respective attributes using the get function, namely self.range, self.attackSpeed, and self.damage.

c) Loading Images

First the size is defined by getting the height of the sprite sheet. An empty list of animation is created for each frame to be appended to this list later on. A for loop is created and each frame is created using the subsurface method with x coordinate, y coordinate, width, and height as its parameters. The loop continues until the end of each spritesheet. Lastly, the animation list is returned to be used for animation.

d) Update Turret Activity

When the entity target is in range of the turret range, play the turret animation. If it is out of range and before the turret shoots the next target, another conditional statement ensures that the turret doesn't directly pick a target and shoot after killing. To do this the current time will be used to be subtracted by the last time the turret shot its bullet. If this value was greater than the corresponding turret attack speed, then it will rotate and shoot.

e) Picking a Target

Two variables, set to 0, will be used to define the x and y coordinates. Using a for loop to go through every entity in its group, if an entity is still alive calculate its Euclidean distance. This is basically the distance of the longest side in a right-angled triangle. After calculating the distance, check if that distance is smaller than the turret's range. If it is then its target becomes from None to entity and the angle is calculated using a similar method with calculating entity face direction. The target's health is reduced according to the turret's damage and the player will get as much money as them damaging those entities. Lastly, play the turret sound effect and break the loop.

f) Turret Animation

Similar to how applying a delay for turrets when shooting, it is also applied to how the animation of a turret is cycled through, with a shorter 'cooldown'. It will add the frame index by one when its not

over the length of the animation list. If it was however, the frame index is reset to zero, self.lastShot is recorded and self.target is reset to None.

g) Upgrading Turrets

This function basically handles turret upgrading and the turret filtering is applied here. When a turret is upgraded, self.upgradeLevel is increased by 1 and this will be used to determine whether the upgrade button should appear or not. The white transparent circle will scale based on the turret's range however, the red circle stays the same with its fixed value of 30.

h) The Draw Method

Similar to how the entities are drawn, with the exception of the angle because initially the turret angle was 90 so that it points upwards when placed on a tile hence, the angle is subtracted by 90. If a turret is selected, then show the white and red transparent circle.

6. main.py

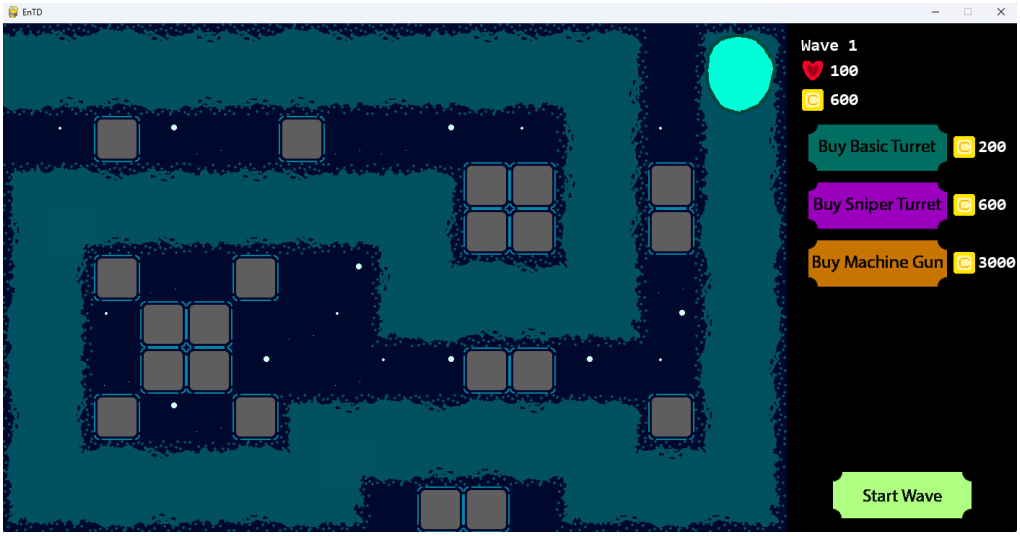
This is the main file of the game, the one to be run to open the game. It imports all the classes from the game files above with an additional json library imported which will be used to access the tmj file of the map. The screen resolution is set according to the values from the Settings class. Then some game variables are defined to determine the conditions of corresponding purpose. After that is loading images for map, entities, turrets, and the fonts for GUI essentials. Not to mention the json file is also loaded by first opening the file and then storing the data in mapData. Then the audio is loaded as well as the necessary functions. The first function is drawText and this will be used to texts on the screen. Next is displayData and will show the health, the money and the current wave with a black background. Next is the createTurret function and the purpose of this function is to filter out the places where turrets can be placed which is a tile with a value of 28, representing a grey platform. This also filters out which turret button was clicked by the user. Next is selectTurret with the sole purpose of when a player selects a placed turret on the map. Then clearSelectedTurret is used to deselect and remove the range image when clicking outside the selected turret area with the condition it is in the game area not the turret panel. Lastly, the reselectTurret function has

a sole purpose of ensuring that once a player selects a turret to buy, if they were to cancel they should be able to select the same turret again. After the functions, objects and clickable buttons are made.

For the game loop it is divided into two parts, the update and the draw section. The update section keeps track of waves and some audio. The draw section draws everything on the screen as well as keep track of when a certain audio will play if an ‘event’ occurs. For instances, when the player presses the start wave button the music starts playing.

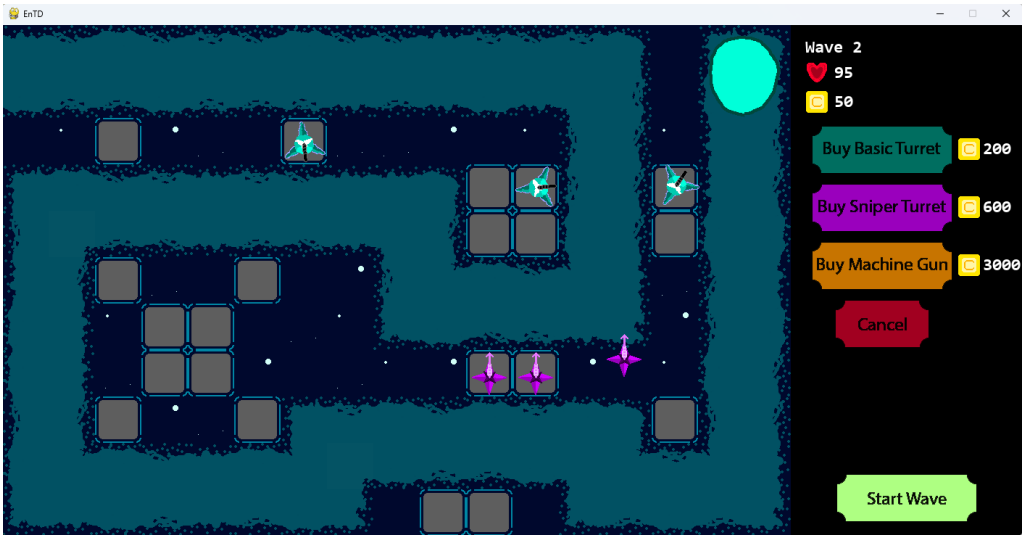
E. Screenshots of the Game

1. Game Screen (Before wave starts)

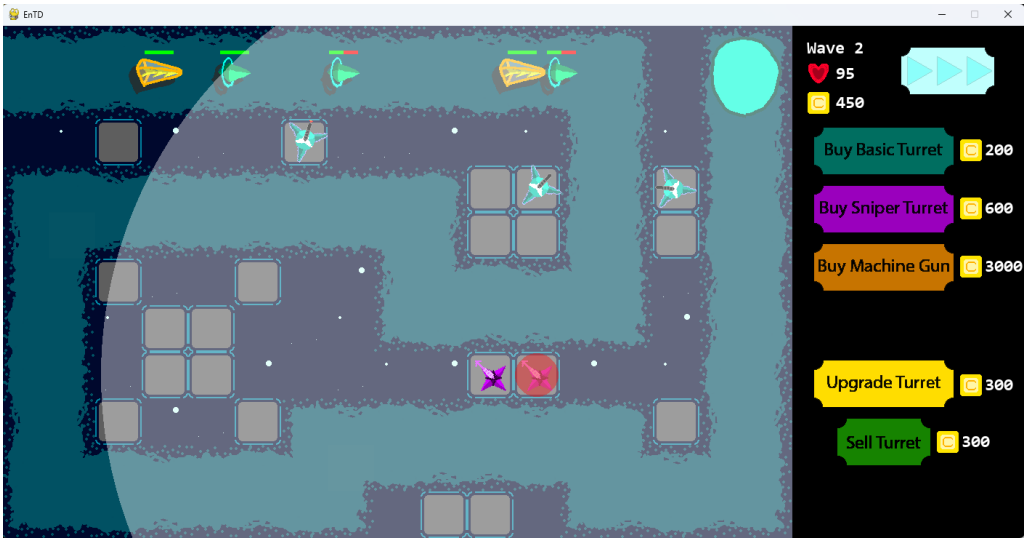


2. Game Screen (During wave)

3. Buying and Placing Turret

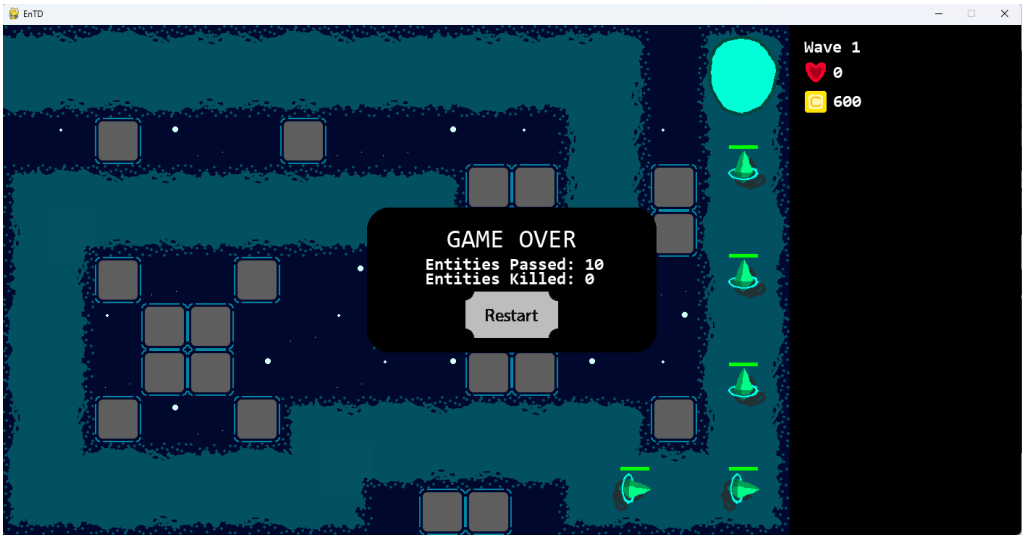


4. Selecting Turret



5. **Game Over Screen (Win)**

6. **Game Over Screen (Lose)**



F. Lessons Learnt/Reflection

During the duration of this final project, I can say it was quite heavy. Especially when coding is quite new to me and an experience I haven't quite yet felt before. One thing is for sure is that when making a big project using python as the code, making classes and defining functions does make it look neater and more readable. Back then I used to think that classes overcomplicate things and putting all the code in a single file is the way to go. However, I can see the benefits of using classes that is beneficial for me as well as other teammates in case it was a group project. Honestly speaking, I could have done more by adding a bit more here and there into the code however, with how bad my time management is, I was kind of surprised I finished this project. This also brings me to another topic to be improved, time management. I sort

of learned the hard way how time management affects assignment execution performance. Nevertheless, a mistake was a mistake and I'll learn from those mistakes.

G. Resources

1. Pygame learning
   https://www.pygame.org/docs/index.html
2. Tiled Learning
   https://youtu.be/ZwaomOYGuYo?si=vgUrMdHq_r5atKe5
3. Music and sound effects
   https://www.y2meta.com/en163