Algorithm Programing

FINAL PROJECT: Pynal Maze

Made by:
Giancarlo Prawiro
L1AC | 2902671820

Introduction:

This is a simple maze game to show off what I have learned in class. I wanted to make a maze game as a callback to my first game I made in scratch which is also a maze game. I wanted the maze layout to be easily configurable which is why it uses tile maps. Initially I wanted to do a maze game with multiple levels and sprites but because of me not wanting to feature creep, over complicated things and time constraints I decided to scale back the scope and just make a simple maze game. I also used a tutorial on how to make an RPG but because I wanted to make it my own I used the tutorial as a foundation and made it a maze game.

Project Goal:

Make a maze game that has a player, a wall that will send the player back when they touch it, a goal so they can win and an easy config file so the player can change the layout using the tile map.

Project Specification:

Game mechanics:

- A player that can move around with the arrow keys
- A wall that when the player collides with it, it puts the player back at starting position
- A goal that the player can touch and win

Config file:

- A tilemap to help with the positions of the objects

Solution Design:

File Structure:

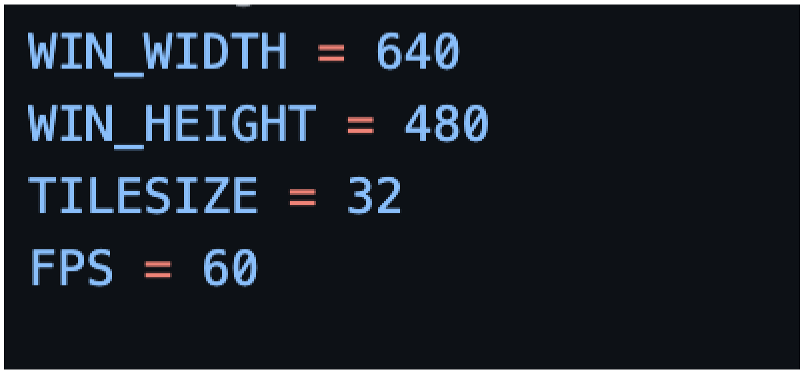
- config.py
- sprites.py
- main.py

How it works:

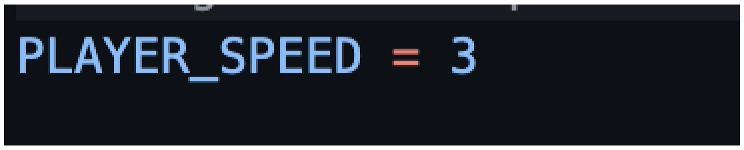
To make it easier to explain I will be explain it by the different files:

Config.py:
This file is to make everything easier to config so instead of finding each of the individual code I want to config I can just config this file

- Win is the window size, tile size is how big each object is and fps is the frames per second

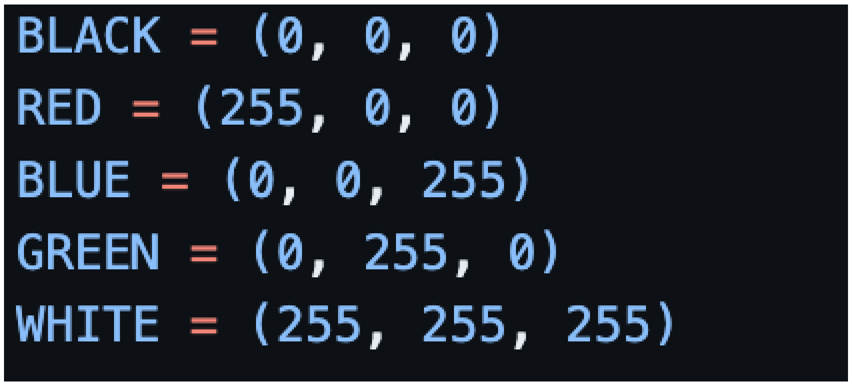


```
WIN_WIDTH = 640
WIN_HEIGHT = 480
TILESIZE = 32
FPS = 60
```



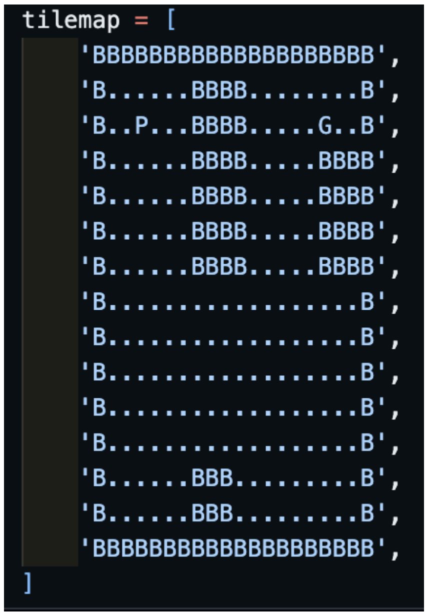
```
PLAYER_SPEED = 3
```

- How fast the players go
- This makes it easier to change color so instead of figuring out what number to use for a color I can just type the color



```
BLACK = (0, 0, 0)
RED = (255, 0, 0)
BLUE = (0, 0, 255)
GREEN = (0, 255, 0)
WHITE = (255, 255, 255)
```

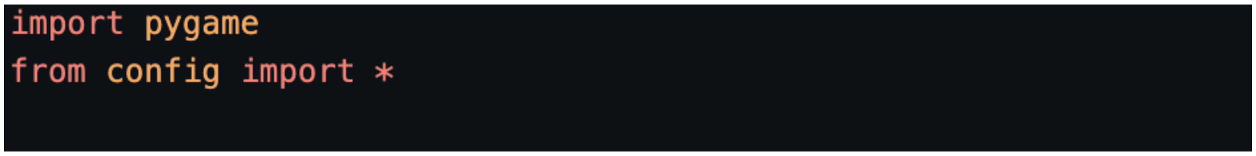
- Tile map makes it so the game can read the position of each object using a list, B is for the wall, P is for the player, G is for the goal and . is nothing



```
tilemap = [
    'BBBBBBBBBBBBBBBBBBBB',
    'B......BBBB........B',
    'B..P...BBBB.....G..B',
    'B......BBBB.....BBBB',
    'B......BBBB.....BBBB',
    'B......BBBB.....BBBB',
    'B......BBBB.....BBBB',
    'B..................B',
    'B..................B',
    'B..................B',
    'B..................B',
    'B..................B',
    'B......BBB.........B',
    'B......BBB.........B',
    'BBBBBBBBBBBBBBBBBBBB',
]
```

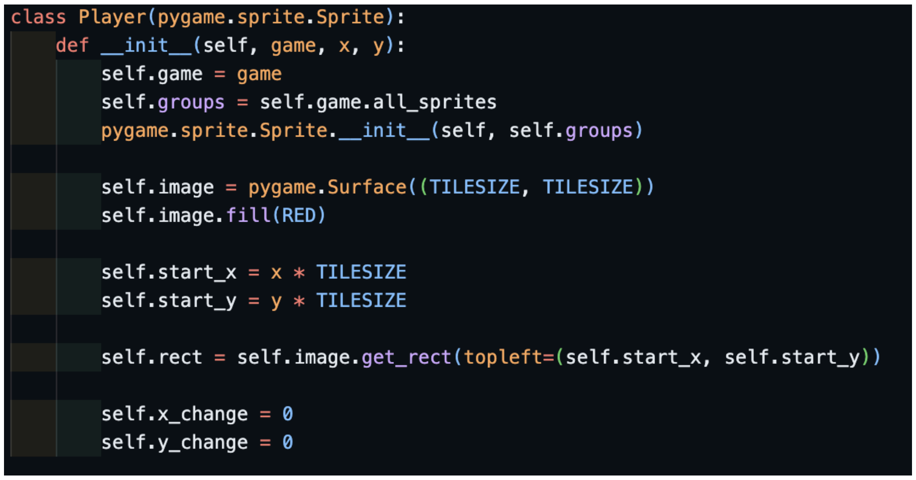
Sprite.py:

- This is to import the pygame module and to read the code from the config.py file

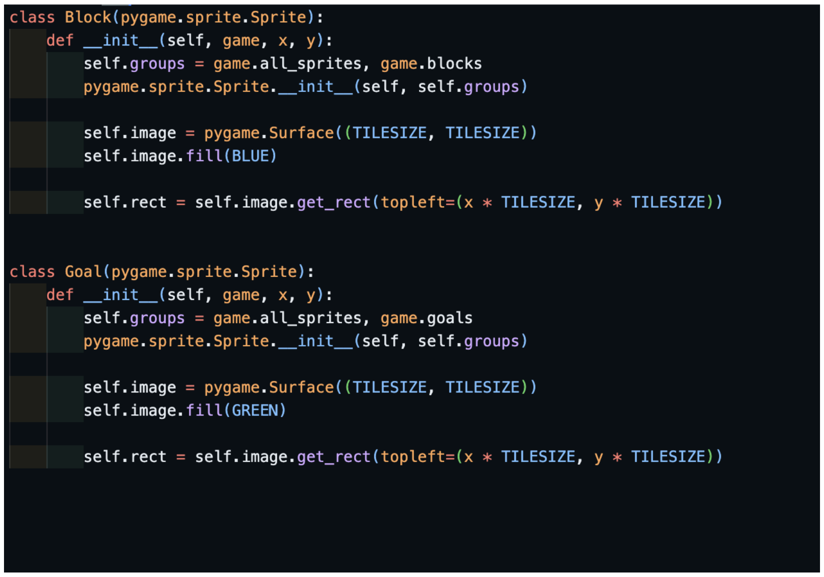


```
import pygame
from config import *
```

- This is to make the sprite and what the objects look like and size

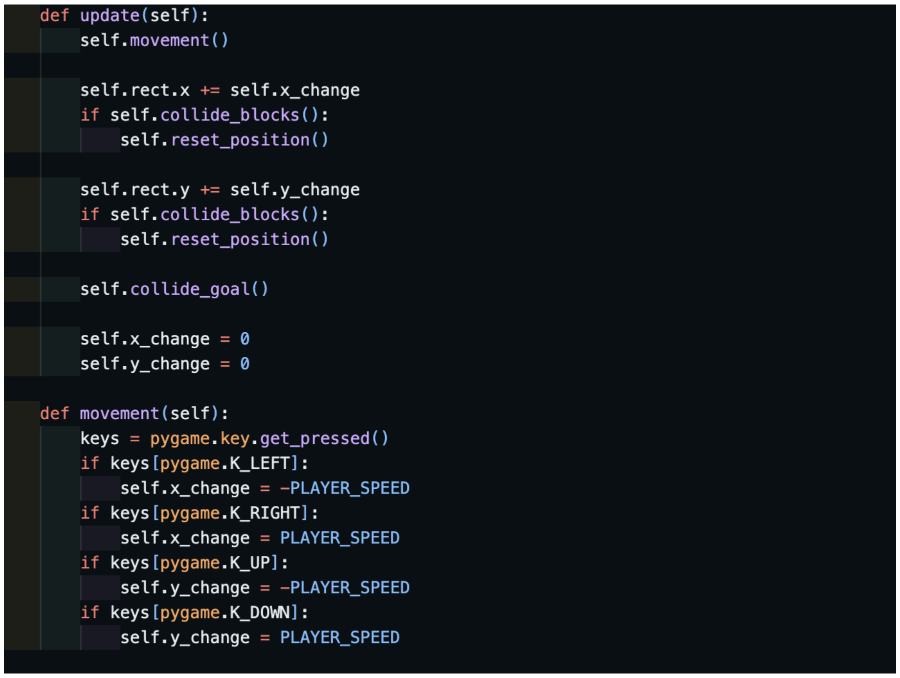


```
class Player(pygame.sprite.Sprite):
    def __init__(self, game, x, y):
        self.game = game
        self.groups = self.game.all_sprites
        pygame.sprite.Sprite.__init__(self, self.groups)

        self.image = pygame.Surface((TILESIZE, TILESIZE))
        self.image.fill(RED)

        self.start_x = x * TILESIZE
        self.start_y = y * TILESIZE

        self.rect = self.image.get_rect(topleft=(self.start_x, self.start_y))

        self.x_change = 0
        self.y_change = 0
```



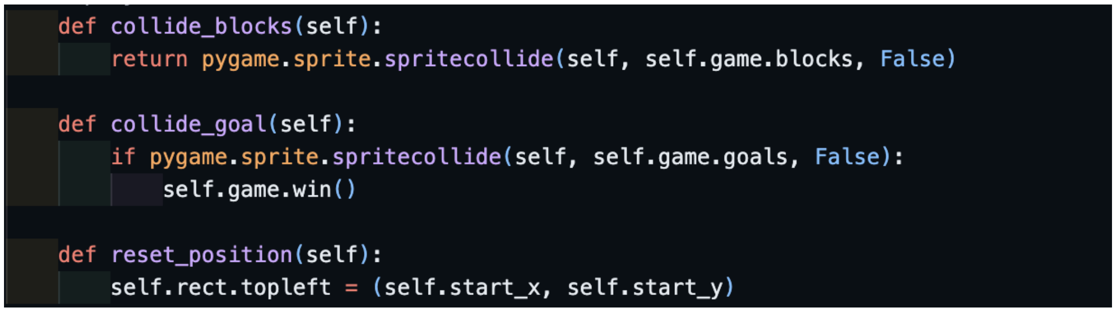
```python
class Block(pygame.sprite.Sprite):
    def __init__(self, game, x, y):
        self.groups = game.all_sprites, game.blocks
        pygame.sprite.Sprite.__init__(self, self.groups)

        self.image = pygame.Surface((TILESIZE, TILESIZE))
        self.image.fill(BLUE)

        self.rect = self.image.get_rect(topleft=(x * TILESIZE, y * TILESIZE))


class Goal(pygame.sprite.Sprite):
    def __init__(self, game, x, y):
        self.groups = game.all_sprites, game.goals
        pygame.sprite.Sprite.__init__(self, self.groups)

        self.image = pygame.Surface((TILESIZE, TILESIZE))
        self.image.fill(GREEN)

        self.rect = self.image.get_rect(topleft=(x * TILESIZE, y * TILESIZE))
```

- This is for the collision and player control



```python
def update(self):
    self.movement()

    self.rect.x += self.x_change
    if self.collide_blocks():
        self.reset_position()

    self.rect.y += self.y_change
    if self.collide_blocks():
        self.reset_position()

    self.collide_goal()

    self.x_change = 0
    self.y_change = 0

def movement(self):
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        self.x_change = -PLAYER_SPEED
    if keys[pygame.K_RIGHT]:
        self.x_change = PLAYER_SPEED
    if keys[pygame.K_UP]:
        self.y_change = -PLAYER_SPEED
    if keys[pygame.K_DOWN]:
        self.y_change = PLAYER_SPEED
```

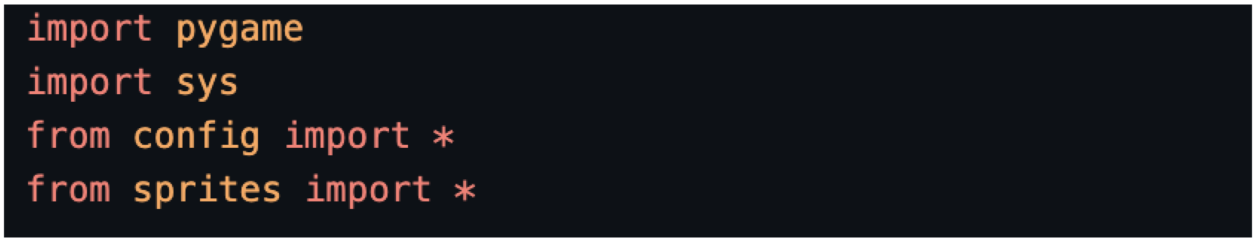
- Collision for the wall so the players get sent back when they touch the wall



```python
def collide_blocks(self):
    return pygame.sprite.spritecollide(self, self.game.blocks, False)

def collide_goal(self):
    if pygame.sprite.spritecollide(self, self.game.goals, False):
        self.game.win()

def reset_position(self):
    self.rect.topleft = (self.start_x, self.start_y)
```

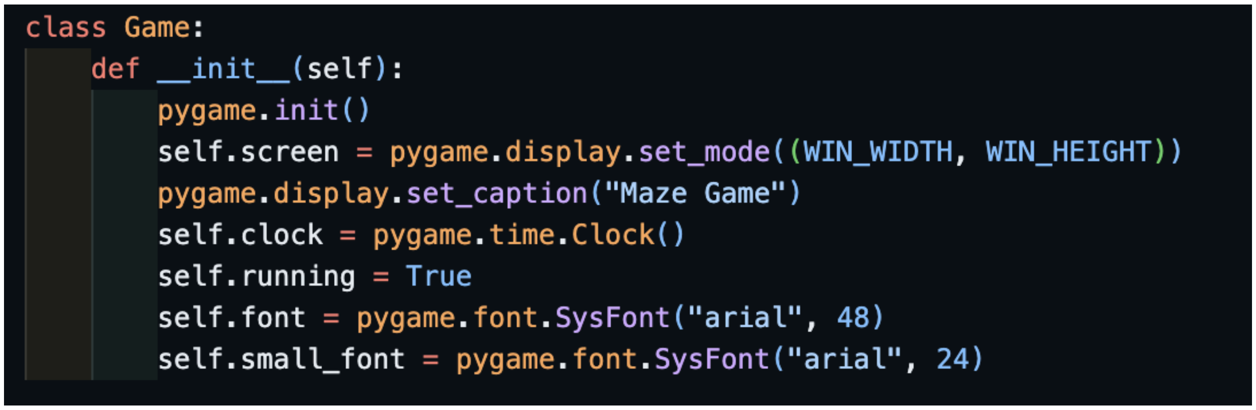
Main.py:

- This imports the pygame module as well as the sys module, the code also imports the config.py and sprites.py file so the file can get the read the code



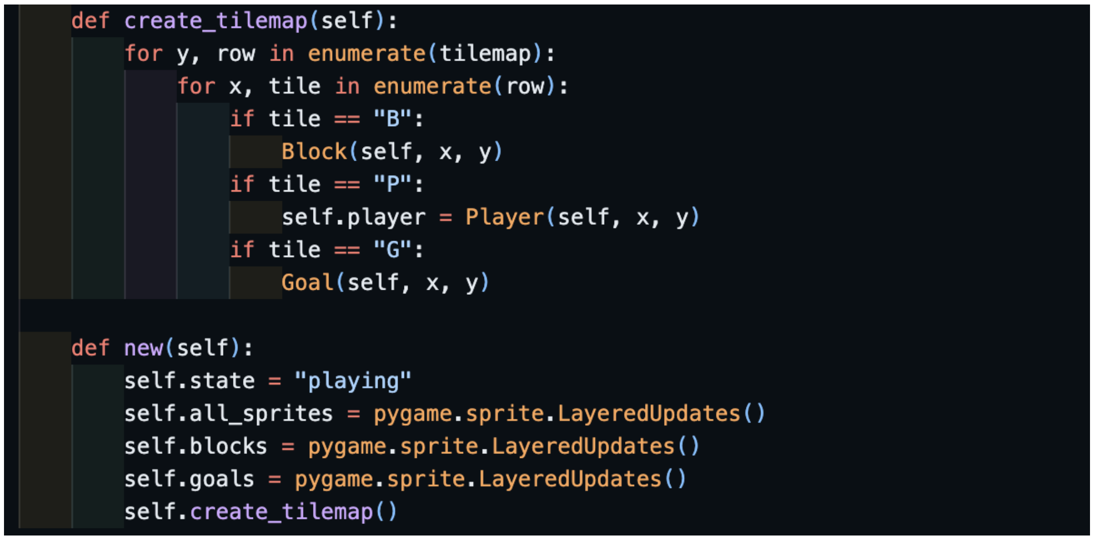
```
import pygame
import sys
from config import *
from sprites import *
```

- This is to set up the game itself including the windows size and fps which it gets from config.py. It also chooses which font the game will use



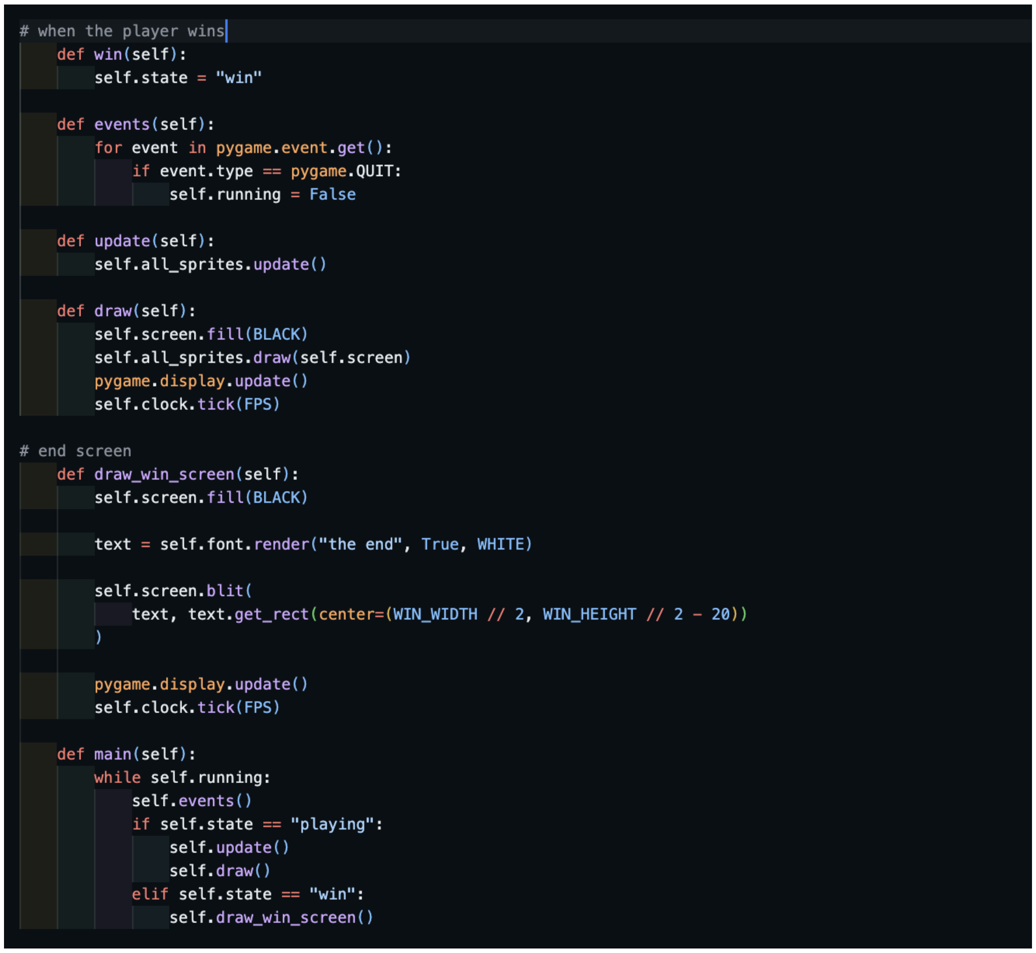
```
class Game:
    def __init__(self):
        pygame.init()
        self.screen = pygame.display.set_mode((WIN_WIDTH, WIN_HEIGHT))
        pygame.display.set_caption("Maze Game")
        self.clock = pygame.time.Clock()
        self.running = True
        self.font = pygame.font.SysFont("arial", 48)
        self.small_font = pygame.font.SysFont("arial", 24)
```

- This creates the tilemap which will position every object according to the tilemap in config.py



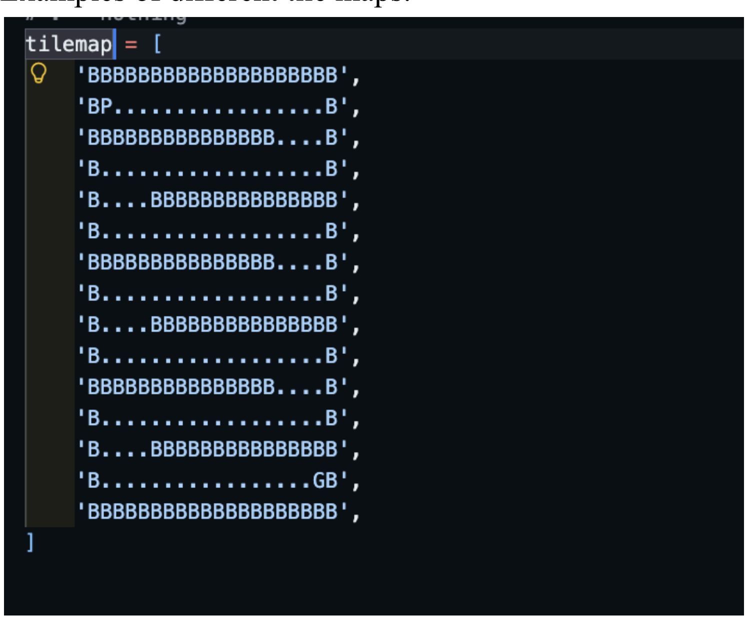
```
    def create_tilemap(self):
        for y, row in enumerate(tilemap):
            for x, tile in enumerate(row):
                if tile == "B":
                    Block(self, x, y)
                if tile == "P":
                    self.player = Player(self, x, y)
                if tile == "G":
                    Goal(self, x, y)

    def new(self):
        self.state = "playing"
        self.all_sprites = pygame.sprite.LayeredUpdates()
        self.blocks = pygame.sprite.LayeredUpdates()
        self.goals = pygame.sprite.LayeredUpdates()
        self.create_tilemap()
```

- This is code for the win screen and win state

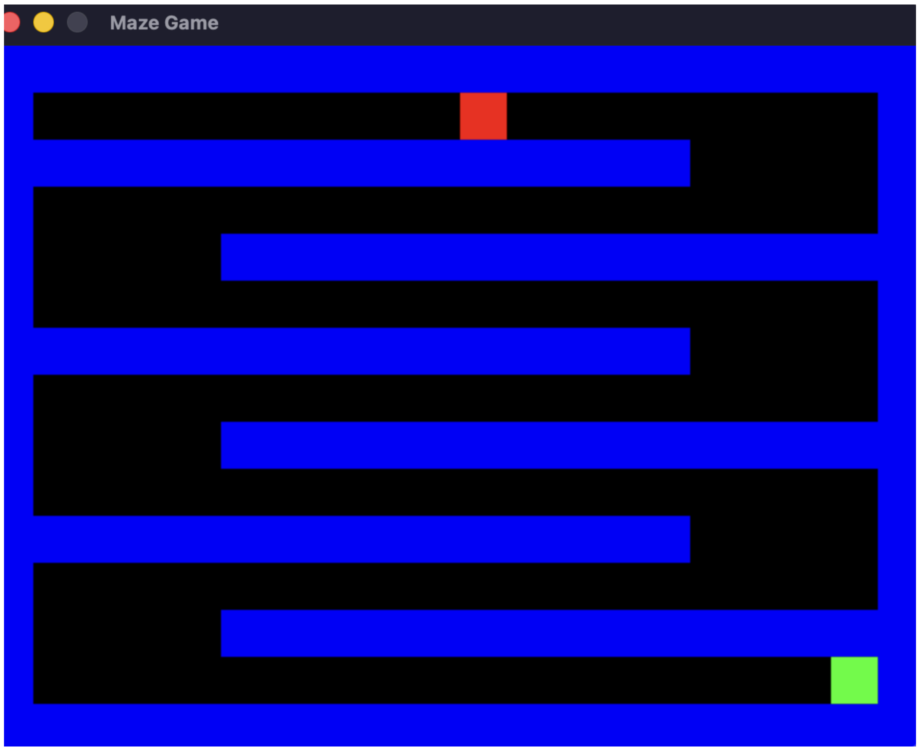


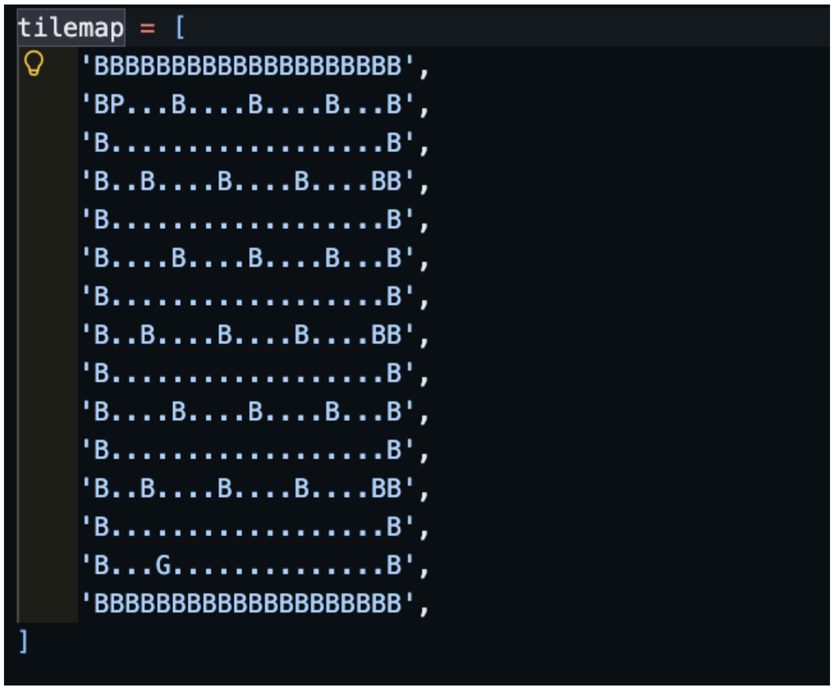
```python
# when the player wins
    def win(self):
        self.state = "win"

    def events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False

    def update(self):
        self.all_sprites.update()

    def draw(self):
        self.screen.fill(BLACK)
        self.all_sprites.draw(self.screen)
        pygame.display.update()
        self.clock.tick(FPS)

# end screen
    def draw_win_screen(self):
        self.screen.fill(BLACK)

        text = self.font.render("the end", True, WHITE)

        self.screen.blit(
            text, text.get_rect(center=(WIN_WIDTH // 2, WIN_HEIGHT // 2 - 20))
        )

        pygame.display.update()
        self.clock.tick(FPS)

    def main(self):
        while self.running:
            self.events()
            if self.state == "playing":
                self.update()
                self.draw()
            elif self.state == "win":
                self.draw_win_screen()
```

Examples of different tile maps:

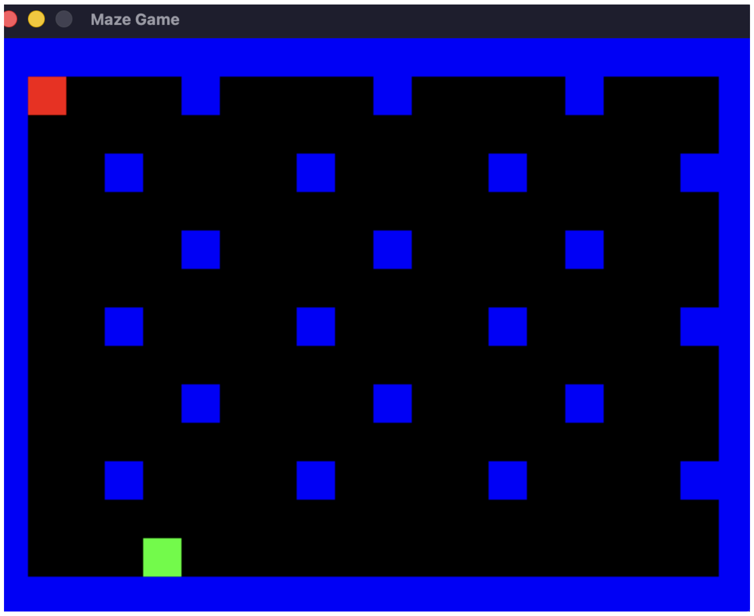


```python
tilemap = [
    'BBBBBBBBBBBBBBBBBBBB',
    'BP.................B',
    'BBBBBBBBBBBBBBB....B',
    'B..................B',
    'B....BBBBBBBBBBBBBBB',
    'B..................B',
    'BBBBBBBBBBBBBBB....B',
    'B..................B',
    'B....BBBBBBBBBBBBBBB',
    'B..................B',
    'BBBBBBBBBBBBBBB....B',
    'B..................B',
    'B....BBBBBBBBBBBBBBB',
    'B.................GB',
    'BBBBBBBBBBBBBBBBBBBB',
]
```





```
tilemap = [
    'BBBBBBBBBBBBBBBBBBBB',
    'BP...B....B....B...B',
    'B..................B',
    'B..B....B....B....BB',
    'B..................B',
    'B....B....B....B...B',
    'B..................B',
    'B..B....B....B....BB',
    'B..................B',
    'B....B....B....B...B',
    'B..................B',
    'B..B....B....B....BB',
    'B..................B',
    'B...G..............B',
    'BBBBBBBBBBBBBBBBBBBB',
]
```



Statement on the use of Artificial Intelligence:

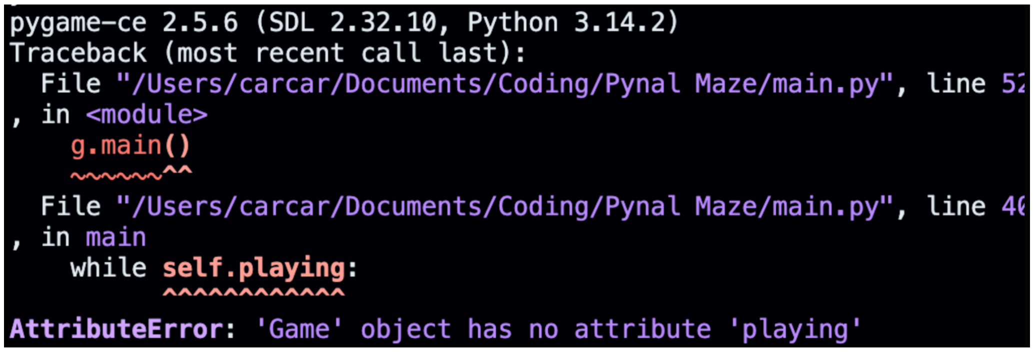
AI Tools:

- Chat-GPT 5.2
- Gemini 3

I only use AI when there is an error that I didn't know how to fix, figuring out how to make it so when the players hit the wall they go back because before they would just collide with it and help figure out how to make the end screen. I also use gemini to help me generate different tilemaps so I can test out different layouts

Prompts used:

- "How to fix this bug"



```
pygame-ce 2.5.6 (SDL 2.32.10, Python 3.14.2)
Traceback (most recent call last):
  File "/Users/carcar/Documents/Coding/Pynal Maze/main.py", line 52
, in <module>
    g.main()
    ~~~~~~^^
  File "/Users/carcar/Documents/Coding/Pynal Maze/main.py", line 40
, in main
    while self.playing:
          ^^^^^^^^^^^^
AttributeError: 'Game' object has no attribute 'playing'
```

- "How to make it so the player goes back to the start when they collide with the wall"
- "How to make a goal" it gave me the code and i had to implement it
- The AI asked if I wanted an end screen so I said "sure"

- “can you generate me different tile maps tilemap” and it generated many tile maps

References:

- Shawcode Pygame RPG Tutorial part 1 - 5
  https://youtube.com/playlist?list=PLkkm3wcQHjT7gn81Wn-e78cAyhwBW3FIc&si=JKuGZQKssTg7v0es
- Google Gemini 3
  https://gemini.google.com/app/
- Chat-GPT 5.2
  https://chatgpt.com