Infected City

Analysis **4**
- Proposal 4
- Stakeholders 4
- Research 5
- Features 6
- Limitations 6
- Requirements 7
- Success Criteria 8

Prototype 1 **9**
- Design 9
  - Decomposition 9
  - Structure of solution 10
  - Key variables/data structures/classes 10
  - Define solution fully 11
  - Usability features 16
  - Test data 17
  - Post development data 17
- Development 17
  - Testing 17
  - Review 17

Prototype 2 **18**
- Design 18
  - Decomposition 18
  - Structure of solution 19
  - Key variables/data structures/classes 20
  - Define solution fully 22
  - Usability features 31
  - Test data 32
  - Post development data 32
- Development 32
  - Testing 32
  - Review 32

Prototype 3 **33**
Design 33
Decomposition 33
Structure of solution 33
Key variables / Data structures / Classes 35
Define solution fully 36
Usability features 39
Test data 39
Post development data 39
Development 39
Testing 39
Review 40

Prototype 4 **40**
Design 40
Decomposition 40
Structure of solution 41
Key variables / Data structures / Classes 43
Define solution fully 44
Usability features 62
Test data 62
Post development data 63
Development 63
Testing 63
Review 63

Prototype 5 **64**
Design 64
Decomposition 64
Structure of solution 65
Key variables / Data structures / Classes 67
Define solution fully 68
Usability features 121
Test data 122
Post development data 122
Development 123
Testing 123
Review 123

Prototype 6 **125**

Design 125
Decomposition 125
Structure of solution 125
Key variables / Data structures / Classes 125
Define solution fully 125
Usability features 125
Test data 125
Post development data 125
Development 125
Testing 125
Review 125

Evaluation 126
Annotated evidence 126
Evaluation 126
Evaluating met criteria 126
Evaluating unmet criteria 127
Maintenance issues and limitations 127
Development to fix limitations 128

Analysis

Proposal

In this fast-paced world most of the people are stressed out at the end of the day. I like working on this project because my purpose is to make the players blur out their problems and enjoy the little time they have by playing this adventurous game, keeping them entertained all along.

According to research carried out by Gallup it is said that only 15% of the people love their job which means 85% hate their job. Most of the students are stressed out by homework and exams. I believe that this project can fulfill my purpose by allowing the user to be tension free while keeping the game entertaining and fun.

This game can be developed using a computational approach. For this game I will be using python and a tiled map editor. The complexity of this project lies in coding an intelligent AI, for example when the player is close to an enemy, the enemy should be able to figure out the shortest path to reach the player using an algorithm (A* algorithm). I intend to make this game more engaging and entertaining by adding colourful graphics and animations.

Stakeholders

This game is unsuitable for children because there will be fighting animations, instructions, clues (so users should be able to have some sort of strategic approach). Therefore this game is mainly developed for teenagers and adults to play during their free time.

For this game my stakeholders will be ## (2 adults) who will be playing this game, so I would be able to monitor their movements and to get an understanding of how the user might think. So I would be able to improve AI performance and get rid of any glitches if found.
As for teenagers these ## (2 teens) will be my stakeholders, since teens' cognitive ability might be different from adults I will be able to record both parties movements and make this game better.

Research

Existing similar solutions:

Phantasy Star



Overview:
Phantasy Star is a role-playing video game developed by Saga and released for the Master system in 1987. One of the earliest Japanese RPGs for consoles. Phantasy star tells the story of Alis on her journey to defeat the evil ruler of her start system.

Parts that I can apply to my solution:
As seen from the image above I can see that, it is a tile based rpg game with a top-down view. Using a tile based system can be very efficient when adding or removing objects from the map. From my research I have found that there are tile map editors which makes this task even easier.

I intend to make my game with an elevated viewpoint (top down or overhead perspective) so that the player will have a sufficient amount of view to move around and explore objects.

For camera movement I have noticed that it is better when the player is fixed in the middle when the player moves around (the map moves and not the player). So that it will look like a scrolling camera with the player being fixed in the middle of the game window.

Features

Intelligent AI - this will allow for more complex gameplay, make the game interesting and challenging

Controls - This game will have 5 keys in total, 4 keys for movements and 1 key for shooting / attacking so that it will be easy for the users to understand the controls and will allow for a wider range of audiences. This game would be suitable for beginners who never played a game. I intend to give the player with an option to either play with arrow keys or WASD keys for the convenience of some players.

Strategy based - This game will provide users with instructions to follow. The user will be able to win this game by using a suitable winning strategy.

The user should be able to experience new maps, characters, weapons and music as the player levels up.

Ability to save and load the game.

Limitations

Graphics - This is a 2D game, 3D graphics is not suitable for this game because it is coded in python and since this game is developed for a wider range of audiences 3D graphics cannot be supported by all the devices

Controls - The main controls would be from the keyboard with the use of a mouse / touchpad, this game doesn't support any external controls such as game consoles. Since every computer has a keyboard and mouse this game could be played by almost everyone.

Language - This game is in English language, so the users should be able to read and understand English to follow the instructions

Realism - This game will not look real simply because it is time-consuming and a waste of time to code each and every part of the player as an object because this is an educational project.

Requirements

Hardware requirements

Computer:
This game can be played by almost all of the desktop computers or laptops with a fairly fast processor.

Keyboard:
Since the player is controlled by the 5 basic keys (arrow keys or WASD keys and spacebar), A keyboard is essential to play this game.

Display:
Most of the desktop computers come with a monitor and a laptop has an inbuilt screen so there will be no issue in playing this game. External monitors would work for this game but it might not be as smooth since this game is built in a laptop in it's resolution size.

Software requirements

Operating system:
MacOs, Windows and linux should work fine

Python interpreter:
Python, pygame and pytmx should be installed in the computer for this game to work.

Success Criteria

In order for this project to be successful it should meet the following criteria:

This game should be coded in object oriented programming

The game should be challenging and need some form of logical thinking in order to win. Since this game is played by teens and adults in their free time I am not intending to make this game extremely difficult or easy so that the player does not get too stressed or bored. My goal is to make the player enjoy and entertain themselves while playing this game.

The game should have fairly good animations for example walking, fighting, running. It should have new characters and new maps as the player levels up so that the game can become more interesting and fun to play.

The game should have a loading system so that the player can save the game and quit. The user should be able to reload the game from the place where they left off. This is an essential feature since the targeted audiences are usually busy.

The game should have an option to look at the name of the person with the highest score.

The AI should be intelligent and must be able to make smart decisions in any given situation. For example when the player comes close to an enemy, the enemy should be able to find the shortest path to reach the player efficiently.

The game should have simple and easy controls so that the player would be able to play this game with little to no effort.

The game should have an ‘options’ category in the menu system so that the user can turn on / off certain functions such as background music and bullet sound.

The game should have a scoring system which is defined as follows:

1. The player can collect coins by moving around the map while defeating the enemy.
2. Each coin would be worth 5 points
3. There should be special coins that are worth more than 5 points (8, 10 etc...) placed at critical points (near enemy or fire)
4. When the player gets hit by the enemy the health should decrease by 5 percent

5. When the player levels up there should be a bonus point given to the player
6. The player should be able to buy and sell guns by exchanging points
7. The user to score the highest points would have a trophy next to their name.

There should be inventories for the player such as knife, gun, health / food.
The player should be able to increase their health by consuming food / health.

The player should be able to consume some magic potion to get different kinds of strengths such as fast movement, invisibility, reduced hit damage etc...

The game should have objects such as fire / water to decrease the player's health or to slow down the player

Furthermore the game should have two options, one to play against the AI and the other option to play one on one with a human. WASD will be assigned for user 1 and arrow keys for user 2. Before the match begins the user can choose the weapon and their preferred outfit. Both the users will have the same function as the 'player', but with two different score sets. The first user to die will be the loser.

Prototype 1

Design

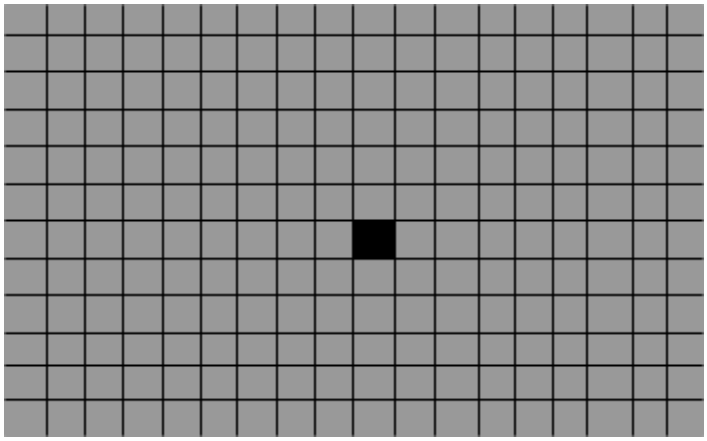
Decomposition

To start off I will have a window display 1024 by 768 pixels in grey background. Since this is a tile based game I will be drawing a grid to allow me to look at the tiles so that it will be easy for me to understand and place the objects in the place I wish to. Each tile size will be 32 by 32 pixels

After creating the grid, I will create a 'Player' class with a player which can move in eight directions, north, north-east, east, south-east, south, south-west, west, north-west, north.

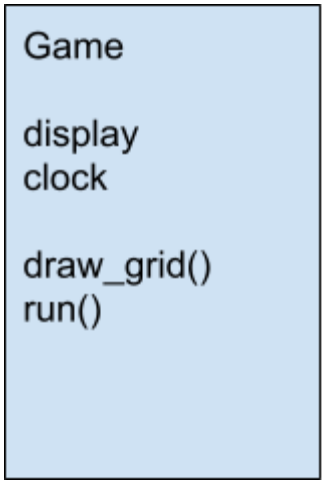
I will add an image for the player by downloading non-copyrighted images from the internet. Then I will create a 'Wall' class which will be a black square and allow the player to be able to collide with the wall.

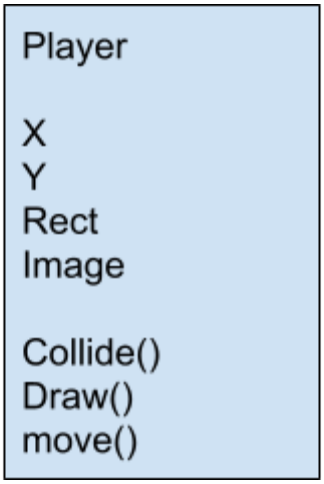
Structure of solution

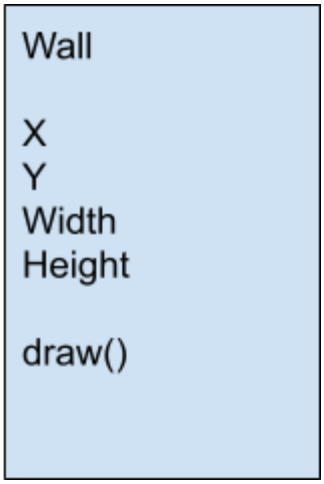


There will be 32 tiles (32 * 32 = 1024 the screen width) in the x axis and 24 (24 * 32 = 768 the screen height) in the y axis.

Key variables/data structures/classes







Key variables

Variable	Data type	How it is used
Width	An integer value that will store the width of the screen window	Create a display for the pygame window
height	An integer value that will store the height of the screen	Create a display for the pygame window
Tilesize	An integer value that stores	When drawing horizontal and

	the tile width and height	vertical grids on the screen
Image 1 - 4	String value holding the name of the image	When blitting the player on the screen
x, y	An array storing 2 integer values	Assign the player's position on the screen
walking	boolean	To return if the player is walking or standing

Input and output

input	process	output
Arrow keys	Identify which arrow key is being pressed and call the appropriate move function	The player will move in the direction that is being pressed

Define solution fully

In this prototype there will be a player and a series of walls displaying on the screen, and the player will be able to collide with the wall.
Collision :
This function checks the player's and wall's coordinates continuously , if the player's 'rect' is overlapping with the wall's rect it means that the player has collided with the wall, so the 'collide' function will return the boolean True.

I will be coding in python in an object oriented approach so that it would be easy to duplicate classes and use the code in another game. By coding in an object oriented approach it would be much easier to fix bugs and allow other developers to understand and edit the code

sgame.py | constants.py

```python
# Screen
WIDTH = 1024     # 32 tiles (64 * 18)
HEIGHT = 768    # 24 tiles (64 * 12)
FPS = 60

# Tile
TILESIZE = 32

# Colors
LIGHTGREY = (40, 40, 40)
DARKGREY = (100, 100, 100)
BLACK = (0, 0, 0)
BGCOLOR = DARKGREY
```

```
pygame 2.0.0.dev6 (SDL 2.0.10, python 3.8.0)
```

Line 1, Column 1 UTF-8 Spaces: 4 Python

sgame.py

```python
import pygame
import os
from constants import *
from images import *


class Game:
    def __init__(self):
        pygame.init()
        self.win = pygame.display.set_mode((WIDTH, HEIGHT))
        self.clock = pygame.time.Clock()

    def draw_grid(self):
        for x in range(0, WIDTH, TILESIZE):     # Vertical lines
            pygame.draw.line(self.win, BLACK, (x, 0), (x, HEIGHT))
        for y in range(0, HEIGHT, TILESIZE):    # Horizontal lines
            pygame.draw.line(self.win, BLACK, (0, y), (WIDTH, y))

    def run(self):
        run = True
        while run:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    run = False

            player.move()

            game.win.fill(DARKGREY)
            game.draw_grid()
            game.clock.tick(FPS)
            #image = pygame.draw.rect(win, BLACK, (x, y, 32, 32))

            player.draw(game.win)
            wall1.draw(game.win)
            pygame.display.update()

        pygame.quit()
```

```
pygame 2.0.0.dev6 (SDL 2.0.10, python 3.8.0)
```

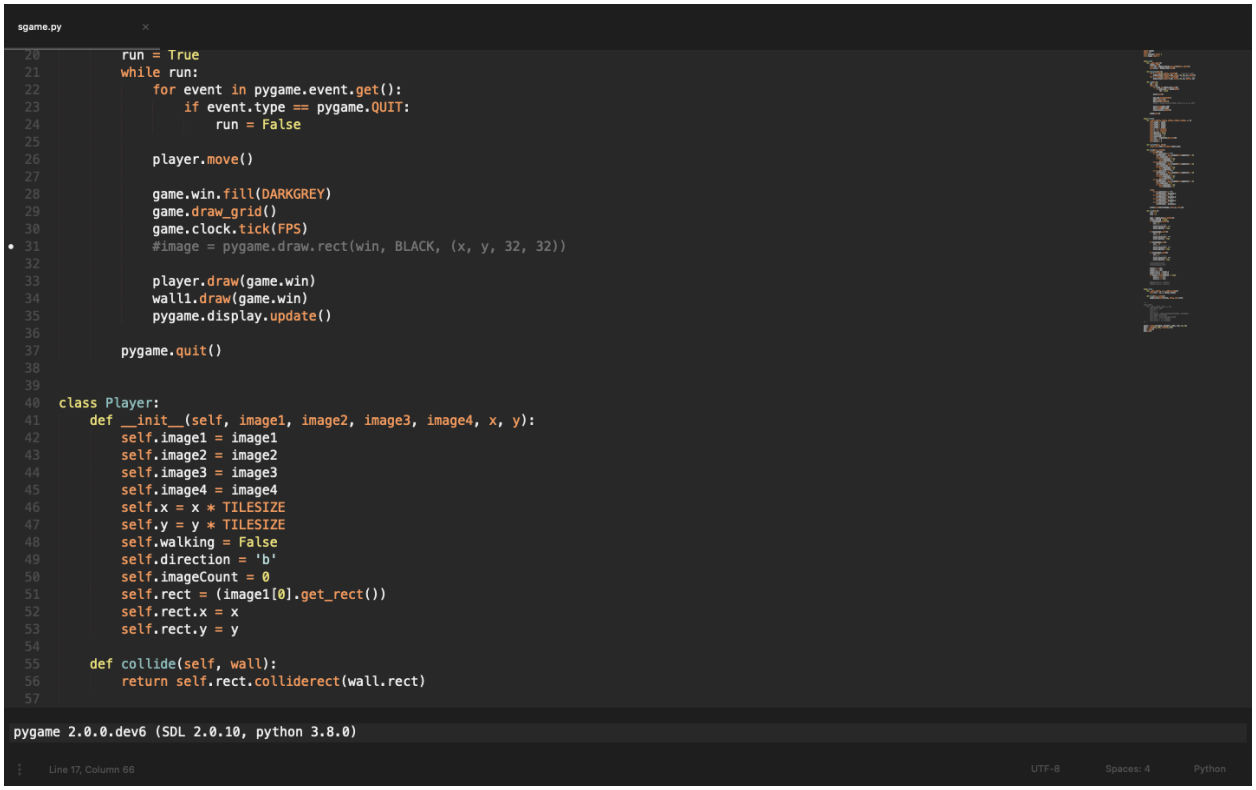
Line 17, Column 66 UTF-8 Spaces: 4 Python

In my game class I have initialised the game and created a window of width 1024 and height 768 pixels. I have included all the constants and colors in my constants file.

The ‘draw_grid’ function loops through each coordinates in the x axis from 0 to width (1024) by a gap of tile size (32) and draws a black line from top to bottom for each x (vertical lines), Then it loops through each coordinates in the y axis from 0 to height (768) by a gap of tile size (32) and draws a black line from left to right for each y coordinate (horizontal lines)

I have defined a method to draw grids, so that it would be easy for me to know where to place the player or the wall on the screen.I will comment out that method once I have finished my project.

I have defined a method ‘run’ which is my main loop, it includes all the other methods and runs the game.



sgame.py × constants.py ×

```
    def draw(self, screen):
        if self.walking:
            if self.direction == 'f':
                currentImage = self.image1[self.imageCount // 5]
                self.imageCount += 1
                if self.imageCount > 44:
                    self.imageCount = 0
            if self.direction == 'b':
                currentImage = self.image2[self.imageCount // 5]
                self.imageCount += 1
                if self.imageCount > 44:
                    self.imageCount = 0
            if self.direction == 'r':
                currentImage = self.image3[self.imageCount // 5]
                self.imageCount += 1
                if self.imageCount > 44:
                    self.imageCount = 0
            if self.direction == 'l':
                currentImage = self.image4[self.imageCount // 5]
                self.imageCount += 1
                if self.imageCount > 44:
                    self.imageCount = 0

        else:
            if self.direction == 'f':
                currentImage = standing_f
            if self.direction == 'b':
                currentImage = standing_b
            if self.direction == 'r':
                currentImage = standing_r
            if self.direction == 'l':
                currentImage = standing_l

        screen.blit(currentImage, (self.x, self.y))

    def move(self):
        velx = 0
```

pygame 2.0.0.dev6 (SDL 2.0.10, python 3.8.0)

Line 71, Column 65 UTF-8 Spaces: 4 Python

sgame.py constants.py ×

```
    def move(self):
        velx = 0
        vely = 0

        keys = pygame.key.get_pressed()
        player.walking = False
        if keys[pygame.K_RIGHT]:
            velx = 3
            #player.x += velx
            player.direction = 'r'
            player.walking = True

        if keys[pygame.K_LEFT]:
            velx = -3
            #player.x -= velx
            player.direction = 'l'
            player.walking = True

        if keys[pygame.K_UP]:
            vely = -3
            #player.y -= vely
            player.direction = 'f'
            player.walking = True

        if keys[pygame.K_DOWN]:
            vely = 3
            #player.y += vely
            player.direction = 'b'
            player.walking = True

        player.x += velx
        player.y += vely
        player.rect.x = player.x
        player.rect.y = player.y
        if player.collide(wall1) == True:
            player.x -= velx
            player.y -= vely
```

pygame 2.0.0.dev6 (SDL 2.0.10, python 3.8.0)

Line 121, Column 34 UTF-8 Spaces: 4 Python

In my ‘Player’ class I have initialized all the variables. The direction is set to ‘b’ - backwards so that when the game starts the player will face backwards.

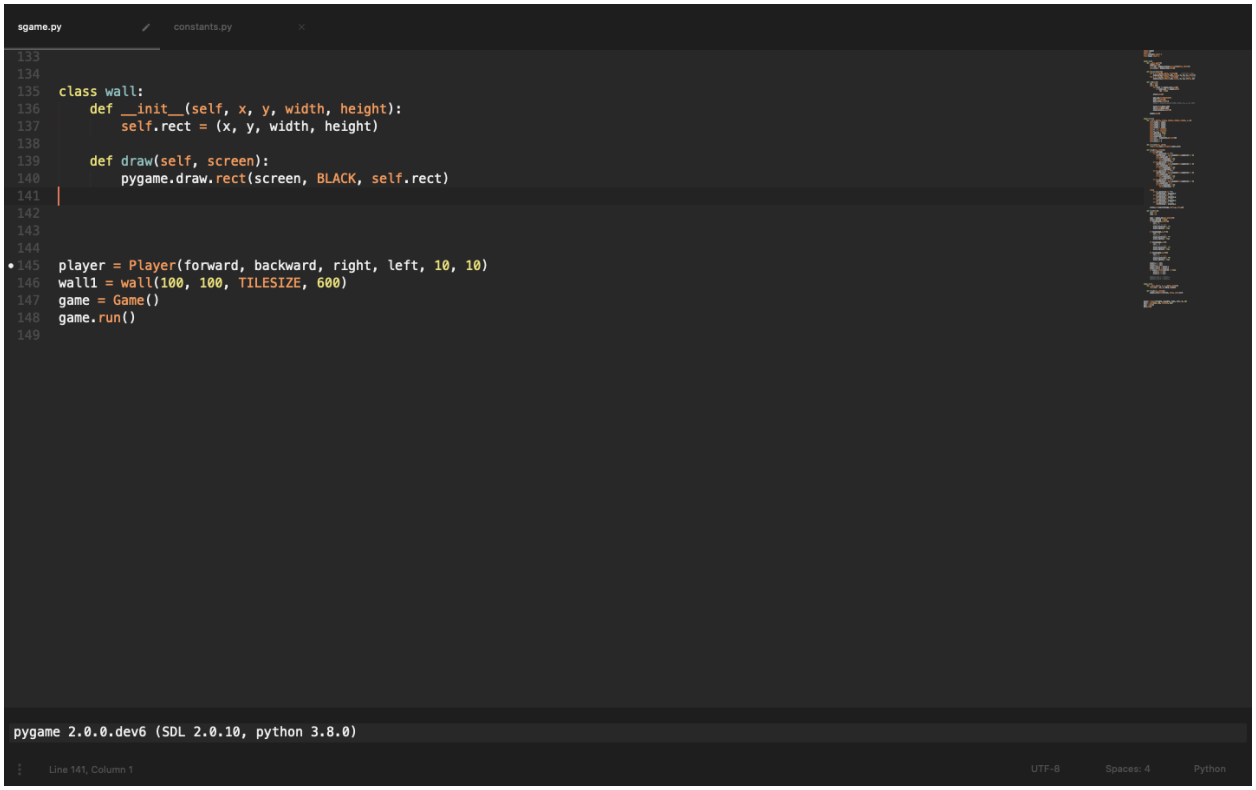
Since my image is always moving I have 4 different set of images for each movement

I have defined a method for collision so that it will return a boolean if it collides with the wall. The draw method prints the images according to the movement of the player and keeps the window updated

Since the top left coordinate is (0, 0) when the player moves to the right the x value increases and when it moves to the left the x value decreases, when the player moves down the y value increases and when it moves up the y value decreases

The move method is the one which controls the player. When the player presses the right key the velocity of x is set to positive 3 and when the player presses the left key the velocity of x is set to negative 3 so this would increase or decrease the players ‘x’ coordinate once the player presses a key, which will appear like the player is moving left or right, the same principle is applied on y axis when the player presses the down key the velocity of y is set to 3 and when the player presses the up key the velocity of y is set to negative 3

I have checked for the collision between the player and the wall using the ‘collide’ function, if this function returns True then I have taken back the movement the player was about to do. For example if the wall is at (x, y) and the player is at (x - velx, y), if the player presses the right button the player’s coordinate then would become (x, y) which means it has now collided with the wall so the ‘collide’ function would return true but by taking back that step I will be able to avoid the player move right. So even if the player presses the right key the player won’t be able to move right.



In my wall class I have created a square of width and height of a tilesize (32 pixels). I have got its rect (x, y, width, height) from the inbuilt function in order to check for collision with the player.

The draw method draws the rectangle onto the screen, with the help of the ‘draw_grid’ function it is easy to know the exact coordinates of the wall.

Finally I have made two objects (instance of each class) for the player and the wall.

game.run() keeps the game running until an error is found or till the user quits.

Usability features

The player is able to move in eight directions right, left, up, down and diagonally in a set velocity.

The player can collide with the wall

Test data

I have tested all the possible movements and it moves as it should be moving, since it has no boundaries it can move out of the screen but it won't be an issue when I place the wall around the screen.

Post development data

For my next prototype i will be focusing on creating enemies, the player having the ability to shoot the enemy and a wall class with a map folder.

I will change the player's image because the current image cannot show diagonal movements.

I will also be creating a new text folder which will be my map, so when I need to place a wall into the screen it would be really easy for me to place it by just adding it into the map folder.

I will create an enemy class and code it's collision function separately.

This will be a tile based game.

Development

Testing

Using the link below you will be able to have a look at my first prototype in which I check for movements, images and wall collisions.
(change the video quality to 720p)

Prototype 1 testing video

Review

The player is able to move around the screen, for now I haven't limited it with boundaries because I think when I insert a wall into the edges it should solve this issue. Since the image of the player cannot show it's movement when moving diagonally I will find a new image for it to solve this problem.
The player is able to collide with the wall and it doesn't overlap with the wall
I have found that when the player moves diagonally the player doesn't move instead it gets stuck to the wall. So the user will not be able to press two keys and move at the same time.
The background window is working as it should be.

Prototype 2

Design

Decomposition

In this prototype I will be creating an enemy class which should be exactly the same as the player class only without the movement method. For the time period I will be coding a movement method with a W, A, S, D key to move which will be removed later because it will be the AI which will perform the enemy movement.

I will be creating a new text folder with numbers or letters assigned to a particular object (i.e player, wall, enemy).

The player and the enemy should be able to collide with the wall and should not overlap with each other.

I will be creating a bullet class so that the player will be able to shoot. The bullet should be able to collide with the enemy and the wall. When the bullet hits the enemy its health should decrease so that when the health is zero or less the enemy will be deleted from the enemy list. When the bullet hits the wall the bullet's index should be taken off from the list so that it will look like it disappears.

Structure of solution

map.txt

```
1111111111111111111111111111111
...............1.......1.......1
......E........1.......1..E....1
...............1.......1.......1
111111111111...11...1111111...11
1..............................1
1...................E..........1
1..............................1
1.........11111111111.......111
1...P.....1...............E.....
1.........1....................
1.........1...1111...1.......111
1.........1...1111...1.........1
1....E....1...1111...1.........1
1.........1..........1...E.....1
1.........1..........1.........1
1.........111111111111.........1
1..............................1
1..11111.......................1
1......1.......E.....11111111..1
1......1.............1.........1
1......1.............1.........1
1......1.............1.........1
1111111111111111111111..11111111
```

Line 25, Column 1 UTF-8 Spaces: 4 Plain Text

This is my map text folder. It is 32 tiles horizontally and 24 tiles vertically. ‘1’ represents the wall, ‘P’ represents the player and ‘E’ represents the enemy. By using this method it will be easy for me to add or delete objects in the future.

I will no longer use the image I used in prototype 1 since it cannot perform diagonal movement. Therefore I will be using a much simpler image for the player and the enemy which I got from the internet.

Player
-

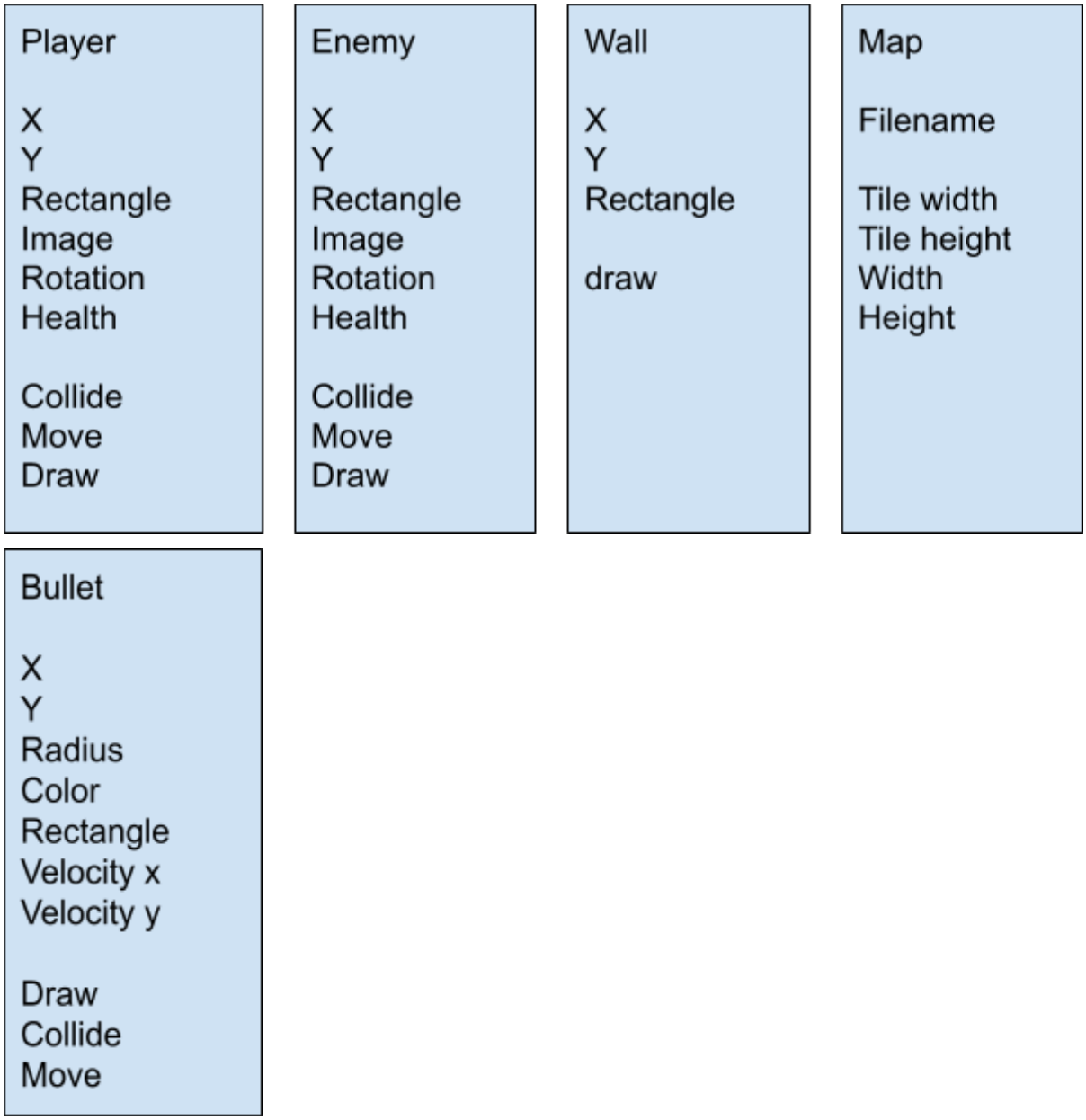


Enemy
-



For the bullet image, I will be using the inbuilt function to make a black circle.

Key variables/data structures/classes



Key variables

Name	Data type	How it is used
wall_list	A large 2D array that will store the instances of the wall in the game	This will be used to get the coordinates of each wall from the wall list
enemy_list	A large 2D array that will store all the instances of the enemy	Get coordinates of each enemy from the enemy list
player_list	2D array that will store the instance of the player	Get coordinates of the player
bullets	A large 2D array that will store all the bullets that is being fired	Get coordinates of each bullet in the list to check if the bullet has gone out of the screen
health	An integer value storing the players health	To check if the player has died
rotation	An integer value storing the rotation of the image	To transform the image in the rotation value
x, y	Integer values storing the players location	Get the players coordinates
velx, vely	Integer values storing the velocity in x axis and y axis	to move the player by a fixed velocity
tile_width tile_height	An integer value storing the number of tiles across the x axis and y axis	Can be used to find out the length and height of the map when finding the player / enemy / wall in the map.

Input and output

input	process	output
spacebar	Triggers the bullet function	Fires a bullet in the screen

Define solution fully

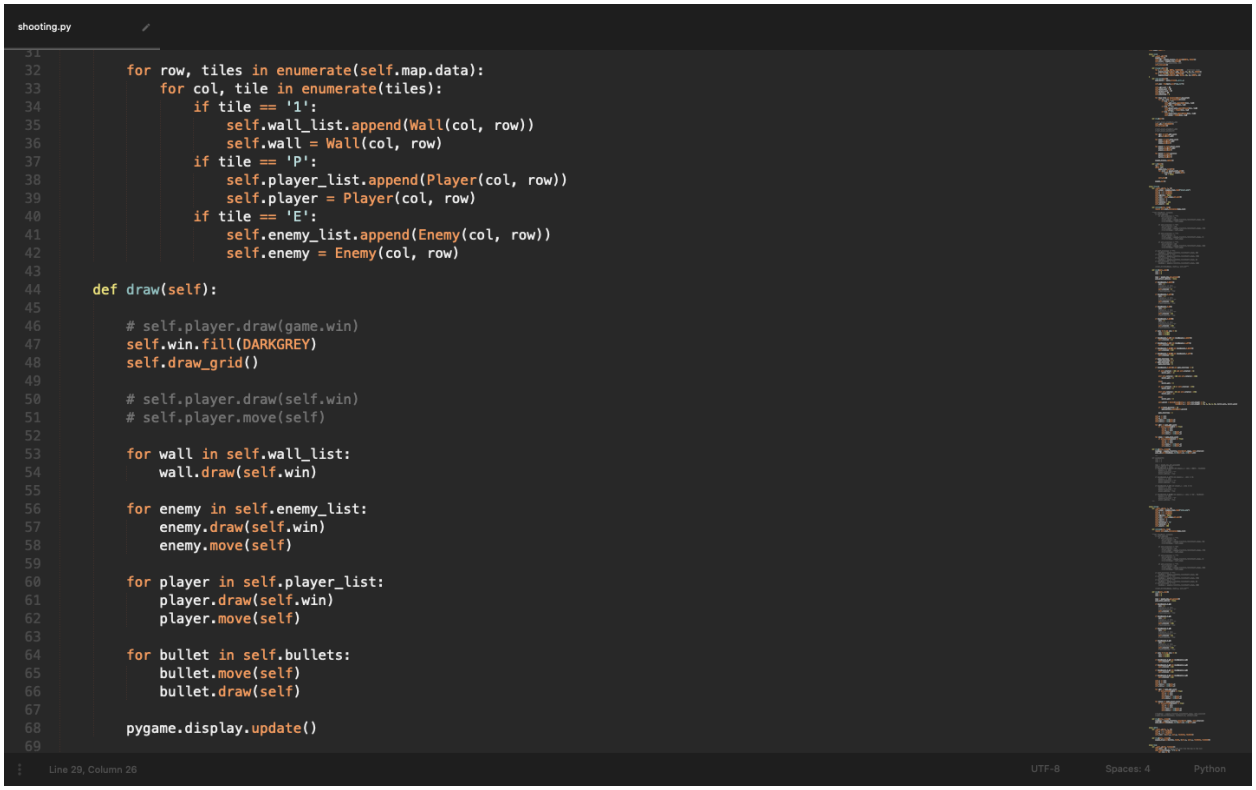
Updated code



From the map text folder I have allocated numbers and letters to different objects so that I can create multiple duplicate objects like walls and enemies.

I have created lists for each class to store the instance of that particular class.

I have gone through each row and column in the map folder to find the coordinates of each tile with the special letter / number and made an instance of that class while storing it onto the list.



In this draw function the ‘draw’ method and ‘move’ method is called for each object from the list. So this is the method which basically draws or moves the objects on the screen.



I have added health and changed the image from my previous player class. I have added a collide method that checks if the player is colliding with any other object and if it does it will return true.

shooting.py

```
    def move(self, game):
        velx = 0
        vely = 0

        keys = pygame.key.get_pressed()
        game.player.walking = False

        if keys[pygame.K_RIGHT]:
            velx = 3
            # player.x += velx
            # gselfdirection = 'r'
            self.rotation = 0
            # gselfwalking = True

        if keys[pygame.K_LEFT]:
            velx = -3
            # player.x -= velx
            # gselfdirection = 'l'
            self.rotation = 180
            # gselfwalking = True

        if keys[pygame.K_UP]:
            vely = -3
            # player.y -= vely
            # gselfdirection = 'f'
            self.rotation = 90
            # gselfwalking = True

        if keys[pygame.K_DOWN]:
            vely = 3
            # player.y += vely
            # gselfdirection = 'b'
            self.rotation = 270
            # gselfwalking = True

        if velx != 0 and vely != 0:
            velx *= 0.7071
            vely *= 0.7071
```

Line 91, Column 24 UTF-8 Spaces: 4 Python

The player will spawn facing east at a rotation of 0 degrees then the player's image will change according to the player's movement / direction. I rotated the image using the inbuilt transform image function, so when the player wants to travel left I have set the rotation angle to 180 degrees so that the image would rotate to 180 degrees anticlockwise, when the player wants to travel south the rotation would be 270 degrees, when the player wants to travel north east the rotation would be 45 degrees and so on.

shooting.py

```
        if velx != 0 and vely != 0:
            velx *= 0.7071
            vely *= 0.7071

        if keys[pygame.K_UP] and keys[pygame.K_RIGHT]:
            self.rotation = 45

        if keys[pygame.K_UP] and keys[pygame.K_LEFT]:
            self.rotation = 135

        if keys[pygame.K_DOWN] and keys[pygame.K_RIGHT]:
            self.rotation = 315

        if keys[pygame.K_DOWN] and keys[pygame.K_LEFT]:
            self.rotation = 225

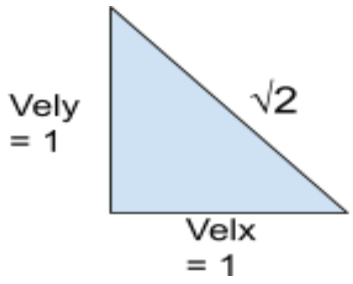
        # checking for collisions
        self.x += velx
        self.y += vely
        self.rect.x = int(self.x)
        self.rect.y = int(self.y)

        for wall in game.wall_list:
            if self.collide(wall) == True:
                self.x -= velx
                self.y -= vely
                self.rect.x = int(self.x)
                self.rect.y = int(self.y)

        for enemy in game.enemy_list:
            if self.collide(enemy) == True:
                self.x -= velx
                self.y -= vely
                self.rect.x = int(self.x)
                self.rect.y = int(self.y)
```

Line 182, Column 34 UTF-8 Spaces: 4 Python

I had an issue when the player moved diagonally because the player moved too fast, in order to solve this issue I used pythagoras theorem to let the player move in the set speed limit which is to multiply the velocity with the square root of 2 divide it by 2 (0.7071 4s.f)



I have checked for collisions between the player and the wall / enemy separately. If the collide method returns true, the player won't be able to move or overlap with the wall / enemy.

shooting.py

```
class Bullet(object):
    def __init__(self, x, y, radius, color, velx, vely):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.rect = pygame.Rect(0, 0, 0, 0)
        self.velx = velx
        self.vely = vely

    def draw(self, game):
        self.rect = pygame.draw.circle(
            game.win, self.color, (self.x, self.y), self.radius)

    def collide(self, obj):
        return self.rect.colliderect(obj.rect)

    def move(self, game):
        if self.x < WIDTH and self.x > 0:
            self.x += self.velx

        else:
            game.bullets.pop(game.bullets.index(self))

        if self.y < HEIGHT and self.y > 0:
            self.y += self.vely
        else:
            game.bullets.pop(game.bullets.index(self))

        for enemy in game.enemy_list:
            if self.collide(enemy) == True:
                enemy.health -= 25
                print(enemy.health)
                game.bullets.pop(game.bullets.index(self))
                if enemy.health <= 0:
                    game.enemy_list.remove(enemy)
```

Line 206, Column 1 UTF-8 Spaces: 4 Python

shooting.py

```
        self.vely = vely

    def draw(self, game):
        self.rect = pygame.draw.circle(
            game.win, self.color, (self.x, self.y), self.radius)

    def collide(self, obj):
        return self.rect.colliderect(obj.rect)

    def move(self, game):
        if self.x < WIDTH and self.x > 0:
            self.x += self.velx

        else:
            game.bullets.pop(game.bullets.index(self))

        if self.y < HEIGHT and self.y > 0:
            self.y += self.vely
        else:
            game.bullets.pop(game.bullets.index(self))

        for enemy in game.enemy_list:
            if self.collide(enemy) == True:
                enemy.health -= 25
                print(enemy.health)
                game.bullets.pop(game.bullets.index(self))
                if enemy.health <= 0:
                    game.enemy_list.remove(enemy)

        for wall in game.wall_list:
            if self.collide(wall):
                game.bullets.pop(0)
                if game.bullets == []:
                    return True
```

Line 206, Column 1 UTF-8 Spaces: 4 Python

This is my bullet class, it draws a circle on the screen while moving at a certain velocity and checking for collision between walls /enemies.

I have created a ‘bullet_list’ in the ‘Game’ class ‘load_data’ function to store all the bullets fired.
In the move function the bullet only moves if the following conditions satisfy:
The bullets x position should be less than the windows width and greater than 0
The bullets y position should be less than windows height and greater than 0.

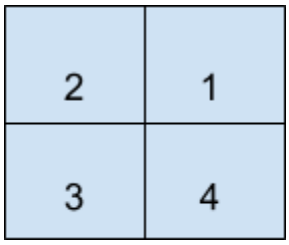
If the above condition satisfies the bullet moves or else it is popped off from the bullet list because it has gone beyond the screen.

It then checks if it collides with any of the enemy from the list, if it has collided with one of the enemy then its health is reduced by 25 (out of 100)
If that enemy’s health reaches zero or less than zero then the enemy is popped from the list so that it will disappear from the screen.

It also checks for collision with the wall, so if it collides with a wall that particular bullet will be popped off from the list.



This part of the code is from the ‘move’ function in ‘Player’ class which creates an instance of the bullet class, for now only the player will be having the ability to shoot. The shoot loop delays the period of time it takes to shoot again so that there will be a visible gap on the screen between bullets.



When the player presses the spacebar it checks for the rotation of the player, if the player faces the third and fourth quadrant (south) which means the player's rotations must be greater than 180 and less than 360, the velocity of the bullet would be positive in y direction. This would allow the bullet to travel in the south direction from the player's position

If the player faces the first or second quadrant (north) the velocity of the bullet in y direction would be negative, the same principle applies in the x axis. When the player is facing the first and fourth quadrant - east (the rotation will be less than 90 and greater than 270) the velocity of the bullet in x axis then would be positive. So when the bullet fires the bullet will travel in the right direction from the player.

Then I have created an instance of the bullet class creating a black circle.
I have only allowed for 5 bullets on the screen so when all 5 bullets have been shot the player needs to wait until the bullet goes off the screen or hits an enemy or hits a wall to regenerate another bullet into the bullet list.

The draw function draws the black circle onto the window screen

shooting.py

```
class Enemy:
    def __init__(self, x, y):
        self.image = pygame.image.load('enemy.png')
        self.x = x * TILESIZE
        self.y = y * TILESIZE
        self.walking = False
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.direction = 'b'
        self.rotation = 0
        self.health = 100

    def collide(self, obj):
        return self.rect.colliderect(obj.rect)

    def move(self, game):
        velx = 0
        vely = 0

        keys = pygame.key.get_pressed()
        game.enemy.walking = False

        if keys[pygame.K_d]:
            velx = 3
            # player.x += velx
            # gselfdirection = 'r'
            self.rotation = 0
            # gselfwalking = True

        if keys[pygame.K_a]:
            velx = -3
            # player.x -= velx
            # gselfdirection = 'l'
            self.rotation = 180
            # gselfwalking = True

        if keys[pygame.K_w]:
```

Line 297, Column 1 UTF-8 Spaces: 4 Python

shooting.py

```
        if keys[pygame.K_w]:
            vely = -3
            # player.y -= vely
            # gselfdirection = 'f'
            self.rotation = 90
            # gselfwalking = True

        if keys[pygame.K_s]:
            vely = 3
            # player.y += vely
            # gselfdirection = 'b'
            self.rotation = 270
            # gselfwalking = True

        if velx != 0 and vely != 0:
            velx *= 0.7071
            vely *= 0.7071

        if keys[pygame.K_w] and keys[pygame.K_d]:
            self.rotation = 45

        if keys[pygame.K_w] and keys[pygame.K_a]:
            self.rotation = 135

        if keys[pygame.K_s] and keys[pygame.K_d]:
            self.rotation = 315

        if keys[pygame.K_s] and keys[pygame.K_a]:
            self.rotation = 225

        self.x += velx
        self.y += vely
        self.rect.x = int(self.x)
        self.rect.y = int(self.y)

        for wall in game.wall_list:
            if self.collide(wall) == True:
                self.x -= velx
```

Line 297, Column 1 UTF-8

The enemies class is the same as the player's class with only a few changes. I have changed the enemies image and it checks for collision with the player and the wall. For now I have allowed the user to control the enemy by using W,A,S,D keys for testing purposes.

shooting.py

```
class Wall:
    def __init__(self, x, y):
        self.x = x * TILESIZE
        self.y = y * TILESIZE
        self.rect = (self.x, self.y, TILESIZE, TILESIZE)

    def draw(self, screen):
        pygame.draw.rect(screen, BLACK, (self.x, self.y, TILESIZE, TILESIZE))




class Map:
    def __init__(self, filename):
        self.data = []  # append each line from the map to the list
        with open(filename, 'rt') as f:
            for line in f:
                self.data.append(line.strip())

        self.tilewidth = len(self.data[0])
        self.tileheight = len(self.data)
        self.width = self.tilewidth * TILESIZE  # map width
        self.height = self.tileheight * TILESIZE    # map height


class Bullet(object):
    def __init__(self, x, y, radius, color, velx, vely):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.rect = pygame.Rect(0, 0, 0, 0)
        self.velx = velx
        self.vely = vely

    def draw(self, game):
        self.rect = pygame.draw.circle(
            game.win, self.color, (self.x, self.y), self.radius)
```

Line 393, Column 44 UTF-8 Spaces: 4 Python

I created a wall class so it is really easy to place a wall anywhere in the map, it is a rectangle with width and height the same as the grid tile size, the draw function draws each square from the map onto the screen at the (x, y) coordinates.

In my ‘Map’ class it opens the text folder (the map) and appends all the data in it onto the data list in the ‘Game’ class so that it can look for special numbers or letters and create an instance at the position by getting the coordinates from the map.

Usability features

The player and enemy can collide with each other and also the wall
The player can shoot and reduce enemy’s health
The bullet can collide with the enemy and the wall
When the enemies health is zero the enemy disappears from the screen
The map made it easy to create a wall, enemy or player by just adding it onto the map

Test data

The player cannot move diagonally when it is in contact with the wall. The shooting method works completely fine, the player can shoot in all 8 directions (including diagonally). The collision function works with the player, enemy and the wall. The bullet animation works well.

Post development data

For my next prototype I will be focusing more on creating an enemy AI, which should be able to make intelligent decisions based on the circumstance and try to attack the player. I will work on fixing the diagonal movement issue where the player gets stuck onto the wall.
I will develop a health bar for the player and the enemy so that when the player attacks the enemy or vice versa the health would decrease by displaying it on top of the player and enemy's image.

Development

Testing

Using the link below you will be able to see my testing video in which I will be testing for players and enemies collision with the wall, firing bullets and the player being able to collide with the enemy and vice versa. The enemy disappears from the screen once the enemies health is zero (with 4 bullets) and bullets disappearing when collided with the wall or enemy

Prototype 2 testing video

Review

I believe that this prototype has more functions and classes than prototype 1 now that it contains all the essential functions needed in this project. In my next prototype i will be fixing the same issue that was caused in prototype 1, where the player was not able to move diagonally while colliding with the wall.

The collision function works well between the bullet and enemy / wall. Other collisions such as player colliding with enemy and vice versa also works as it should be with no errors
The map is better than which was in prototype 1, now it is easy to add or delete objects at the position by editing the map text file.
The player and the enemy movement is smooth

Prototype 3

Design

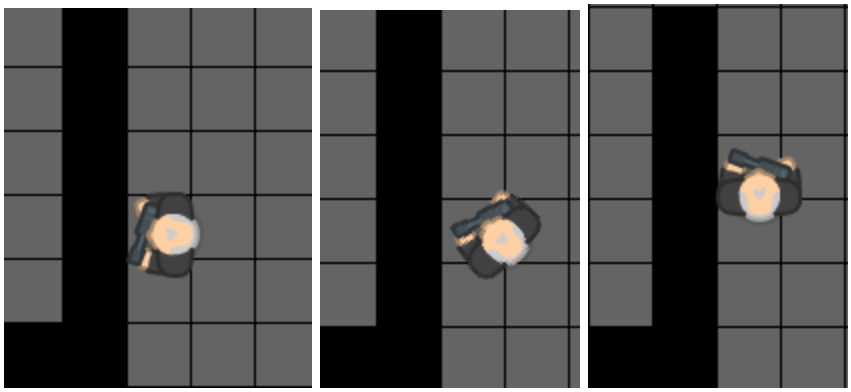
Decomposition

In this prototype I will fix the diagonal collision problem by creating a method to check collision against the wall separately in x-axis (horizontally) and y-axis (vertically) so that it will allow the player to move diagonally while sliding against the wall.

I will create an AI that should be able to follow the player by comparing the position between the player and the enemy.

I will be drawing a health bar at the top of the player's and enemy's image so that it will look attractive and eliminates the need to display the score at the top.

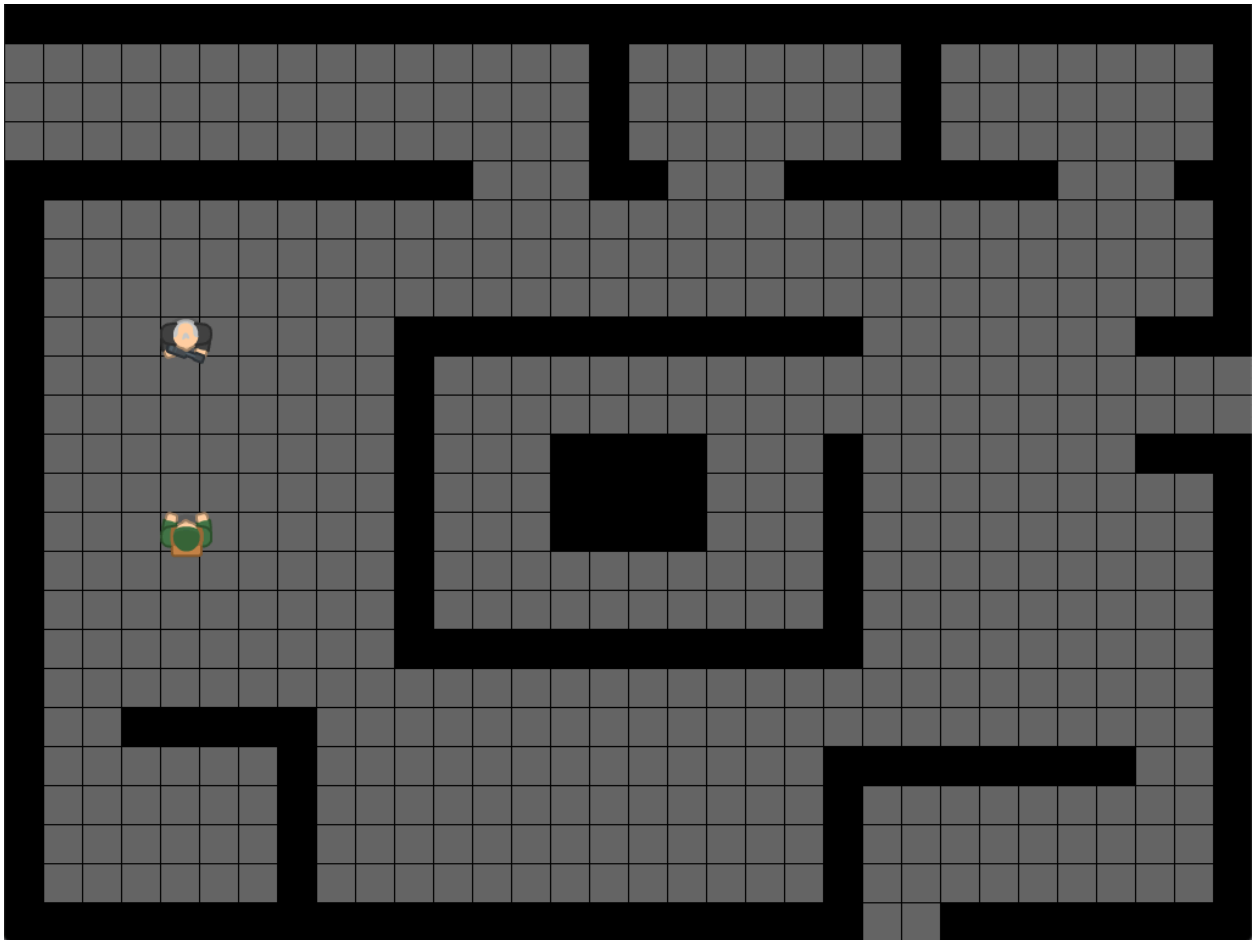
Structure of solution



When the player tries to slide through the wall, let's say if the player is facing left and has collided with the wall, the player will only be able to slide through the wall if the velocity of x is set to zero and velocity of y is a positive / negative number, so whatever the direction in y axis the player wants to go (up / down), the player can slide through the wall facing diagonally.

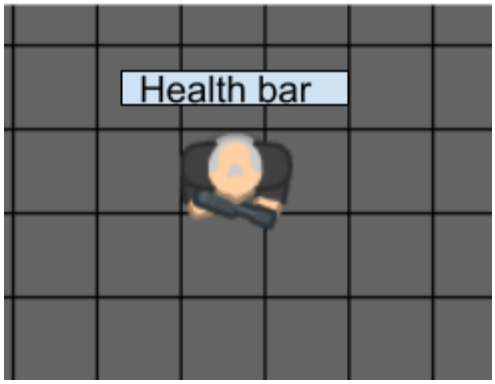
The same principle applies in every direction, So I will be checking for collisions in each direction and if the player has collided with the wall, I will set the velocity of velx / vely that has collided with the wall to zero, so the player won't be able to go through the wall but would be able to slide through the wall when two keys are pressed (diagonal movement)

Enemy AI



In this particular scenario the enemy's y coordinate is 13 and the player's y coordinate is 8, since the enemy is below the player (enemy's y coordinate is greater than player's y coordinate), I will assign the velocity of y for the enemy to -1 (so that the enemy would move up) until it reaches the players coordinate which is 8. So that it will look like the enemy is chasing the player by moving closer to the player. The same principle can be applied in every direction.





For the health bar I will get the current health from the player and then draw a rectangle color coded, green would resemble good health (less than 100), yellow-medium(less than 60), red-critical (less than 30).

Key variables / Data structures / Classes

Updated classes

Player	Enemy	Wall	Map
X Y Rectangle Image Rotation Health Collide_with walls(dir) Move Draw_health Draw	X Y Rectangle Image Rotation Health Collide_with walls(dir) Draw_health Move Draw	X Y Rectangle draw	Filename Tile width Tile height Width Height

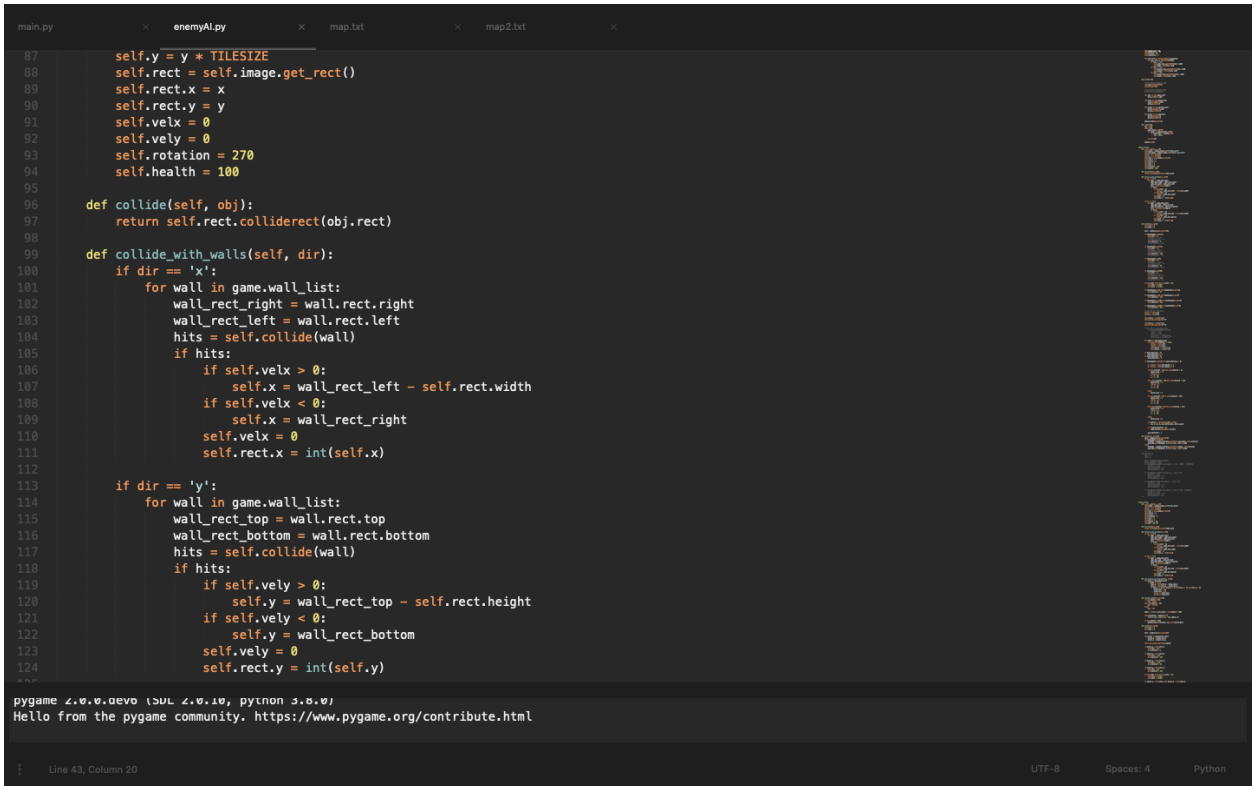
Bullet

X
Y
Radius
Color
Rectangle
Velocity x
Velocity y

Draw
Collide
Move

Define solution fully

Updated code:



```
        self.y = y * TILESIZE
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.velx = 0
        self.vely = 0
        self.rotation = 270
        self.health = 100

    def collide(self, obj):
        return self.rect.colliderect(obj.rect)

    def collide_with_walls(self, dir):
        if dir == 'x':
            for wall in game.wall_list:
                wall_rect_right = wall.rect.right
                wall_rect_left = wall.rect.left
                hits = self.collide(wall)
                if hits:
                    if self.velx > 0:
                        self.x = wall_rect_left - self.rect.width
                    if self.velx < 0:
                        self.x = wall_rect_right
                    self.velx = 0
                    self.rect.x = int(self.x)

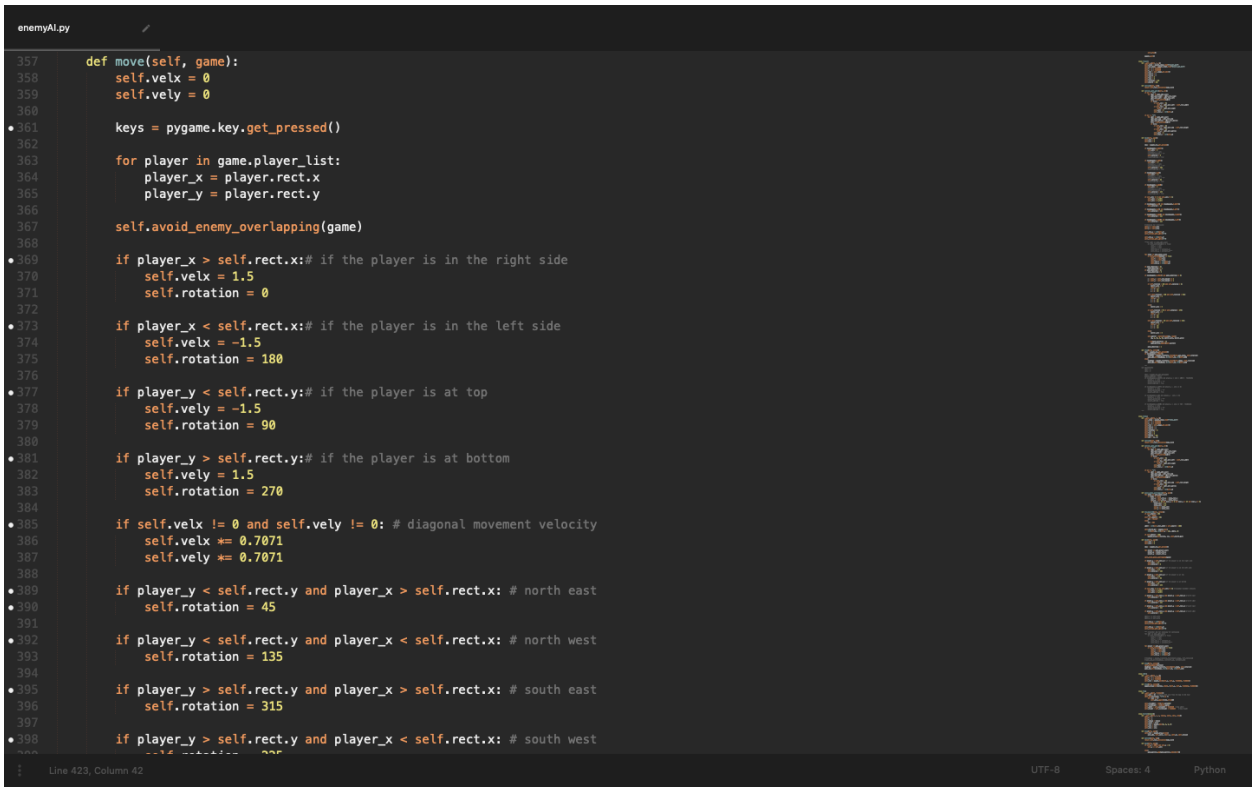
        if dir == 'y':
            for wall in game.wall_list:
                wall_rect_top = wall.rect.top
                wall_rect_bottom = wall.rect.bottom
                hits = self.collide(wall)
                if hits:
                    if self.vely > 0:
                        self.y = wall_rect_top - self.rect.height
                    if self.vely < 0:
                        self.y = wall_rect_bottom
                    self.vely = 0
                    self.rect.y = int(self.y)
```

‘Collide with walls’ method is the newly added code to the player’s and enemy’s class. Firstly it checks if it has collided with the wall, if the ‘collide’ function returns True then, it checks for the

velocity at which it was travelling before colliding with the wall for each x and y directions separately. For example if the velocity at which it was travelling in the x direction (horizontally) is positive then it should have been travelling in the east direction, therefore the player's 'x' position will be assigned to the difference between the left side of the wall and width of the player so that the player will be positioned to the right of the wall.

The velocity in the x axis is set to zero so that the player won't be able to go through the wall or move right in the x direction, even if the user keeps pressing the right key. But the player would be able to move in the y axis with respect to the key the user presses, so this will allow the player to slide up or down through the wall.

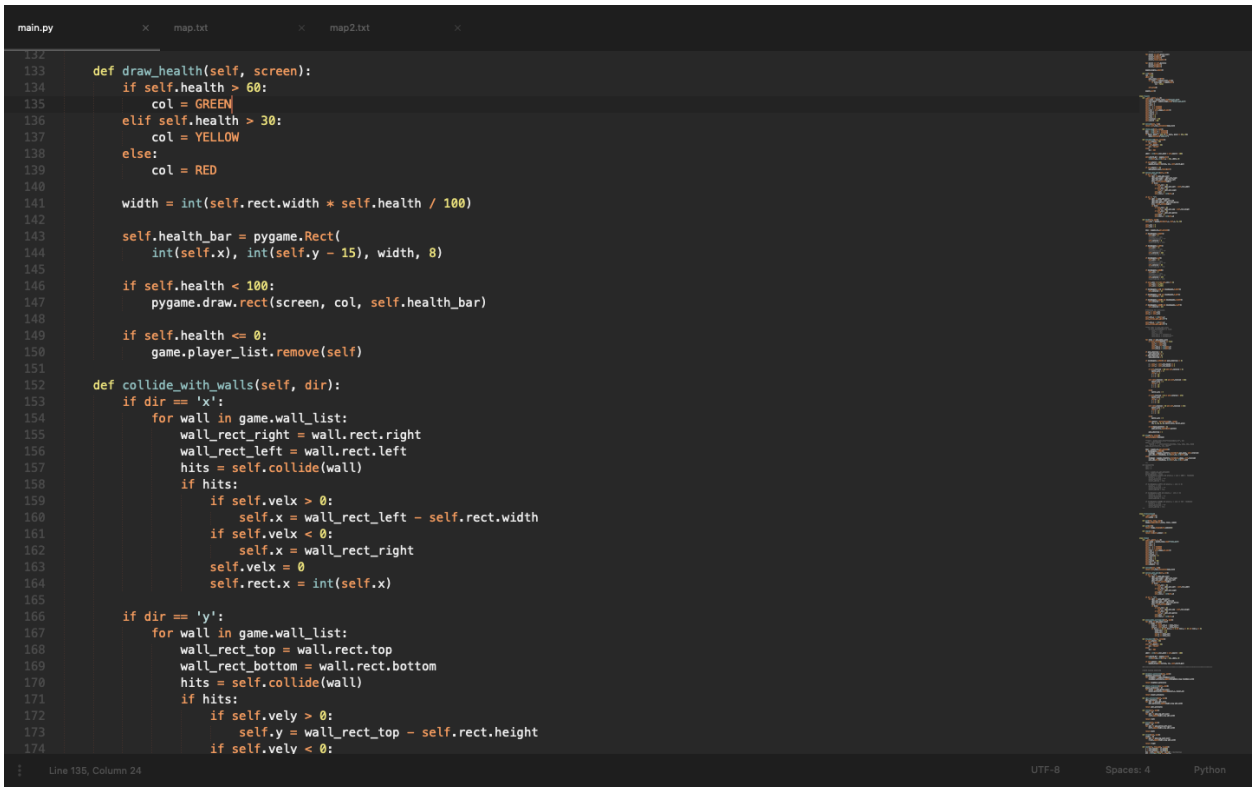
The same principle is applied in the y direction allowing the player to slide left or right through the wall.



This is the 'move' method in 'Enemy' class. When this function is called it checks for the position of the player and the enemy. If the player's x position is greater than the enemy's x position then the player must be to the right of the enemy's x position, therefore I have set the enemy's velocity of x to +1.5 so that it will increase it's x position by 1.5 until it reaches the player's x position.
I have also allocated a rotation value so that when the player is drawn on the screen it will transform the original image (which is pointing east - rotation of 0 degrees) to the desired

direction in which the velocity is travelling i.e north, south, east, west, north east, north west, south east, south west.

The principle is same for other directions too, if the player is left to the enemy then the velocity of x will be -1.5, if the player is at the top of enemy's position then the velocity in y axis is -1.5 and if the player is at the bottom of the players position then the velocity in y axis is 1.5.




```python
    def draw_health(self, screen):
        if self.health > 60:
            col = GREEN
        elif self.health > 30:
            col = YELLOW
        else:
            col = RED

        width = int(self.rect.width * self.health / 100)

        self.health_bar = pygame.Rect(
            int(self.x), int(self.y - 15), width, 8)

        if self.health < 100:
            pygame.draw.rect(screen, col, self.health_bar)

        if self.health <= 0:
            game.player_list.remove(self)

    def collide_with_walls(self, dir):
        if dir == 'x':
            for wall in game.wall_list:
                wall_rect_right = wall.rect.right
                wall_rect_left = wall.rect.left
                hits = self.collide(wall)
                if hits:
                    if self.velx > 0:
                        self.x = wall_rect_left - self.rect.width
                    if self.velx < 0:
                        self.x = wall_rect_right
                    self.velx = 0
                    self.rect.x = int(self.x)

        if dir == 'y':
            for wall in game.wall_list:
                wall_rect_top = wall.rect.top
                wall_rect_bottom = wall.rect.bottom
                hits = self.collide(wall)
                if hits:
                    if self.vely > 0:
                        self.y = wall_rect_top - self.rect.height
                    if self.vely < 0:
```

I have a draw_health function in both players and the enemy's class.When this function is called it checks for the current health of the enemy or the player. Then assigns the colour to the rectangle which will be drawn onto the screen. If the health is greater than 60 the colour will be green, if the health is greater than 30 the color will be yellow and if it is less than 30 it will be red. The width of the health bar is dynamic; it changes its size based on the current health.
If the health goes less than or equal to zero then the player will be removed from the player's list so that it will look like the player died / disappeared from the screen. I have not created an end screen yet but in the future once the player dies there will be an end screen that will be displayed on the screen.

Usability features

The player and enemy has the ability to slide through the wall diagonally
The enemy can move without human interaction (using the AI)
The enemy can chase the player in any direction
The player and the enemy have a health bar at the top of their image which shows their remaining health.

Test data

I have tested the diagonal movement for the player and the enemy in all 8 directions to see if they can slide through the wall without getting stuck to the wall which was an issue from prototype 2. The solution for that problem was to check for collision in each direction individually and set the appropriate velocity of x or y axis to zero. This has solved the problem and now the enemy and player can slide through the wall in any direction.

The AI works as it was supposed to work, the enemy is able to figure out where the player's position is and the enemy tries to move towards the player. I have checked in all directions and it works perfectly fine.

The health bar works well and it has alleviated the appearance of the game. I have checked moving around with the health bar and it always stays above the player / enemy's image.

Post development data

The AI I have created is not intelligent enough. I will be focusing on creating an intelligent AI in my next prototype by programming a path finding algorithm (i.e A start search algorithm). Furthermore I intend to create new maps so that the player will be able to experience different maps and layouts when reaching an end point

Development

Testing

Using the link below you will be able to see my testing video in which I will be testing for the player's and enemy's diagonal movement when colliding with the wall, health bar showing up on the screen and the AI coming after the player

Prototype 3 testing video

Review

In this prototype I have fixed the diagonal movement issue when colliding with the walls that was caused in prototype 2 and made a simple AI so that the enemy would be able to chase the player.
The diagonal movement is now smooth and works perfectly fine when colliding with the wall in any given direction.
The AI however is not intelligent enough to figure out the way to reach the player. As you saw on the testing video when there is a wall between the player and the enemy the enemy comes near the wall and stops because it doesn't know how to reach the player. In my next prototype I will be solving this issue by making the AI more intelligent using a path finding algorithm. (i.e dijkstra, a star search algorithm).

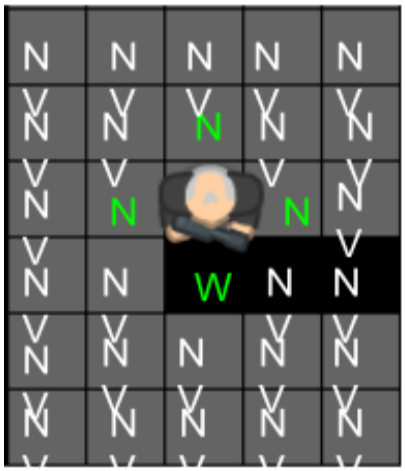
Prototype 4

Design

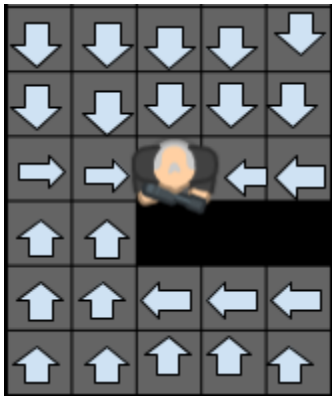
Decomposition

I will code a path finding algorithm for my enemy class so that the enemy would be able to find the shortest path between the player and enemy using a star search algorithm.
To start off I will try to find all the possible paths the enemy could take to reach the player (breadth first search) and then build my way up to reduce the number of searches it takes to get to the enemy by improving the code. (I have broken down the enemy AI pathfinding algorithm into 2 parts; part 1 is to find neighbours and implement breadth first search algorithm, part 2 is to code A* start search algorithm)

Structure of solution

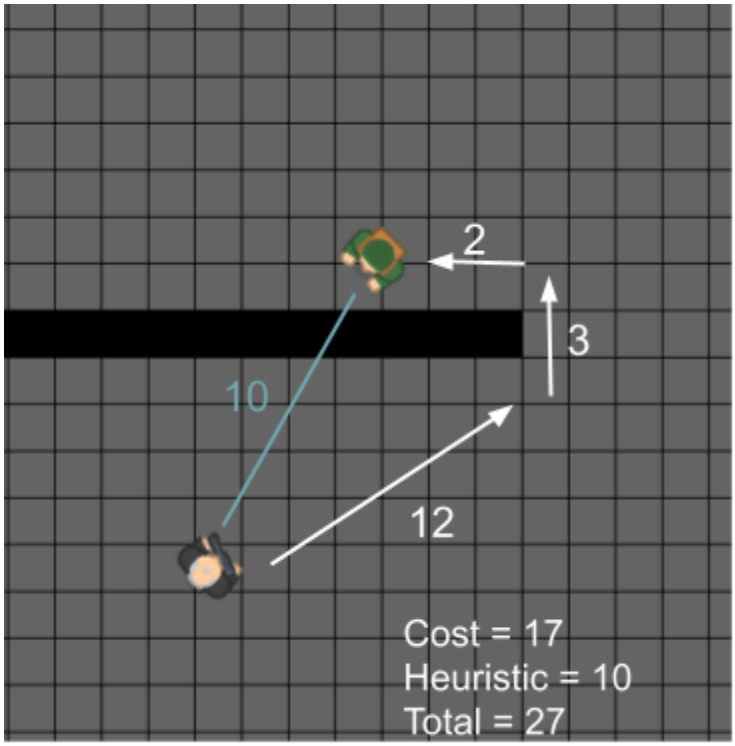


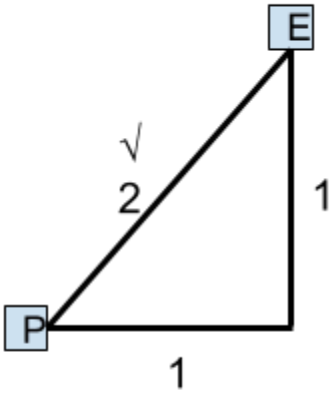
To begin a breadth first search algorithm I will need a frontier, in this case it will be my players position, After that I will have to find the neighbours associated with the frontier. Therefore I will need a method in my enemies class to find all the neighbours for the player and then make them frontiers and find their neighbours and so on excluding the walls. This will give me a neighbor's list with all possible routes the enemy can take
This will allow me to find all the neighbours in the map until I reach the end node



I will draw arrows on the screen to get a visual on how to get from one node to the player. Similar to the image above.

A* star search algorithm



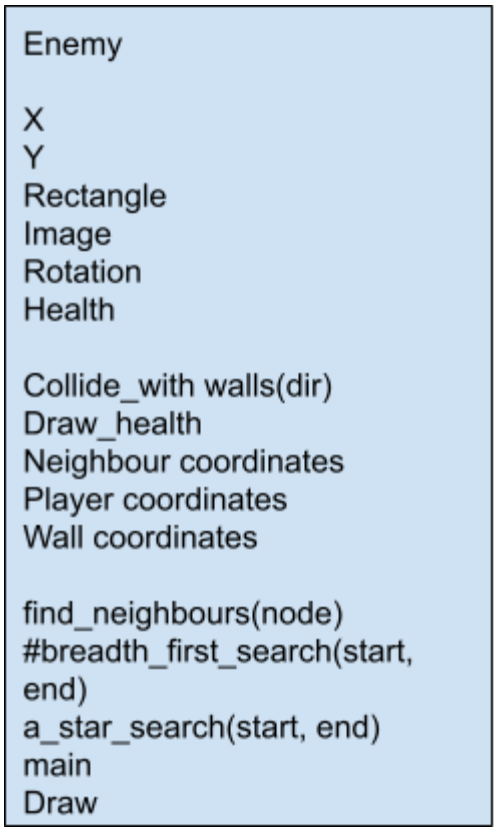


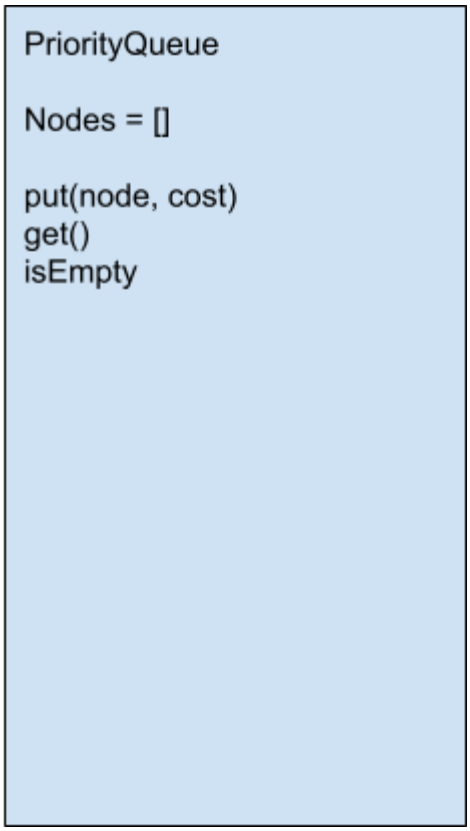
I should be able to get the cost for each movement for example if it is moving 1 tile to the right (12, 5) to (13, 5) then the cost would be 1, but if it has to move diagonally (12, 5) to (13, 6) then the cost would be √2. #####By doing this to each node till I reach the enemy, this would give me the actual cost to go from the player's position to the enemy's position (Cost). The total cost (Total) can be calculated by adding the actual cost (Cost) plus the heuristic which is a positive value that will be returned by subtracting the player's and enemy's coordinates.

In order to reduce the number of searches, heuristic would be extremely useful. This way the algorithm will get an estimate of where the player is and prioritise only the paths that have less total cost and heuristic while ignoring others.

Key variables / Data structures / Classes

Updated / new class





Key variables

Name	Data type	How is it used
neigbour_cordinates	A large 2D array storing the '.' in the map.	Get the neighbours coordinates to allow the searching algorithm to traverse through the map while avoiding walls / obstacles
wall_cordinates	A large 2D array storing each of the wall's position	Used in the searching algorithm to identify walls
frontier	A large 2D array (queue)	Stores every node that the algorithm has taken for the enemy to reach the player
path	A large 2D dictionary that	This is the path used by the

	stores the direction to reach from one node to another	enemy to move from one node to another
arrows	A large 2D dictionary storing the images of the arrow in each direction	Used to blit the arrow on the screen to look at the path the searching algorithm has taken

Define solution fully

```
    # path finding algorithm

    def neighbour_cordinates(self, game):
        neighbour_cordinates = []
        for neighbour in game.neighbours_list:
            neighbour_cordinates.append((neighbour.row, neighbour.col))

        return neighbour_cordinates

    def player_cordinates(self, game):
        player_cordinates = []
        for player in game.player_list:
            player_cordinates.append((player.row, player.col))

        return player_cordinates

    def wall_cordinates(self, game):
        wall_cordinates = []
        for wall in game.wall_list:
            wall_cordinates.append((wall.row, wall.col))

        return wall_cordinates

    def find_neigbours(self, node, game):
        node_x = node[0]
        node_y = node[1]
        right = (node_x + 1, node_y)
        left = (node_x - 1, node_y)
        up = (node_x, node_y - 1)
        down = (node_x, node_y + 1)

        neighbour_cordinates = [right, left, up, down]

        wall_cordinates = self.wall_cordinates(game)

        for i in wall_cordinates:
            wx, wy = i
```

```
pygame 2.0.0.dev6 (SDL 2.0.10, python 3.8.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
(0, 0)
```

Line 470, Column 48 UTF-8 Spaces: 4 Python

```
    def find_neigbours(self, node, game):
        node_x = node[0]
        node_y = node[1]
        right = (node_x + 1, node_y)
        left = (node_x - 1, node_y)
        up = (node_x, node_y - 1)
        down = (node_x, node_y + 1)

        neighbour_cordinates = [right, left, up, down]

        wall_cordinates = self.wall_cordinates(game)

        for i in wall_cordinates:
            wx, wy = i

            if wx == up[0] and wy == up[1]:
                neighbour_cordinates.remove(up)

            elif wx == down[0] and wy == down[1]:
                neighbour_cordinates.remove(down)

            elif wx == right[0] and wy == right[1]:
                neighbour_cordinates.remove(right)

            elif wx == left[0] and wy == left[1]:
                neighbour_cordinates.remove(left)

        return neighbour_cordinates

    def breadth_first_search(self, start, end, game):
        frontier = deque()
        frontier.append(start)
        path = {}
        path[start] = None

        while len(frontier) > 0:
            current = frontier.popleft()
            if current == end:
```

```
pygame 2.0.0.dev6 (SDL 2.0.10, python 3.8.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
(0, 0)
```

Line 470, Column 48 UTF-8

I have defined neighbour coordinates, player coordinates and wall coordinates so that in the future it'll be easy to get each of their coordinates.

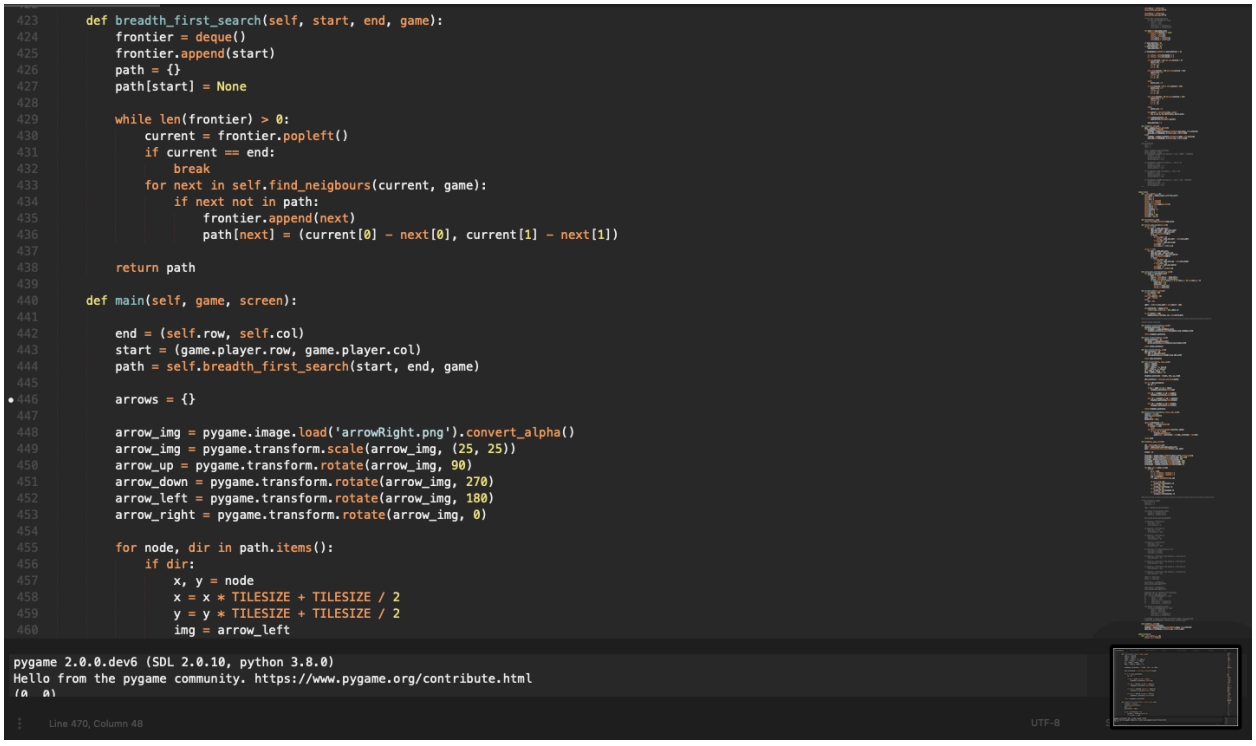
To find the neighbours of a node I have defined a method ‘find_neighbours’ which takes in an argument ‘node’.

I have assigned ‘node_x’ to the ‘x’ position of the ‘node’ and ‘node_y’ to the ‘y’ position of the ‘node’. Frontier would be (node_x, node_y).

To find the neighbours for frontier I have added 1 to node_x to find the the coordinates to the right of the frontier, deleted 1 in node_x to find left of the frontier, added 1 in node_y to find coordinate below frontier and deleted 1 in node_y to find the top coordinate.

I created a neighbour coordinates list to store all the neighbours we have found from above to that particular node / frontier.

In order to find the coordinates in which the player could travel, that is excluding the wall, I found all the wall coordinates from the function that I defined previously and then checked x and y coordinates of the wall against our neighbour coordinates (up, down, left and right) to see if it matches the wall coordinates, if it did match then I removed that coordinate from the neighbour coordinates list hence eliminating all the walls in neighbour coordinates .



```
    def breadth_first_search(self, start, end, game):
        frontier = deque()
        frontier.append(start)
        path = {}
        path[start] = None

        while len(frontier) > 0:
            current = frontier.popleft()
            if current == end:
                break
            for next in self.find_neigbours(current, game):
                if next not in path:
                    frontier.append(next)
                    path[next] = (current[0] - next[0], current[1] - next[1])

        return path

    def main(self, game, screen):

        end = (self.row, self.col)
        start = (game.player.row, game.player.col)
        path = self.breadth_first_search(start, end, game)

        arrows = {}

        arrow_img = pygame.image.load('arrowRight.png').convert_alpha()
        arrow_img = pygame.transform.scale(arrow_img, (25, 25))
        arrow_up = pygame.transform.rotate(arrow_img, 90)
        arrow_down = pygame.transform.rotate(arrow_img, 270)
        arrow_left = pygame.transform.rotate(arrow_img, 180)
        arrow_right = pygame.transform.rotate(arrow_img, 0)

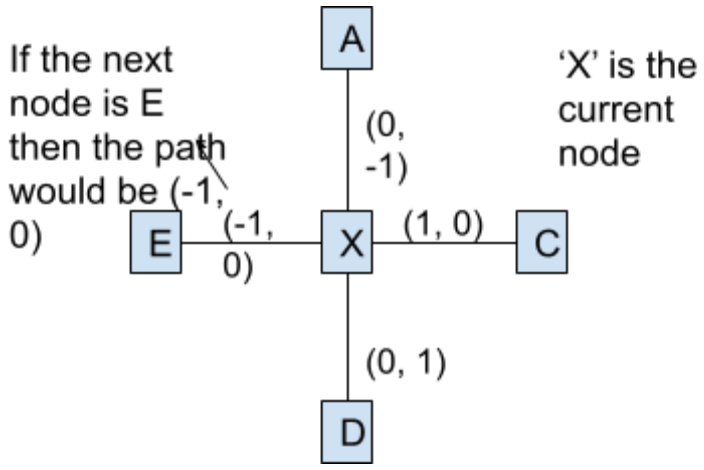
        for node, dir in path.items():
            if dir:
                x, y = node
                x = x * TILESIZE + TILESIZE / 2
                y = y * TILESIZE + TILESIZE / 2
                img = arrow_left
```

```
pygame 2.0.0.dev6 (SDL 2.0.10, python 3.8.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

Since a regular queue made using a list is slow when it re-indexes each item in the list while removing or appending, I decided to go with deque from collections module, double-ended

queue which only focuses on the item at each end which is perfect for our searching algorithm because we are bothered by only finding the the item at the start and end.

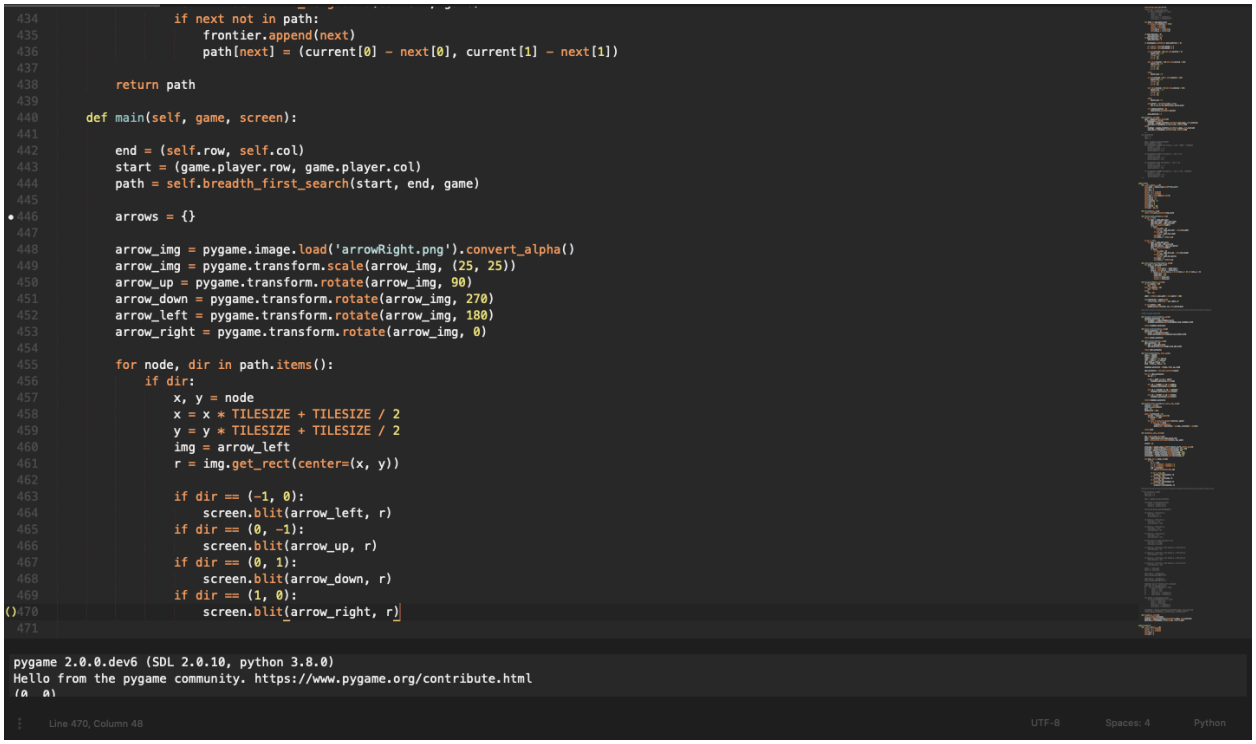
I have made the variable ‘frontier’ a double-ended queue, and appended my ‘start’ node into it. I created a path dictionary to keep track on how the node went from one node to another.



As long as there are nodes inside the frontier, I have set the ‘current’ variable equal to the first node from the frontier queue, If the current node is equal to the end node then the algorithm must have found a path so it will break from the while loop and return the path, if not it will look at the neighbours of current node by implementing the ‘find_neighbours’ functions to find all the the neighbours of that node and append them to the ‘path’ dictionary.

To keep record on how it traversed through the nodes the ‘path[next]’ will return one of the following for each node:
(-1, 0) - from right
(1, 0) - from left
(0, 1) - from up
(0, -1) - from down

It will then return the path



```
                if next not in path:
                    frontier.append(next)
                    path[next] = (current[0] - next[0], current[1] - next[1])

        return path

    def main(self, game, screen):

        end = (self.row, self.col)
        start = (game.player.row, game.player.col)
        path = self.breadth_first_search(start, end, game)

        arrows = {}

        arrow_img = pygame.image.load('arrowRight.png').convert_alpha()
        arrow_img = pygame.transform.scale(arrow_img, (25, 25))
        arrow_up = pygame.transform.rotate(arrow_img, 90)
        arrow_down = pygame.transform.rotate(arrow_img, 270)
        arrow_left = pygame.transform.rotate(arrow_img, 180)
        arrow_right = pygame.transform.rotate(arrow_img, 0)

        for node, dir in path.items():
            if dir:
                x, y = node
                x = x * TILESIZE + TILESIZE / 2
                y = y * TILESIZE + TILESIZE / 2
                img = arrow_left
                r = img.get_rect(center=(x, y))

                if dir == (-1, 0):
                    screen.blit(arrow_left, r)
                if dir == (0, -1):
                    screen.blit(arrow_up, r)
                if dir == (0, 1):
                    screen.blit(arrow_down, r)
                if dir == (1, 0):
                    screen.blit(arrow_right, r)
```

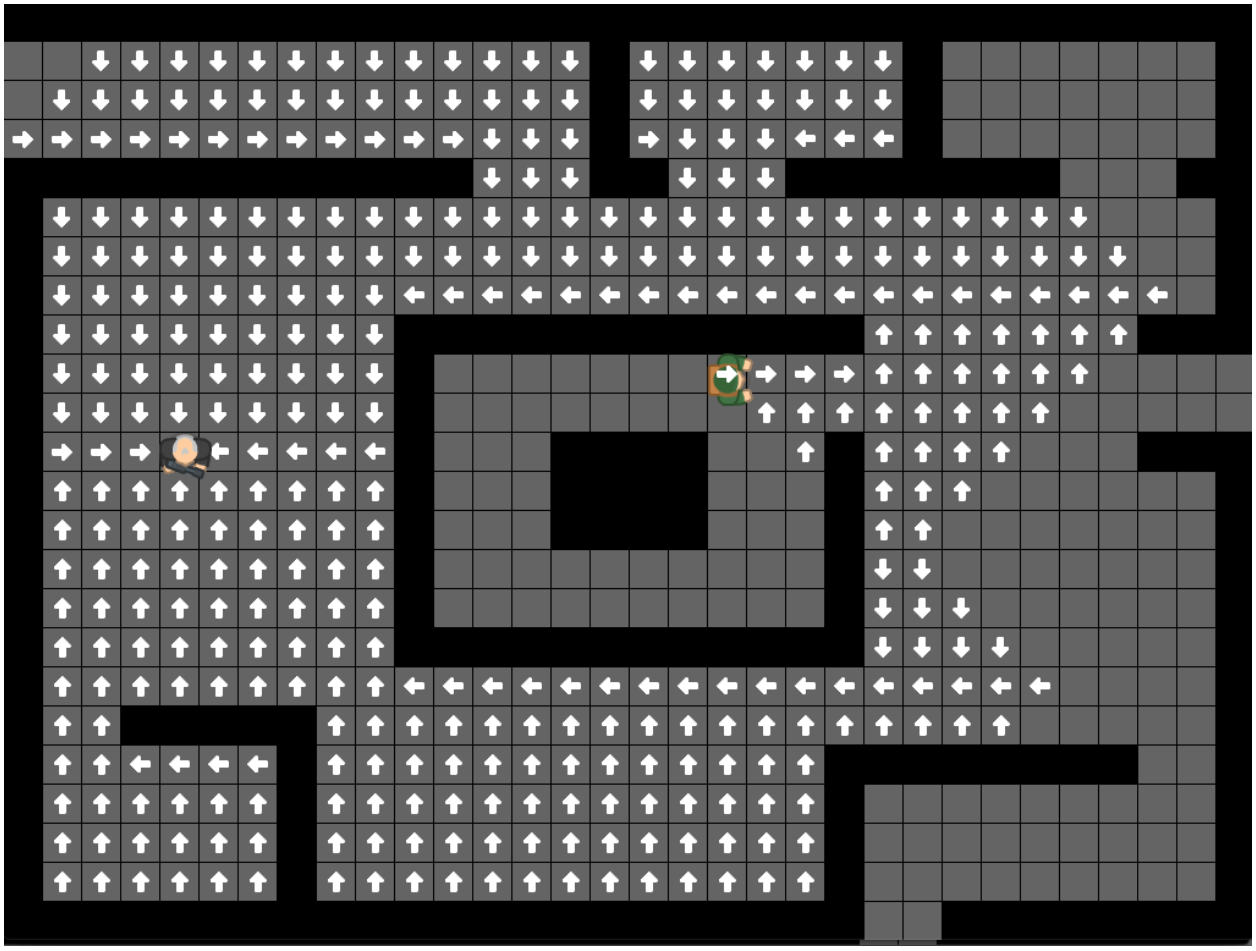
```
pygame 2.0.0.dev6 (SDL 2.0.10, python 3.8.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

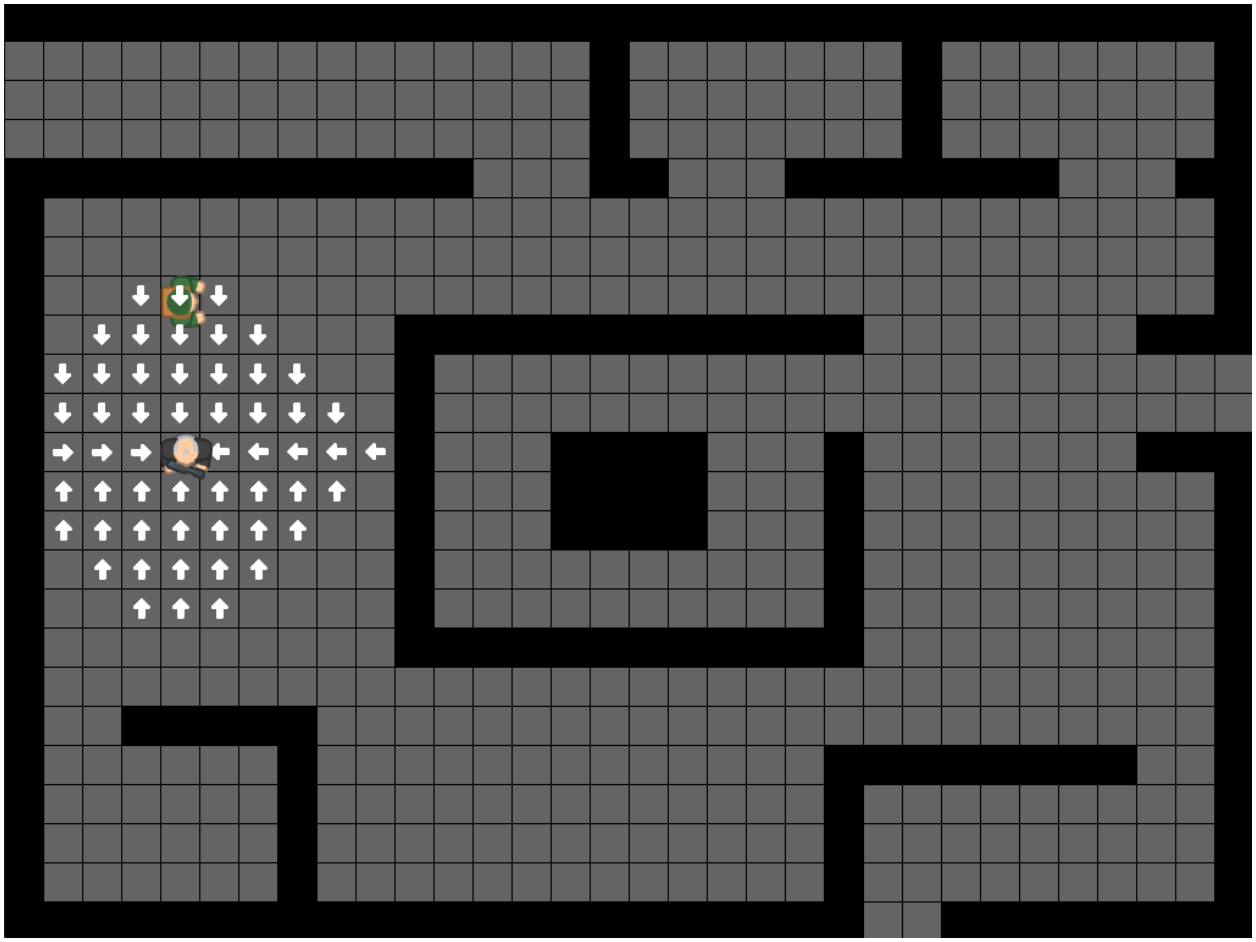
This is my main function which brings all the other methods together in my enemy class. I have assigned my end node to be the enemy's current position, and start node to the player's current position, so that when I run 'breadth_first_search' algorithm it would start finding neighbours from the start node and figure it's way till the end node.

To visually look at how the algorithm found it's path I got the 'node' and 'dir' from the 'path' dictionary I defined at the start, which would return the node and direction to reach the end node / enemy.

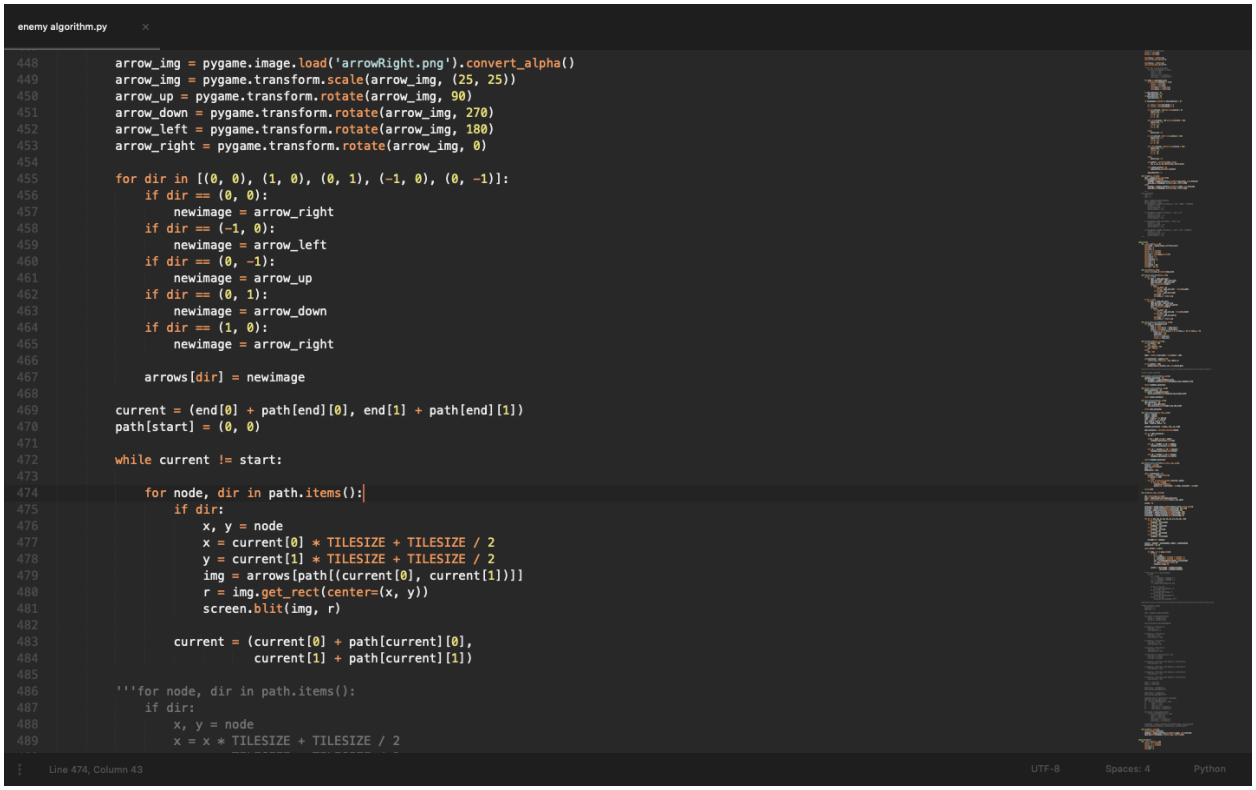
If the direction is (-1, 0), it means that the path is from right to left (west), if direction is (1, 0), it means that the path is from left to right (east), if direction is (0, 1), it means that the path is from top to bottom (south) and if direction is (0, -1) it means that the path is from bottom to top (north).

I have transformed my arrow image according to the direction it has come from i.e if it is (-1, 0) then the arrow should point to left, (1, 0) - arrow should point to right, (0, 1) - arrow should point down and (0, -1) - arrow should point up.





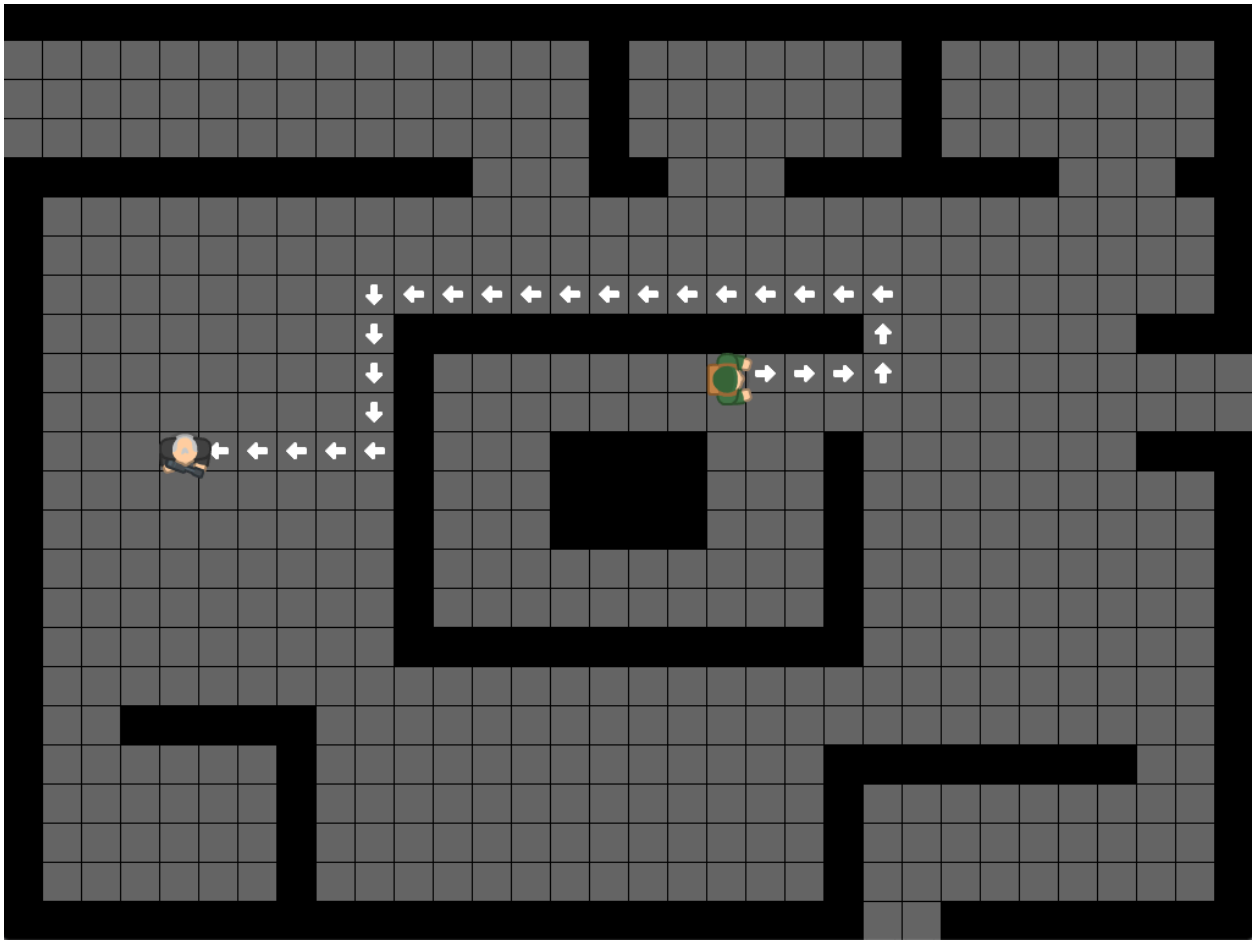
The images from above shows the path it takes for the enemy to get to the player while avoiding the walls.



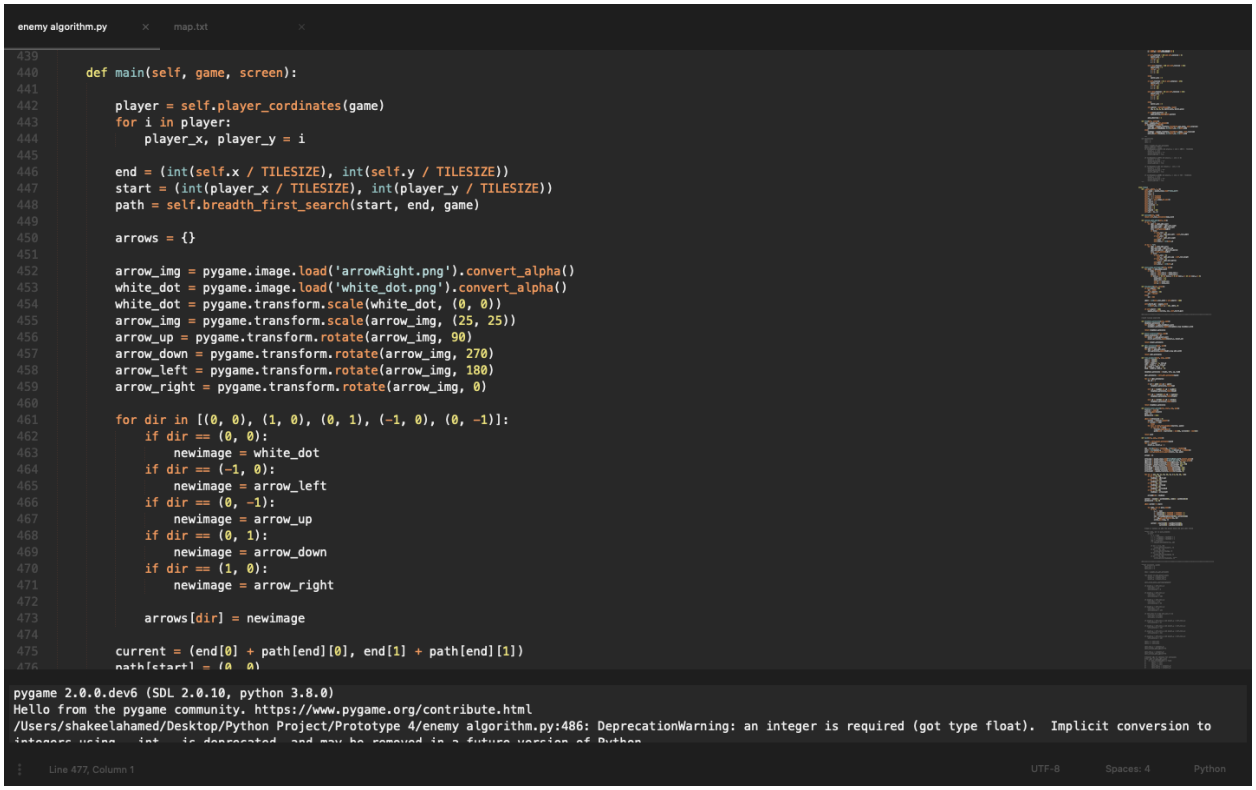
To see only one path from start node to end node I have modified the code, ###I have appended the arrow images to the path node based on the direction each node has.

As long as my current node hasn't reached the end node, I have blitted each arrow on the x and y position of that 'node' by finding the direction for each node and returning their corresponding path e.g. (5, 4):(1, 0) - At (5,4) on the screen there will be an arrow facing east

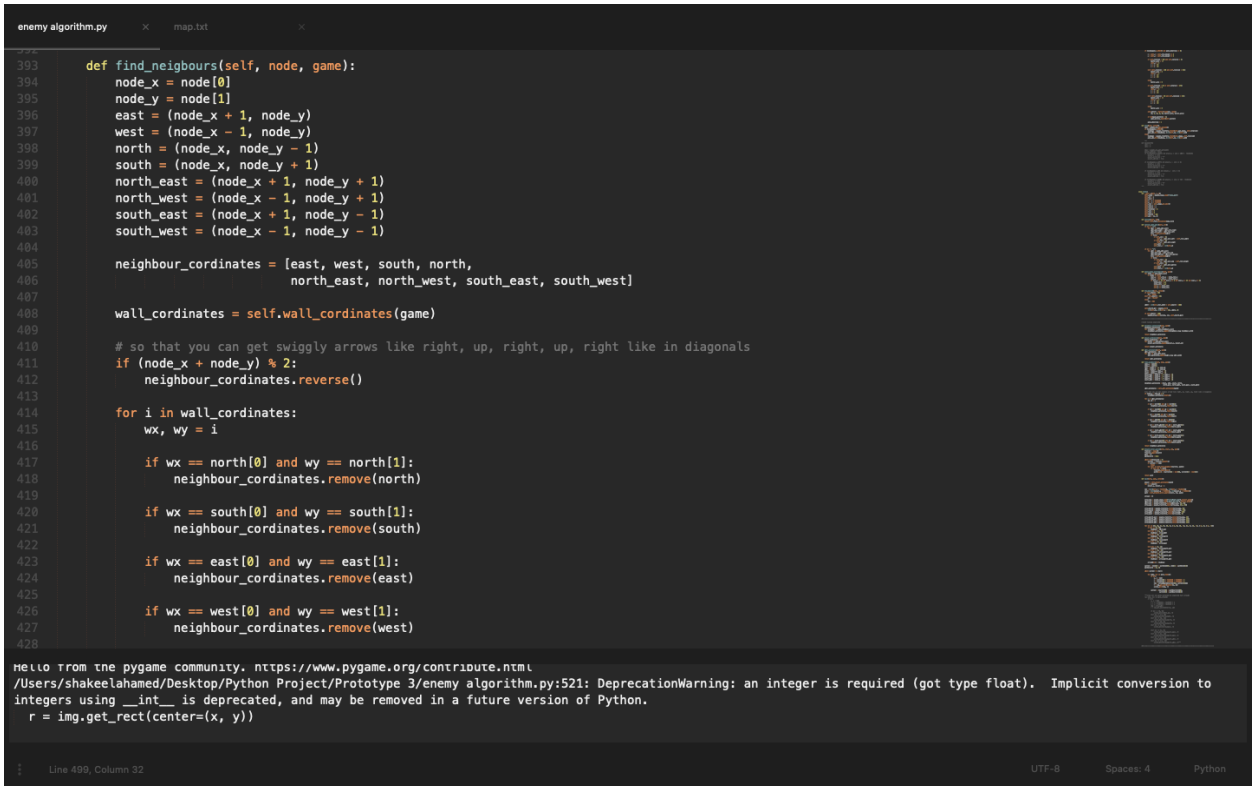
For the next 'current' node I have incremented it by identifying the path for the next node and adding it into the current node. If the current node is (10, 5) and path for the current node is (-1, 0) then the current node becomes (9, 5), then it will get it's path and so on until it reaches the end node.

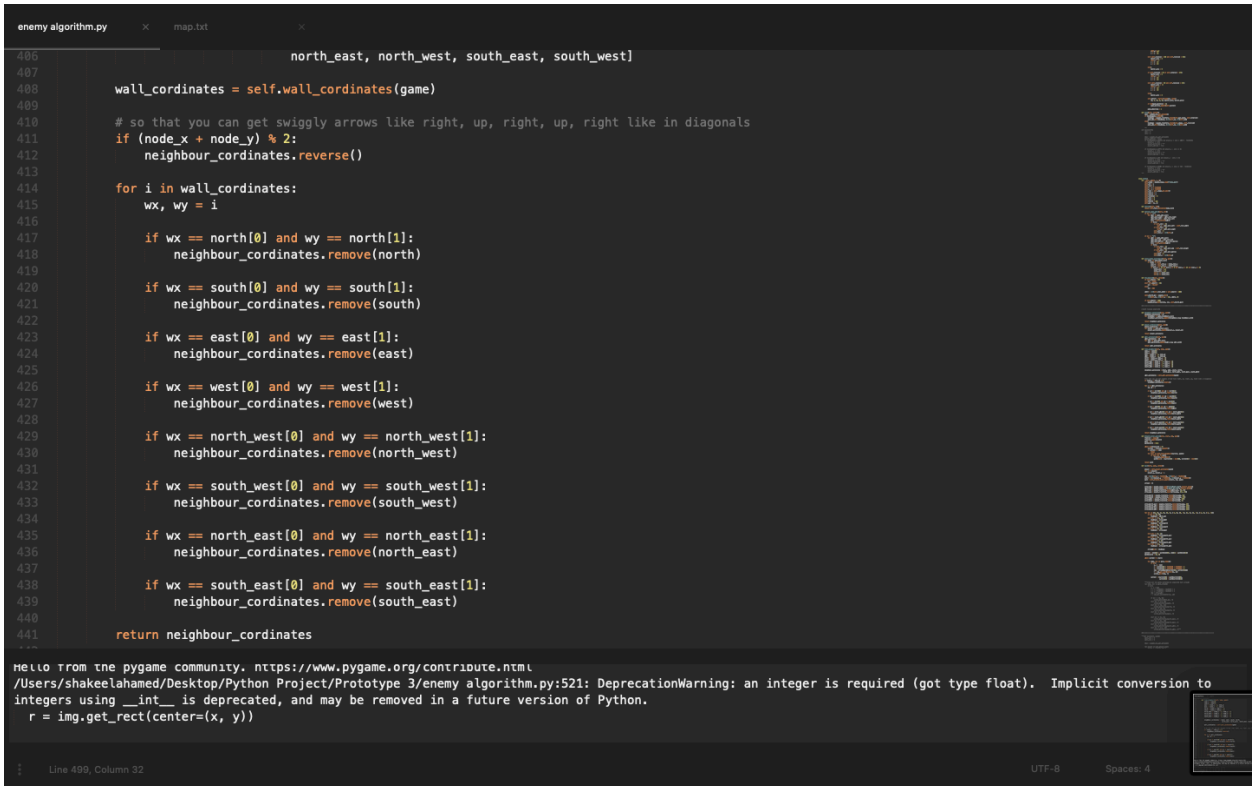


The image above shows a possible shortest path that the enemy could take to reach the player



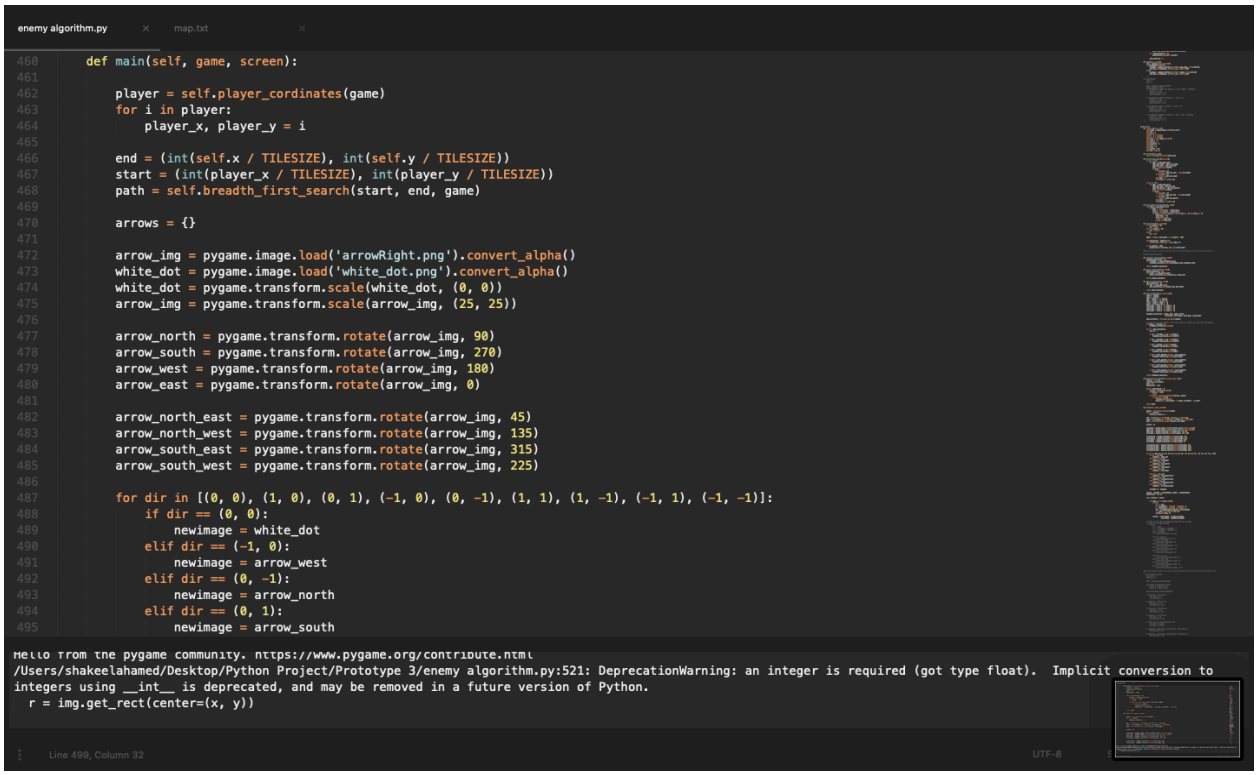
The path would change each time the player or enemy moves from one place to another, I have assigned the ‘start’ node to the player’s current position and the ‘end’ node to the enemy’s current position.





Since from the beginning of the project I wanted for the player and enemy to be able to move diagonally, I modified the ‘find_neighbours’ function to be able to find neighbours diagonally i.e

north east, north west, south east and south west. I appended them to the neighbour coordinates list. After getting all the coordinates in all 8 directions for each node, I looped through all the wall coordinates and then removed the neighbour coordinates if it matched with one of the wall coordinates. So that the path finding algorithm won't try to go through a wall.



enemy algorithm.py map.txt

```
        for dir in [(0, 0), (1, 0), (0, 1), (-1, 0), (0, -1), (1, 1), (1, -1), (-1, 1), (-1, -1)]:
            if dir == (0, 0):
                newimage = white_dot
            elif dir == (-1, 0):
                newimage = arrow_west
            elif dir == (0, -1):
                newimage = arrow_north
            elif dir == (0, 1):
                newimage = arrow_south
            elif dir == (1, 0):
                newimage = arrow_east

            elif dir == (1, 1):
                newimage = arrow_south_east
            elif dir == (1, -1):
                newimage = arrow_north_east
            elif dir == (-1, 1):
                newimage = arrow_south_west
            elif dir == (-1, -1):
                newimage = arrow_north_west

            arrows[dir] = newimage

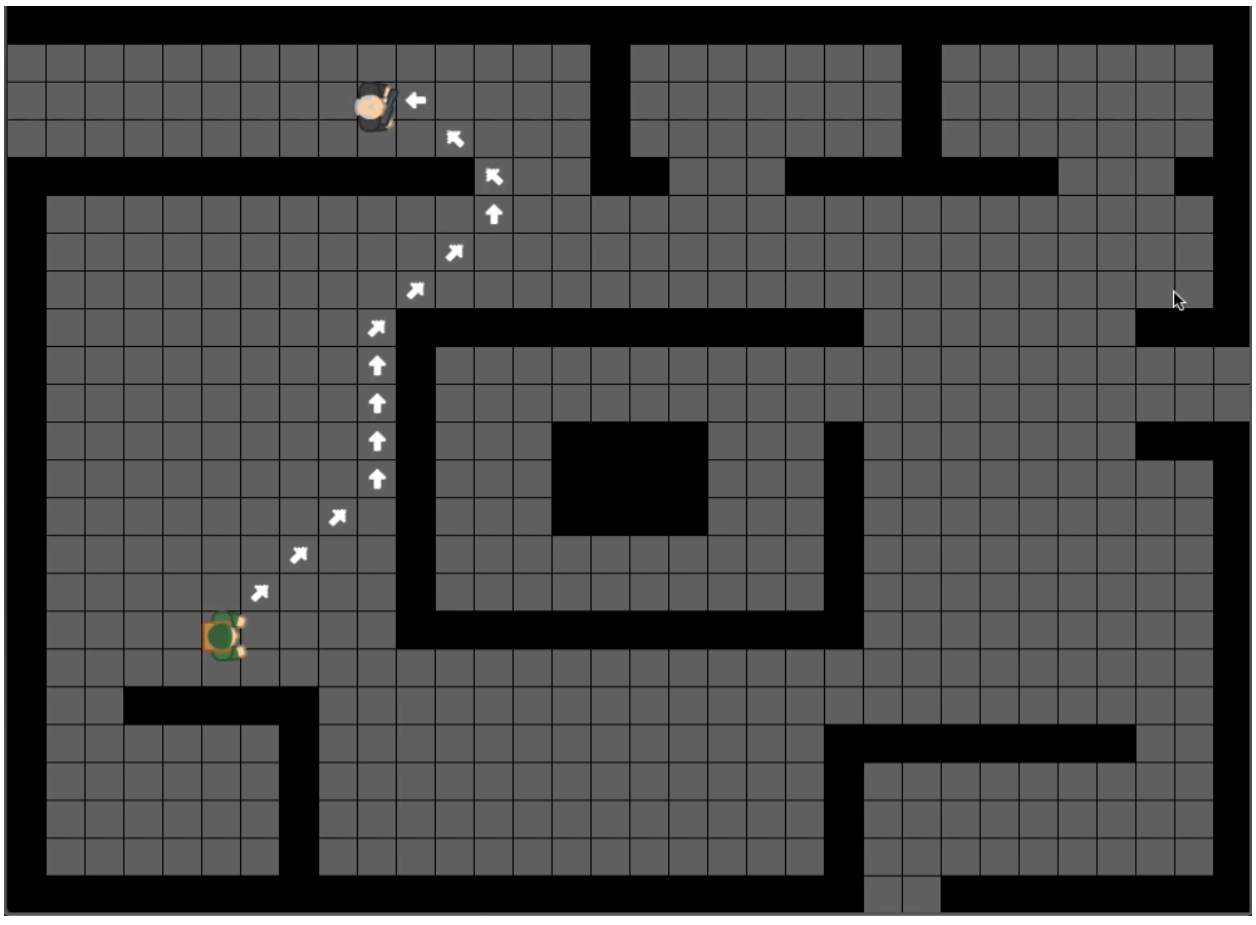
        current = (end[0] + path[end][0], end[1] + path[end][1])
        path[start] = (0, 0)

        while current != start:

            for node, dir in path.items():
                if dir:
                    x, y = node
                    x = current[0] * TILESIZE + TILESIZE / 2
                    y = current[1] * TILESIZE + TILESIZE / 2
                    img = arrows[path[(current[0], current[1])]]
                    r = img.get_rect(center=(x, y))
                    screen.blit(img, r)
```

```
Hello from the pygame community. https://www.pygame.org/contribute.html
/Users/shakeelahamed/Desktop/Python Project/Prototype 3/enemy algorithm.py:521: DeprecationWarning: an integer is required (got type float).  Implicit conversion to
integers using __int__ is deprecated, and may be removed in a future version of Python.
  r = img.get_rect(center=(x, y))
```

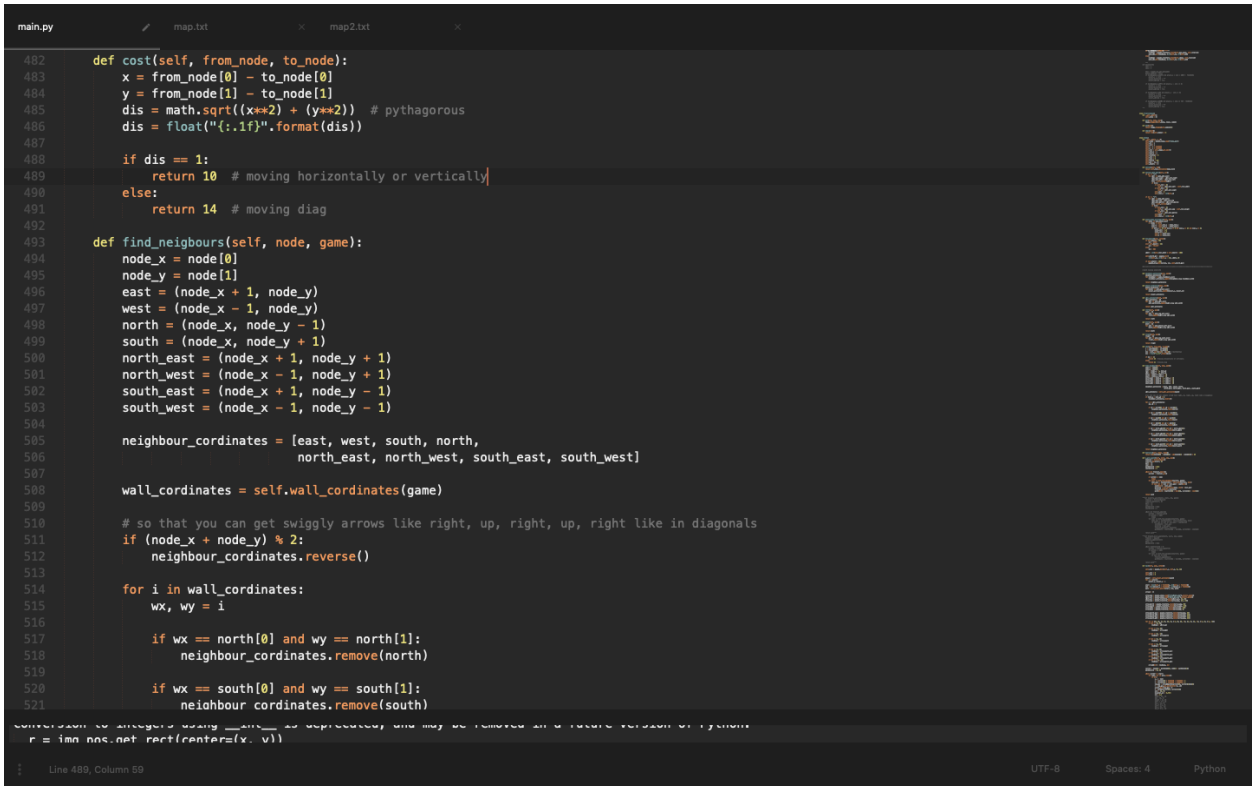
Line 499, Column 32 UTF-8



Updated the main function from enemy class, so that the arrow image would transform its rotation in all 8 directions (north, south, east, west, north east, north west, south east, south west)

#==========

Breadth first search is inefficient and slows down the computer when there are too many obstacles like the walls all around the map. The player rapidly slows down when I move away from the enemy since ‘breadth first search’ considers every ‘node’ in the map, therefore it is not intelligent. The solution to this problem is to introduce ‘A star search algorithm’ which will take into account the actual cost and the heuristic which will reduce the number of searches the algorithm performs.



I have introduced a new function ‘cost’ to find the cost it it takes it takes to go from one node to another node, I have used pythagoras theorem to find out if the next node is in a straight line (horizontally or vertically) or if the next node is in a diagonal line (y = x, y = -x).

If the modulus of the distance is 1 then it means that it is travelling in x or y axis / horizontally or vertically therefore then the cost of it would be 1 (I multiplied the cost by 10 just to avoid decimals) therefore it would return 10.
If the modulus of distance isn't one then it must be moving diagonally therefore the cost would return 14. (1.4 * 10).

djikstra.py map.txt

```
296
297 class PriorityQueue:
298     def __init__(self):
299         self.nodes = []
300
301     def put(self, node, cost):
302         heapq.heappush(self.nodes, (cost, node))
303
304     def get(self):
305         return heapq.heappop(self.nodes)[1]
306
307     def empty(self):
308         return len(self.nodes) == 0
309
310
311 class Enemy:
312     def __init__(self, x, y):
313         self.image = pygame.image.load('enemy.png')
314         self.col = y
315         self.row = x
316         self.x = x * TILESIZE
317         self.y = y * TILESIZE
318         self.rect = self.image.get_rect()
319         self.rect.x = x
320         self.rect.y = y
321         self.rotation = 0
322         self.velx = 0
323         self.vely = 0
324         self.health = 100
325         self.pos = (x, y)
326         self.weights = {}
327
328     def collide(self, obj):
329         return self.rect.colliderect(obj.rect)
330
331     def collide_with_walls(self, dir):
332         if dir == 'x':
333             for wall in game.wall_list:
334                 wall_rect_right = wall.rect.right
335                 wall_rect_left = wall.rect.left
```

```
Hello from the pygame community. https://www.pygame.org/contribute.html
```

Line 308, Column 36 UTF-8 Spaces: 4 Python

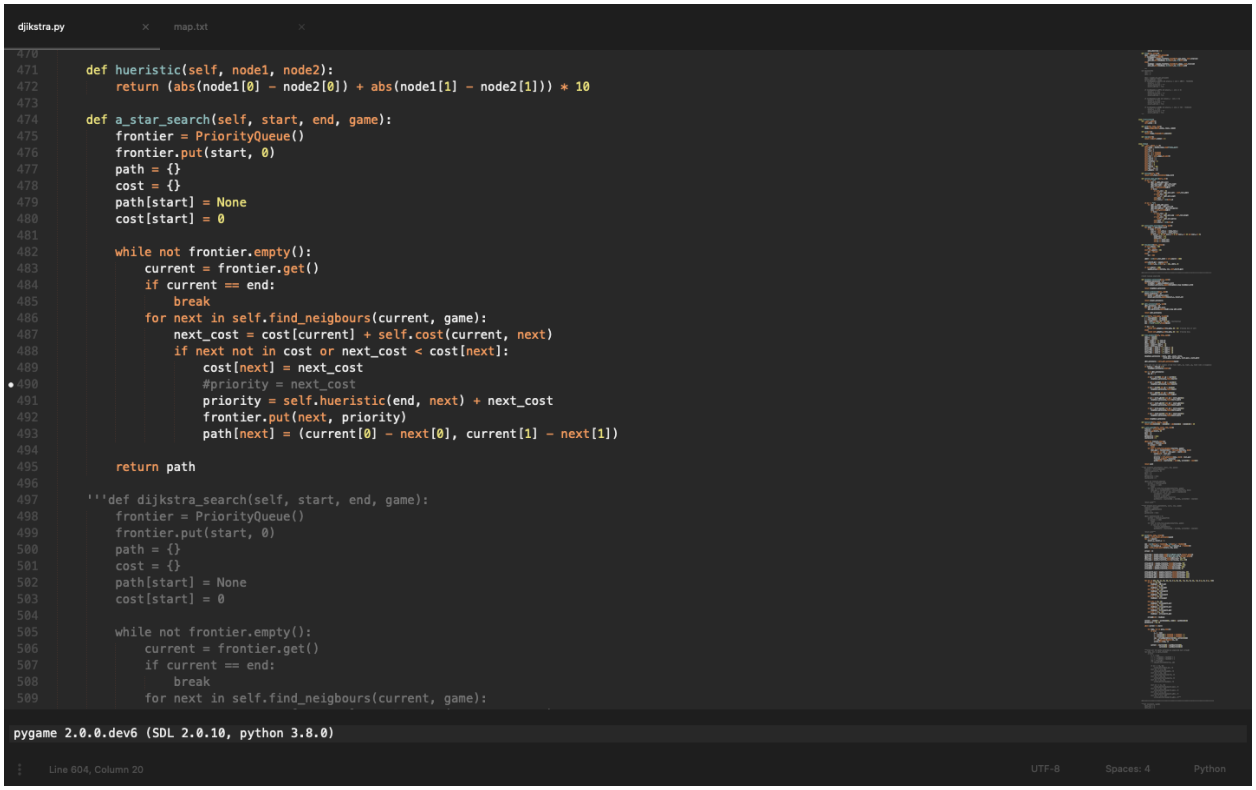
I have imported the heapq module from python library, which is a type of list / queue which allocates a priority for each item in the list (ordered by priority). When I pull off an item from heapq it would be the one with the highest priority out of all the others.

I created a 'PriorityQueue' class which has a 'nodes' list containing all the nodes and their corresponding weight / cost. In the 'put' function I am able to push / append 'node' and 'cost' onto the 'nodes' list..

The 'nodes' list will be ordered according to the priority of cost in ascending order. The highest priority is the lowest cost.

In the 'get' function, it will return / pop the node which has the highest priority / lowest cost.

‘Empty’ function checks if the list is empty by finding out the length of the list. If the length is zero then the list must be empty.



I defined a function for heuristic in my enemy’s class to find an estimate that takes to travel from one node to another. To avoid decimals I multiplied it by 10 like I did previously. In the ‘heuristic’ function, it takes two arguments ‘node 1’ and ‘node 2’, I then calculate the heuristic value by using the ‘abs’ function which only returns the positive value of a number, to find the difference between node 1 and node 2, and adding them up together.

(Node 1 ‘X ‘ coordinate - Node 2 ‘X’ coordinate) will give me a positive number
(Node 1 ‘Y’ coordinate - Node 2 ‘Y’ coordinate) will also be a positive since I’m using absolute value.
The ‘heuristic’ value would then be the addition of the difference between them.

A star search algorithm is similar to breadth first search algorithm but the number of searches carried out is rapidly decreased.

This time my frontier will be the ‘PriorityQueue’ I made previously, I have appended the start node into its frontier with a cost of 0.

I created a ‘path’ dictionary like before to keep track of the direction for each node, so that we can show the path on the screen later.

The ‘cost’ dictionary will keep track of the cost it takes to move from one node to another for each node.

while the frontier is not empty, the ‘current’ node will be the lowest cost we get from the priority queue ‘get method’. If ‘current’ node is equal to the end node it means that we have reached a feasible solution and therefore it will break from the while loop and return the path

If not then we loop through all the neighbours of the current node with ‘find_neighbours(current)’ function and then find the next_cost, the next_cost is whatever the current cost plus the cost it takes to move to the next tile.

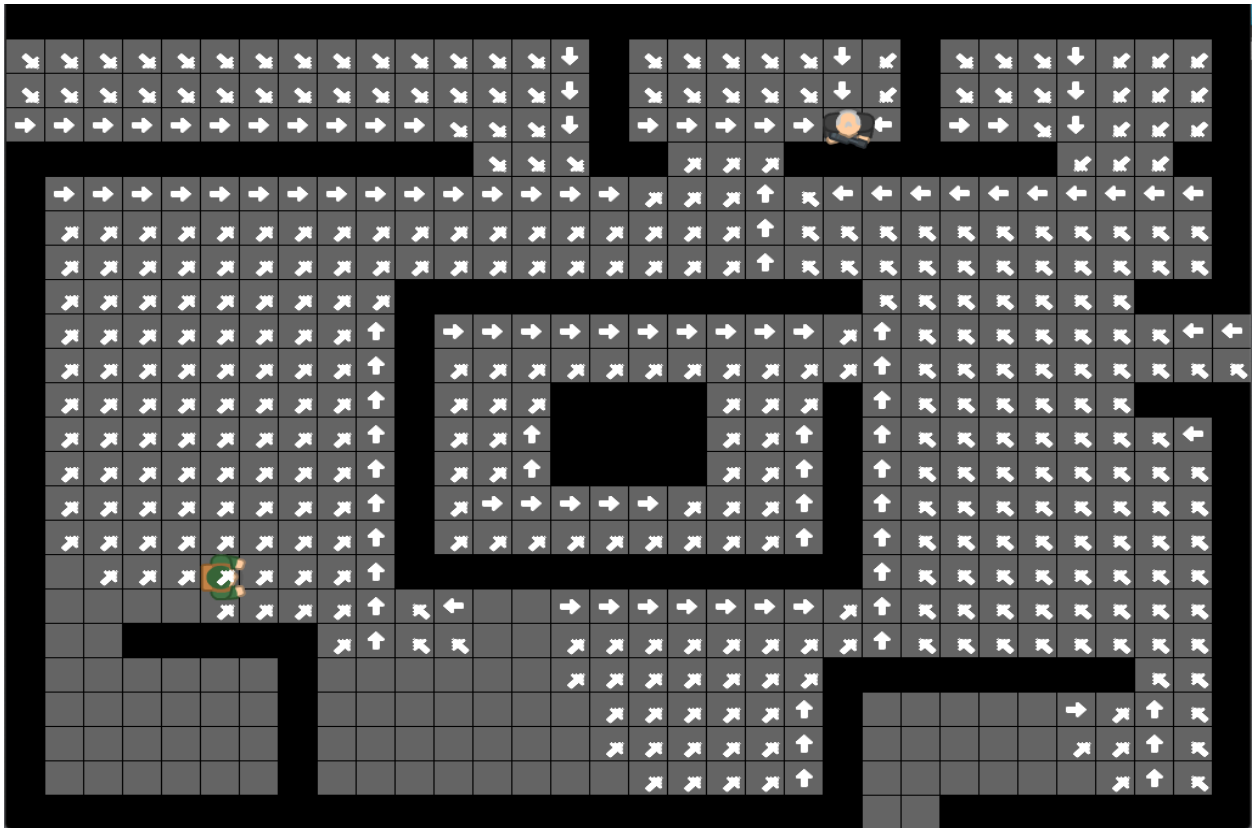
If the nodes ‘next_cost’ is not in the dictionary or if we have found a next_cost which is less than the cost we have in the dictionary then the cost will be appended to the cost dictionary.

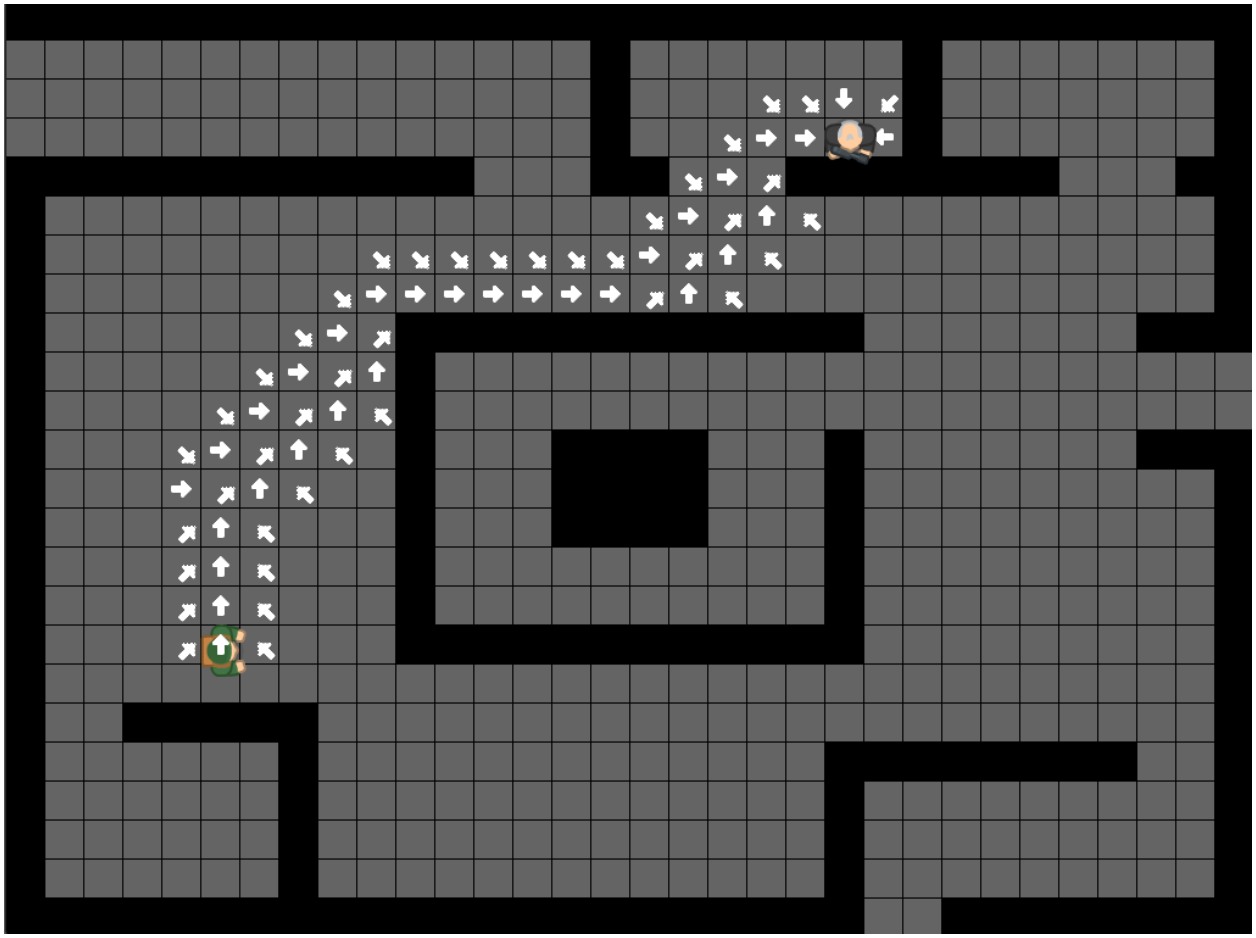
The ‘priority’ variable is the addition of the ‘next_cost’ plus the heuristic value between the ‘end’ node and the ‘next’ node.

I have then appended the ‘node’ and the ‘priority’ (cost) to the frontier list which will order them by priority

I will then append the path for the current node by finding the difference between the current node and the node it is going to go next. I.e if the current node is (1, 2) and the next node is (2, 2), then the ‘path’ would ((2 - 1), (2 - 2)) = (1, 0)

After returning the path I have updated the ‘path’ variable to ‘a_star_search’ in my ‘main’ function in ‘Enemy’ class.





From the images above we can compare the difference between a star search algorithm and breadth first search algorithm, the number of searches the algorithm has done has rapidly decreased.

Usability features

The enemy can now find the shortest path to the player by avoiding the walls.

Test data

I have tested the path it makes by moving the player and enemy to different places in the map e.g. behind the wall. The algorithm always finds its path from the enemy to the player (avoiding the walls).

The ‘a star search’ algorithm works perfectly fine and always tries to find the best possible shortest path

However when there is a complex map with a lot of walls, the algorithm slows down a bit, but still it finds its way to the player.

When the player moves far away from the enemy, the player slows down since the algorithm takes a lot of time to figure out its path as the map expands. Therefore closer the enemy, faster the algorithm finds its path.

Post development data

I am satisfied with the current algorithm, since this is far more better than the AI in prototype 4.
For my next prototype I will work on the movement of enemy, allowing it to follow the arrows to reach the player
I will create new maps so that when the player reaches an end point, the player should be able to move to the next map.
I will work on modifying the map and adding inventories such as health.

Development

Testing

Using the links below you would be able to see both breadth first search algorithm and a star search algorithm, in which I will go around the walls to look at the path the algorithm makes from the enemy to reach the player.

Breadth first search testing video

A star search algorithm testing video

Review

In this prototype I have programmed a pathfinding algorithm for the enemy to find the shortest path to the player without going through walls.

Initially I programmed a breadth first search algorithm to find all the possible paths and then improved the algorithm to reduce the number of searches using ‘a star search’ algorithm by finding ‘cost’ and ‘heuristic’ values.

No matter where the player is, the enemy can now find a feasible path to reach the player.
In my next prototype I will program the movement of the enemy so that it can follow the arrow to reach the player.

Prototype 5

Design

Decomposition

In this prototype I will work on allowing the enemy to move by following the path returned by ‘a star search’ algorithm. For example if the path returns (-1, 0) that means that the enemy should travel in the west direction by doing this to all the paths returned by the algorithm this will allow the enemy to move in all eight directions.

I will be working on my map, to make it look better. I will be using an application called ‘Tiled’ map editor which is an open source application. I will download sprite sheets from the website named ‘Kenney’ which has no copyright restrictions and free to use.

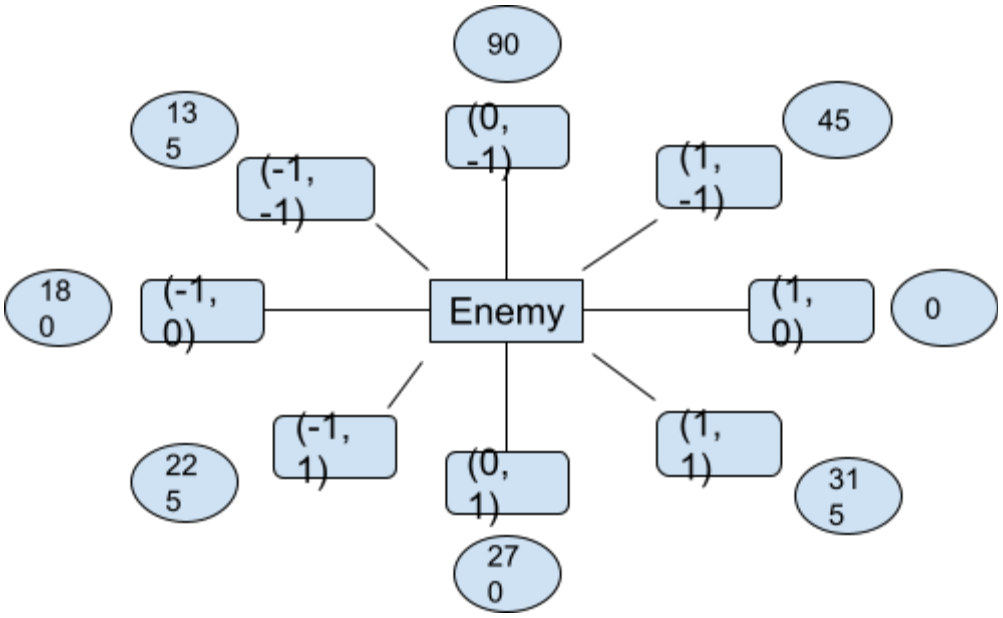
I will be working on making the player move to a different map when the player reaches some end point.

I will add a health inventory so that the player can reach out to the health ‘object’ and increase the health by a fraction.

I will be making a start screen and an end screen for the game.

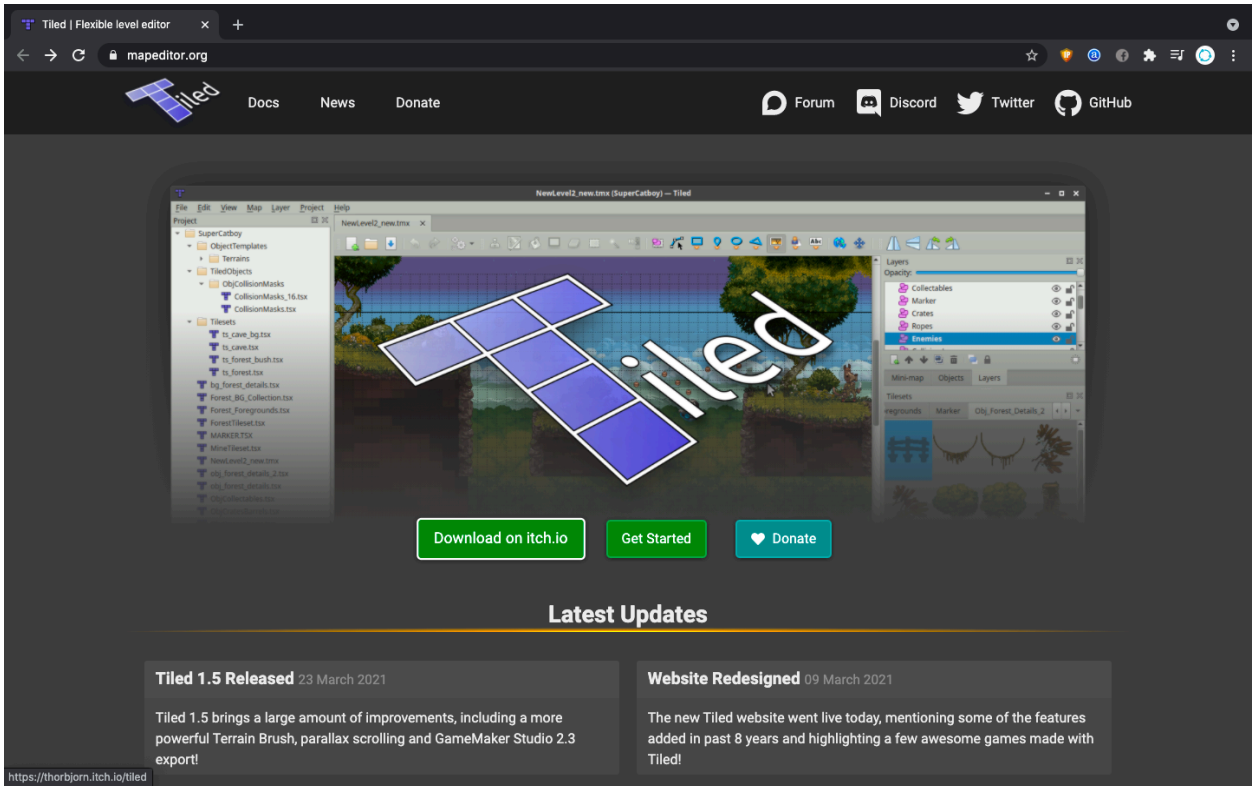
I will also be working on options such as pause, save and load game.

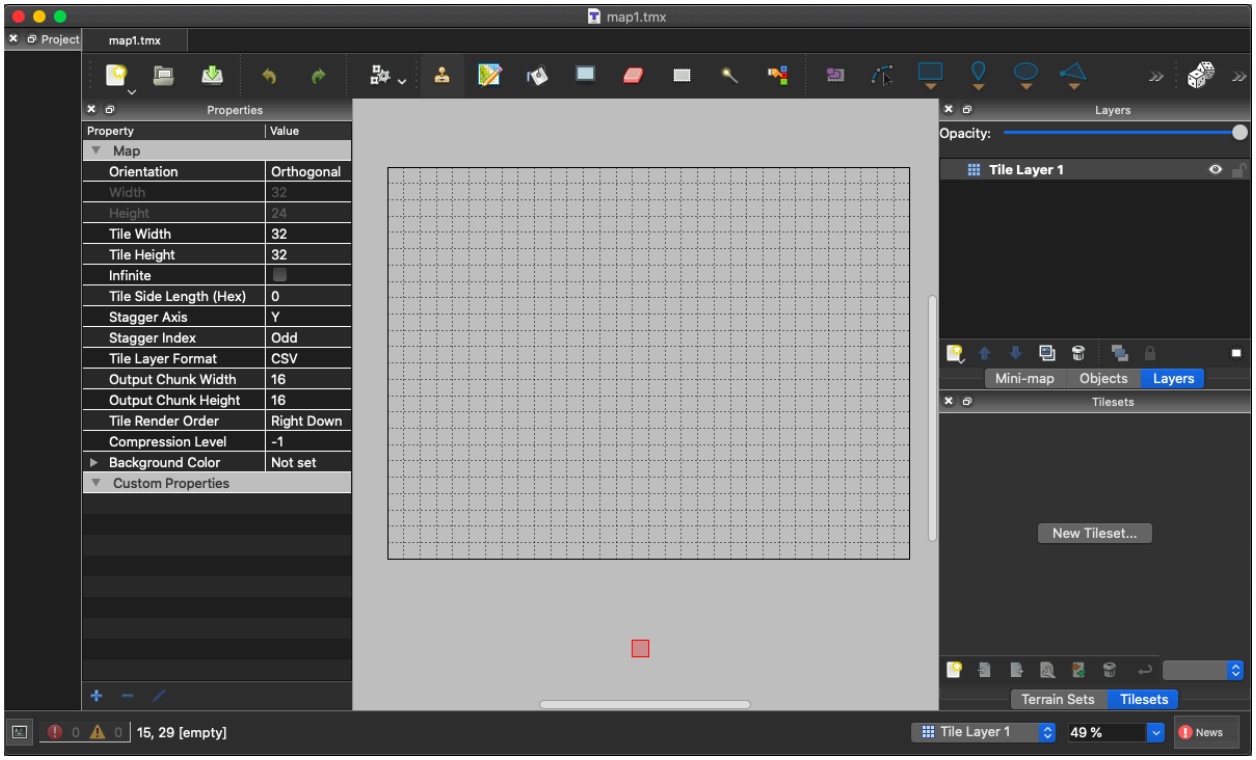
Structure of solution



The algorithm will decide which path to take out of all these 8 directions. If the direction is (0, -1) then I will set the velocity of y to -1 and the rotation of the player's image to 90 degrees, if the direction is (1, 0) I will set the velocity of x to 1 and the rotation to 0 and so on...

For diagonal movement if the direction is (1, -1) I will set the velocity of x to √2 / 2 and velocity of y to -√2 / 2 and so on… The value √2 is from Pythagoras theorem, I have divided it by 2 (√2 / 2) since it takes two steps to move diagonally (i.e. right and then up) which will give me a speed of 0.7071.. So that when the enemy moves diagonally it would not look like it is moving too fast or too slow.





This website provides an application that helps us to draw items/objects on the map very easily and efficiently rather than manually typing ‘special numbers / letters’ on the map. In my previous

map I had 1's and .'s. 1's will make a wall, .'s for empty space. P to create an instance of player and E to create an instance of enemy.

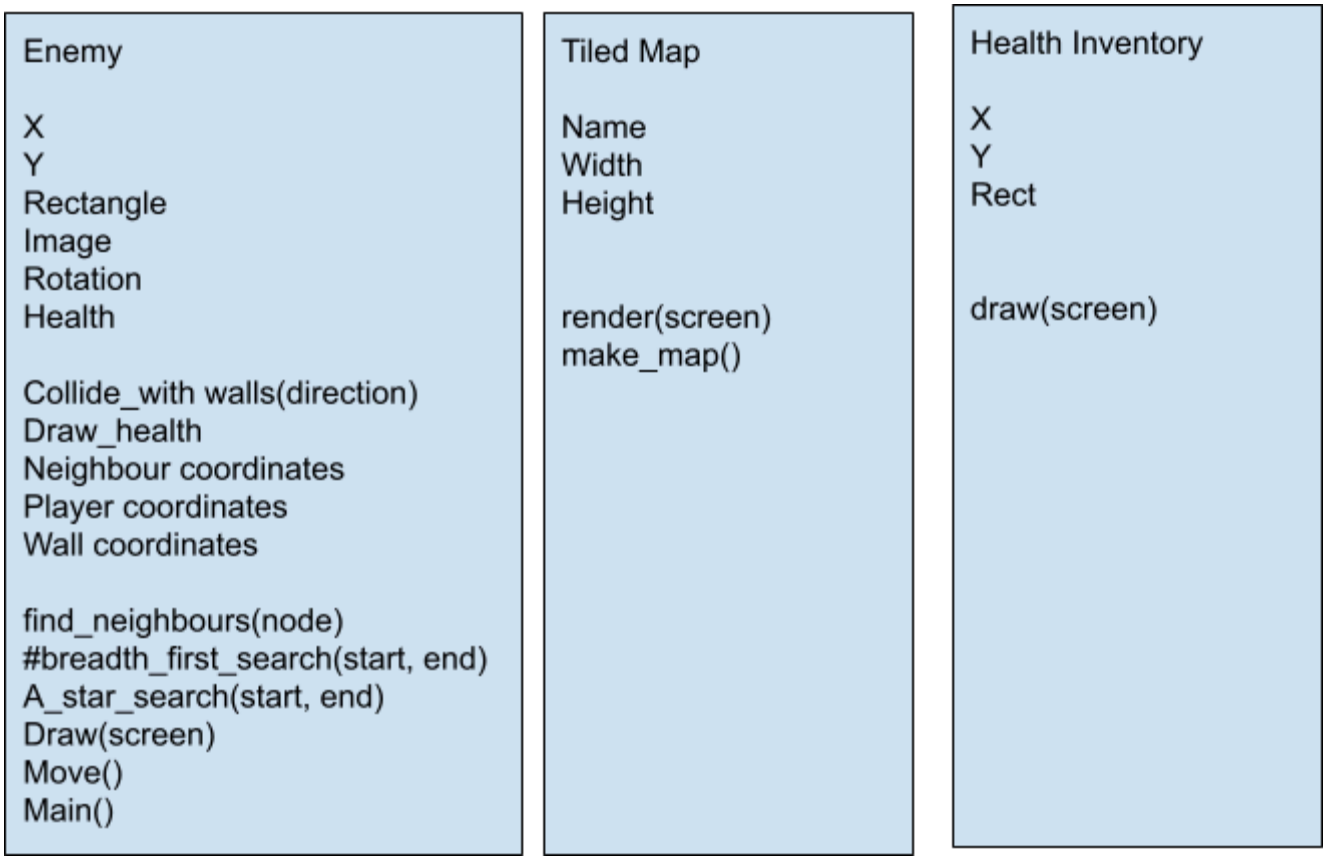
To create a new map in the 'Tiled' application, I just have to enter the tile width, tile height and the number of tiles I need (horizontally and vertically).

In order to work with this application, I need to import pytmx library which is a map loader for python/pygame for games development. It provides smart tile loading with a fast and efficient storage base.

I will create a new class for each inventory so that I can make instances of it later when moving to different maps.

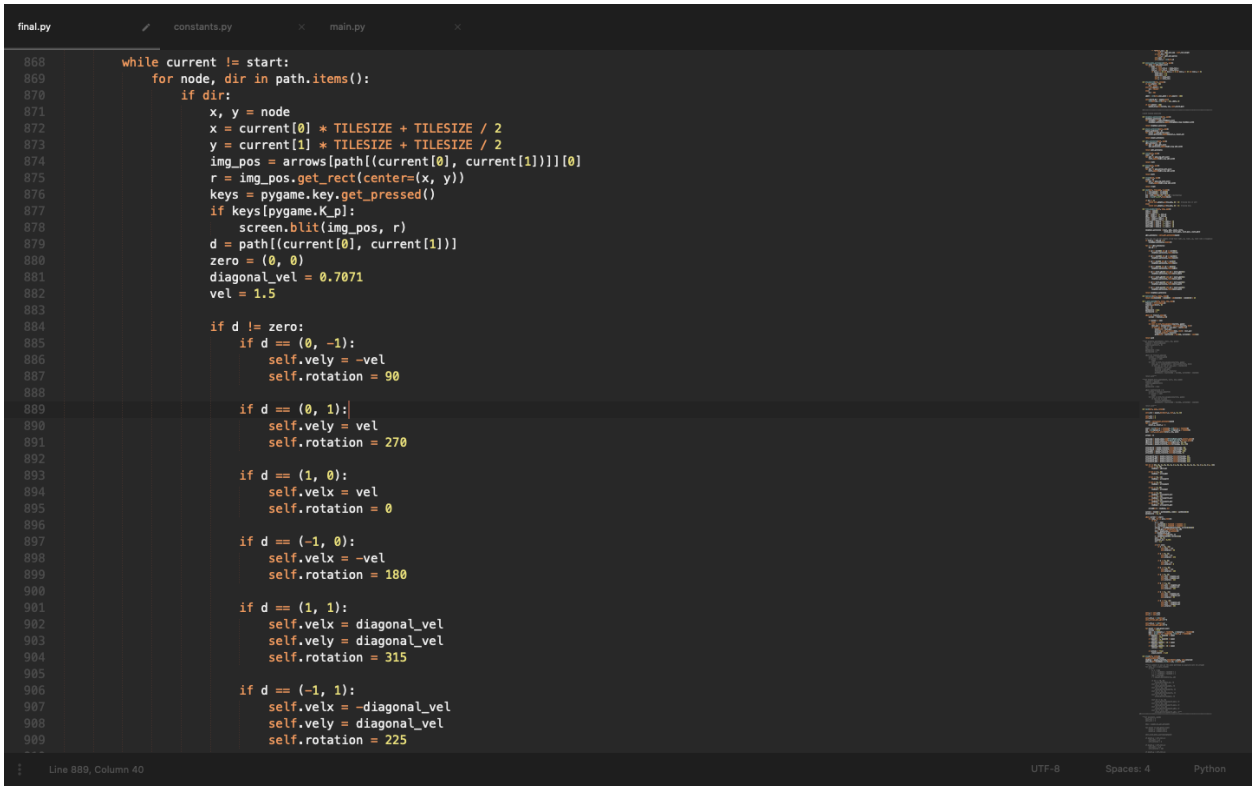
Key variables / Data structures / Classes

Updated classes



Input	Process	Output
Start new game	Trigger the load data function in game class to create new instances for each class	New game with default players, enemy's, walls, map will be spawned on the screen
Load game	Get the data from the python text file and make instances of each class for that saved data	The game will continue from the place where the user left off
Pause, enter button	When the pause icon is clicked it triggers the pause function to pause the game.	The game pauses, when the enter button is pressed the game resumes
home	When the home icon is pressed it triggers the save function to write the data onto the python text file	The data is written on the python text file.

Define solution fully



```
final.py  constants.py  main.py

while current != start:
    for node, dir in path.items():
        if dir:
            x, y = node
            x = current[0] * TILESIZE + TILESIZE / 2
            y = current[1] * TILESIZE + TILESIZE / 2
            img_pos = arrows[path[(current[0], current[1])]][0]
            r = img_pos.get_rect(center=(x, y))
            keys = pygame.key.get_pressed()
            if keys[pygame.K_p]:
                screen.blit(img_pos, r)
            d = path[(current[0], current[1])]
            zero = (0, 0)
            diagonal_vel = 0.7071
            vel = 1.5

            if d != zero:
                if d == (0, -1):
                    self.vely = -vel
                    self.rotation = 90

                if d == (0, 1):
                    self.vely = vel
                    self.rotation = 270

                if d == (1, 0):
                    self.velx = vel
                    self.rotation = 0

                if d == (-1, 0):
                    self.velx = -vel
                    self.rotation = 180

                if d == (1, 1):
                    self.velx = diagonal_vel
                    self.vely = diagonal_vel
                    self.rotation = 315

                if d == (-1, 1):
                    self.velx = -diagonal_vel
                    self.vely = diagonal_vel
                    self.rotation = 225

Line 889, Column 40   UTF-8   Spaces: 4   Python
```



```
final.py  constants.py  main.py

                    self.vely = vel
                    self.rotation = 270

                if d == (1, 0):
                    self.velx = vel
                    self.rotation = 0

                if d == (-1, 0):
                    self.velx = -vel
                    self.rotation = 180

                if d == (1, 1):
                    self.velx = diagonal_vel
                    self.vely = diagonal_vel
                    self.rotation = 315

                if d == (-1, 1):
                    self.velx = -diagonal_vel
                    self.vely = diagonal_vel
                    self.rotation = 225

                if d == (1, -1):
                    self.velx = diagonal_vel
                    self.vely = -diagonal_vel
                    self.rotation = 45

                if d == (-1, -1):
                    self.velx = -diagonal_vel
                    self.vely = -diagonal_vel
                    self.rotation = 135



        self.y += self.vely
        self.x += self.velx

        self.rect.x = int(self.x)
        self.collide_with_walls('x')

        self.rect.y = int(self.y)
        self.collide_with_walls('y')

Line 882, Column 30   UTF-8   Spaces: 4   Python
```

For my movement method I am continuing the code from the last prototype where I left off by drawing the path on the screen.

I have assigned the variable d, which will get the direction of the current node that it is pointing at, The direction could be from any of the eight directions like I have described in the ‘structure of solution’ part.

I have also included a rotation value, so that when the player is moving in a given direction the player’s image will also be rotated in the direction that it is travelling.

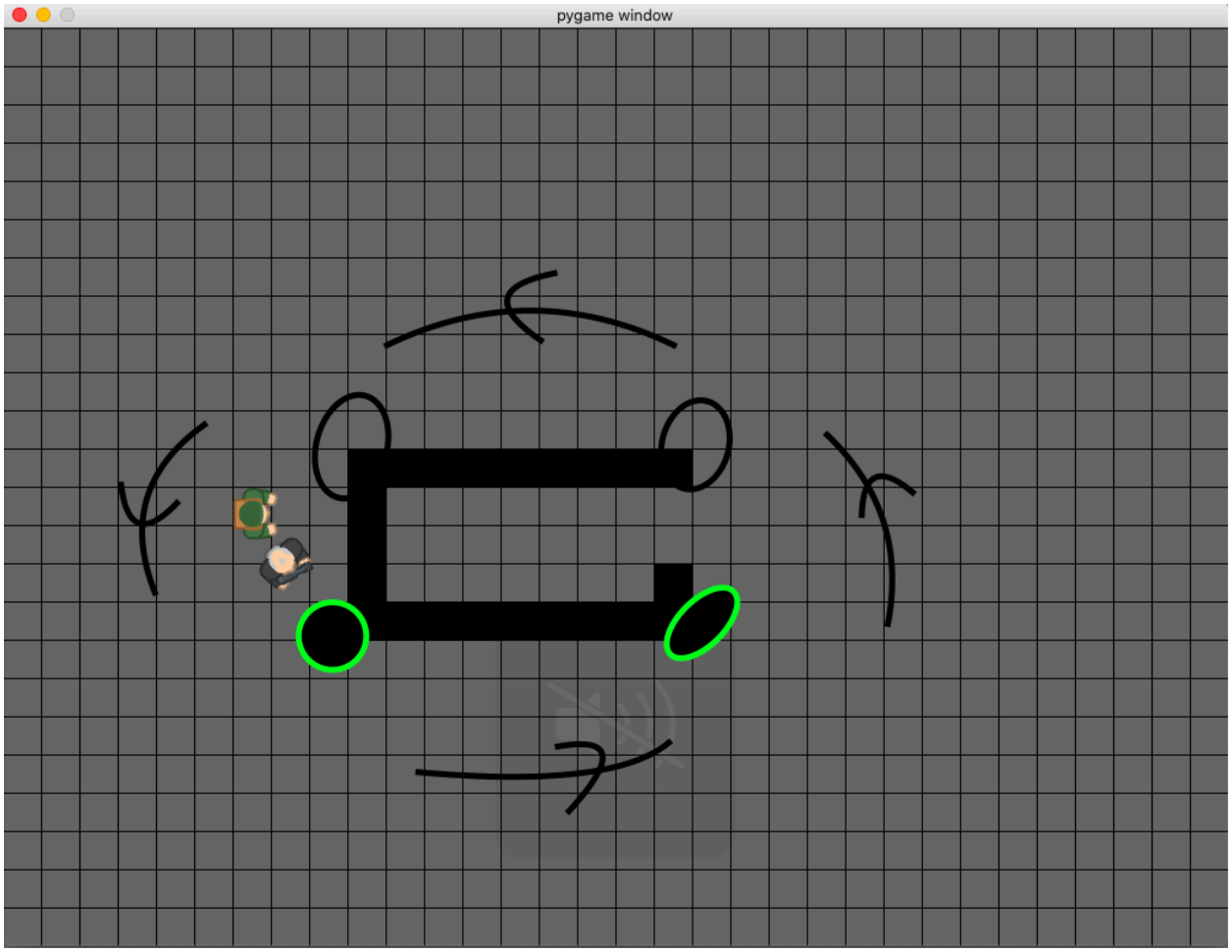
If d is zero (0, 0) then the node is not pointing towards anything.

While d is not zero, if d is (0, -1) then it must be pointing up therefore I have set my velocity of y to -1.5 and the rotation to 90. I have applied the same principle to every direction.

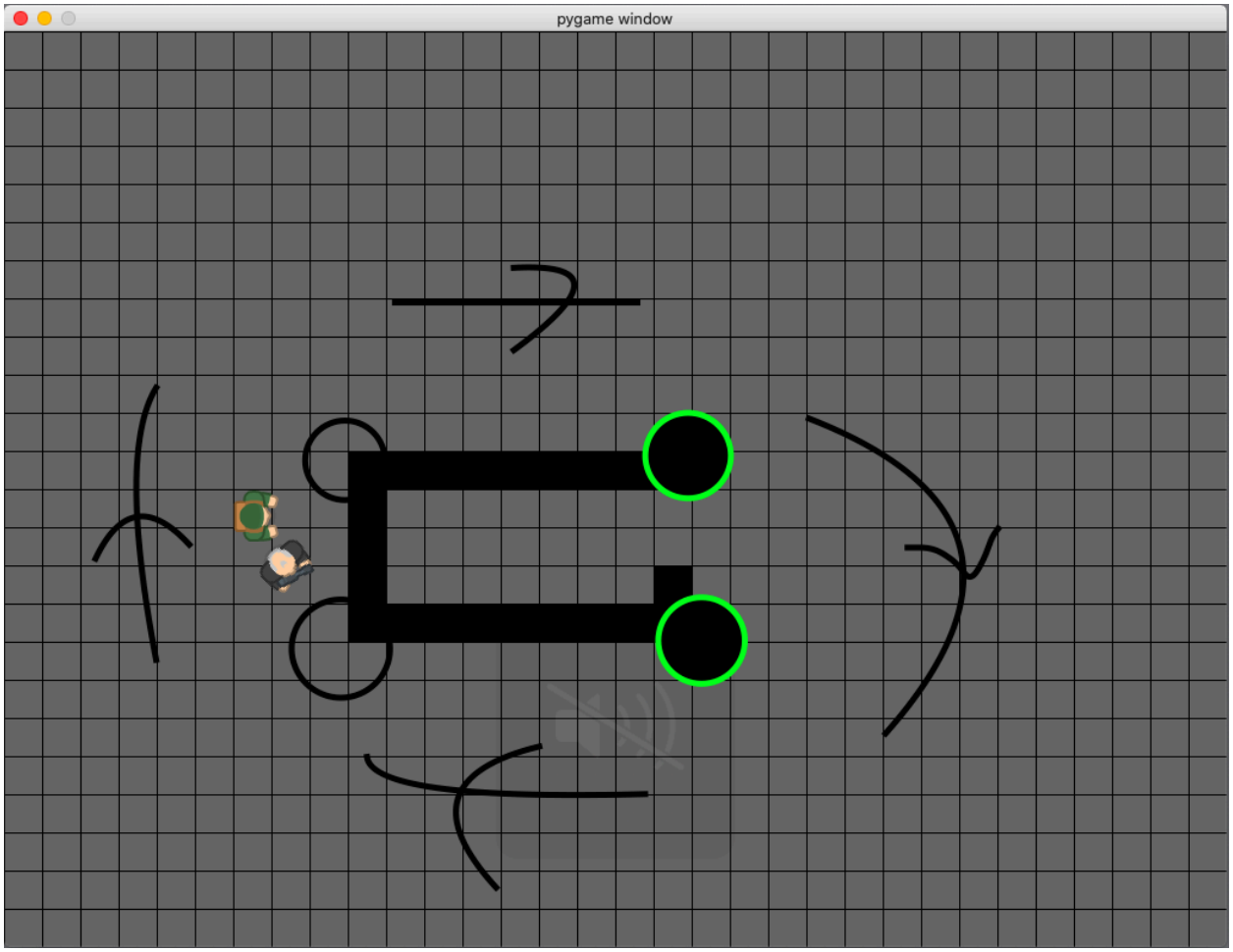
Inorder for the enemy to move I have set the ‘x’ and ‘y’ position of the enemy to change by the velocity that it gets for each node according to the direction each node has. #### (the direction is returned from the path dictionary for each node).

I have called the ‘collide_with_walls’ function for both ‘x’ and ‘y’ axis on the enemy, that I coded in prototype 3 which solved the issue of sliding through the walls for the player.

Movement Issue Clip



When the player is travelling in clockwise direction and when the enemy follows the player using the algorithm. The enemy can move through the green corners (bottom left and bottom right) but it gets stuck at the top left and top right. As you can see from the video clip above the algorithm shows the direction but still the enemy doesn't move. It just stops at that location until the direction is changed.



When the player is moving in anticlockwise direction and when the enemy is following the player, The enemy gets stuck at top left and bottom left corner.

Bottom right corner works well in both clockwise and anticlockwise directions.

I tried adjusting the player's and enemy's position by a scale but it didn't work.

```
constants.py   final.py   map2.txt

1101        self.row = y
1102        self.col = x
1103
1104
1105    class Wall:
1106        def __init__(self, x, y):
1107            self.row = x
1108            self.col = y
1109            self.x = x * TILESIZE
1110            self.y = y * TILESIZE
1111            self.rect = pygame.Rect(self.x, self.y, TILESIZE, TILESIZE)
1112
1113        def draw(self, screen):
1114            pygame.draw.rect(screen, BLACK, (self.x, self.y, TILESIZE, TILESIZE))
1115
1116
1117    class topLeft_wall:
1118        def __init__(self, x, y):
1119            self.row = x
1120            self.col = y
1121            self.x = x * TILESIZE
1122            self.y = y * TILESIZE
1123            self.rect = pygame.Rect(self.x, self.y, TILESIZE, TILESIZE)
1124
1125
1126    class bLeft_wall:
1127        def __init__(self, x, y):
1128            self.row = x
1129            self.col = y
1130            self.x = x * TILESIZE
1131            self.y = y * TILESIZE
1132            self.rect = pygame.Rect(self.x, self.y, TILESIZE, TILESIZE)
1133
1134
1135    class topRight_wall:
1136        def __init__(self, x, y):
1137            self.row = x
1138            self.col = y
1139            self.x = x * TILESIZE
1140            self.y = y * TILESIZE
1141            self.rect = pygame.Rect(self.x, self.y, TILESIZE, TILESIZE)
1142

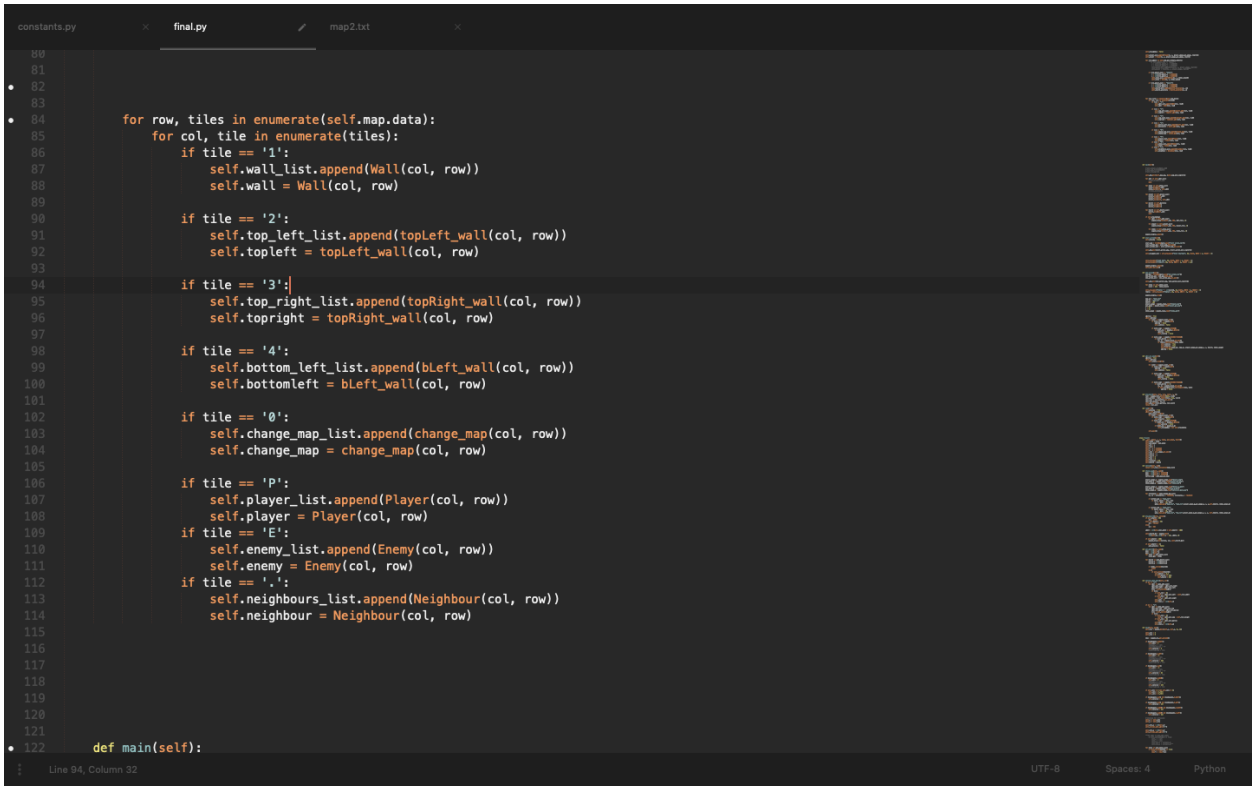
Line 15, Column 28   UTF-8   Spaces: 4   Python
```

Therefore as a temporary fix I introduced new classes for top left, top right and bottom left corners, So that I can know their exact locations(x, y) on the map

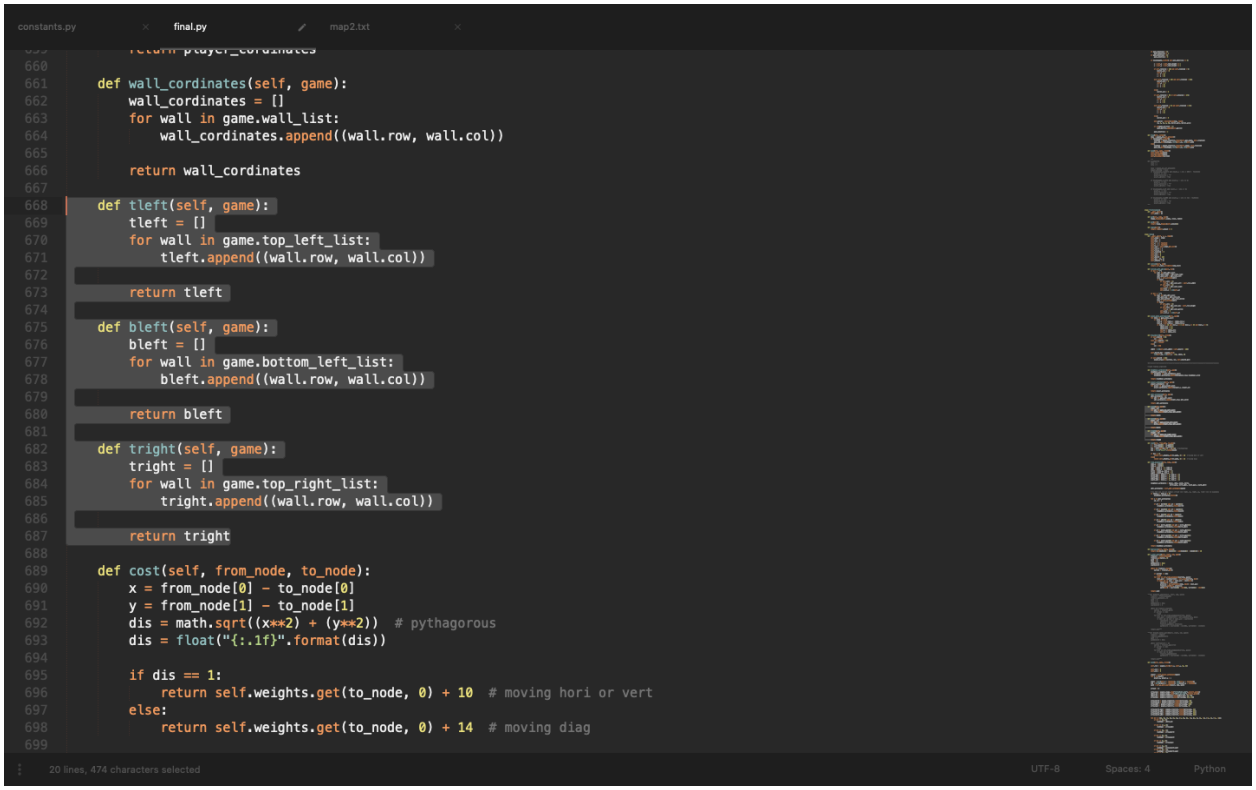
```
constants.py   final.py   map2.txt

1   ..............................
2   ..............................
3   ..............................
4   ..............................
5   ..............................
6   ..............................
7   ..............................
8   ..............................
9   ..............................
10  ........2.........3............
11  .........111111111............
12  .........1....................
13  .........1....................
14  .....P...1.......1............
15  .........111111111............
16  ........4.....................
17  ..............................
18  ..............................
19  ............E.................
20  ..............................
21  ..............................
22  ..............................
23  ..............................
24  ..............................
25

Line 17, Column 33   UTF-8   Spaces: 4   Plain Text
```



I implemented the same technique I used to find the position of walls (‘1’) to find the position of top left (‘2’), top right (‘3’) and bottom left (‘4’) corners.



```
constants.py   final.py   map2.txt

    def wall_cordinates(self, game):
        wall_cordinates = []
        for wall in game.wall_list:
            wall_cordinates.append((wall.row, wall.col))

        return wall_cordinates

    def tleft(self, game):
        tleft = []
        for wall in game.top_left_list:
            tleft.append((wall.row, wall.col))

        return tleft

    def bleft(self, game):
        bleft = []
        for wall in game.bottom_left_list:
            bleft.append((wall.row, wall.col))

        return bleft

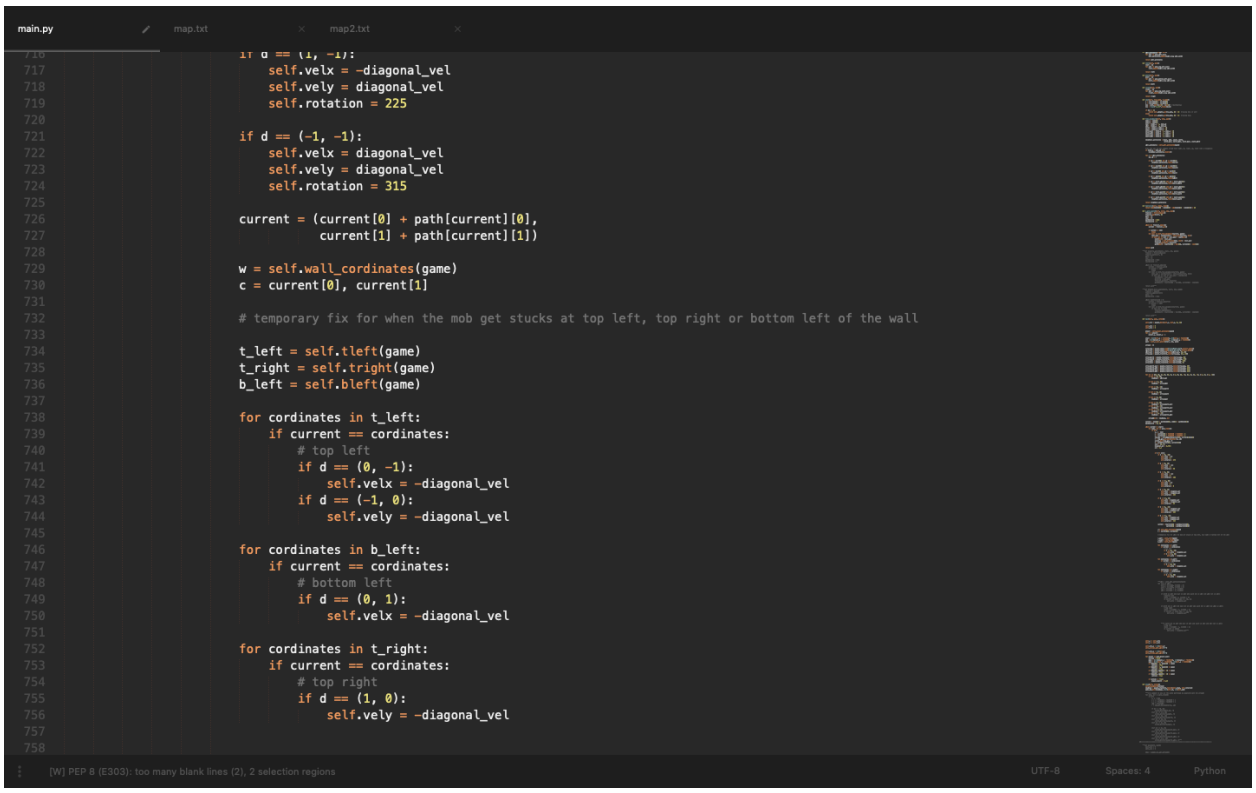
    def tright(self, game):
        tright = []
        for wall in game.top_right_list:
            tright.append((wall.row, wall.col))

        return tright

    def cost(self, from_node, to_node):
        x = from_node[0] - to_node[0]
        y = from_node[1] - to_node[1]
        dis = math.sqrt((x**2) + (y**2))  # pythagorous
        dis = float("{:.1f}".format(dis))

        if dis == 1:
            return self.weights.get(to_node, 0) + 10  # moving hori or vert
        else:
            return self.weights.get(to_node, 0) + 14  # moving diag

20 lines, 474 characters selected                                   UTF-8   Spaces: 4   Python
```



```
main.py   map.txt   map2.txt

            self.velx = -diagonal_vel
            self.vely = diagonal_vel
            self.rotation = 225

        if d == (-1, -1):
            self.velx = diagonal_vel
            self.vely = diagonal_vel
            self.rotation = 315

        current = (current[0] + path[current][0],
                   current[1] + path[current][1])

        w = self.wall_cordinates(game)
        c = current[0], current[1]

        # temporary fix for when the mob get stucks at top left, top right or bottom left of the wall

        t_left = self.tleft(game)
        t_right = self.tright(game)
        b_left = self.bleft(game)

        for cordinates in t_left:
            if current == cordinates:
                # top left
                if d == (0, -1):
                    self.velx = -diagonal_vel
                if d == (-1, 0):
                    self.vely = -diagonal_vel

        for cordinates in b_left:
            if current == cordinates:
                # bottom left
                if d == (0, 1):
                    self.velx = -diagonal_vel

        for cordinates in t_right:
            if current == cordinates:
                # top right
                if d == (1, 0):
                    self.vely = -diagonal_vel

[W] PEP 8 (E303): too many blank lines (2), 2 selection regions                UTF-8   Spaces: 4   Python
```

Back to when I coded the movement part for the enemy

I assigned the variable ‘t_left’ to find the coordinate of the top left corner, ‘t_right’ to find the top right corner and ‘b_left’ to find the bottom left corner of the wall in the map.

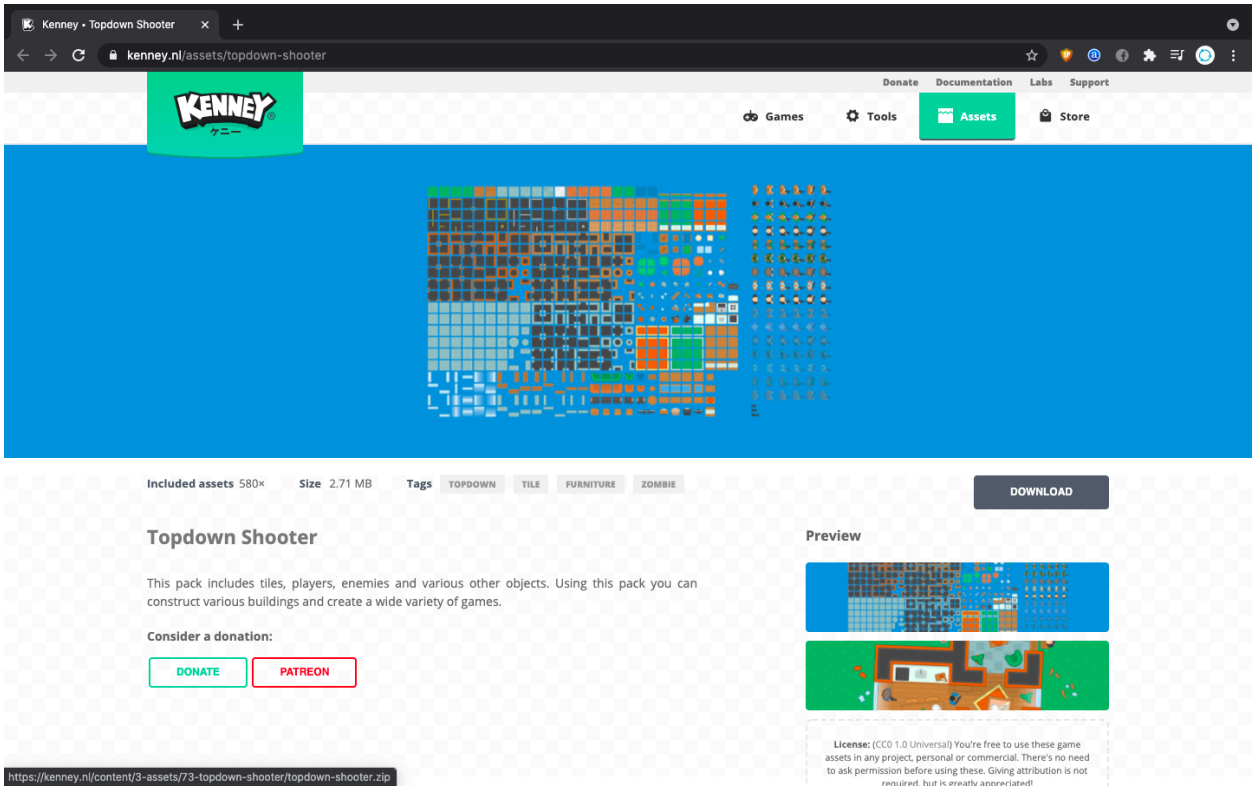
And then I printed out the d values that I obtained at those particular corners and understood the enemies behaviour at those coordinates.

I assigned a small velocity (0.7071) at those corners to the direction I wanted it to travel.

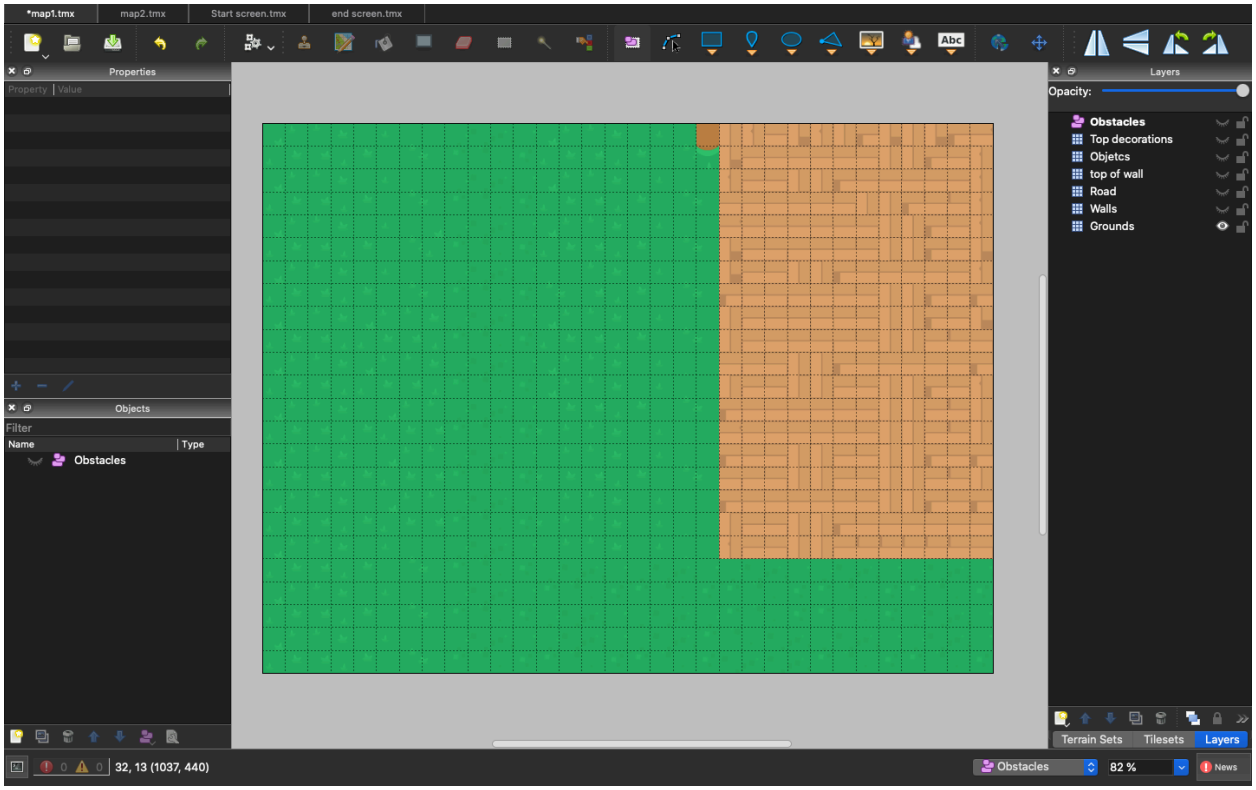
For example at the bottom left corner when d was equal to (0, 1) which means that the enemy wanted to travel up, I needed the enemy to go left and then only move up.

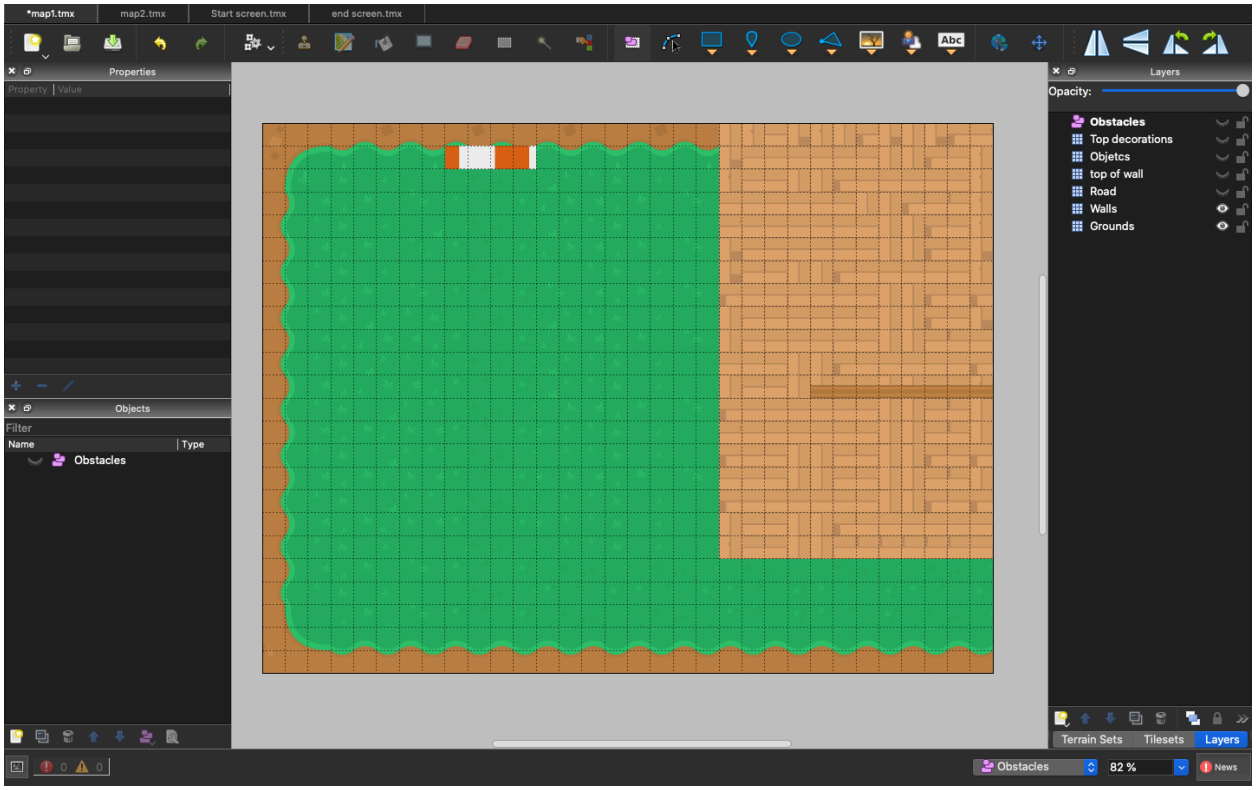
At the top left corner I had to check for both possible movements, travel left and then down or travel up and then right. So I assigned the appropriate velocity for x and y by studying their d values.

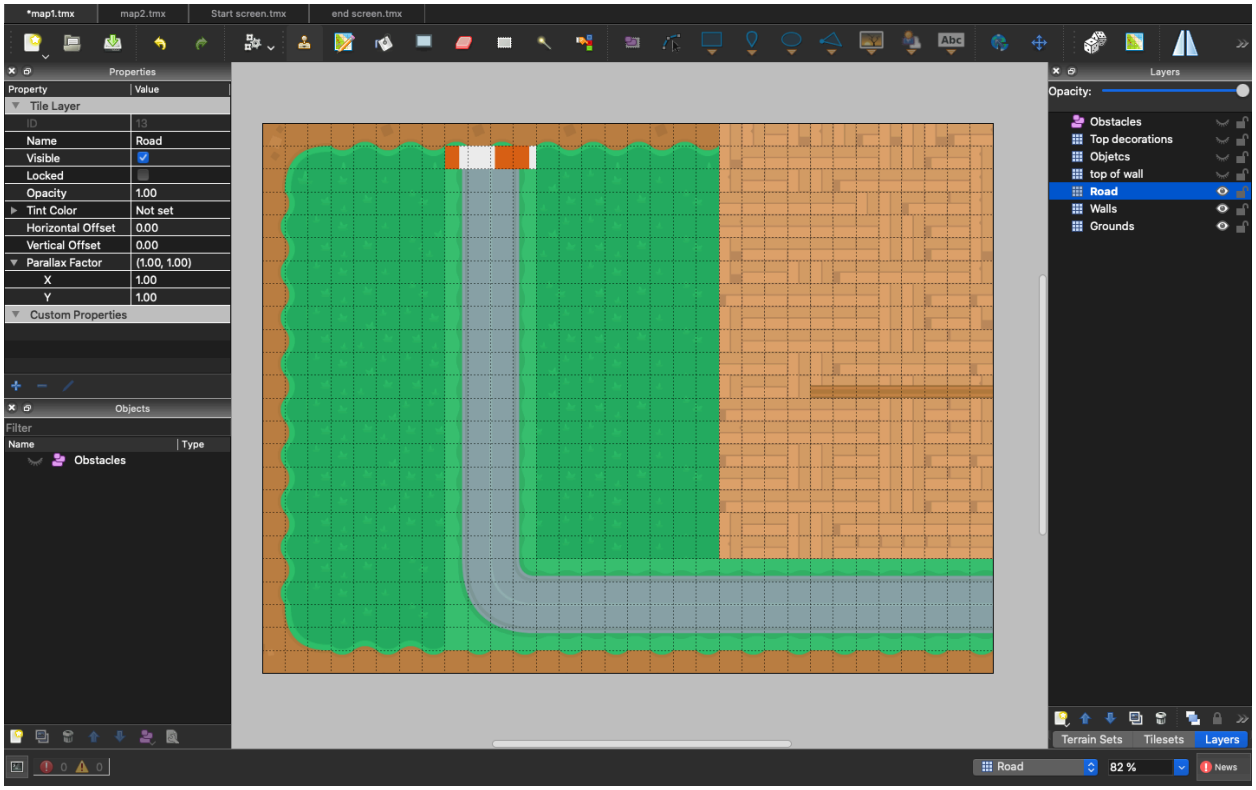
Therefore by using this temporary fix I am now able to give the enemy a little push, for it to move beyond the point where it gets stuck.

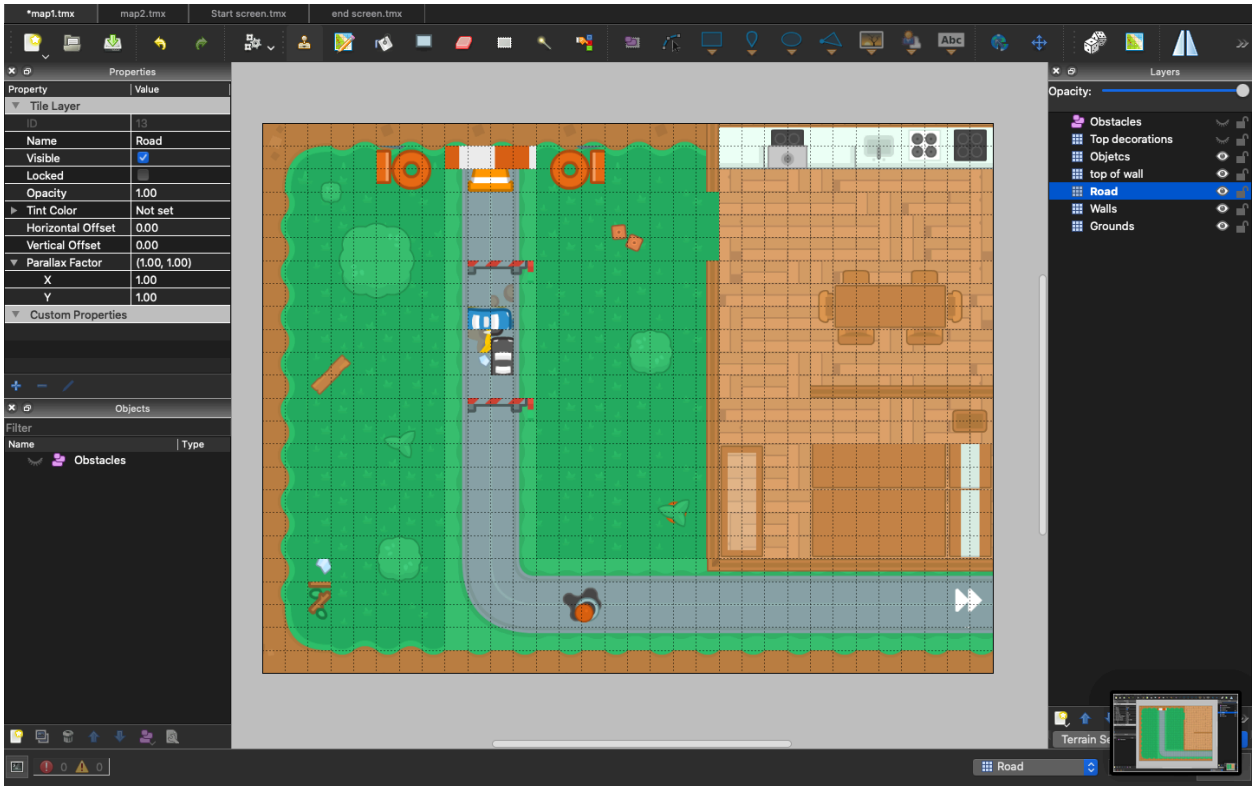


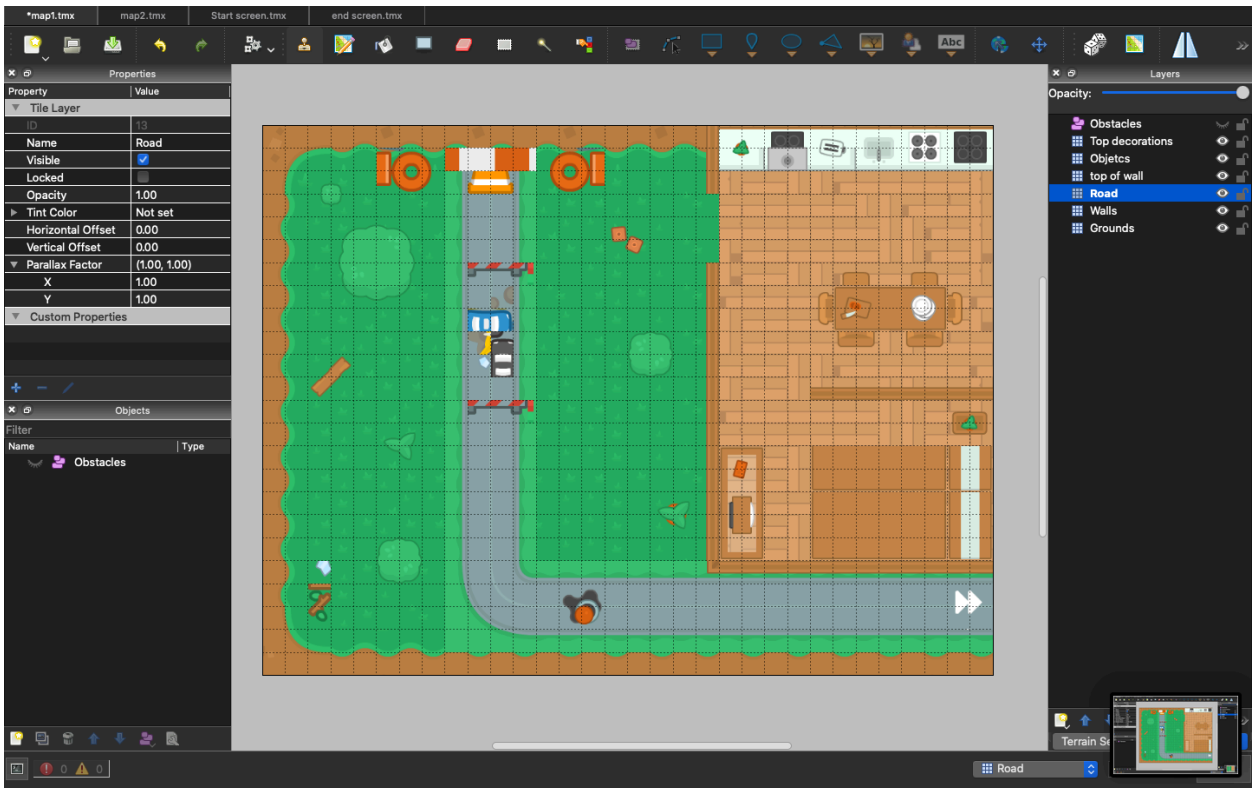
I downloaded a few sprite sheets from this website named ‘Kenney’.


*map1.tmx
map2.tmx
Start screen.tmx
end screen.tmx
Properties
Property
Value
Objects
Filter
Name
Type
Obstacles
Layers
Opacity:
Obstacles
Top decorations
Objetcs
top of wall
Road
Walls
Grounds
Terrain Sets
Tilesets
Layers
32, 13 (1037, 440)
Obstacles
82 %
News

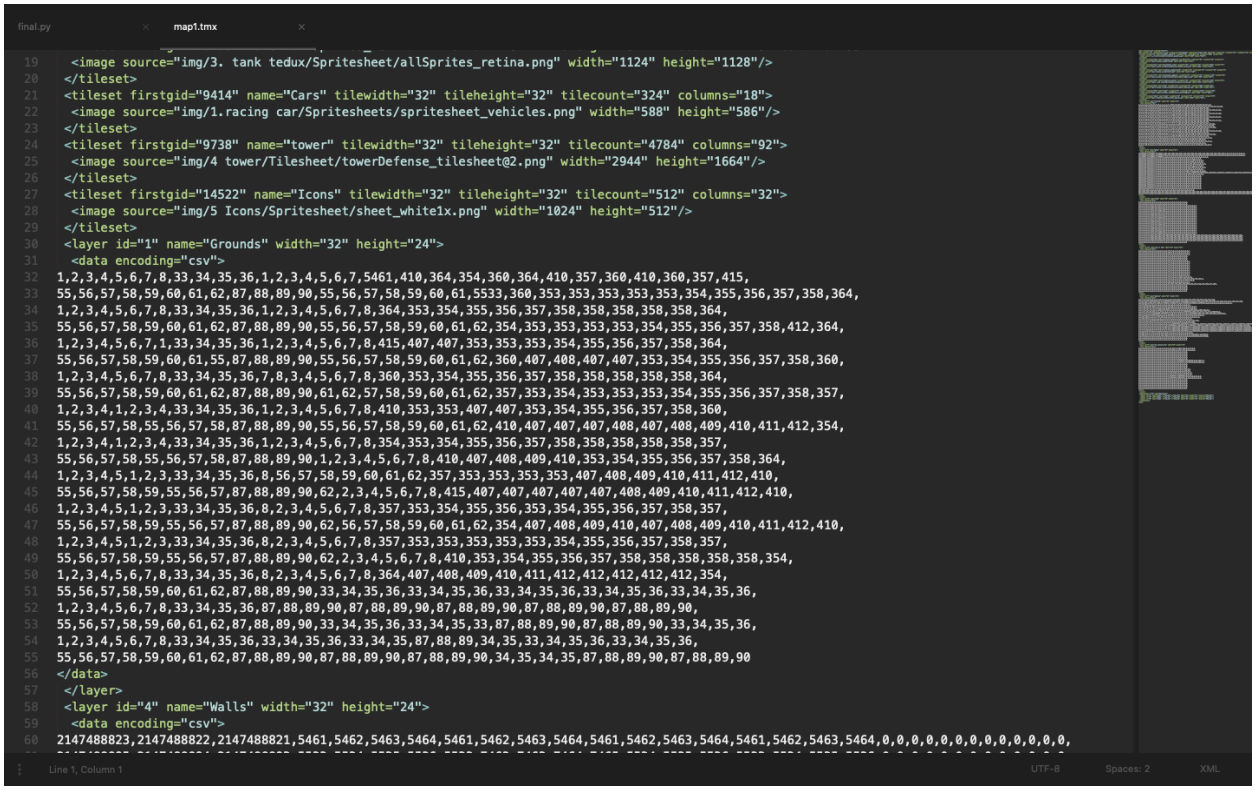

*map1.tmx
map2.tmx
Start screen.tmx
end screen.tmx
Properties
Property
Value
Objects
Filter
Name
Type
Obstacles
Layers
Opacity:
Obstacles
Top decorations
Objetcs
top of wall
Road
Walls
Grounds
Terrain Sets
Tilesets
Layers
Obstacles
82 %
News


*map1.tmx
map2.tmx
Start screen.tmx
end screen.tmx
Properties
Property
Value
Tile Layer
ID
13
Name
Road
Visible
Locked
Opacity
1.00
Tint Color
Not set
Horizontal Offset
0.00
Vertical Offset
0.00
Parallax Factor
(1.00, 1.00)
X
1.00
Y
1.00
Custom Properties
Objects
Filter
Name
Type
Obstacles
Layers
Opacity:
Obstacles
Top decorations
Objetcs
top of wall
Road
Walls
Grounds
Terrain Sets
Tilesets
Layers
Road
82 %
News

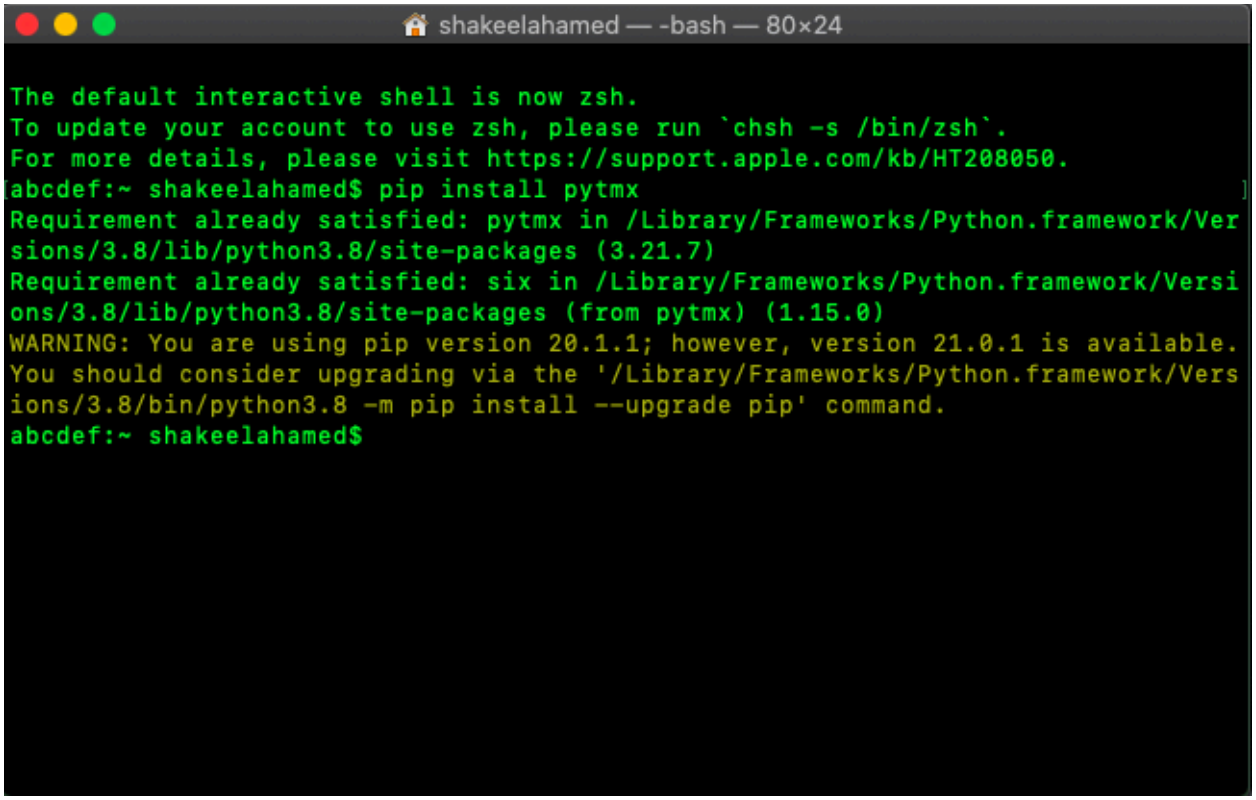




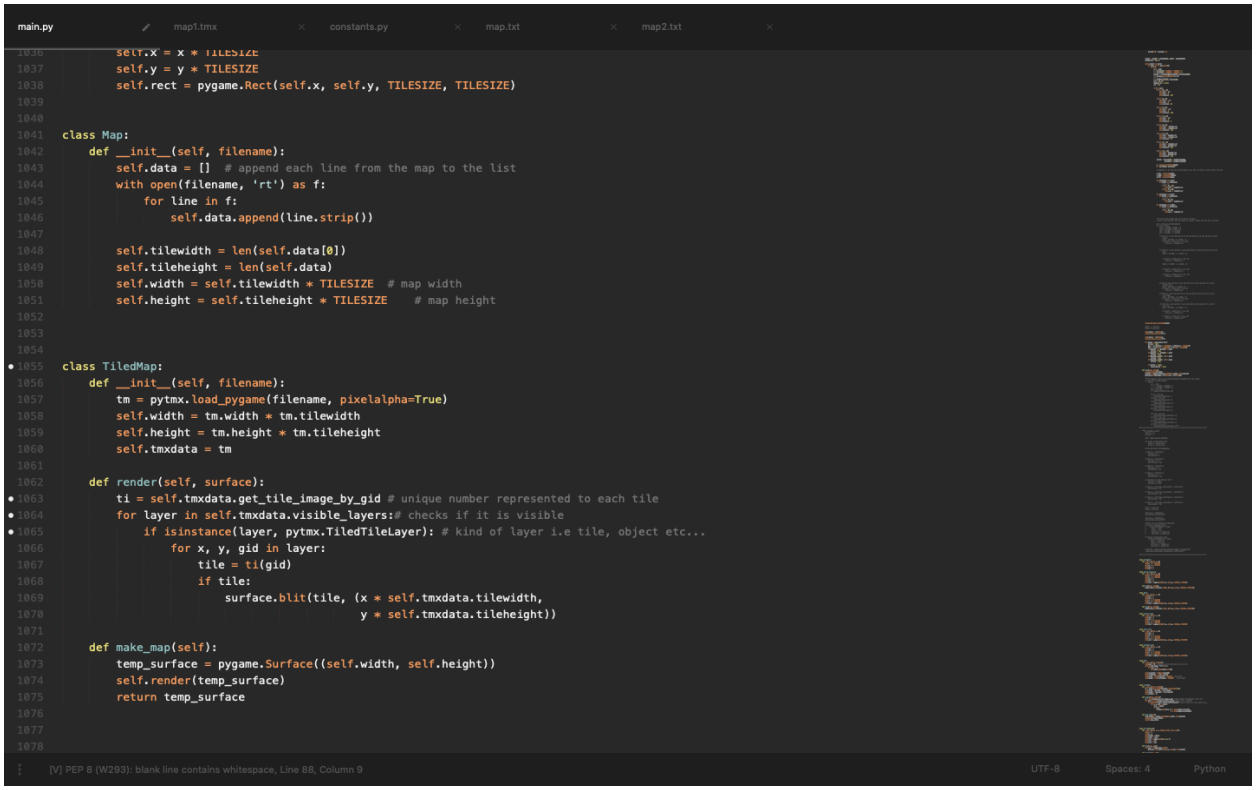
The images above shows each stage of producing this map



This is how my new Map file looks like, this map is the same as my old Map that had 1's and .'s, except now it has a lot of different tile images assigned to each layer by a unique number.



To work with this map editor, I had to install pytmx library and import it into my main code



I kept my Map class I wrote before because I will need that to represent the walls (1's) later

I created a new class called 'TIled Map' to load the map onto the game.

I have assigned tm to load the tmx file from the map and set pixel alpha to true so that I can get the transparency that comes with the images when I made the map through the Tiled editor.

The total width of the map is the number of tiles across the map multiplied by the tile width, In my case it would be (32 * 32 = 1024)

The total height would be number of tiles in y axis multiplied by the tile height, (24 * 32 = 768)

I have assigned the variable tmx data to store all the data of the map for reference

Render function takes in a surface and draws all the tiles from the map editor onto the screen.

The unique number assigned to each image in my tmx file is called gid (global identifier)

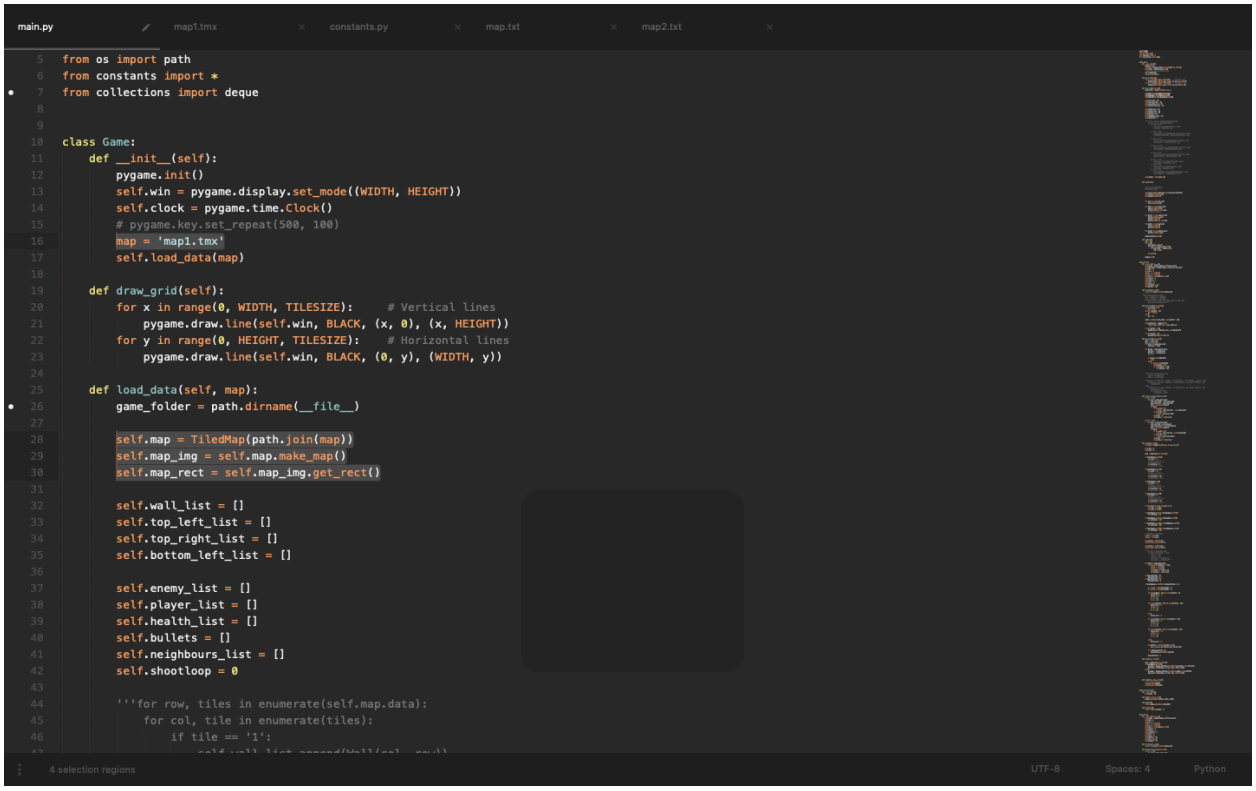
I have assigned the ‘ti’ variable to find the tile image by gid (global identifier), the unique number assigned to each image in my tmx file. Then the inbuilt function will return the tile image

I have looped through the layers in my map that are visible and then checked if it is a tile layer, if it is, then I have got the x, y and gid (the image id) of each tile in that layer.

I have assigned the variable ‘tile’ to be the ‘image’ of that ‘gid’.

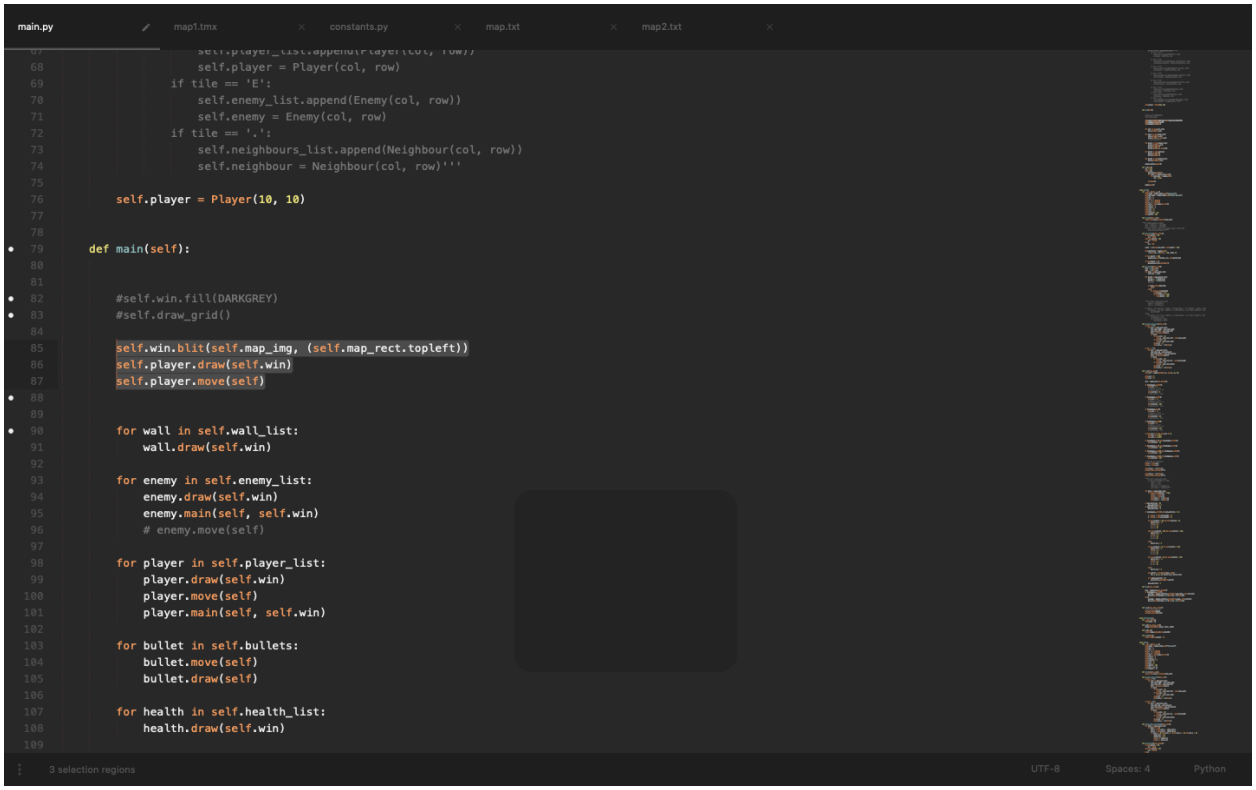
Finally I have blit each of those tiles onto the screen

The ‘make map’ function creates a temporary surface to draw the map onto, then render the map onto that temporary surface and then return that surface.



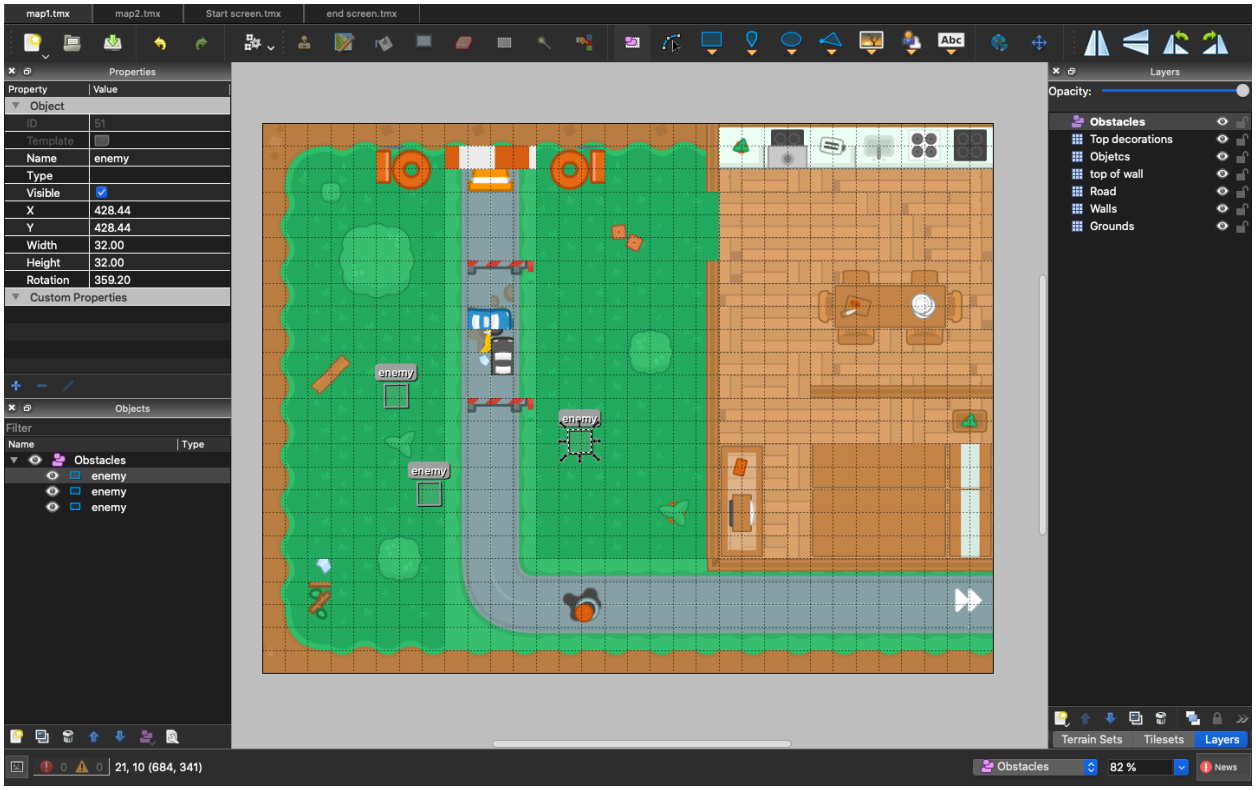
In my main game class where I used to load my previous map, I have now assigned the variable ‘map’ to create an instance of the ‘Tiled map’ class that loads up the map I created using ‘tiled map editor’.

‘Map_img’ creates the map with a surface, ‘map_rect’ gives me the x, y, width and height of the map so that I will be able to locate it onto the screen when I blit



In my main function inside the ‘Game’ class, I have blit them onto the screen using the ‘map_img’ and ‘map_rect’ variables.

I also created an instance of the player at the location (10, 10).



In my Tiled editor, I created an ‘object’ layer for the enemy so that It would be easy for me to place the enemy wherever I want in the map in the future.

```
        self.map_out = TiledMap(path.join(map_out))
        self.map_img = self.map_out.make_map()
        self.map_rect = self.map_img.get_rect()

        self.map = Map(path.join(map_in))

        self.change_map_list = []  # 0's

        self.health_list = []

        self.wall_list = []
        self.top_left_list = []
        self.top_right_list = []
        self.bottom_left_list = []

        self.enemy_list = []
        self.player_list = []
        self.bullets = []
        self.neighbours_list = []
        self.shootloop = 0

        self.draw_debug = False


        for tile_object in self.map_out.tmxdata.objects:
            if tile_object.name == 'enemy':
                x = int(tile_object.x / TILESIZE)
                y = int(tile_object.y / TILESIZE)
                self.enemy_list.append(Enemy(x, y, enemy_image))
                self.enemy = Enemy(x, y, enemy_image)


        for row, tiles in enumerate(self.map.data):
            for col, tile in enumerate(tiles):
                if tile == '1':
                    self.wall_list.append(Wall(col, row))
                    self.wall = Wall(col, row)

                if tile == '2':
```

From tmx data objects I looped through to find any objects named ‘enemy’ and got their exact coordinates from the map, This way I will be able to make an instance of enemy class at those coordinates like I used to do in my previous Map where I had ‘E’ assigned for enemies.



The map works perfectly fine, the player and the enemy can move around the map.

main.py | map.txt | map1.tmx | constants.py | map2.txt | map2.tmx

```
11111111111111111111111111111111
1....1111111111....1111111111111
1....111.11.111....1..........1
1.11...........................1
1..............1..............1
1...11..........1..............1
1...11...11........1.....11.11.1
1.................1....111111.1
1........11........1....111111.1
1........11.....11.1.....11.11.1
1.1.......1.....11.1...........1
1..1...............1....11111111
1........11........1.........111
1....11............1.........111
1....11............111..11111111
1..................111..11111111
1................11111..11111111
1................11111..11111111
1....11............111..11111111
1.1..11............111111111111
1.1..........11................0
1.1..........11................0
1..............................0
11111111111111111111111111111111
```

Line 25, Column 1 | UTF-8 | Spaces: 4 | Plain Text

1’s used to represent walls in my previous map but in this map it will represent all the obstacles, So that the player and the enemy won’t be able to go through the obstacles in the new map.

main.py | constants.py | map.txt | map1.tmx | map2.txt

```
        self.draw_debug = False

        for row, tiles in enumerate(self.map.data):
            for col, tile in enumerate(tiles):
                if tile == '1':
                    self.wall_list.append(Wall(col, row))
                    self.wall = Wall(col, row)
                if tile == '2':
                    self.top_left_list.append(topLeft_wall(col, row))
                    self.topleft = topLeft_wall(col, row)

                if tile == '3':
                    self.top_right_list.append(topRight_wall(col, row))
                    self.topright = topRight_wall(col, row)

                if tile == '4':
                    self.bottom_left_list.append(bLeft_wall(col, row))
                    self.bottomleft = bLeft_wall(col, row)

                '''if tile == 'H':
                    self.health_list.append(Health_inventory(col, row))
                    self.health_inventory = Health_inventory(col, row)

                if tile == '2':
                    self.top_left_list.append(topLeft_wall(col, row))
                    self.topleft = topLeft_wall(col, row)

                if tile == '3':
                    self.top_right_list.append(topRight_wall(col, row))
                    self.topright = topRight_wall(col, row)

                if tile == '4':
                    self.bottom_left_list.append(bLeft_wall(col, row))
                    self.bottomleft = bLeft_wall(col, row)

                if tile == 'P':
                    self.player_list.append(Player(col, row))
```

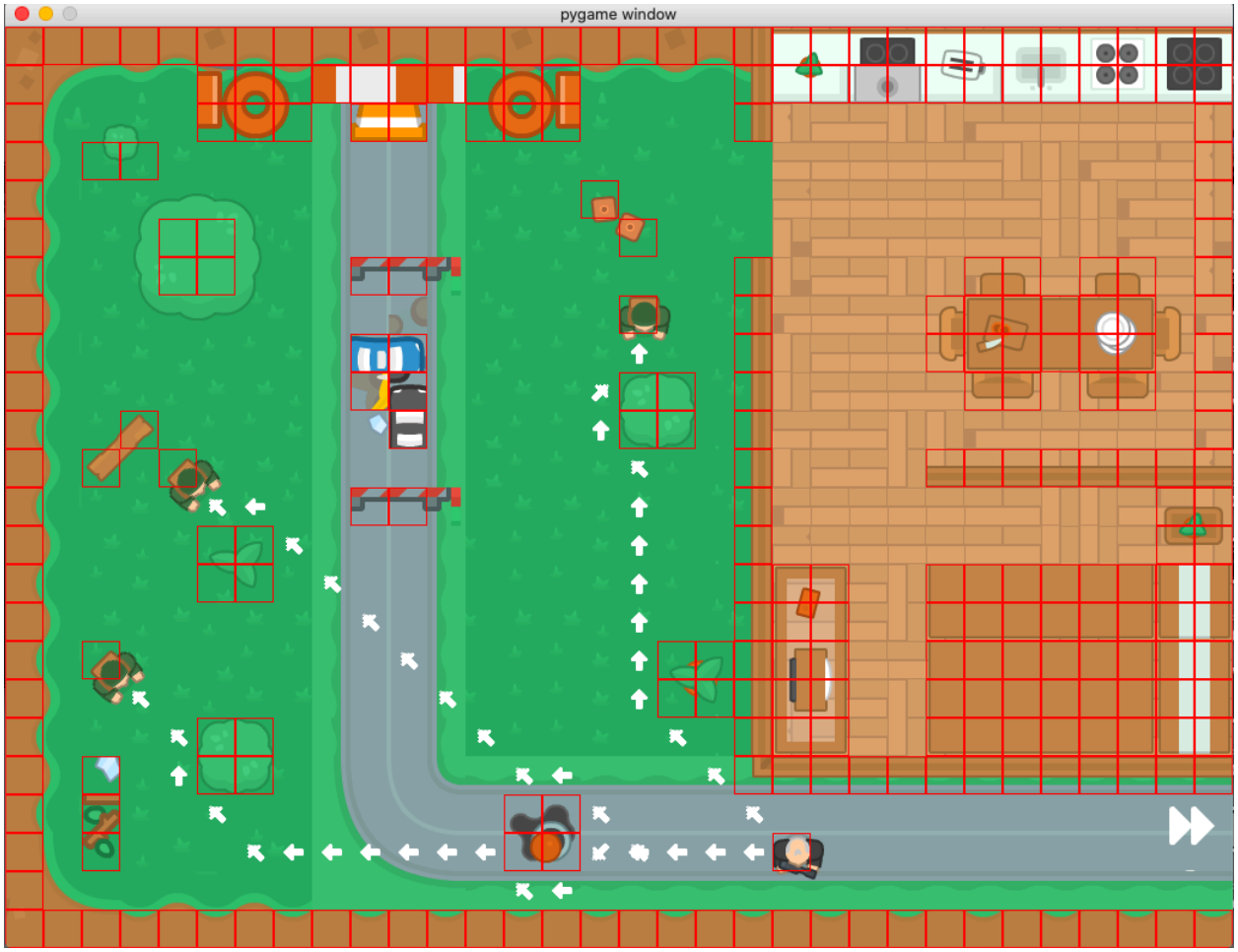
```
integers using __int__ is deprecated, and may be removed in a future version of Python.
  self.rect = pygame.Rect(self.x, self.y, 32, 32)
```

5 lines, 180 characters selected | UTF-8 | Spaces: 4 | Python

This code is from my previous map where I used to loop through the text file to find 1's and then made an instance of a wall at that location. I'm using this same code for the new map

```
123
124         if self.draw_debug:
125             for wall in self.wall_list:
126                 pygame.draw.rect(self.win, RED, wall.rect, 1)
127
128         pygame.display.update()
129
130     def run(self):
131         run = True
132         while run:
133             game.clock.tick(FPS)
134             for event in pygame.event.get():
135                 if event.type == pygame.QUIT:
136                     run = False
137
138                 if event.type == pygame.KEYDOWN:
139                     if event.key == pygame.K_ESCAPE:
140                         run = False
141                     if event.key == pygame.K_d:
142                         self.draw_debug = not self.draw_debug
```

I made a debug tool so that it will draw a border around the obstacles in the map, which will make it easier to fix any bugs in the future and to know what and where something is colliding. The debug function can be turned on and off by pressing the letter 'd'



The player and the enemy cannot move through the boxes and the path finding algorithm works perfectly fine.

final.py untitled map.txt

```
1111111111111111111111111111111
1....1111111111....1111111111111
1....111.11.111....1...........1
1.11............................1
1..............1...............1
1...11..........1..............1
1...11...11........1.....11.11.1
1..................1....111111.1
1........11........1....111111.1
1........11.....11.1.....11.11.1
1..1......1.....11.1...........1
1.1.................1....11111111
1........11........1..........11
1....11............1..........11
1....11............111..11111111
1..................111..11111111
1................11111..11111111
1................11111..11111111
1....11............111..11111111
1.1..11............1111111111111
1.1..........11.................0
1.1..........11.................0
1...............................0
1111111111111111111111111111111
```

3 selection regions UTF-8 Spaces: 4 Plain Text

final.py untitled

```python
class topRight_wall:
    def __init__(self, x, y):
        self.row = x
        self.col = y
        self.x = x * TILESIZE
        self.y = y * TILESIZE
        self.rect = pygame.Rect(self.x, self.y, TILESIZE, TILESIZE)

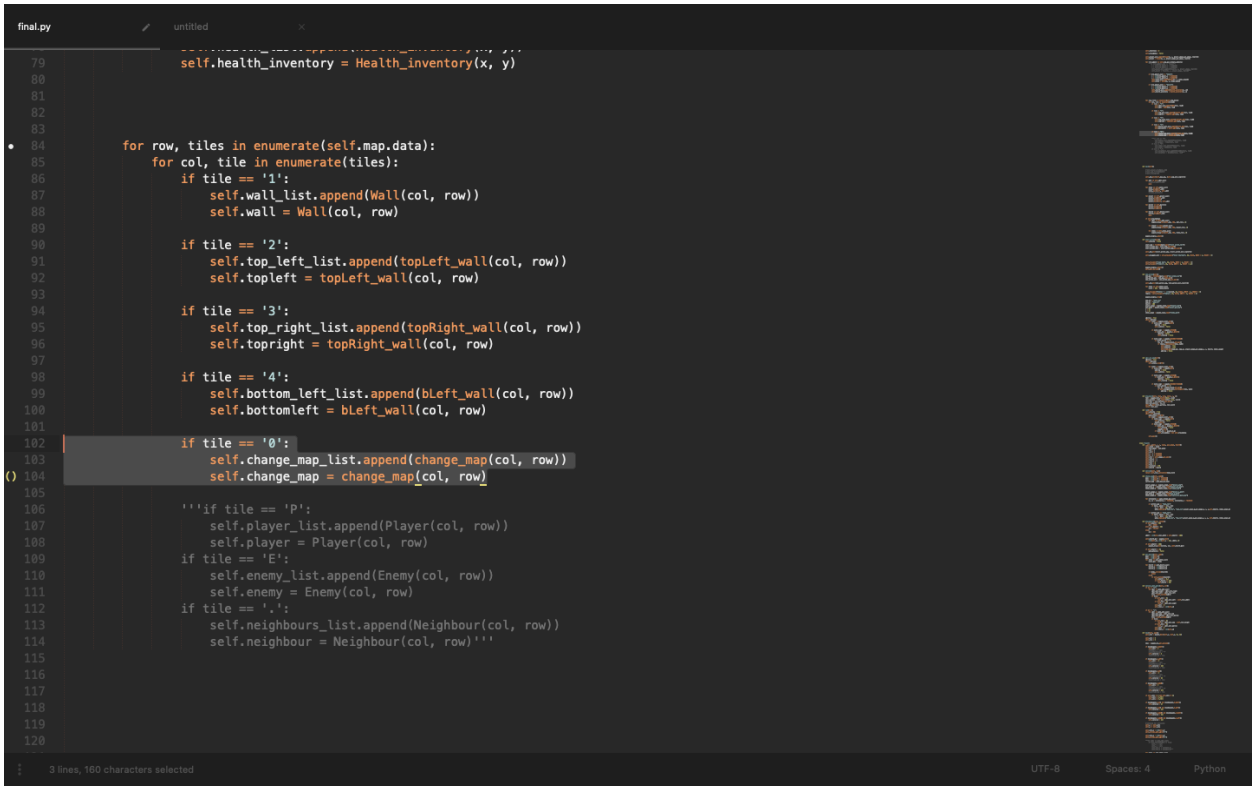

class change_map:
    def __init__(self, x, y):
        self.row = x
        self.col = y
        self.x = x * TILESIZE
        self.y = y * TILESIZE
        self.rect = pygame.Rect(self.x, self.y, TILESIZE, TILESIZE)


class TiledMap:
    def __init__(self, filename):
        tm = pytmx.load_pygame(filename, pixelalpha=True)
        self.width = tm.width * tm.tilewidth
        self.height = tm.height * tm.tileheight
        self.tmxdata = tm
        self.name = filename

    def render(self, surface):
        # unique number represented to each tile
        ti = self.tmxdata.get_tile_image_by_gid
        for layer in self.tmxdata.visible_layers:  # checks if it is visible
            # Kind of layer i.e tile, object etc...
            if isinstance(layer, pytmx.TiledTileLayer):
                for x, y, gid in layer:
                    tile = ti(gid)
                    if tile:
                        surface.blit(tile, (x * self.tmxdata.tilewidth,
                                            y * self.tmxdata.tileheight))

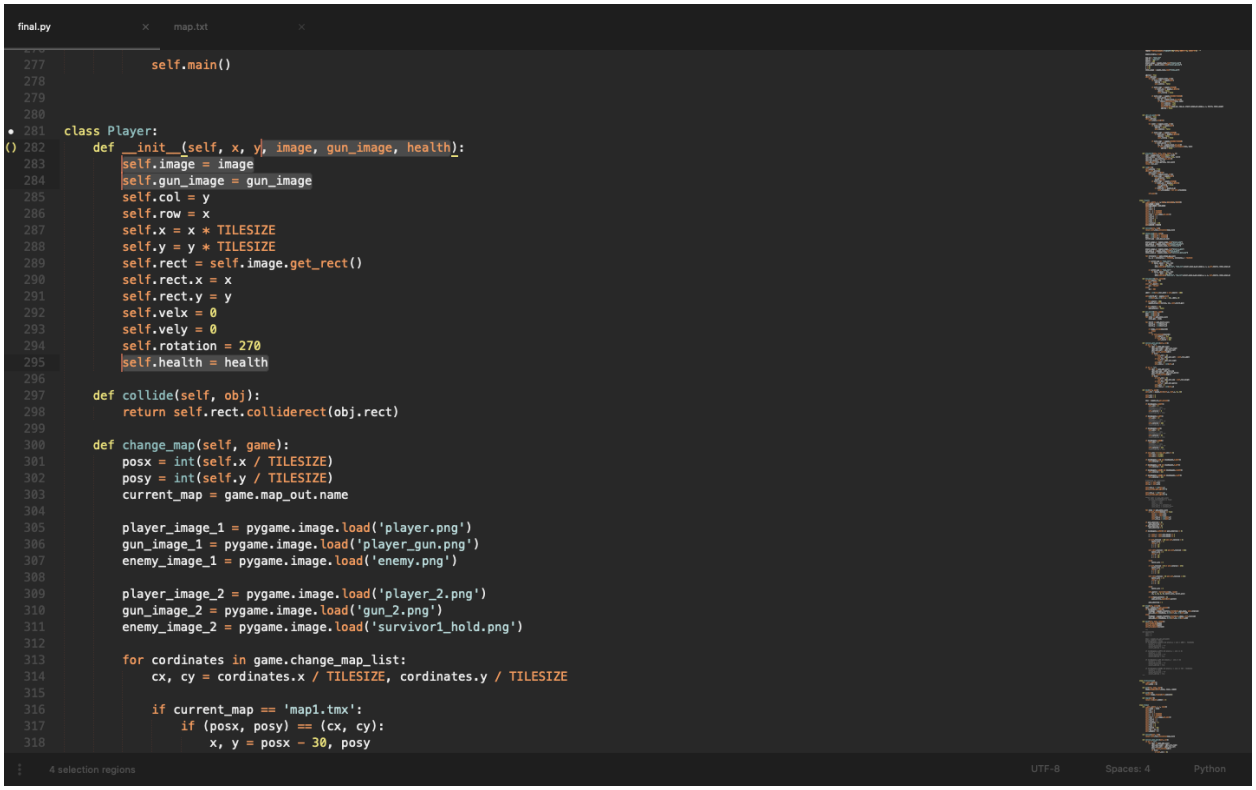
    def make_map(self):
        temp_surface = pygame.Surface((self.width, self.height))
        self.render(temp_surface)
        return temp_surface
```

7 lines, 217 characters selected UTF-8 Spaces: 4 Python

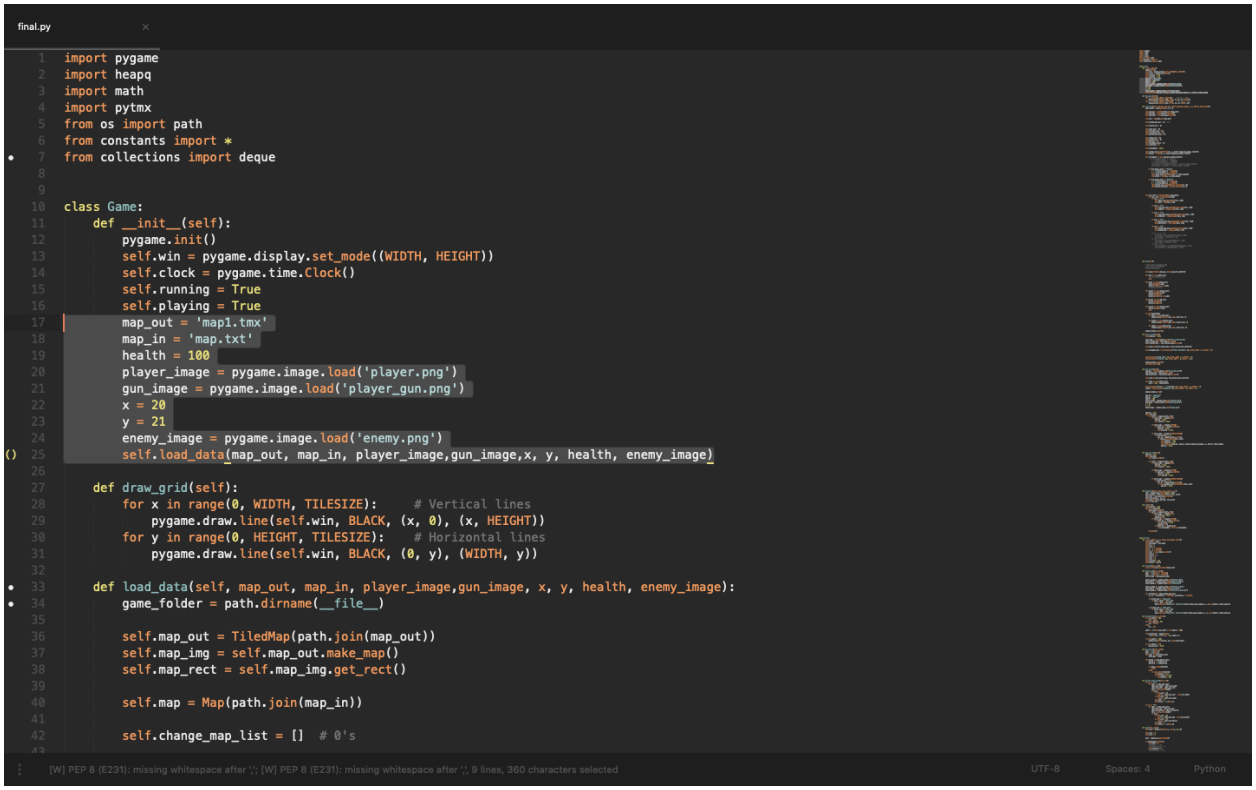


At the bottom right of the map I added 0's to get the location of it in the map, so that when the player goes near any of the 0, I can make the player appear to be moving onto a next level. As you can see from my tiled editor map I have left a road that leads to an unknown path.

I have created a class called 'change_map' to get the coordinates of 0's in the map. And then added those coordinates onto the 'change_map_list' so that I can get those coordinates to check against the player's coordinates later.



I have edited my players class so that I can insert the player's 'image', 'gun_image' and 'health' in the 'load_data' function in the 'Game' class, because I am creating an instance of the player only in 'load_data' function. When the player moves onto a new map, I would be able to change the images and sync the health from the previous map.



I have passed on the variables needed to make an instance of the player and the enemy at the start. In my ‘load_data’ function I have taken the following arguments:

‘map_out’ - outer appearance of the map (the map from ‘Tiled’ editor)

‘map_in’ - text file that has walls for the ‘Tiled’ map so that the player and enemy won’t be able to go through any obstacles

‘health’ - this is player’s health and I have set this to 100

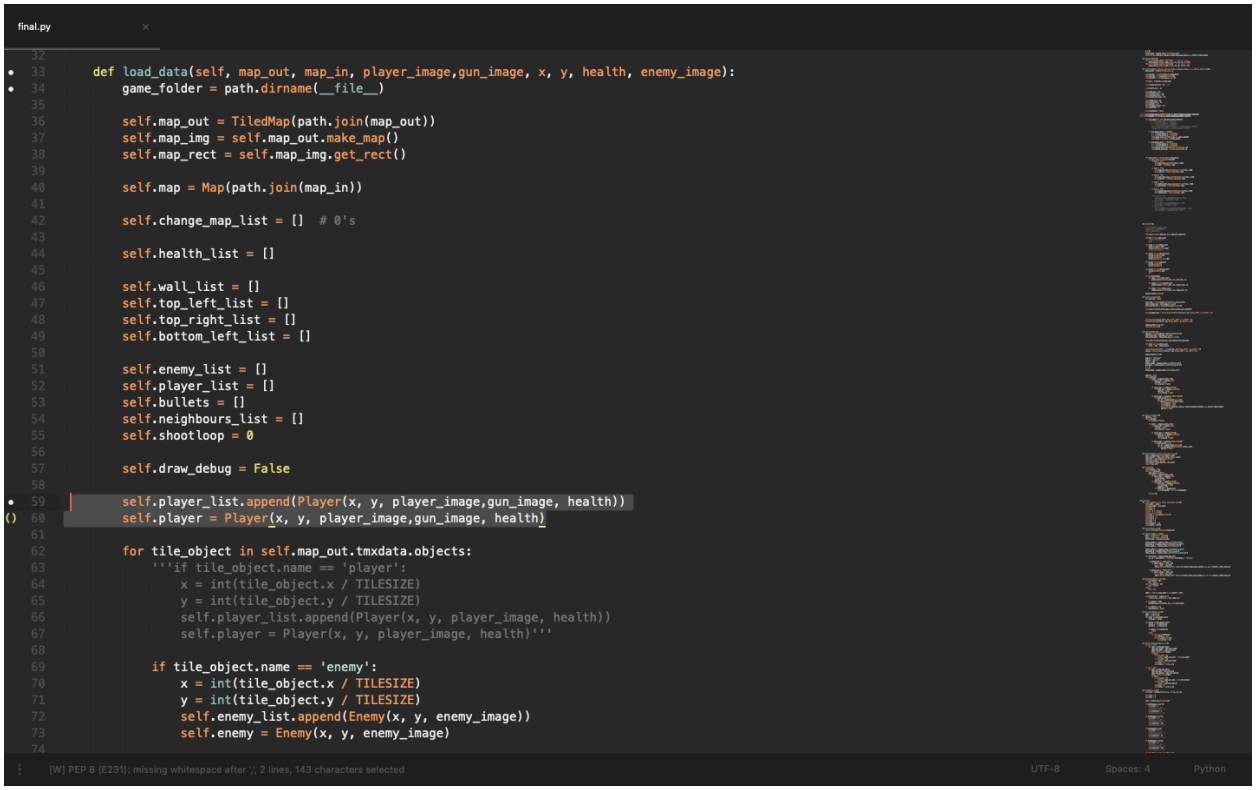
‘player_image’ - the player’s image for first level

‘Gun_image’ - shooting image for the player

‘x’ - the x coordinate of the player when spawning on the map

‘y’ - the y coordinate of the player when spawning on the map

‘enemy_image’ - the enemy’s image for the first level



Now I have all the arguments I needed to create an instance of the player, enemy and the map from the ‘load_data’ function

final.py

```
        self.rect.x = x
        self.rect.y = y
        self.velx = 0
        self.vely = 0
        self.rotation = 270
        self.health = health

    def collide(self, obj):
        return self.rect.colliderect(obj.rect)

    def change_map(self, game):
        posx = int(self.x / TILESIZE)
        posy = int(self.y / TILESIZE)
        current_map = game.map_out.name

        player_image_1 = pygame.image.load('player.png')
        gun_image_1 = pygame.image.load('player_gun.png')
        enemy_image_1 = pygame.image.load('enemy.png')

        player_image_2 = pygame.image.load('player_2.png')
        gun_image_2 = pygame.image.load('gun_2.png')
        enemy_image_2 = pygame.image.load('survivor1_hold.png')

        for cordinates in game.change_map_list:
            cx, cy = cordinates.x / TILESIZE, cordinates.y / TILESIZE

            if current_map == 'map1.tmx':
                if (posx, posy) == (cx, cy):
                    x, y = posx - 30, posy
                    game.load_data('map2.tmx', 'map2.txt',player_image_2,gun_image_2, x, y,self.health, enemy_image_2)

            if current_map == 'map2.tmx':
                if (posx, posy) == (cx, cy):
                    x, y = posx + 30, posy
                    game.load_data('map1.tmx', 'map.txt',player_image_1,gun_image_1, x, y, self.health, enemy_image_1)

    def draw_health(self, screen):
        if self.health > 60:
            col = GREEN
        elif self.health > 30:
            col = YELLOW
        else:
            col = RED
```

[W] PEP 8 (E231): missing whitespace after ';' [W] PEP 8 (E231): missing whitespace after ';' 25 lines, 1114 characters selected — UTF-8 — Spaces: 4 — Python

final.py

```
                self.x += 2
                x = x - 10
                y = y - 10

            else:
                bullet_velx = 0

            self.bullet = Bullet(round(x), round(
                y), 6, (0, 0, 0), bullet_velx, bullet_vely)

            if len(game.bullets) < 5:
                game.bullets.append(self.bullet)

            game.shootloop = 1

    def draw(self, screen):
        keys = pygame.key.get_pressed()
        if keys[pygame.K_SPACE]:
            newimage = pygame.transform.rotate(self.gun_image, self.rotation)
            game.win.blit(newimage, (int(self.x), int(self.y)))
        else:
            newimage = pygame.transform.rotate(self.image, self.rotation)
            game.win.blit(newimage, (int(self.x), int(self.y)))

    def main(self, game, screen):
        self.change_map(game)
        self.get_health(game)
        self.draw_health(screen)

        '''
    def move(self):
        velx = 3
        vely = 3

        keys = pygame.key.get_pressed()
        player.walking = False
        if keys[pygame.K_RIGHT] and player.x + velx < WIDTH - TILESIZE:
            player.x += velx
            player.direction = 'r'
            player.walking = True

        if keys[pygame.K_LEFT] and player.x - velx >= 0:
```

21 characters selected — UTF-8 — Spaces: 4 — Python

I created a new function called ‘change_map’ in my player’s class

posx and posy gets the integer value of the row and column the player is currently at.

The variable ‘current_map’ returns the name of the map that the player is currently in.

‘player_image_1’, ‘gun_image_1’, ‘enemy_image_1’ loads up the images for the player and enemy for the first level.

‘player_image_2’, ‘gun_image_2’, ‘enemy_image_2’ loads up the images for the player and enemy for the second level.

To find the coordinates of an ending point (the 0’s at the bottom right of the map) I looped through the ‘change_map_list’ I created before which stored the coordinates of 0’s, and assigned the variable cx, cy for the coordinate of each ‘0’ in the map.

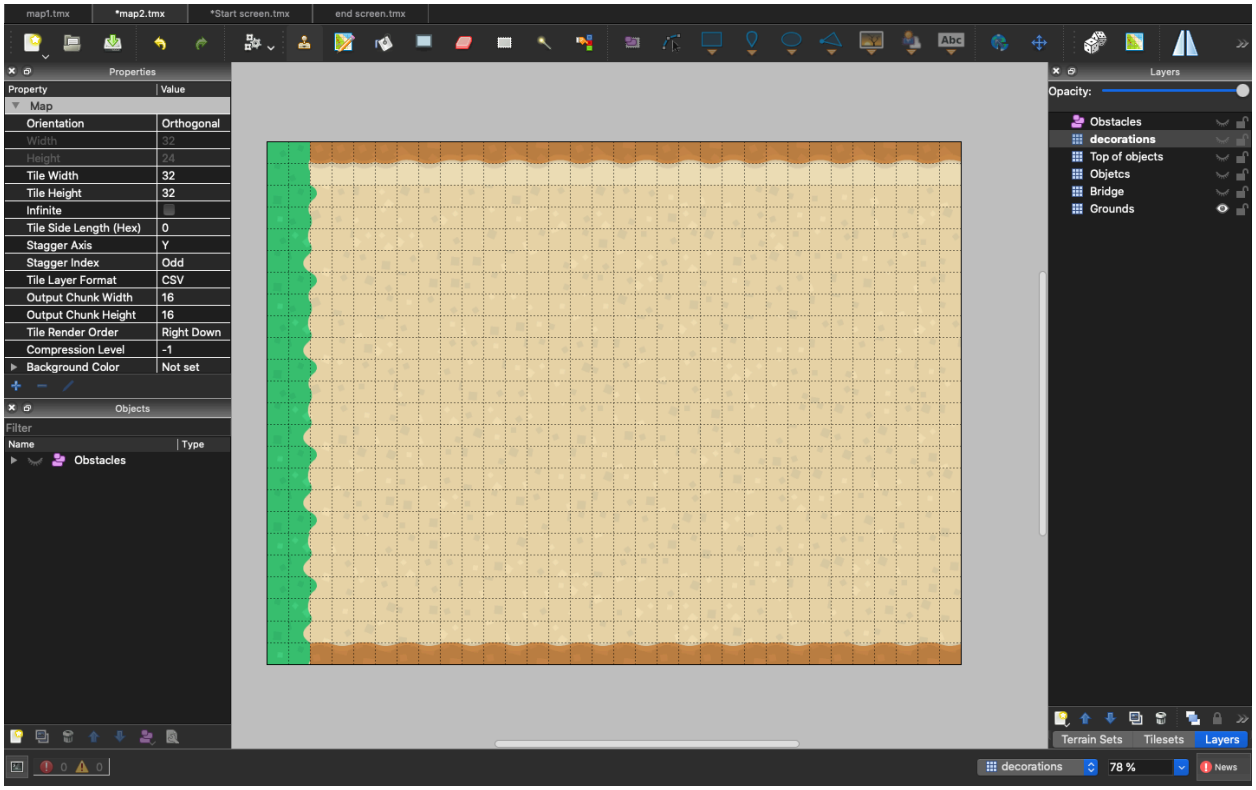
If the current map is equal to ‘map1.tmx’ which is the map I created from ‘Tiled map editor’, and if the player position is equal to any of the ‘0’ from the map (the end of a the road from the map), then I called the ‘load_data’ function to make a new instance for each object.

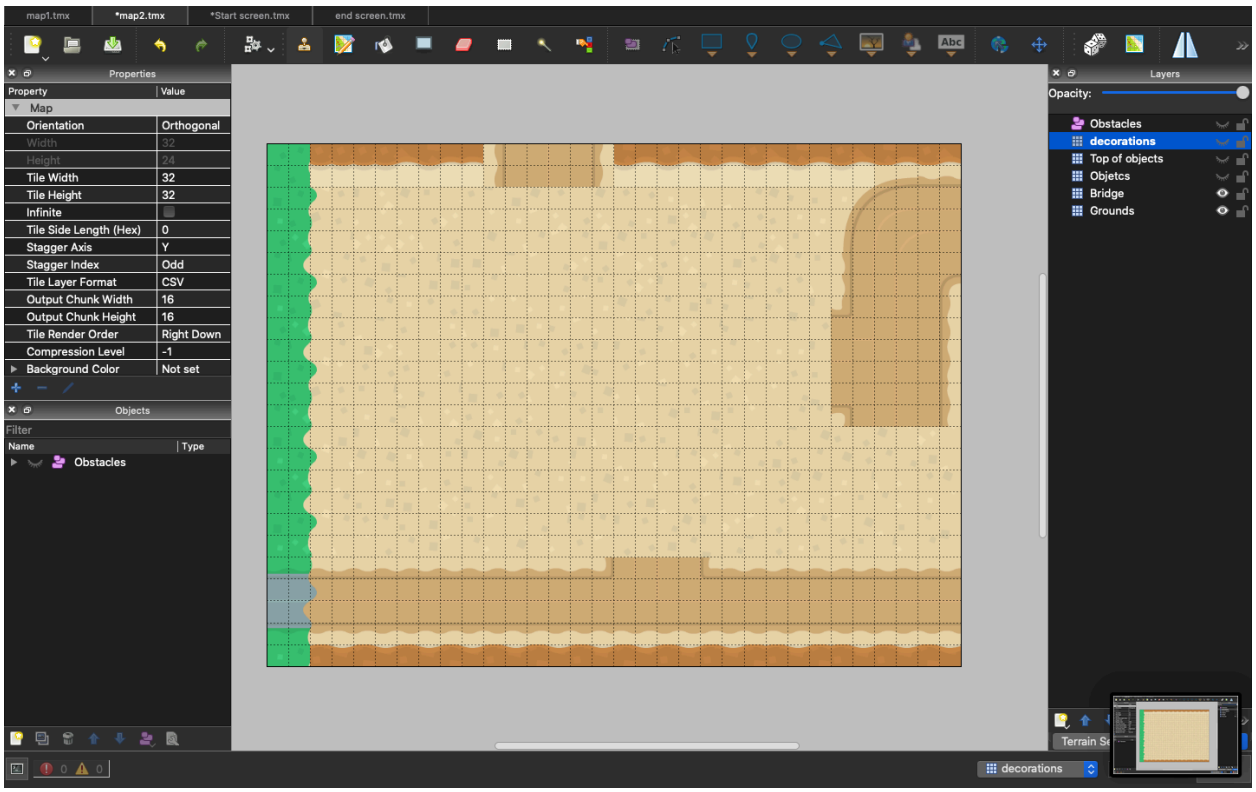
Since I want to make it appear like the player is moving smoothing from one map to another, I got the new coordinates for the x and y position for the player by deducting 30 from the x position and leaving the same y position.

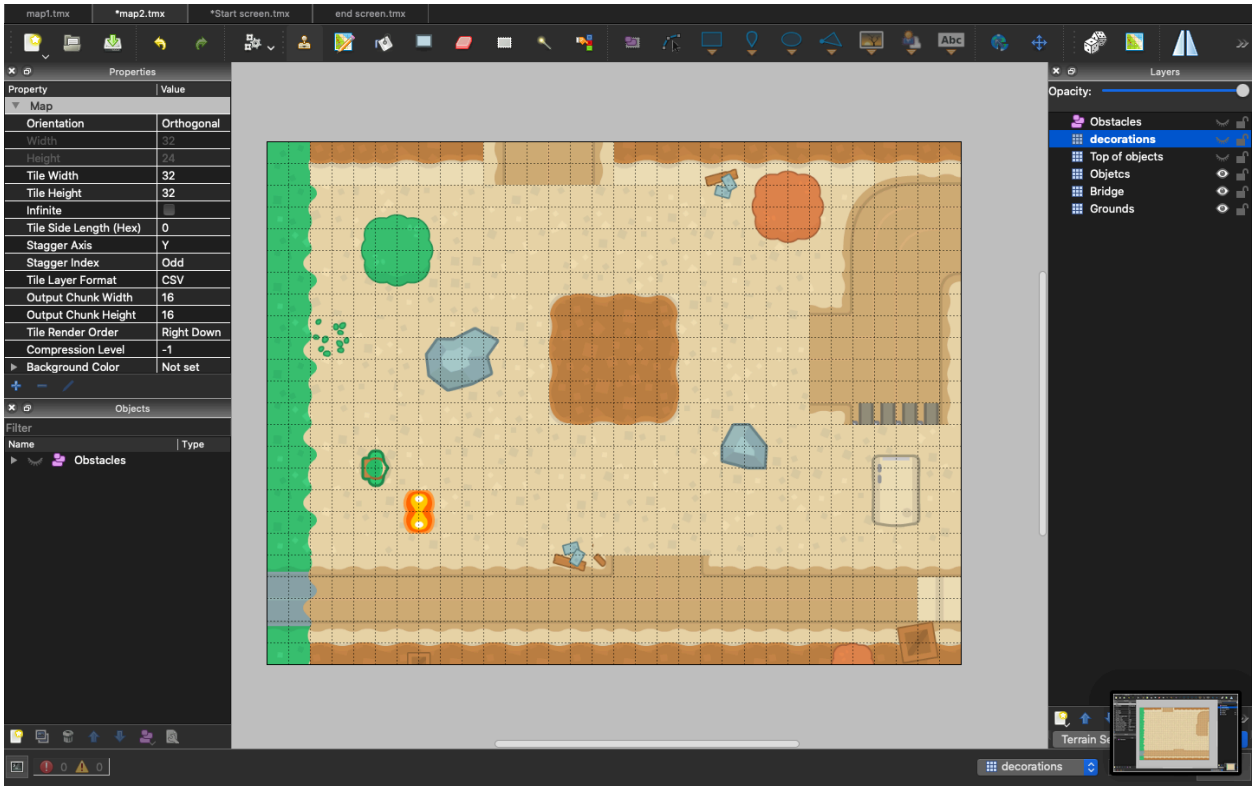
For example when the player reaches ‘0’ in the map, the player’s x coordinate would be 31 (since there are only 32 tiles across the x axis). When I deduct 30 the player’s x position would be 1 and the y coordinate will remain the same.

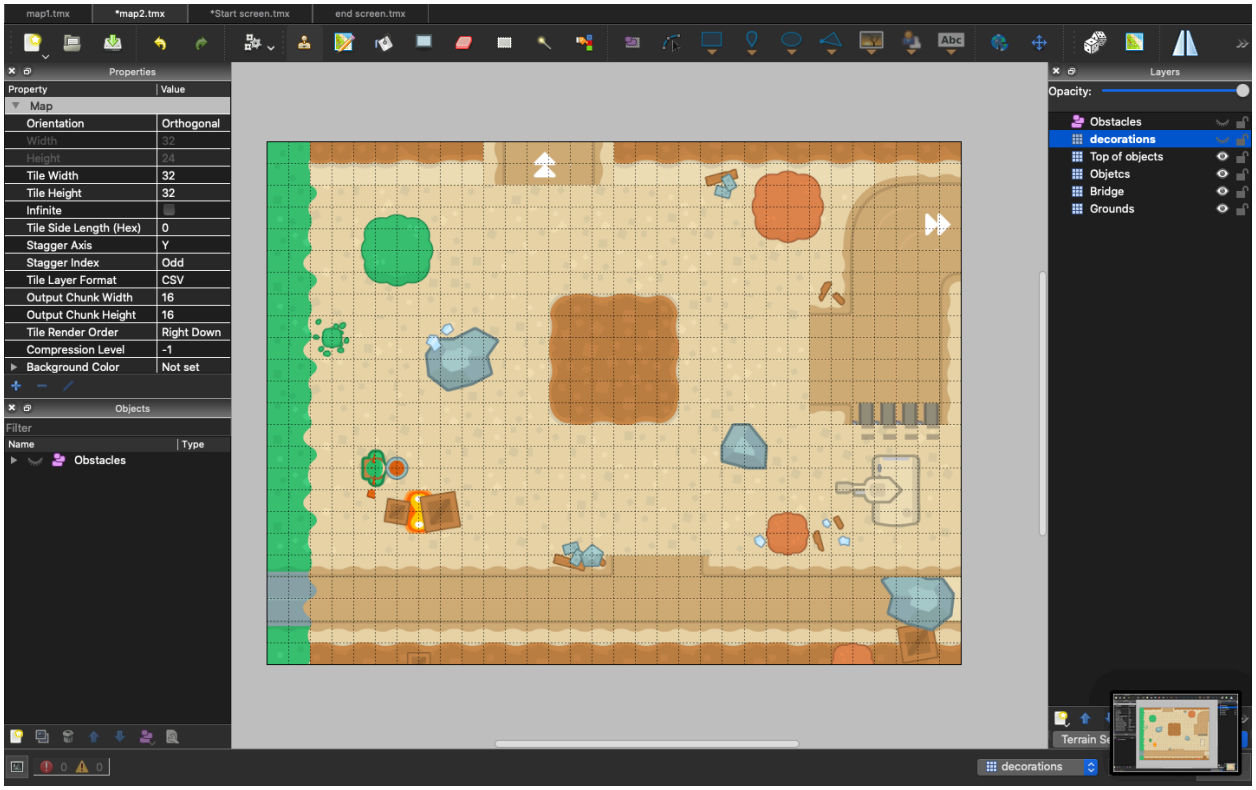
So now in the new map the player will spawn at (1, y position from previous map)

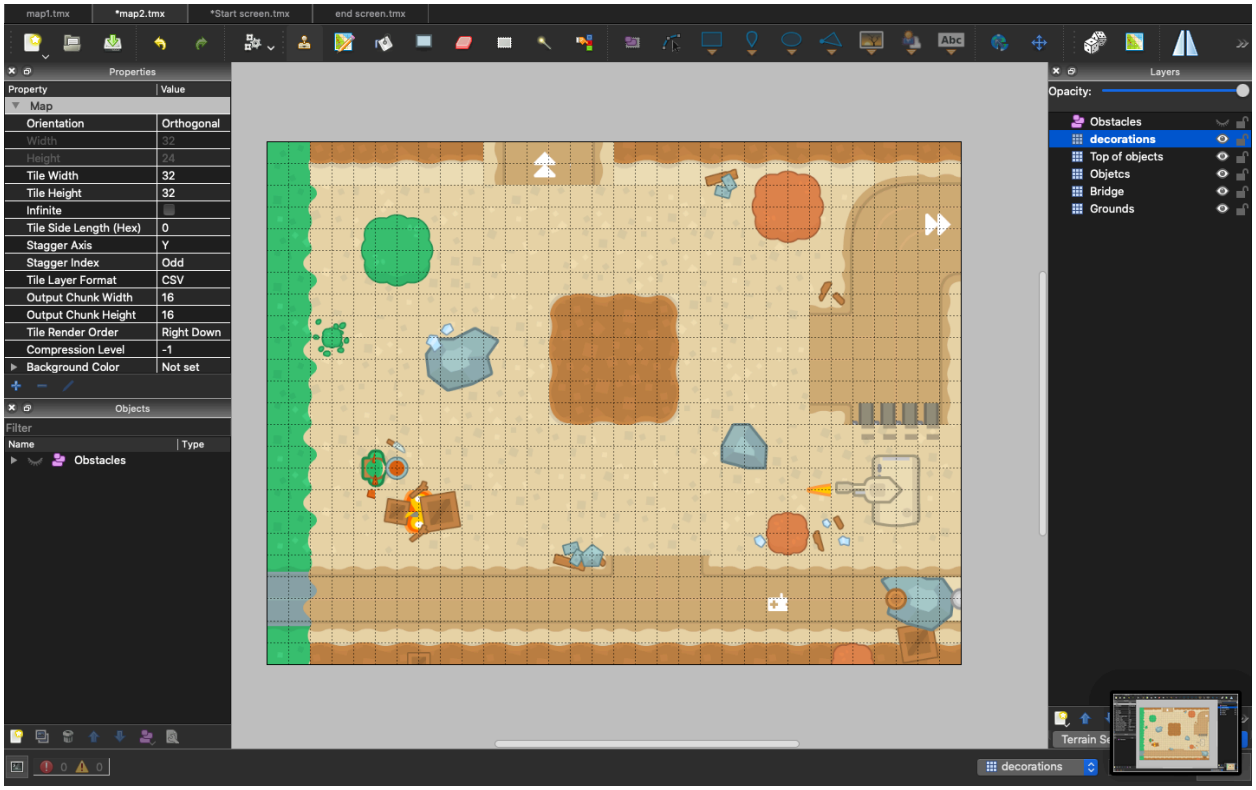
I have assigned the x and y variables for the new coordinates

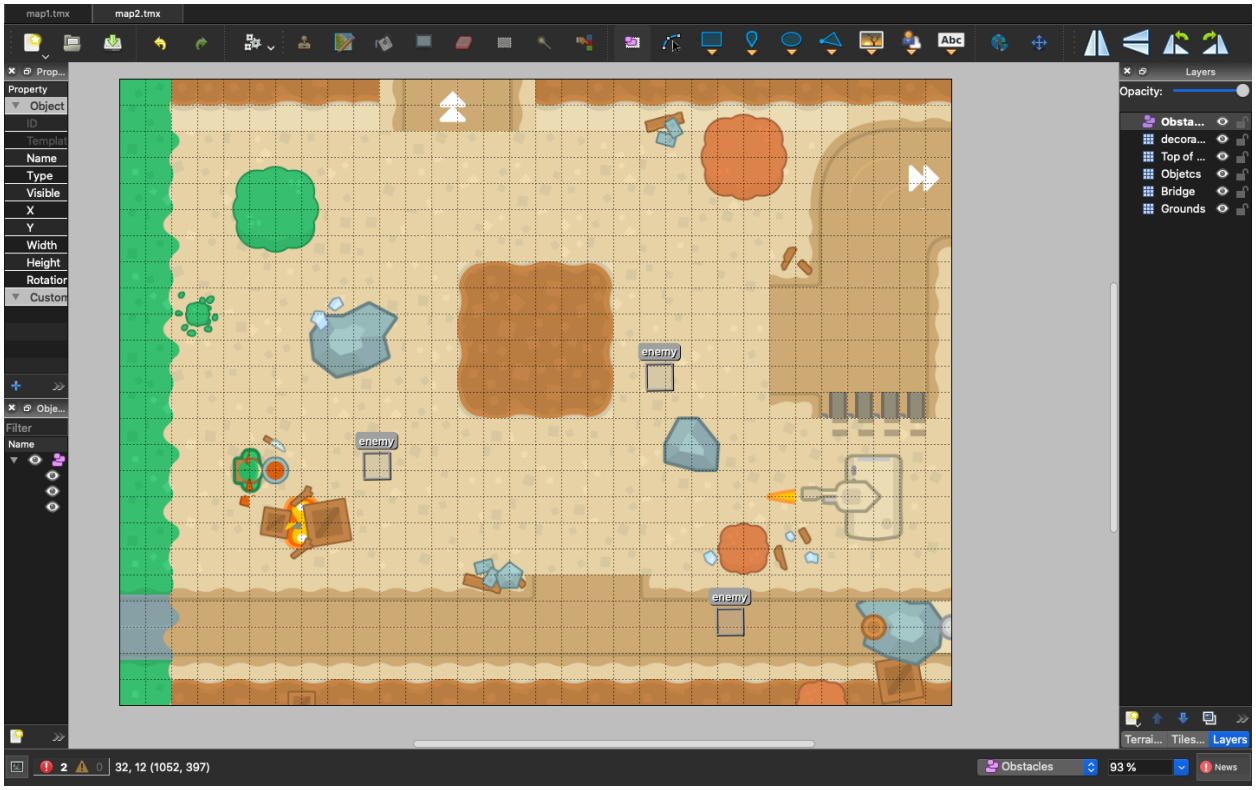

map1.tmx
*map2.tmx
*Start screen.tmx
end screen.tmx
Properties
Property
Value
Map
Orientation
Orthogonal
Width
32
Height
24
Tile Width
32
Tile Height
32
Infinite
Tile Side Length (Hex)
0
Stagger Axis
Y
Stagger Index
Odd
Tile Layer Format
CSV
Output Chunk Width
16
Output Chunk Height
16
Tile Render Order
Right Down
Compression Level
-1
Background Color
Not set
Objects
Filter
Name
Type
Obstacles
Layers
Opacity:
Obstacles
decorations
Top of objects
Objetcs
Bridge
Grounds
Terrain Sets
Tilesets
Layers
decorations
78 %
News


map1.tmx
*map2.tmx
*Start screen.tmx
end screen.tmx
Properties
Property
Value
Map
Orientation
Orthogonal
Width
32
Height
24
Tile Width
32
Tile Height
32
Infinite
Tile Side Length (Hex)
0
Stagger Axis
Y
Stagger Index
Odd
Tile Layer Format
CSV
Output Chunk Width
16
Output Chunk Height
16
Tile Render Order
Right Down
Compression Level
-1
Background Color
Not set
Objects
Filter
Name
Type
Obstacles
Layers
Opacity:
Obstacles
decorations
Top of objects
Objetcs
Bridge
Grounds
decorations


map1.tmx
*map2.tmx
*Start screen.tmx
end screen.tmx
Properties
Property
Value
Map
Orientation
Orthogonal
Width
32
Height
24
Tile Width
32
Tile Height
32
Infinite
Tile Side Length (Hex)
0
Stagger Axis
Y
Stagger Index
Odd
Tile Layer Format
CSV
Output Chunk Width
16
Output Chunk Height
16
Tile Render Order
Right Down
Compression Level
-1
Background Color
Not set
Objects
Filter
Name
Type
Obstacles
Layers
Opacity:
Obstacles
decorations
Top of objects
Objetcs
Bridge
Grounds
decorations


map1.tmx
*map2.tmx
*Start screen.tmx
end screen.tmx
Properties
Property
Value
Map
Orientation
Orthogonal
Width
32
Height
24
Tile Width
32
Tile Height
32
Infinite
Tile Side Length (Hex)
0
Stagger Axis
Y
Stagger Index
Odd
Tile Layer Format
CSV
Output Chunk Width
16
Output Chunk Height
16
Tile Render Order
Right Down
Compression Level
-1
Background Color
Not set
Objects
Filter
Name
Type
Obstacles
Layers
Opacity:
Obstacles
decorations
Top of objects
Objetcs
Bridge
Grounds
decorations


map1.tmx
*map2.tmx
*Start screen.tmx
end screen.tmx
Properties
Property
Value
Map
Orientation
Orthogonal
Width
32
Height
24
Tile Width
32
Tile Height
32
Infinite
Tile Side Length (Hex)
0
Stagger Axis
Y
Stagger Index
Odd
Tile Layer Format
CSV
Output Chunk Width
16
Output Chunk Height
16
Tile Render Order
Right Down
Compression Level
-1
Background Color
Not set
Objects
Filter
Name
Type
Obstacles
Layers
Opacity:
Obstacles
decorations
Top of objects
Objetcs
Bridge
Grounds
decorations



I created a new map called map2.tmx and made walls for the obstacle and saved it as map2.txt. So now the map 2 is also ready to be used

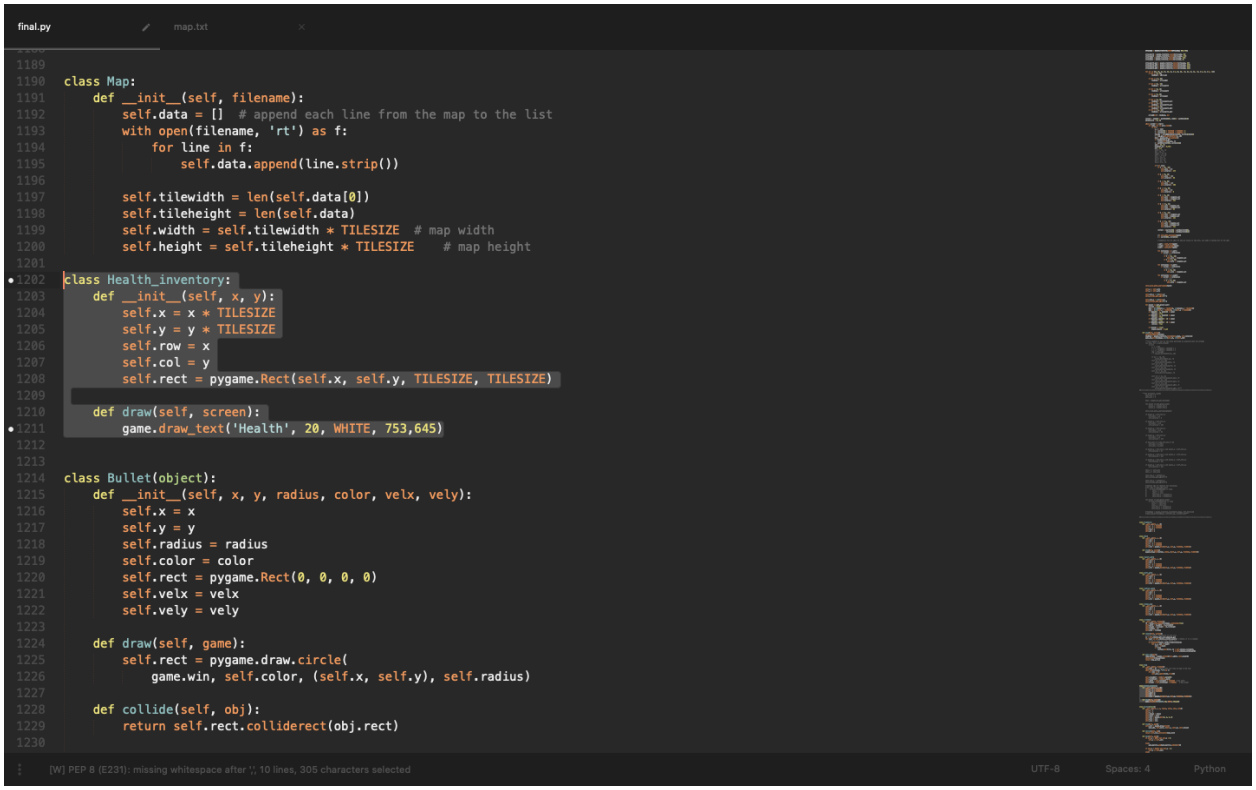
I made an instance of the player, enemy and maps by calling out the function ‘load_data’ and passing on the arguments of the new map (map2.tmx and map2.txt), player_image_2, gun_image_2, x, y, health (player’s health from previous map) and enemy_image_2.

I have also allowed the player to go back to the original map (map1) by going close to the beginning of the road (bottom left) by making an instance for each object like I have done in map 2 but with level 1 images. And the player will be spawning at (30, y position) so that it will appear to be smooth when moving between maps


pygame window



Now the player is able to move back and forth between map1 and map2



I created a new class ‘health-inventory’ that has x, y, rect, and a draw function that will draw the text ‘health’ on the map.

```
final.py                                    map.txt

58
59          self.player_list.append(Player(x, y, player_image,gun_image, health))
60          self.player = Player(x, y, player_image,gun_image, health)
61
62          for tile_object in self.map_out.tmxdata.objects:
63              '''if tile_object.name == 'player':
64                  x = int(tile_object.x / TILESIZE)
65                  y = int(tile_object.y / TILESIZE)
66                  self.player_list.append(Player(x, y, player_image, health))
67                  self.player = Player(x, y, player_image, health)'''
68
69              if tile_object.name == 'enemy':
70                  x = int(tile_object.x / TILESIZE)
71                  y = int(tile_object.y / TILESIZE)
72                  self.enemy_list.append(Enemy(x, y, enemy_image))
73                  self.enemy = Enemy(x, y, enemy_image)
74
75              if tile_object.name == 'health':
76                  x = int(tile_object.x / TILESIZE)
77                  y = int(tile_object.y / TILESIZE)
78                  self.health_list.append(Health_inventory(x, y))
79                  self.health_inventory = Health_inventory(x, y)
80
81
82
83
84          for row, tiles in enumerate(self.map.data):
85              for col, tile in enumerate(tiles):
86                  if tile == '1':
87                      self.wall_list.append(Wall(col, row))
88                      self.wall = Wall(col, row)
89
90                  if tile == '2':
91                      self.top_left_list.append(topLeft_wall(col, row))
92                      self.topleft = topLeft_wall(col, row)
93
94                  if tile == '3':
95                      self.top_right_list.append(topRight_wall(col, row))
96                      self.topright = topRight_wall(col, row)
97
98                  if tile == '4':
99                      self.bottom_left_list.append(bLeft_wall(col, row))

5 lines, 271 characters selected                                    UTF-8    Spaces: 4    Python
```

In map 2 I have created an obstacle named ‘health’, and made an instance of ‘health_inventory’ at that location and added it to the list.

```
final.py                                    map.txt

321             if current_map == 'map2.tmx':
322                 if (posx, posy) == (cx, cy):
323                     x, y = posx + 30, posy
324                     game.load_data('map1.tmx', 'map.txt',player_image_1,gun_image_1, x, y, self.health, enemy_image_1)
325
326     def draw_health(self, screen):
327         if self.health > 60:
328             col = GREEN
329         elif self.health > 30:
330             col = YELLOW
331         else:
332             col = RED
333
334         width = int(self.rect.width * self.health / 100)
335
336         self.health_bar = pygame.Rect(
337             int(self.x), int(self.y - 15), width, 8)
338
339         if self.health < 100:
340             pygame.draw.rect(screen, col, self.health_bar)
341
342         if self.health <= 0:
343             game.playing = False
344
345     def get_health(self, game):
346         posx = int(self.x)
347         posy = int(self.y)
348         for enemy in game.enemy_list:
349             enemy_pos = enemy
350
351         for health in game.health_list:
352             health_x = int(health.x)
353             health_y = int(health.y)
354
355             if enemy.collide(health):
356                 break
357             else:
358                 if self.collide(health):
359                     self.health += 0.5
360                     if self.health >= 100:
361                         self.health = 100
362
363     def collide_with_walls(self, dir):

17 lines, 512 characters selected                                    UTF-8    Spaces: 4    Python
```

From the ‘health_inventory’ list I created, I have looped through the list and assigned the variable ‘health_x’ and ‘health_y’ to get the coordinates of it on the map.

I am using the collide function I used at the start of prototype 2 to check collisions between two rects to find if an object has collided with another.

If the enemy is colliding with the health object then I did not allow the player to increase its health, but if the enemy is not colliding and if the player is colliding with the health object coordinates (health_x, health_y) then I have set the health to increase by 0.5.

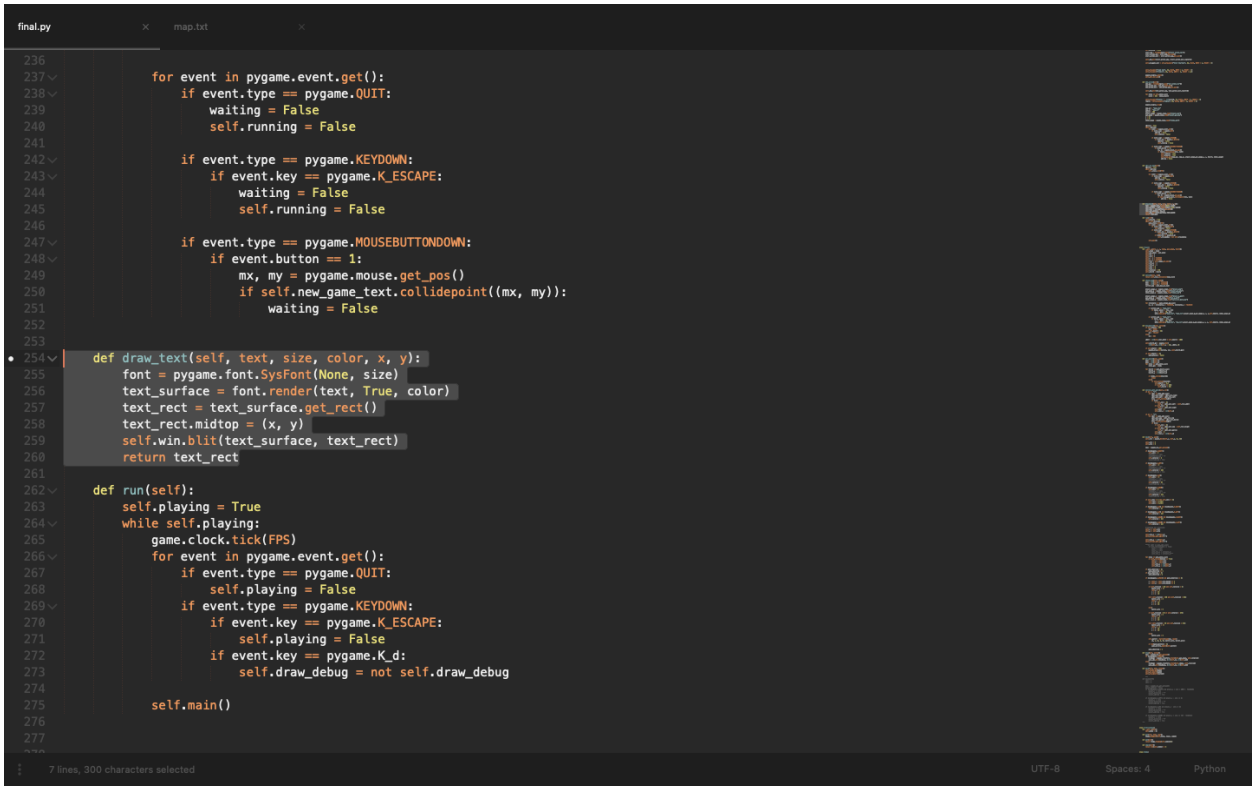
I have set a limit of 100 so that player won’t be able to increase the health more than 100


pygame window
Health



Once the player collides with the health rect, the player's health increases by 0.5.

Drawing start screen and end screen for the game



This ‘draw_text’ function is in my ‘Game’ class, it draws a text on the screen, which takes in the following arguments:

‘text’ - the actual text to be written on the screen

‘Size’ - the size of the font

‘color’ - rgb color for the text

‘x’, ‘y’ - coordinates for the text to appear on the screen

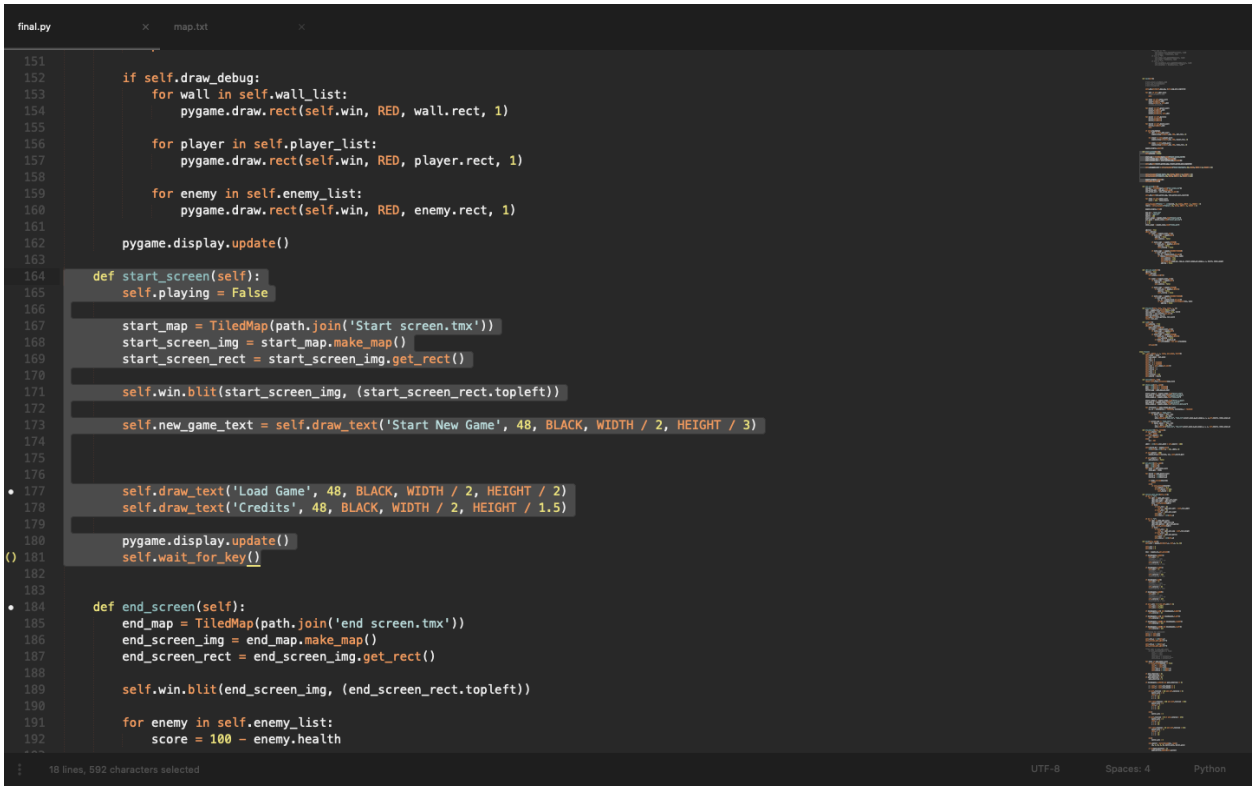
I have assigned the font with the default style and the ‘size’ from the arguments I have passed

‘text_surface’ renders the text with the color I pass in

‘text_rect’ gets the x, y, width and height of the text

And then I have blit the ‘text_surface’ at the ‘text_rect’

I have also returned the text_rect for future purpose



```
        if self.draw_debug:
            for wall in self.wall_list:
                pygame.draw.rect(self.win, RED, wall.rect, 1)

            for player in self.player_list:
                pygame.draw.rect(self.win, RED, player.rect, 1)

            for enemy in self.enemy_list:
                pygame.draw.rect(self.win, RED, enemy.rect, 1)

        pygame.display.update()

    def start_screen(self):
        self.playing = False

        start_map = TiledMap(path.join('Start screen.tmx'))
        start_screen_img = start_map.make_map()
        start_screen_rect = start_screen_img.get_rect()

        self.win.blit(start_screen_img, (start_screen_rect.topleft))

        self.new_game_text = self.draw_text('Start New Game', 48, BLACK, WIDTH / 2, HEIGHT / 3)



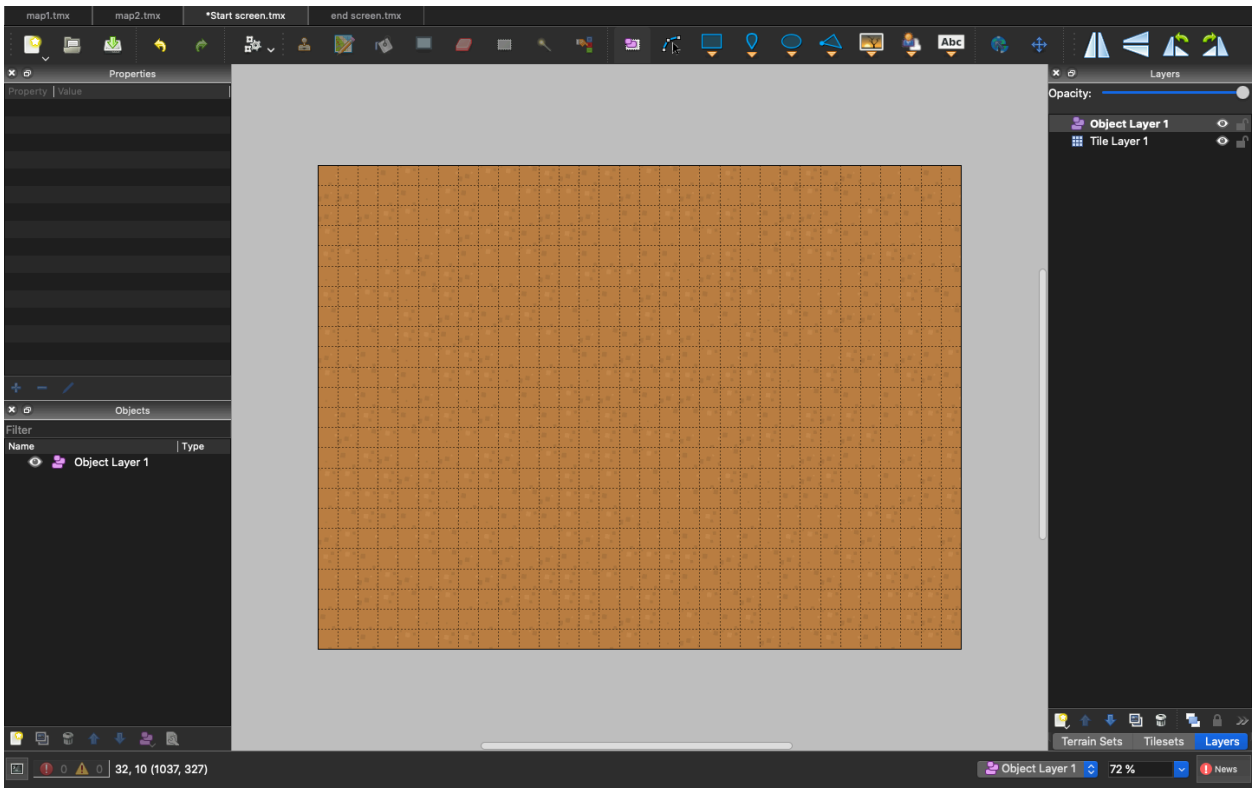
        self.draw_text('Load Game', 48, BLACK, WIDTH / 2, HEIGHT / 2)
        self.draw_text('Credits', 48, BLACK, WIDTH / 2, HEIGHT / 1.5)

        pygame.display.update()
        self.wait_for_key()


    def end_screen(self):
        end_map = TiledMap(path.join('end screen.tmx'))
        end_screen_img = end_map.make_map()
        end_screen_rect = end_screen_img.get_rect()

        self.win.blit(end_screen_img, (end_screen_rect.topleft))

        for enemy in self.enemy_list:
            score = 100 - enemy.health
```

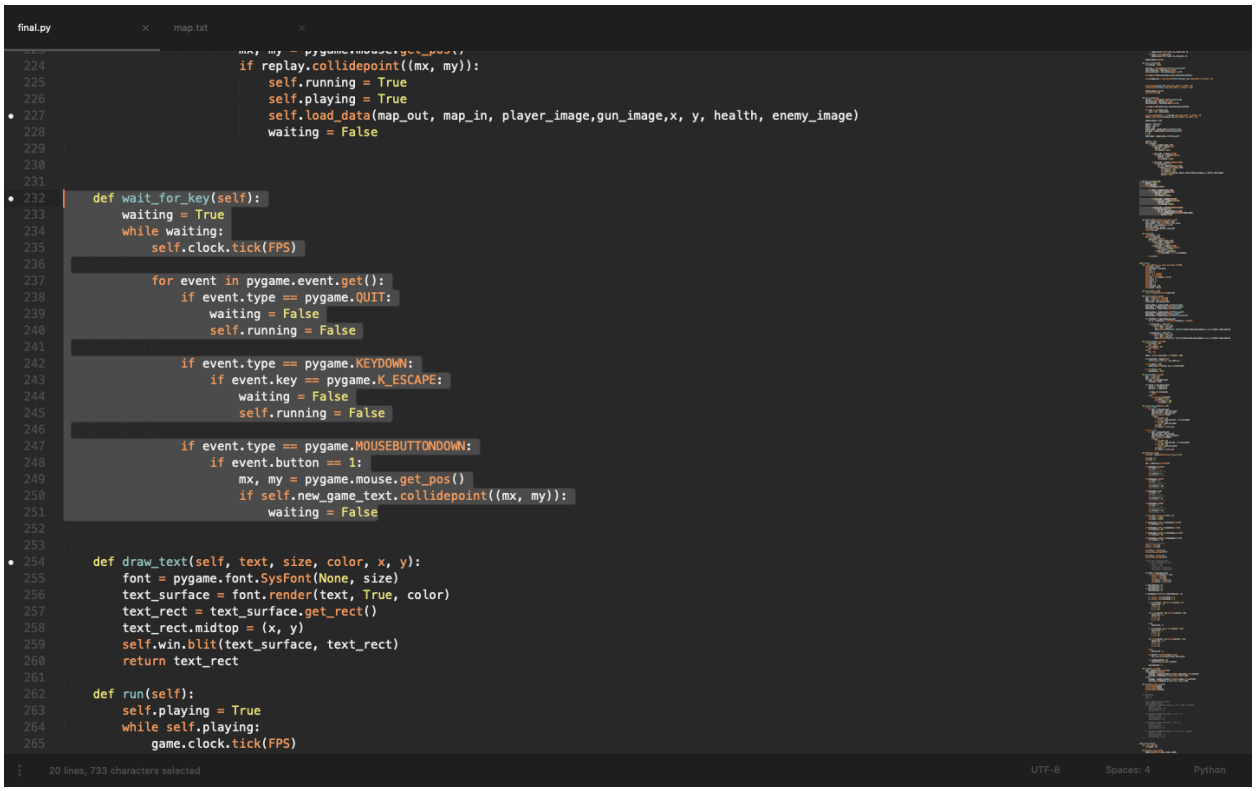


I created a normal background for the start screen, so that I can blit it onto the screen when the game begins

I have assigned the variable ‘new_game_text’ to draw the text ‘Start New Game’ onto the screen

I have also drawn ‘Load Game’ and ‘Options’ for the menu

I have called the function ‘wait_for_key’ so that the game won’t begin unless the user does some action

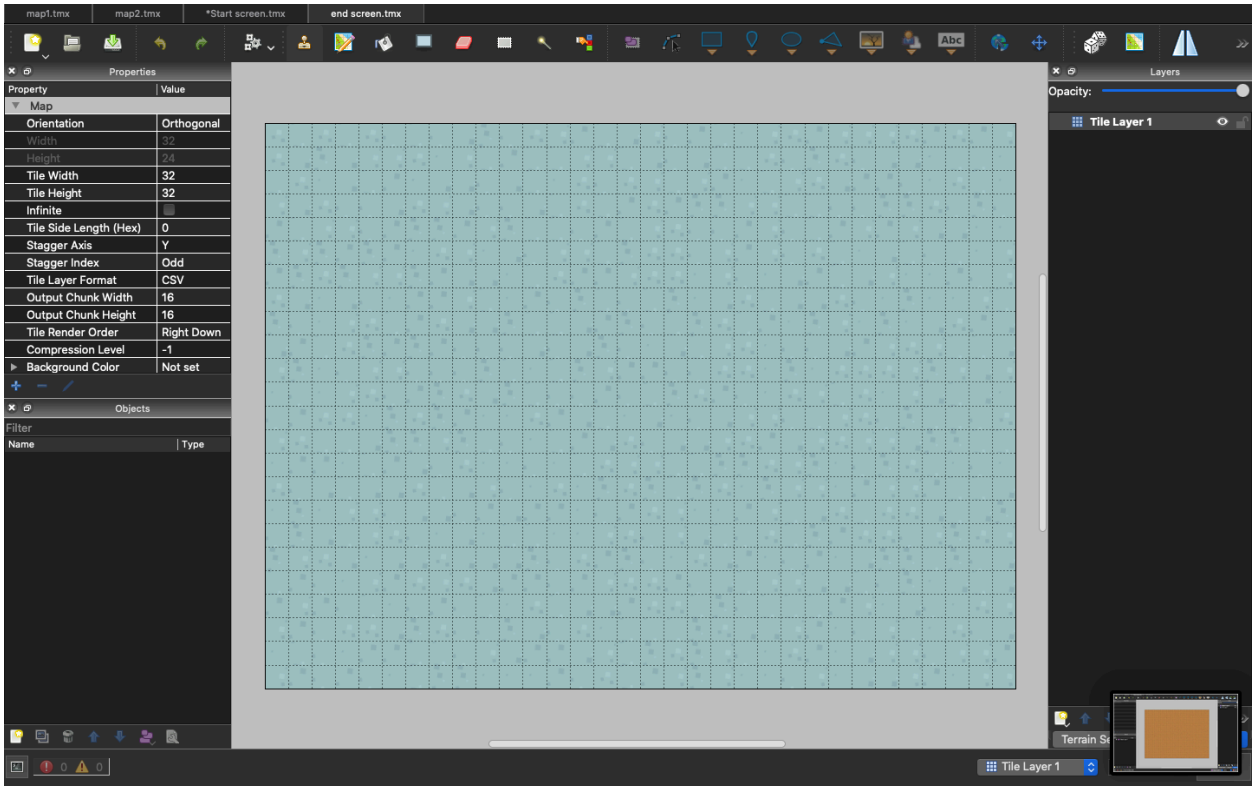


This is my ‘wait_for_key’ function, when the user presses quit button or presses the escape button on the keyboard the game will end.

I have allowed for mouse clicks and assigned the variable ‘mx’, ‘my’ to get the coordinates of my mouse position, then from the text I drew on the screen (‘Start New Game’), I will check if the user's mouse coordinate (mx, my) is clicked at the rect of that text. If so then It will stop the waiting loop



When I click on ‘Start New Game’ the game begins



I made this background for the end screen

final.py × map.txt ×

```
184     def end_screen(self):
185         end_map = TiledMap(path.join('end screen.tmx'))
186         end_screen_img = end_map.make_map()
187         end_screen_rect = end_screen_img.get_rect()
188 
189         self.win.blit(end_screen_img, (end_screen_rect.topleft))
190 
191         for enemy in self.enemy_list:
192             score = 100 - enemy.health
193 
194         self.draw_text('Score: ' + str(score), 48, BLACK, WIDTH / 2, HEIGHT / 3)
195         replay = self.draw_text('Replay', 48, BLACK, WIDTH / 2, HEIGHT / 2)
196 
197         pygame.display.flip()
198 
199         map_out = 'map1.tmx'
200         map_in = 'map.txt'
201         health = 100
202         player_image = pygame.image.load('player.png')
203         gun_image = pygame.image.load('player_gun.png')
204         x = 20
205         y = 21
206         enemy_image = pygame.image.load('enemy.png')
207 
208 
209         waiting = True
210         while waiting:
211             for event in pygame.event.get():
212                 if event.type == pygame.QUIT:
213                     waiting = False
214                     self.running = False
215 
216                 if event.type == pygame.KEYDOWN:
217                     if event.key == pygame.K_ESCAPE:
218                         waiting = False
219                         self.running = False
220 
221                 if event.type == pygame.MOUSEBUTTONDOWN:
222                     if event.button == 1:
223                         mx, my = pygame.mouse.get_pos()
224                         if replay.collidepoint((mx, my)):
225                             self.running = True
```

Line 190, Column 1 UTF-8 Spaces: 4 Python

final.py × map.txt ×

```
188 
189         self.win.blit(end_screen_img, (end_screen_rect.topleft))
190 
191         for enemy in self.enemy_list:
192             score = 100 - enemy.health
193 
194         self.draw_text('Score: ' + str(score), 48, BLACK, WIDTH / 2, HEIGHT / 3)
195         replay = self.draw_text('Replay', 48, BLACK, WIDTH / 2, HEIGHT / 2)
196 
197         pygame.display.flip()
198 
199         map_out = 'map1.tmx'
200         map_in = 'map.txt'
201         health = 100
202         player_image = pygame.image.load('player.png')
203         gun_image = pygame.image.load('player_gun.png')
204         x = 20
205         y = 21
206         enemy_image = pygame.image.load('enemy.png')
207 
208 
209         waiting = True
210         while waiting:
211             for event in pygame.event.get():
212                 if event.type == pygame.QUIT:
213                     waiting = False
214                     self.running = False
215 
216                 if event.type == pygame.KEYDOWN:
217                     if event.key == pygame.K_ESCAPE:
218                         waiting = False
219                         self.running = False
220 
221                 if event.type == pygame.MOUSEBUTTONDOWN:
222                     if event.button == 1:
223                         mx, my = pygame.mouse.get_pos()
224                         if replay.collidepoint((mx, my)):
225                             self.running = True
226                             self.playing = True
227                             self.load_data(map_out, map_in, player_image,gun_image,x, y, health, enemy_image)
228                             waiting = False
229 
230 
```

Line 190, Column 1 UTF-8

This is my ‘end_screen’ function. Which is similar to the start screen function

For now I have built a simple scoring system that produces the health taken away from the enemy so that I would be able to see how the screen would look like when the game ends.

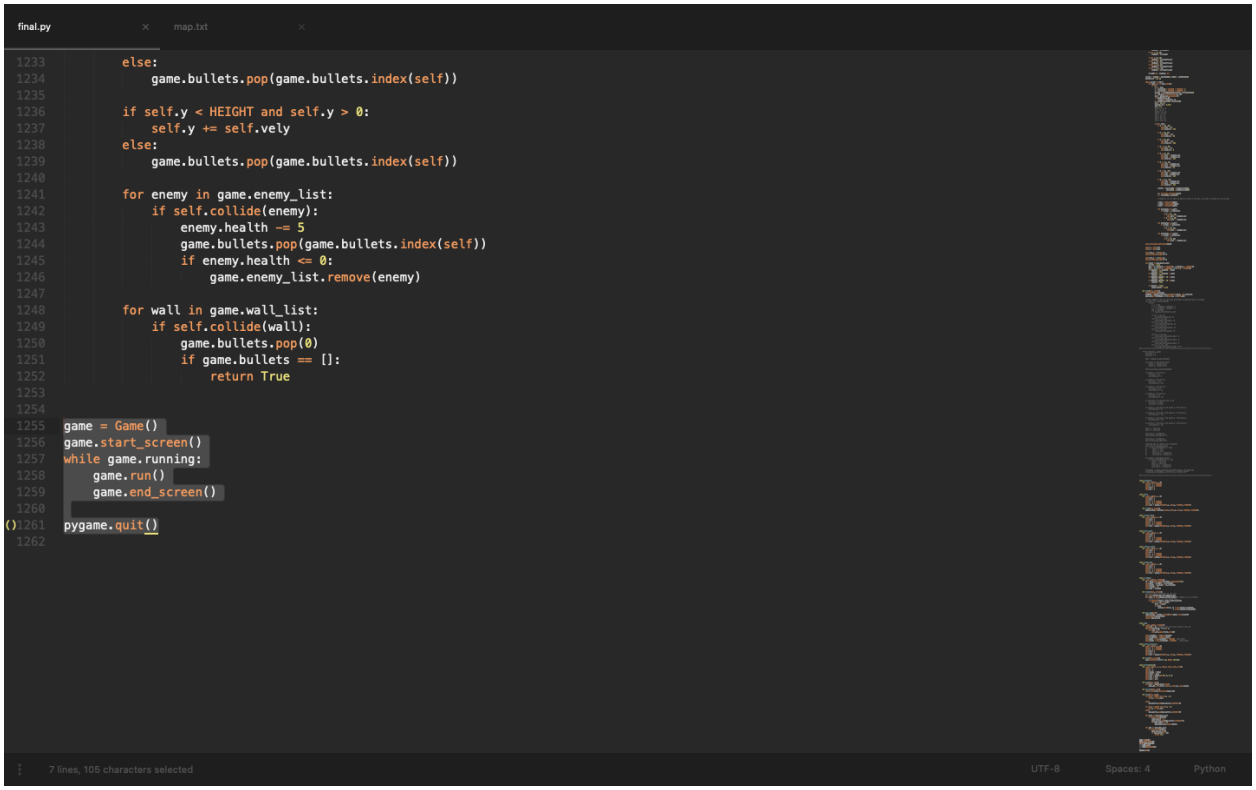
I have drawn a text ‘Score:’ on the screen which will show the score earned by the player.

And a ‘Play again’ text so that when the player presses this text the player should be able to play the game again.

I am running through a waiting loop again just like the ‘wait_for_key’ function but with different functions.

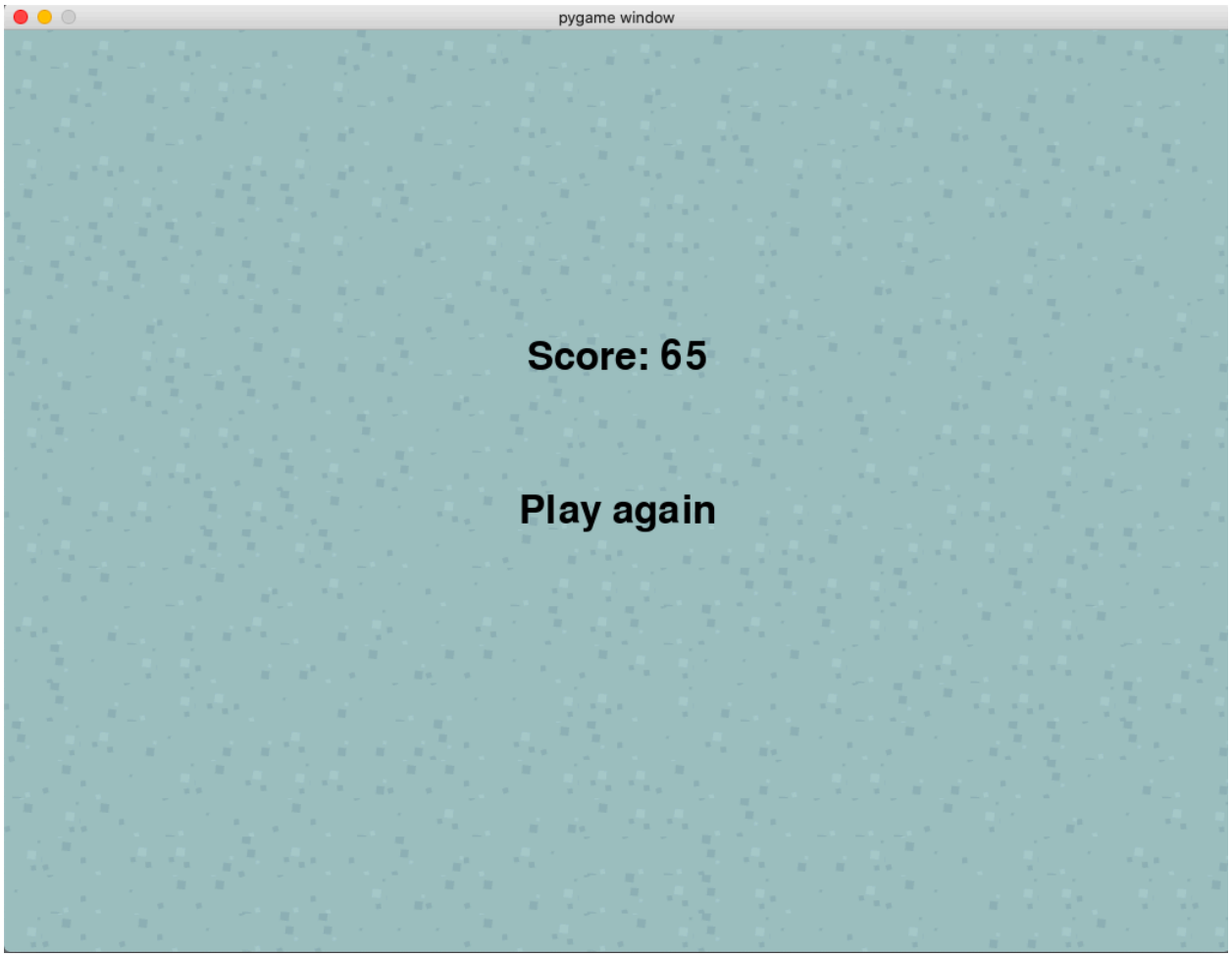
If the player presses the escape button or quit button the game will end

But if the player presses the ‘Play again’ text that is on the screen I have made a new instance for the whole game by calling the ‘load_data’ function and passing on the arguments like I did at the start of the code



I have called the ‘start_screen’ function which will show the start screen to the user.

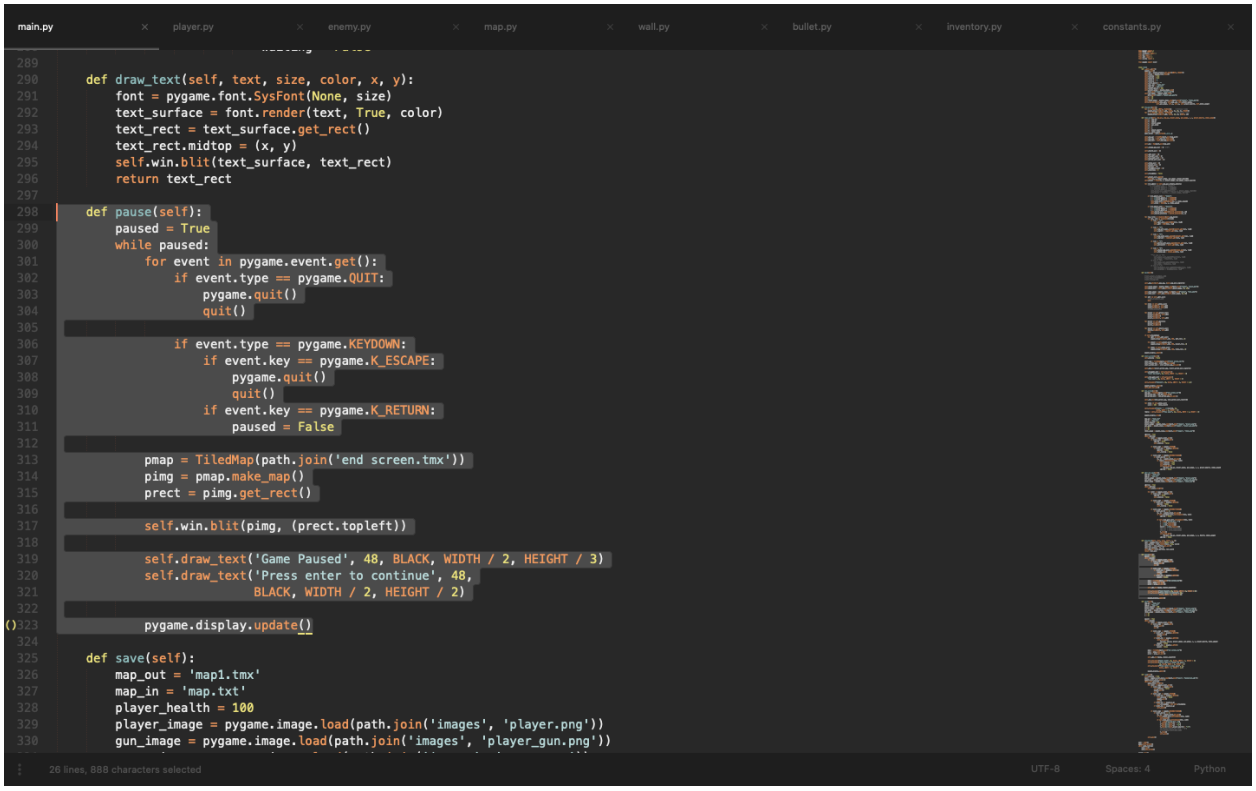
While the game is running I have called the ‘run’ function and then ‘end_screen’ function so that if the player presses the ‘Play again’ text ‘game.running’ will be true and the game will run again



This is my end screen, when I press the ‘Play again’ text the game restarts and works as it should.

New functions

Pause function

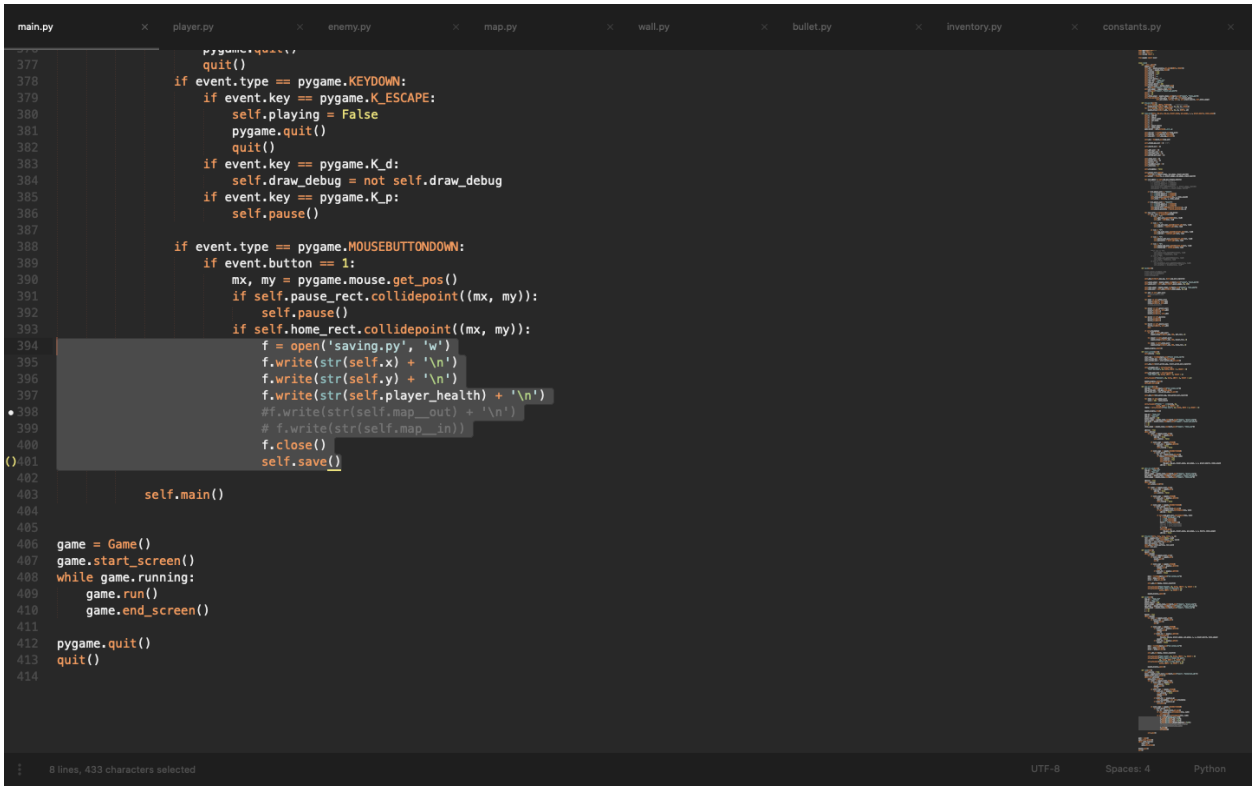


I have added a pause function on the game window so that the player would be able to pause and resume the game.

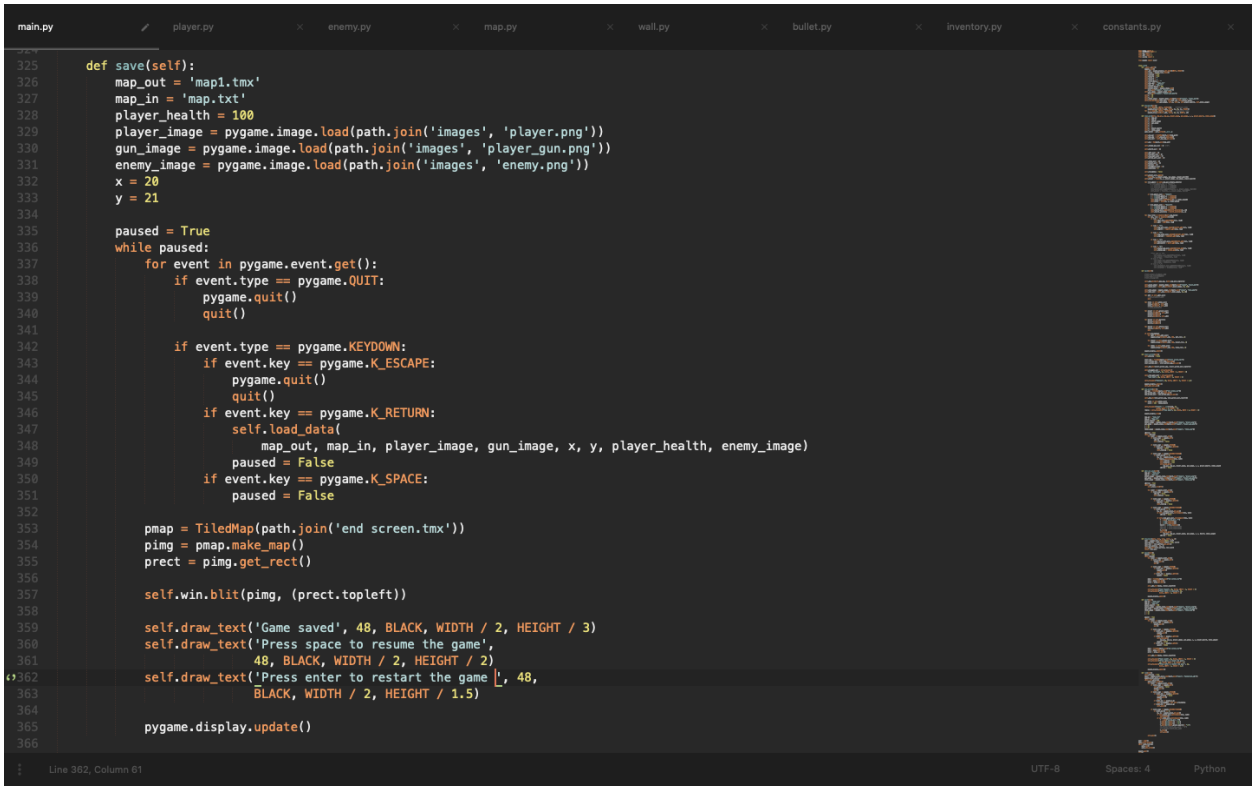
I have added text ‘Game Paused’ and ‘Press enter to continue’ on the ‘end screen’ image that I previously created from the tiled map editor.

I have assigned the variable ‘paused’ to True, while it is true I have checked for any inputs from the user. If the user wishes to quit the game by pressing the escape button or the close window icon the game will end. But if the player presses the return button the game will be resumed

Saving and loading the game



When the user presses the home button, I have created a python text file which will write the player's current health and its position onto the file.




```
    def save(self):
        map_out = 'map1.tmx'
        map_in = 'map.txt'
        player_health = 100
        player_image = pygame.image.load(path.join('images', 'player.png'))
        gun_image = pygame.image.load(path.join('images', 'player_gun.png'))
        enemy_image = pygame.image.load(path.join('images', 'enemy.png'))
        x = 20
        y = 21

        paused = True
        while paused:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    quit()

                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_ESCAPE:
                        pygame.quit()
                        quit()
                    if event.key == pygame.K_RETURN:
                        self.load_data(
                            map_out, map_in, player_image, gun_image, x, y, player_health, enemy_image)
                        paused = False
                    if event.key == pygame.K_SPACE:
                        paused = False

            pmap = TiledMap(path.join('end screen.tmx'))
            pimg = pmap.make_map()
            prect = pimg.get_rect()

            self.win.blit(pimg, (prect.topleft))

            self.draw_text('Game saved', 48, BLACK, WIDTH / 2, HEIGHT / 3)
            self.draw_text('Press space to resume the game',
                           48, BLACK, WIDTH / 2, HEIGHT / 2)
            self.draw_text('Press enter to restart the game ', 48,
                           BLACK, WIDTH / 2, HEIGHT / 1.5)

            pygame.display.update()
```

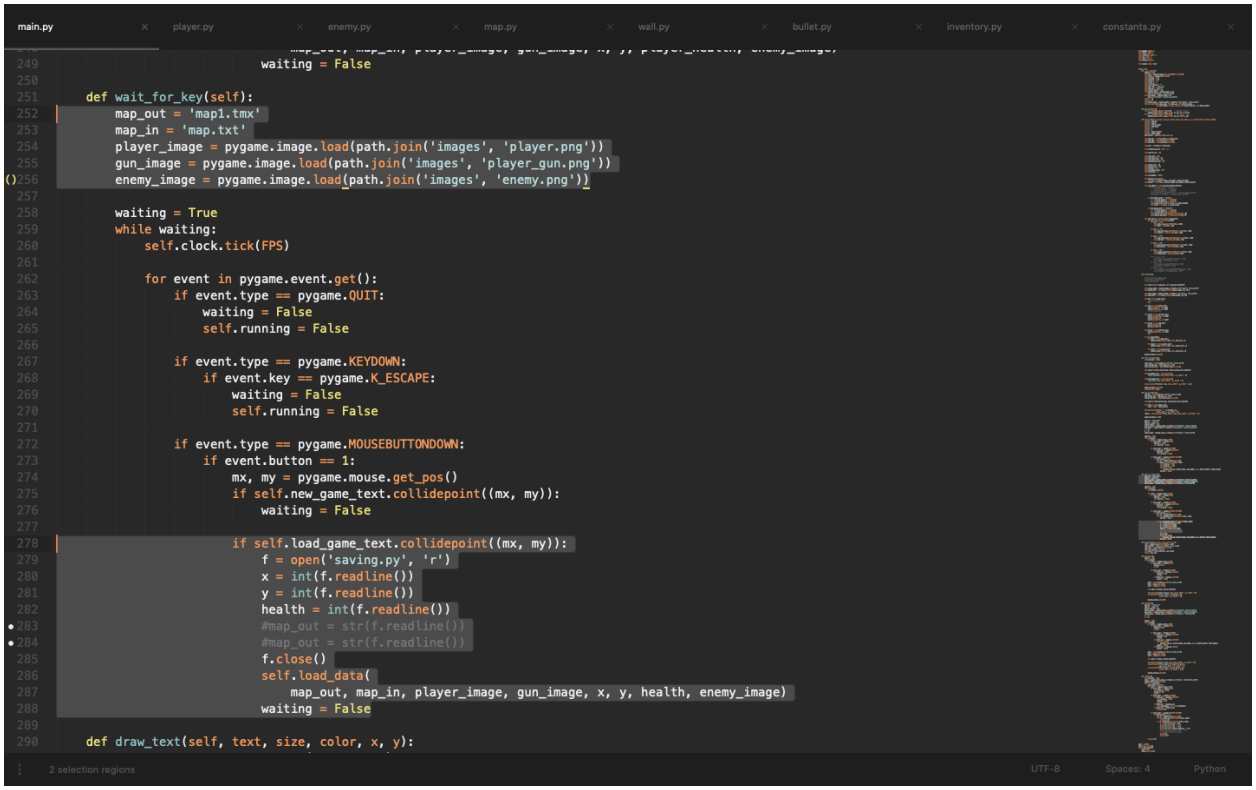
The save function is then called after the player's health and position is written onto the file.

I have added 'Game saved', 'Press space to continue' and 'Press enter to restart the game' texts on the screen.

While the game is paused it will check for any inputs from the user, if the user presses the escape button or quit icon the game will end. But if the player presses the return button I have created a new instance for the game by calling the 'load_data' function from the 'Game' class.

Therefore It will create a new game allowing the user to play the game from the beginning

If the player presses the space button 'paused' will be false hence the game will be resumed.



Assuming that the player has saved the game earlier

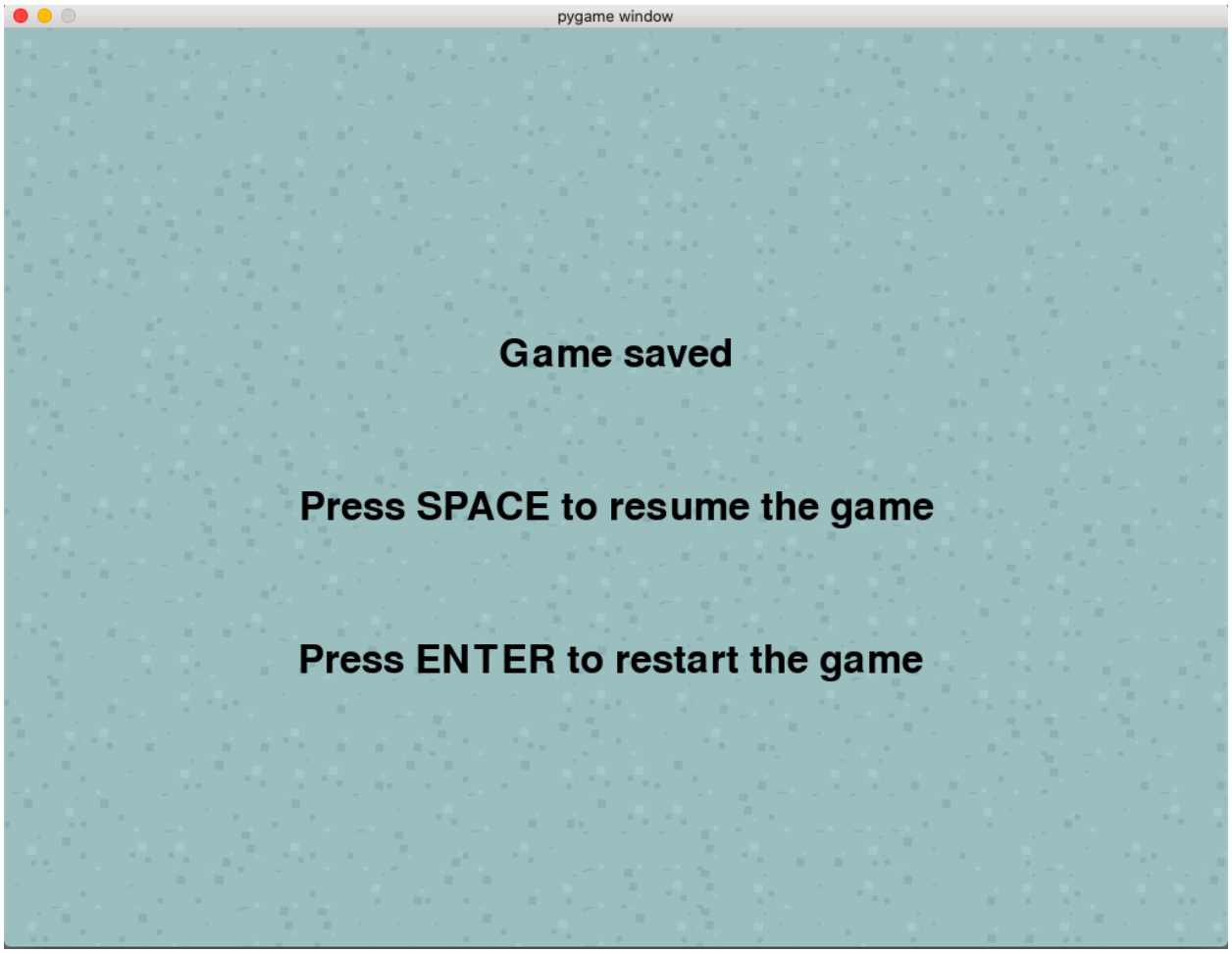
When the player opens up the game and clicks on the ‘Load game’ text, I have opened the python text file and assigned the variables ‘x’, ‘y’ and ‘health’ to retrieve the (x, y) coordinates and the health of the player at the time the user pressed the home icon.

Then I have called the ‘load_data’ method from the ‘Game’ class to make an instance of the game with the exact coordinates and health of the player when the user saved the game.


pygame window



When I press the pause icon this screen pops up. When I press enter button the game continues from where I left off



When I press the home icon this screen pops up, when I press the space button the game resumes and when I press the enter button the game restarts.

Usability features

The enemy can move around the map and chase the player while avoiding obstacles.

The new map can be blitted on the pygame window, the player and enemy can move around on the new map

When the player goes near the end of the road the map changes to a different map which appears to be like moving onto a next level.

The player can pause and resume the game

The player can save and load the game

The player can increase the health by going near the health object in ‘map 2’

The player can move around between the two maps having the same health.

The start screen allows the user to click ‘Start New Game’ so that the game can begin and when the player dies, the end screen shows the ‘score’ and ‘Start again’ text, When the player clicks the ‘Start again’ text, the game restarts.

Test data

The enemy had a movement issue where the enemy gets stuck at bottom left, top left and top right corners of the wall, I fixed it with a temporary fix (refer define solution fully), but that wouldn’t be an efficient way to solve this problem. I will be working on this in my next prototype.

I’ve been playing it for a while and also had my stakeholders play the game, all of the functions work well.

I tested the movement of the enemy in all eight directions, the map changes when the player reaches the bottom right and when the player goes back to map 1 the player spawns at bottom left.

The enemy can reduce the player’s health by colliding.

When I go near the health inventory object the player health increases by 0.5 and stops once it reaches 100.

Post development data

In my next prototype I will be working on a scoring system for example collecting coins, additional points when moving onto next level, additional points when the player kills / shoots the enemy etc..
I thought of having a gun shop in some map so that the player would be able to buy and sell guns that has different features in speed, image and the damage that has on the enemy

An inventory for the player would make the game more entertaining so the player would be able to pick up and drop knives, guns and other assets that may be required to enter some other level.

Also having objects such as water and fire on different maps so when the player passes through water the speed decreases and when the player steps into fire the health decreases.

I will also be adding instructions

Development

Testing

In the video clip below you will be able to see me testing the following:
Using the debug tool, movement of enemies, increasing the player's health when going near the health object, start screen, end screen and being able to move from one map to another back and forth.

game clip

Testing movement

##updated video clip showing pause, saving and loading functions

Updated Clip

Review

In this prototype I have allowed the enemy to move so that he can chase and reduce the player's health. As you have seen from the clip above it works perfectly fine.

The player is now able to move from one map to another giving a sense of new level in the game. I will be working on various different maps and layouts in the future.

The issue with enemies movement was fixed by my temporary fix but since it is not efficient, I will work on a suitable method in the next prototype.

The enemies image flickers around the corners of the wall, I will be also fixing that in my next prototype

Other than that the game works very well.

From this point I will be trying to make the game look better and make the code efficient.

Prototype 6

Design

Decomposition

Structure of solution

Key variables / Data structures / Classes

Define solution fully

Usability features

Test data

Post development data

Development

Testing

Review

Evaluation

Annotated evidence

You can view the full code by clicking the link below

Full code

Evaluation

Evaluating met criteria

The game has met the following success criterias:

The game is coded in object oriented programming having seperate folders for classes

The game is challenging as the player moves onto the next level, more enemies are spawned at each level

The appearance/layout of the game is exciting with the first level being a mini city raided by zombies, cars being crashed and oil spilled on the road. As the player moves onto the next level (map 2) there is a different scenario like a tank is shooting the zombie and cars. ## change the layout

The loading game option works, the user can save the game by pressing the home button and reload the game at the same position that the user left.

The player can pause and resume the game

The AI is very intelligent because the enemy finds the shortest path to reach the player using a star search algorithm. The enemy can move by following the path and attacking the enemy by reducing the player's health

This game has simple controls, arrow keys to move and spacebar to shoot.

This game has a simple menu system at the start and at the end

The player and enemy can collide with each other by reducing each other's health

The user has the option to pause and resume the game

The user has the option to turn on and off the background music

The player is able to move between the maps, as the player moves onto the new map the images of the characters and the map layout changes. Giving a sense of new level to the user

Level 2 (map 2) has a health pack so that the player would be able to increase the health by a fraction by going near the health object

Evaluating unmet criteria

The game isn't extremely challenging or difficult, the user doesn't need much of a strategy to win this game.

There are no animations such as walking, fighting, running

The game doesn't show highscores

The is no working 'options' category

The scoring system is simple, doesn't have any points table

There player cannot buy and sell guns yet

There is no additional inventory other than health.

There are no objects such as fire / water in the game yet

This game doesn't have one on one human battle feature

Maintenance issues and limitations

The game takes time to load the game, and when the player moves onto a new map the game consumes more time, It happens only once. The next time the player moves into a new map the game doesn't consume much time as it did for the first time.

The player slows down when the enemy goes far away from the player or when there are many walls in the map, the path finding algorithm expands as the player moves away from the enemy, causing the algorithm to consider every path in the map.

Development to fix limitations

The games performance can be improved by playing it on a computer with an 8GB RAM or more. So that it will take less time to load the game onto the screen.

A faster processor would solve the problem where the player slows down when the enemy moves far away from the player or when there are a lot of walls in the map.