#Game Save class

import pygame

class GameSave(object):

#the constructor method of the class including all the information needed for game loading as the parameters

def \_\_init\_\_(self, maze, CurrentScore, lives, moves, Time):

#set up all the attributes of the class using the arguments provided

self.\_\_savedMaze = maze

self.\_\_CurrentScore = CurrentScore

self.\_\_lives = lives

self.\_\_moves = moves

self.\_\_Time = Time

#method to calculate the player's position in the maze with 5 parameters

def calcPlayerPos(self, mazePos, MAZE\_HEIGHT, MAZE\_WIDTH, midX, midY):

#uses a complex calculation to work out the X and Y values

#this is the inverse operation to the one used when generating the maze

X = (mazePos[0] - (midX - 12.5\*MAZE\_WIDTH)) / 25

Y = (mazePos[1] - (midY - 25\*MAZE\_HEIGHT)) / 25

self.\_\_coords = [int(X),int(Y)]

return self.\_\_coords #returns these coordinates which are to be written to the file

#method to write info to 'mazeInfo' file with 3 parameters

def writeToFile(self, fileName, coords):

#writes the attributes to the file

for item in self.\_\_savedMaze: #uses a FOR loop that iterates across the 'savedMaze' 2D array

fileName.write(str(item)+"\n")

fileName.write(str(self.\_\_CurrentScore)+"\n")

fileName.write(str(self.\_\_lives)+"\n")

fileName.write(str(self.\_\_moves)+"\n")

fileName.write(str(self.\_\_Time)+"\n")

fileName.write(str(coords)) #writes the passed coordinates to the file

#method to load info from the file when loading the maze back with 3 parameters

def loadMaze(self, fileName, MAZE\_HEIGHT, MAZE\_WIDTH):

fileLines = [] #sets up an empty array for the output of the file

maze = [] #resets the maze array

for line in fileName: #uses a FOR loop that iterates across the file

fileLines.append(line) #appends each line of the file to the output array

for num in range(0, len(fileLines)): #uses a FOR loop that iterates across the 'fileLines' array

if num < MAZE\_HEIGHT: #checks if the line number is less than 'MAZE\_HEIGHT'

maze.append(fileLines[num]) #if it is, appends the line to maze

elif MAZE\_HEIGHT <= num < (MAZE\_HEIGHT + 1): #checks if the line is the one after

CurrentScore = int(fileLines[num]) #if it is sets its integer value to 'CurrentScore'

elif (MAZE\_HEIGHT + 1) <= num < (MAZE\_HEIGHT + 2): #checks if the line is the one after

lives = int(fileLines[num]) #if it is sets its integer value to 'lives'

elif (MAZE\_HEIGHT + 2) <= num < (MAZE\_HEIGHT + 3): #checks if the line is the one after

moves = int(fileLines[num]) #if it is sets its integer value to 'moves'

elif (MAZE\_HEIGHT + 3) <= num < (MAZE\_HEIGHT + 4): #checks if the line is the one after

Time = int(fileLines[num]) #if it is sets its integer value to 'Time'

else: #checks if the line is any other one

playerPos = fileLines[num] #if it is sets its value to 'playerPos'

return maze, CurrentScore, lives, moves, Time, playerPos #returns all of these values to the 'Main' program

#Game Save test

import pygame

from time import sleep

from random import randint,random

from Player\_class import Player

from Maze\_objects import Wall, Cell, Empty, Key

from Information\_objects import Scoreboard, LivesC, MovesC, Timer

from GameSave\_class import GameSave

maze = []

MAZE\_HEIGHT = 5

MAZE\_WIDTH = 5

class screens(object):

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

#set the window up with title

self.\_win = pygame.display.set\_mode((1900, 1000), pygame.RESIZABLE)

self.\_winXY = pygame.display.get\_window\_size()

pygame.display.set\_caption("Maths Maze Navigator")

#create sprite groups

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

def display():

pass

def surroundingCells(maze, rand\_wall):

s\_cells = 0

if (maze[rand\_wall[0]-1][rand\_wall[1]] == True):

s\_cells += 1

if (maze[rand\_wall[0]+1][rand\_wall[1]] == True):

s\_cells += 1

if (maze[rand\_wall[0]][rand\_wall[1]-1] == True):

s\_cells +=1

if (maze[rand\_wall[0]][rand\_wall[1]+1] == True):

s\_cells += 1

return s\_cells

def create\_paths(maze, MAZE\_HEIGHT, MAZE\_WIDTH):

starting\_height = int(random()\*MAZE\_HEIGHT)

starting\_width = int(random()\*MAZE\_WIDTH)

if (starting\_height == 0):

starting\_height += 1

if (starting\_height == MAZE\_HEIGHT-1):

starting\_height -= 1

if (starting\_width == 0):

starting\_width += 1

if (starting\_width == MAZE\_WIDTH-1):

starting\_width -= 1

maze[starting\_height][starting\_width] = True

wallCoords = []

wallCoords.append([starting\_height - 1, starting\_width])

wallCoords.append([starting\_height, starting\_width - 1])

wallCoords.append([starting\_height, starting\_width + 1])

wallCoords.append([starting\_height + 1, starting\_width])

maze[starting\_height-1][starting\_width] = False

maze[starting\_height][starting\_width - 1] = False

maze[starting\_height][starting\_width + 1] = False

maze[starting\_height + 1][starting\_width] = False

while (wallCoords):

# Pick a random wall

rand\_wall = wallCoords[int(random()\*len(wallCoords))-1]

# Check if it is a left wall

if (rand\_wall[1] != 0):

if (maze[rand\_wall[0]][rand\_wall[1]-1] == 'None' and maze[rand\_wall[0]][rand\_wall[1]+1] == True):

# Find the number of surrounding cells

s\_cells = surroundingCells(maze, rand\_wall)

if (s\_cells < 2):

# Denote the new path

maze[rand\_wall[0]][rand\_wall[1]] = True

# Mark the new walls

# Upper cell

if (rand\_wall[0] != 0):

if (maze[rand\_wall[0]-1][rand\_wall[1]] != True):

maze[rand\_wall[0]-1][rand\_wall[1]] = False

if ([rand\_wall[0]-1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]-1, rand\_wall[1]])

# Bottom cell

if (rand\_wall[0] != MAZE\_HEIGHT-1):

if (maze[rand\_wall[0]+1][rand\_wall[1]] != True):

maze[rand\_wall[0]+1][rand\_wall[1]] = False

if ([rand\_wall[0]+1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]+1, rand\_wall[1]])

# Leftmost cell

if (rand\_wall[1] != 0):

if (maze[rand\_wall[0]][rand\_wall[1]-1] != True):

maze[rand\_wall[0]][rand\_wall[1]-1] = False

if ([rand\_wall[0], rand\_wall[1]-1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]-1])

# Delete wall

for wall in wallCoords:

if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):

wallCoords.remove(wall)

continue

#Check if it is an upper wall

if (rand\_wall[0] != 0):

if (maze[rand\_wall[0]-1][rand\_wall[1]] == 'None' and maze[rand\_wall[0]+1][rand\_wall[1]] == True):

s\_cells = surroundingCells(maze, rand\_wall)

if (s\_cells < 2):

# Denote the new path

maze[rand\_wall[0]][rand\_wall[1]] = True

# Mark the new walls

# Upper cell

if (rand\_wall[0] != 0):

if (maze[rand\_wall[0]-1][rand\_wall[1]] != True):

maze[rand\_wall[0]-1][rand\_wall[1]] = False

if ([rand\_wall[0]-1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]-1, rand\_wall[1]])

# Leftmost cell

if (rand\_wall[1] != 0):

if (maze[rand\_wall[0]][rand\_wall[1]-1] != True):

maze[rand\_wall[0]][rand\_wall[1]-1] = False

if ([rand\_wall[0], rand\_wall[1]-1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]-1])

# Rightmost cell

if (rand\_wall[1] != MAZE\_WIDTH-1):

if (maze[rand\_wall[0]][rand\_wall[1]+1] != True):

maze[rand\_wall[0]][rand\_wall[1]+1] = False

if ([rand\_wall[0], rand\_wall[1]+1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]+1])

# Delete wall

for wall in wallCoords:

if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):

wallCoords.remove(wall)

continue

# Check the bottom wall

if (rand\_wall[0] != MAZE\_HEIGHT-1):

if (maze[rand\_wall[0]+1][rand\_wall[1]] == 'None' and maze[rand\_wall[0]-1][rand\_wall[1]] == True):

s\_cells = surroundingCells(maze, rand\_wall)

if (s\_cells < 2):

# Denote the new path

maze[rand\_wall[0]][rand\_wall[1]] = True

# Mark the new walls

if (rand\_wall[0] != MAZE\_HEIGHT-1):

if (maze[rand\_wall[0]+1][rand\_wall[1]] != True):

maze[rand\_wall[0]+1][rand\_wall[1]] = False

if ([rand\_wall[0]+1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]+1, rand\_wall[1]])

if (rand\_wall[1] != 0):

if (maze[rand\_wall[0]][rand\_wall[1]-1] != True):

maze[rand\_wall[0]][rand\_wall[1]-1] = False

if ([rand\_wall[0], rand\_wall[1]-1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]-1])

if (rand\_wall[1] != MAZE\_WIDTH-1):

if (maze[rand\_wall[0]][rand\_wall[1]+1] != True):

maze[rand\_wall[0]][rand\_wall[1]+1] = False

if ([rand\_wall[0], rand\_wall[1]+1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]+1])

# Delete wall

for wall in wallCoords:

if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):

wallCoords.remove(wall)

continue

# Check the right wall

if (rand\_wall[1] != MAZE\_WIDTH-1):

if (maze[rand\_wall[0]][rand\_wall[1]+1] == 'None' and maze[rand\_wall[0]][rand\_wall[1]-1] == True):

s\_cells = surroundingCells(maze, rand\_wall)

if (s\_cells < 2):

# Denote the new path

maze[rand\_wall[0]][rand\_wall[1]] = True

# Mark the new walls

if (rand\_wall[1] != MAZE\_WIDTH-1):

if (maze[rand\_wall[0]][rand\_wall[1]+1] != True):

maze[rand\_wall[0]][rand\_wall[1]+1] = False

if ([rand\_wall[0], rand\_wall[1]+1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]+1])

if (rand\_wall[0] != MAZE\_HEIGHT-1):

if (maze[rand\_wall[0]+1][rand\_wall[1]] != True):

maze[rand\_wall[0]+1][rand\_wall[1]] = False

if ([rand\_wall[0]+1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]+1, rand\_wall[1]])

if (rand\_wall[0] != 0):

if (maze[rand\_wall[0]-1][rand\_wall[1]] != True):

maze[rand\_wall[0]-1][rand\_wall[1]] = False

if ([rand\_wall[0]-1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]-1, rand\_wall[1]])

# Delete wall

for wall in wallCoords:

if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):

wallCoords.remove(wall)

continue

# Delete the wall from the list anyway

for wall in wallCoords:

if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):

wallCoords.remove(wall)

return maze

def generate\_maze(MAZE\_HEIGHT, MAZE\_WIDTH, maze, walls, cells, all\_sprites, midX, midY):

visited = [] #set up an empty array for the cells the player has visited

#set up a FOR loop for the rows of the maze and visited

for x in range(0, MAZE\_HEIGHT): #initiates a FOR loop for the maze/visited rows

visitedRow = [] #initialise the 'visitedRow' as an empty array

mazeRow = [] #initialise the 'mazeRow' as an empty array

for y in range(0, MAZE\_WIDTH): #set up a nested FOR loop for the maze/visited columns

mazeRow.append("None") #append the value which will be used for empty sections of the maze

visitedRow.append(False) #append the value which will be used for cells the player hasn't visited yet

maze.append(mazeRow) #append the 'mazeRow' full of 'None' strings to 'maze'

visited.append(visitedRow) #append the 'visitedRow' full of 'False' booleans to 'visited'

#calls the 'create\_paths' function which returns the 'maze' array

maze = create\_paths(maze, MAZE\_HEIGHT, MAZE\_WIDTH)

#set up a FOR loop for the rows of the maze

for i in range(0, MAZE\_HEIGHT):

for j in range(0, MAZE\_WIDTH): #set up a nested FOR loop for the maze columns

if (maze[i][j] == 'None'): #use an IF statement to check if an item in 'maze' is 'None'

maze[i][j] = False #if it is 'None' sets it to False for the walls of the maze

#sets the entrance and exit of the maze

maze[0][1] = 'None'

maze[MAZE\_WIDTH-1][MAZE\_WIDTH-2] = 'None'

#set up a FOR loop for the rows of the maze

for x in range(0, MAZE\_HEIGHT):

for y in range(0, MAZE\_WIDTH): #set up a nested FOR loop for the maze columns

X = (25\*x)+(midX-((25\*MAZE\_WIDTH) / 2)) #set the X coordinate to be 25x bigger than maze index

Y = (25\*y)+(midY-(25\*MAZE\_HEIGHT)) #set the Y coordinate to be 25x bigger than maze index

if maze[x][y] == False: #check if maze index is a wall

wall = Wall([X,Y]) #set up a wall sprite with X and Y as the location

walls.add(wall) #add it to the 'walls' group

all\_sprites.add(wall) #add it to the 'all\_sprites' group

elif maze[x][y] == True: #check if maze index is a cell

cell = Cell([X,Y]) #set up a cell sprite with X and Y as the location

cells.add(cell) #add it to the 'cells' group

all\_sprites.add(cell) #add it to the 'all\_sprites' group

else: #otherwise:

empty = Empty([X,Y]) #set up an empty sprite with X and Y as the location

all\_sprites.add(empty) #adds it to the 'all\_sprites' group

if x == (MAZE\_WIDTH-1): #if it is the exit of the maze

keys = pygame.sprite.Group() #create a sprite group for keys

key = Key("Key.png", [X,Y]) #set up a key sprite with X and Y as the location

all\_sprites.add(key) #adds it to the 'all\_sprites' group

keys.add(key) #adds it to the 'keys' group

return keys, visited #returns the 'keys' group to the 'pygame\_window' class and also visited 2D array

class load\_game():

def \_\_init\_\_(self, maze, CurrentScore, lives, moves, Time, playerPos, midX, midY):

self.\_\_savedMaze = maze

self.\_\_CurrentScore = CurrentScore

self.\_\_lives = lives

self.\_\_moves = moves

self.\_\_Time = Time

self.\_\_playerPos = playerPos

self.\_\_midX = midX

self.\_\_midY = midY

def loadMaze(self, midX, midY):

all\_sprites = pygame.sprite.Group()

walls = pygame.sprite.Group()

cells = pygame.sprite.Group()

#set up a FOR loop for the rows of the maze

for x in range(0, MAZE\_HEIGHT):

for y in range(0, MAZE\_WIDTH): #set up a nested FOR loop for the maze columns

X = (25\*x)+(midX-((25\*MAZE\_WIDTH) / 2)) #set the X coordinate to be 25x bigger than maze index

Y = (25\*y)+(midY-(25\*MAZE\_HEIGHT)) #set the Y coordinate to be 25x bigger than maze index

if maze[x][y] == False: #check if maze index is a wall

wall = Wall([X,Y]) #set up a wall sprite with X and Y as the location

walls.add(wall) #add it to the 'walls' group

all\_sprites.add(wall) #add it to the 'all\_sprites' group

elif maze[x][y] == True: #check if maze index is a cell

cell = Cell([X,Y]) #set up a cell sprite with X and Y as the location

cells.add(cell) #add it to the 'cells' group

all\_sprites.add(cell) #add it to the 'all\_sprites' group

else: #otherwise:

empty = Empty([X,Y]) #set up an empty sprite with X and Y as the location

all\_sprites.add(empty) #adds it to the 'all\_sprites' group

return walls, cells, all\_sprites

def loadGame(self, all\_sprites, MAZE\_HEIGHT, MAZE\_WIDTH):

self.\_all\_sprites = all\_sprites

pixelX = (25\*int(self.\_\_playerPos[1]))+(self.\_\_midX-((25\*MAZE\_WIDTH) / 2))

pixelY = (25\*int(self.\_\_playerPos[4]))+(self.\_\_midY-(25\*MAZE\_HEIGHT))

player = Player("Player.png",[pixelX, pixelY])

self.\_\_locX1 = self.\_\_midX - 250

self.\_\_locX2 = self.\_\_midX - 100

self.\_\_locX3 = self.\_\_midX + 100

self.\_\_locX4 = self.\_\_midX + 250

self.\_\_locY = self.\_\_midY + (self.\_\_midY / 2)

self.\_\_scoreBoard = Scoreboard(15,"white",100,150,self.\_\_locX1,self.\_\_locY)

self.\_\_livesCounter = LivesC(15,"white",100,150,self.\_\_locX2,self.\_\_locY)

self.\_\_movesCounter = MovesC(15,"white",100,150,self.\_\_locX3,self.\_\_locY)

self.\_\_timer = Timer(15,"white",100,150,self.\_\_locX4,self.\_\_locY)

self.\_all\_sprites.add(player)

self.\_all\_sprites.add(self.\_\_scoreBoard)

self.\_all\_sprites.add(self.\_\_livesCounter)

self.\_all\_sprites.add(self.\_\_movesCounter)

self.\_all\_sprites.add(self.\_\_timer)

return self.\_all\_sprites, player, self.\_\_scoreBoard, self.\_\_livesCounter, self.\_\_movesCounter, self.\_\_timer

class pygame\_window(screens):

def \_\_init\_\_(self, MAZE\_HEIGHT, MAZE\_WIDTH, maze):

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

#assign parameters to variables

self.\_\_maze\_height = MAZE\_HEIGHT

self.\_\_maze\_width = MAZE\_WIDTH

self.\_\_maze = maze

#initiate pygame

pygame.init()

#Calculate where the middle of the screen is

self.\_\_winX = self.\_winXY[0]

self.\_\_winY = self.\_winXY[1]

self.\_\_midX = self.\_\_winX / 2

self.\_\_midY = self.\_\_winY / 2

#instantiate sprites

self.\_\_player = Player("Player.png",[600,300])

self.\_\_locX1 = self.\_\_midX - 250

self.\_\_locX2 = self.\_\_midX - 100

self.\_\_locX3 = self.\_\_midX + 100

self.\_\_locX4 = self.\_\_midX + 250

self.\_\_locY = self.\_\_midY + (self.\_\_midY / 2)

self.\_\_scoreBoard = Scoreboard(15,"white",100,150,self.\_\_locX1,self.\_\_locY)

self.\_\_livesCounter = LivesC(15,"white",100,150,self.\_\_locX2,self.\_\_locY)

self.\_\_movesCounter = MovesC(15,"white",100,150,self.\_\_locX3,self.\_\_locY)

self.\_\_timer = Timer(15,"white",100,150,self.\_\_locX4,self.\_\_locY)

#Set variables for Scoreboard, Lives counter, Moves counter and Timer

self.\_\_CurrentScore = 0

self.\_\_lives = 300

self.\_\_moves = 0

self.\_\_Time = 0

#instantiate the 'GameSave' class

self.\_\_gameSave = GameSave(maze, self.\_\_CurrentScore, self.\_\_lives, self.\_\_moves, self.\_\_Time)

#create sprite groups

self.\_\_walls = pygame.sprite.Group()

self.\_\_cells = pygame.sprite.Group()

#add sprites to all\_sprites

self.\_all\_sprites.add(self.\_\_player)

self.\_all\_sprites.add(self.\_\_scoreBoard)

self.\_all\_sprites.add(self.\_\_livesCounter)

self.\_all\_sprites.add(self.\_\_movesCounter)

self.\_all\_sprites.add(self.\_\_timer)

def display(self):

#call the function to generate the maze and 'catch' the 2 variables it returns

keys, visited = generate\_maze(self.\_\_maze\_height, self.\_\_maze\_width, self.\_\_maze, self.\_\_walls,

self.\_\_cells, self.\_all\_sprites, self.\_\_midX, self.\_\_midY)

coords = [0, 1]

run = True

#initiate a while loop until run is no longer True

while run:

#initiate a for loop to detect events

for event in pygame.event.get():

#allows the user to quit the game

if event.type == pygame.QUIT:

run = False #sets run to False which breaks the loop

if event.type == pygame.KEYDOWN: #checks if a keyboard key is pressed

if event.key == pygame.K\_f: #if the key is f

pygame.display.toggle\_fullscreen()

if event.key == pygame.K\_l:

file = open("mazeInfo.txt", 'r')

maze, CurrentScore, lives, moves, Time, playerPos = self.\_\_gameSave.loadMaze(file, MAZE\_HEIGHT, MAZE\_WIDTH)

currentGame = load\_game(maze, CurrentScore, lives, moves, Time, playerPos, self.\_\_midX, self.\_\_midY)

self.\_\_walls, self.\_\_cells, self.\_all\_sprites = currentGame.loadMaze(self.\_\_midX, self.\_\_midY)

self.\_all\_sprites, self.\_\_player, self.\_\_scoreBoard, self.\_\_livesCounter, self.\_\_movesCounter, self.\_\_timer = currentGame.loadGame(self.\_all\_sprites, MAZE\_HEIGHT, MAZE\_WIDTH)

if event.type == pygame.MOUSEBUTTONUP: #checks if a keyboard key is pressed

pygame.time.set\_timer(event, 1000)#sets a timer for 1 second

self.\_\_Time += 1

if event.type == pygame.MOUSEBUTTONDOWN:

self.\_\_gameSave = GameSave(self.\_\_maze, self.\_\_CurrentScore, self.\_\_lives, self.\_\_moves, self.\_\_Time)

file = open("mazeInfo.txt", 'w')

self.\_\_gameSave.writeToFile(file, coords)

file.close()

#fills the pygame window with white

self.\_win.fill((0,0,0))

#stores the key that the user presses

pressed\_keys = pygame.key.get\_pressed()

for sprite in self.\_all\_sprites: #set a FOR loop to cycle through the whole list of sprites

self.\_win.blit(sprite.surf, sprite.rect) #blit each sprite on to the screen

pygame.display.update() #update the display with all sprites blitted on

#calls the move method of the player with the key that the user presses as the parameter

self.\_\_moves = self.\_\_player.move(pressed\_keys, self.\_\_moves)

self.\_\_movesCounter.update(self.\_\_moves,15,"white",100,150) #calls the update method of the 'movesCounter'

self.\_\_timer.update(self.\_\_Time,15,"white",100,150) #update the timer

#checks if the player collides with any of the walls

if pygame.sprite.spritecollideany(self.\_\_player, self.\_\_walls):

self.\_\_player.kill() #if it does then despawn the player

#calls the update method of the 'livesCounter'

self.\_\_lives = self.\_\_livesCounter.update(self.\_\_lives,15,"white",100,150)

#checks if the lives has gone below 0

if self.\_\_lives < 0:

print("You lose!") #if it has then runs the lose end screen

#store the cell that the player collides with as 'thisCell'

thisCell = pygame.sprite.spritecollideany(self.\_\_player, self.\_\_cells)

#run a FOR loop through all the cells

for cell in self.\_\_cells:

if thisCell == cell: #check if the current cell is 'thisCell'

#respawn the player on top of the cell

cellLoc = cell.getLocation()

player = Player("Player.png",cellLoc)

coords = self.\_\_gameSave.calcPlayerPos(cellLoc, MAZE\_HEIGHT, MAZE\_WIDTH, self.\_\_midX, self.\_\_midY)

self.\_win.blit(player.surf, player.rect)

pygame.display.update() #updates the display

#checks if the player collides with the keys

if pygame.sprite.spritecollideany(self.\_\_player, keys):

self.\_\_CurrentScore = 1000 - (25 \* self.\_\_moves) #if it does calculate the score

self.\_\_scoreBoard.update(self.\_\_CurrentScore,15,"white",100,150) #calls the update method of the 'scoreBoard'

run = False

#detects if run is set to False

if not run:

sleep(2) #waits for 2 seconds

pygame.quit() #closes the pygame window

class Game(object):

#set the constructor method with 3 parameters

def \_\_init\_\_(self, MAZE\_HEIGHT, MAZE\_WIDTH, maze):

#set the 'display' attributes to the various classes

self.display2 = pygame\_window(MAZE\_HEIGHT, MAZE\_WIDTH, maze)

#set up the run\_main method which will call the display method of the 'pygame\_window' class

def run\_main(self):

self.display2.display() #call the display method of the class assigned to 'display2

#set up an if statement for the main part of the program

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

game = Game(MAZE\_HEIGHT, MAZE\_WIDTH, maze) #instantiate the game class

game.run\_main() #call the run method of the game class

screen = screens() #set up the base class for the other screens

#HighScores class

class HighScores(object):

#the contructor method for the class

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

#set up the scores array and names dictionary

self.\_\_scores = []

self.\_\_nameDict = {}

#the method to get the scores and names from the file

def get\_scores(self, fileName):

for line in fileName: #sets up a FOR loop which cycles through the file

line = line.strip().split(",") #removes the '\n' and splits the line by the comma

self.\_\_scores.append(line[1]) #appends the score to the scores array

self.\_\_nameDict[line[0]] = float(line[1]) #creates a key-value pair for the name and score in the names dictionary

return self.\_\_scores #returns the scores array to the 'Main' program

#merge sort method to sort the scores

def sort\_scores(self, scores):

if len(scores) > 1: #if the length of the array is more than 1...

#calculates the midpoint of the array

m = len(scores)/2

m = round(m, 0)

m = int(m)

#sets up 2 arrays the opposite halves of the array passed in

scores1 = scores[0:m]

scores2 = scores[m:]

#calls itself on each half array

self.sort\_scores(scores1)

self.sort\_scores(scores2)

#sets initial values to 0

i = j = k = 0

#WHILE i and j are less than the lengths of the half arrays

while i < len(scores1) and j < len(scores2):

if scores1[i] <= scores2[j]: #if the current item in array 1 is smaller than the on in array 2...

scores[k] = scores1[i] #stores the current item of the final array as the item in scores1 that is in position i

i += 1 #increments i by one to move to next item in array 1

else: #otherwise...

scores[k] = scores2[j] #stores the current item of the final array as the item in scores2 that is in position j

j += 1 #increments j by one to move to next item in array 2

k += 1 #increments k by one to move to next item in final array

#WHILE i is less than the length of the first half array

while i < len(scores1):

scores[k] = scores1[i] #stores the current item of the final array as the item in scores1 that is in position i

i += 1 #increments i by one to move to next item in array 1

k += 1 #increments k by one to move to next item in final array

#WHILE j is less than the length of the second half array

while j < len(scores2):

scores[k] = scores2[j] #stores the current item of the final array as the item in scores2 that is in position j

j += 1 #increments j by one to move to next item in array 2

k += 1 #increments k by one to move to next item in final array

return scores #return the sorted scores array to the 'Main' program

#the method to sort the names dictionary

def sortNames(self):

sorted\_names = sorted(self.\_\_nameDict.items(), key=lambda x:x[1]) #the keyword method to sort a dictionary

return sorted\_names #returns the sorted dictionary (as array of tuples) to 'Main' program

#the method to check if the score is bigger than the smallest high score

def check\_score(self, score):

last = len(self.\_\_scores) - 1 #calculates the last index

names = [\*self.\_\_nameDict] #stores the keys list of the names dictionary as names

lastName = names[last] #gets the lastName using the last index

if float(self.\_\_scores[last]) < score: #uses an IF statement to see if score is larger than 'scores[last]'

self.\_\_scores.remove(self.\_\_scores[last]) #removes the last score from 'scores'

del self.\_\_nameDict[lastName] #deletes the last keys-value pair in the names dictionary

return True #returns True to the 'Main' program

else: #otherwise...

return False #returns False to the 'Main' program

#the method to add a score/name to the array/dictionary

def add\_score(self, name, score):

self.\_\_scores.append(score) #append the score to the 'scores' array

self.\_\_nameDict[name] = score #creates a new key-value pair in the dictionary

return self.\_\_scores #returns the scores array to the 'Main' program

#the method to write the new high scores to a file

def saveScore(self, fileName, sortedNames):

for item in sortedNames: #goes through the 'sortedNames' array

name = item[0] #gets the name as item[0]

score = item[1] #gets the score as item[1]

text = name+","+str(score)+"\n" #creates the text which will be written to the file

fileName.write(text) #writes the text to the file

def get\_score(self):

return self.\_\_scores, self.\_\_nameDict

#HighScores test

from HighScores\_class import HighScores

from random import randint

score = randint(500, 1050)

high\_scores = HighScores()

fileIn = open('high\_scores.txt', 'r')

scores = high\_scores.get\_scores(fileIn)

fileIn.close()

intScores = []

print("Your score is:",score)

name = input("Enter your name to save this score: ")

if high\_scores.check\_score(score):

scores = high\_scores.add\_score(name, score)

for item in scores:

intScores.append(int(item))

print(high\_scores.sort\_scores(intScores))

sortedNames = high\_scores.sortNames()

sortedNames = sortedNames[::-1]

fileOut = open('high\_scores.txt', 'w')

high\_scores.saveScore(fileOut, sortedNames)

fileOut.close()

else:

print("Score is not high enough!")

#Information objects

import pygame

class Scoreboard(pygame.sprite.Sprite):

#define the constructor method of the class

def \_\_init\_\_(self, size, color, width, height, locationX, locationY):

CurrentScore = 0 #set 'CurrentScore' to 0

score = "Score: "+str(CurrentScore) #create the string that will be blited on to the sprite

pygame.sprite.Sprite.\_\_init\_\_(self) #Call the parent class (Sprite) constructor

self.font = pygame.font.SysFont("Arial", size) #Set up the font for the text

self.textSurf = self.font.render(score, 1, color) #set up the text with the string 'score'

#set up the sprite's surf

self.surf = pygame.Surface((width, height))

self.surf.fill((0,0,255))

#set up the Scoreboard as a rectangle in the specified part of the screen

self.rect = self.surf.get\_rect(center=(locationX,locationY))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [width/2 - W/2, height/2 - H/2])

#define the update method of the class (to update the score)

def update(self, CurrentScore, size, color, width, height):

score = "Score: "+str(CurrentScore) #create the string that will be blited on to the sprite

self.font = pygame.font.SysFont("Arial", size) #Set up the font for the text

self.textSurf = self.font.render(score, 1, color) #set up the text with the string 'score'

#set up the sprite's surf

self.surf = pygame.Surface((width, height))

self.surf.fill((0,0,255))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [width/2 - W/2, height/2 - H/2])

class LivesC(pygame.sprite.Sprite):

#define the constructor method of the class

def \_\_init\_\_(self, size, color, width, height, locationX, locationY):

Lives = 300 #set 'Lives' to 3

lives = "Lives: "+str(Lives) #create the string that will be blited on to the sprite

pygame.sprite.Sprite.\_\_init\_\_(self) #Call the parent class (Sprite) constructor

self.font = pygame.font.SysFont("Arial", size) #Set up the font for the text

self.textSurf = self.font.render(lives, 1, color) #set up the text with the string 'lives'

#set up the sprite's surf

self.surf = pygame.Surface((width, height))

self.surf.fill((0,0,255))

#set up the Lives counter as a rectangle in the specified part of the screen

self.rect = self.surf.get\_rect(center=(locationX,locationY))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [width/2 - W/2, height/2 - H/2])

def update(self, Lives, size, color, width, height):

Lives -= 1 #Decrement the 'Lives' by 1

lives = "Lives: "+str(Lives) #create the string that will be blited on to the sprite

self.font = pygame.font.SysFont("Arial", size) #Set up the font for the text

self.textSurf = self.font.render(lives, 1, color) #set up the text with the string 'lives'

#set up the sprite's surf

self.surf = pygame.Surface((width, height))

self.surf.fill((0,0,255))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [width/2 - W/2, height/2 - H/2])

return Lives #return the new 'Lives' to the Main program

class MovesC(pygame.sprite.Sprite):

#define the constructor method of the class

def \_\_init\_\_(self, size, color, width, height, locationX, locationY):

Moves = 0 #set 'Moves' to 0

moves = "Moves: "+str(Moves) #create the string that will be blited on to the sprite

pygame.sprite.Sprite.\_\_init\_\_(self) #Call the parent class (Sprite) constructor

self.font = pygame.font.SysFont("Arial", size) #Set up the font for the text

self.textSurf = self.font.render(moves, 1, color) #set up the text with the string 'moves'

#set up the sprite's surf

self.surf = pygame.Surface((width, height))

self.surf.fill((0,0,255))

#set up the Moves counter as a rectangle in the specified part of the screen

self.rect = self.surf.get\_rect(center=(locationX,locationY))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [width/2 - W/2, height/2 - H/2])

def update(self, Moves, size, color, width, height):

moves = "Moves: "+str(Moves) #create the string that will be blited on to the sprite

self.font = pygame.font.SysFont("Arial", size) #Set up the font for the text

self.textSurf = self.font.render(moves, 1, color) #set up the text with the string 'moves'

#set up the sprite's surf

self.surf = pygame.Surface((width, height))

self.surf.fill((0,0,255))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [width/2 - W/2, height/2 - H/2])

class Timer(pygame.sprite.Sprite):

#define the constructor method of the class

def \_\_init\_\_(self, size, color, width, height, locationX, locationY):

Time = 0 #set 'Time' to 0

time = "Time: "+str(Time) #create the string that will be blited on to the sprite

pygame.sprite.Sprite.\_\_init\_\_(self) #Call the parent class (Sprite) constructor

self.font = pygame.font.SysFont("Arial", size) #Set up the font for the text

self.textSurf = self.font.render(time, 1, color) #set up the text with the string 'time'

#set up the sprite's surf

self.surf = pygame.Surface((width, height))

self.surf.fill((0,0,255))

#set up the Timer as a rectangle in the specified part of the screen

self.rect = self.surf.get\_rect(center=(locationX,locationY))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [width/2 - W/2, height/2 - H/2])

def update(self, Time, size, color, width, height):

time = "Time: "+str(Time) #create the string that will be blited on to the sprite

self.font = pygame.font.SysFont("Arial", size) #Set up the font for the text

self.textSurf = self.font.render(time, 1, color) #set up the text with the string 'time'

#set up the sprite's surf

self.surf = pygame.Surface((width, height))

self.surf.fill((0,0,255))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [width/2 - W/2, height/2 - H/2])

return Time #return the new 'Time' to the Main program

class SaveButton(pygame.sprite.Sprite):

#define the constructor method of the class

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

super(SaveButton, self).\_\_init\_\_() #Call the parent class (Sprite) constructor

#set up the sprite's surf

self.surf = pygame.Surface((100,100))

self.surf.fill((255,255,255))

#set up the instructionsImage as a rectangle in the middle of the screen

self.image = pygame.image.load('Save.png').convert()

self.rect = self.surf.get\_rect(center=(location))

class LoadButton(pygame.sprite.Sprite):

#define the constructor method of the class

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

super(LoadButton, self).\_\_init\_\_() #Call the parent class (Sprite) constructor

#set up the sprite's surf

self.surf = pygame.Surface((100,100))

self.surf.fill((255,255,255))

#set up the instructionsImage as a rectangle in the middle of the screen

self.image = pygame.image.load('Load.png').convert()

self.rect = self.surf.get\_rect(center=(location))

#Main program

import pygame

from tkinter import \*

from time import sleep

from random import randint,random

from Player\_class import Player

from Maze\_objects import Wall, Cell, Empty, Key

from Information\_objects import Scoreboard, LivesC, MovesC, Timer, SaveButton, LoadButton

from Starting\_objects import Title, Controls, Play, Instructions, Quit

from Other\_objects import controlsImage, instructionsImage, winScreen, loseScreen, SS\_image, LS\_image

from GameSave\_class import GameSave

from HighScores\_class import HighScores

from QS\_objects import Text1, Button1, Button2, Button3, Button4, Button5

MAZE\_HEIGHT = 10

MAZE\_WIDTH = 10

MAZE\_CONSTANT = 25

maze = []

class screens(object):

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

#set the window up with title

self.\_win = pygame.display.set\_mode((1900, 1000), pygame.RESIZABLE)

self.\_winXY = pygame.display.get\_window\_size()

pygame.display.set\_caption("Maths Maze Navigator")

#create sprite groups

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

def display():

pass

#the 'get\_questions' function which imports the questions from a file and returns them as a 2D array

def get\_questions(fileName):

questions = [] #initialises the variable 'questions' as an empty array

for line in fileName: #runs a FOR loop through each line of the file

newLine = line.strip("\n").split(",") #creates the array which will be appended to 'questions'

questions.append(newLine) #appends the array stored in 'newLine' to 'questions'

return questions #returns the 'questions' 2D array to the app class

def surroundingCells(maze, rand\_wall):

s\_cells = 0

if (maze[rand\_wall[0]-1][rand\_wall[1]] == True):

s\_cells += 1

if (maze[rand\_wall[0]+1][rand\_wall[1]] == True):

s\_cells += 1

if (maze[rand\_wall[0]][rand\_wall[1]-1] == True):

s\_cells +=1

if (maze[rand\_wall[0]][rand\_wall[1]+1] == True):

s\_cells += 1

return s\_cells

def create\_paths(maze, MAZE\_HEIGHT, MAZE\_WIDTH):

starting\_height = int(random()\*MAZE\_HEIGHT)

starting\_width = int(random()\*MAZE\_WIDTH)

if (starting\_height == 0):

starting\_height += 1

if (starting\_height == MAZE\_HEIGHT-1):

starting\_height -= 1

if (starting\_width == 0):

starting\_width += 1

if (starting\_width == MAZE\_WIDTH-1):

starting\_width -= 1

maze[starting\_height][starting\_width] = True

wallCoords = []

wallCoords.append([starting\_height - 1, starting\_width])

wallCoords.append([starting\_height, starting\_width - 1])

wallCoords.append([starting\_height, starting\_width + 1])

wallCoords.append([starting\_height + 1, starting\_width])

maze[starting\_height-1][starting\_width] = False

maze[starting\_height][starting\_width - 1] = False

maze[starting\_height][starting\_width + 1] = False

maze[starting\_height + 1][starting\_width] = False

while (wallCoords):

# Pick a random wall

rand\_wall = wallCoords[int(random()\*len(wallCoords))-1]

# Check if it is a left wall

if (rand\_wall[1] != 0):

if (maze[rand\_wall[0]][rand\_wall[1]-1] == 'None' and maze[rand\_wall[0]][rand\_wall[1]+1] == True):

# Find the number of surrounding cells

s\_cells = surroundingCells(maze, rand\_wall)

if (s\_cells < 2):

# Denote the new path

maze[rand\_wall[0]][rand\_wall[1]] = True

# Mark the new walls

# Upper cell

if (rand\_wall[0] != 0):

if (maze[rand\_wall[0]-1][rand\_wall[1]] != True):

maze[rand\_wall[0]-1][rand\_wall[1]] = False

if ([rand\_wall[0]-1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]-1, rand\_wall[1]])

# Bottom cell

if (rand\_wall[0] != MAZE\_HEIGHT-1):

if (maze[rand\_wall[0]+1][rand\_wall[1]] != True):

maze[rand\_wall[0]+1][rand\_wall[1]] = False

if ([rand\_wall[0]+1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]+1, rand\_wall[1]])

# Leftmost cell

if (rand\_wall[1] != 0):

if (maze[rand\_wall[0]][rand\_wall[1]-1] != True):

maze[rand\_wall[0]][rand\_wall[1]-1] = False

if ([rand\_wall[0], rand\_wall[1]-1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]-1])

# Delete wall

for wall in wallCoords:

if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):

wallCoords.remove(wall)

continue

#Check if it is an upper wall

if (rand\_wall[0] != 0):

if (maze[rand\_wall[0]-1][rand\_wall[1]] == 'None' and maze[rand\_wall[0]+1][rand\_wall[1]] == True):

s\_cells = surroundingCells(maze, rand\_wall)

if (s\_cells < 2):

# Denote the new path

maze[rand\_wall[0]][rand\_wall[1]] = True

# Mark the new walls

# Upper cell

if (rand\_wall[0] != 0):

if (maze[rand\_wall[0]-1][rand\_wall[1]] != True):

maze[rand\_wall[0]-1][rand\_wall[1]] = False

if ([rand\_wall[0]-1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]-1, rand\_wall[1]])

# Leftmost cell

if (rand\_wall[1] != 0):

if (maze[rand\_wall[0]][rand\_wall[1]-1] != True):

maze[rand\_wall[0]][rand\_wall[1]-1] = False

if ([rand\_wall[0], rand\_wall[1]-1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]-1])

# Rightmost cell

if (rand\_wall[1] != MAZE\_WIDTH-1):

if (maze[rand\_wall[0]][rand\_wall[1]+1] != True):

maze[rand\_wall[0]][rand\_wall[1]+1] = False

if ([rand\_wall[0], rand\_wall[1]+1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]+1])

# Delete wall

for wall in wallCoords:

if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):

wallCoords.remove(wall)

continue

# Check the bottom wall

if (rand\_wall[0] != MAZE\_HEIGHT-1):

if (maze[rand\_wall[0]+1][rand\_wall[1]] == 'None' and maze[rand\_wall[0]-1][rand\_wall[1]] == True):

s\_cells = surroundingCells(maze, rand\_wall)

if (s\_cells < 2):

# Denote the new path

maze[rand\_wall[0]][rand\_wall[1]] = True

# Mark the new walls

if (rand\_wall[0] != MAZE\_HEIGHT-1):

if (maze[rand\_wall[0]+1][rand\_wall[1]] != True):

maze[rand\_wall[0]+1][rand\_wall[1]] = False

if ([rand\_wall[0]+1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]+1, rand\_wall[1]])

if (rand\_wall[1] != 0):

if (maze[rand\_wall[0]][rand\_wall[1]-1] != True):

maze[rand\_wall[0]][rand\_wall[1]-1] = False

if ([rand\_wall[0], rand\_wall[1]-1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]-1])

if (rand\_wall[1] != MAZE\_WIDTH-1):

if (maze[rand\_wall[0]][rand\_wall[1]+1] != True):

maze[rand\_wall[0]][rand\_wall[1]+1] = False

if ([rand\_wall[0], rand\_wall[1]+1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]+1])

# Delete wall

for wall in wallCoords:

if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):

wallCoords.remove(wall)

continue

# Check the right wall

if (rand\_wall[1] != MAZE\_WIDTH-1):

if (maze[rand\_wall[0]][rand\_wall[1]+1] == 'None' and maze[rand\_wall[0]][rand\_wall[1]-1] == True):

s\_cells = surroundingCells(maze, rand\_wall)

if (s\_cells < 2):

# Denote the new path

maze[rand\_wall[0]][rand\_wall[1]] = True

# Mark the new walls

if (rand\_wall[1] != MAZE\_WIDTH-1):

if (maze[rand\_wall[0]][rand\_wall[1]+1] != True):

maze[rand\_wall[0]][rand\_wall[1]+1] = False

if ([rand\_wall[0], rand\_wall[1]+1] not in wallCoords):

wallCoords.append([rand\_wall[0], rand\_wall[1]+1])

if (rand\_wall[0] != MAZE\_HEIGHT-1):

if (maze[rand\_wall[0]+1][rand\_wall[1]] != True):

maze[rand\_wall[0]+1][rand\_wall[1]] = False

if ([rand\_wall[0]+1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]+1, rand\_wall[1]])

if (rand\_wall[0] != 0):

if (maze[rand\_wall[0]-1][rand\_wall[1]] != True):

maze[rand\_wall[0]-1][rand\_wall[1]] = False

if ([rand\_wall[0]-1, rand\_wall[1]] not in wallCoords):

wallCoords.append([rand\_wall[0]-1, rand\_wall[1]])

# Delete wall

for wall in wallCoords:

if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):

wallCoords.remove(wall)

continue

# Delete the wall from the list anyway

for wall in wallCoords:

if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):

wallCoords.remove(wall)

return maze

def generate\_maze(MAZE\_HEIGHT, MAZE\_WIDTH, maze, walls, cells, all\_sprites, midX, midY):

visited = [] #set up an empty array for the cells the player has visited

#set up a FOR loop for the rows of the maze and visited

for x in range(0, MAZE\_HEIGHT): #initiates a FOR loop for the maze/visited rows

visitedRow = [] #initialise the 'visitedRow' as an empty array

mazeRow = [] #initialise the 'mazeRow' as an empty array

for y in range(0, MAZE\_WIDTH): #set up a nested FOR loop for the maze/visited columns

mazeRow.append("None") #append the value which will be used for empty sections of the maze

visitedRow.append(False) #append the value which will be used for cells the player hasn't visited yet

maze.append(mazeRow) #append the 'mazeRow' full of 'None' strings to 'maze'

visited.append(visitedRow) #append the 'visitedRow' full of 'False' booleans to 'visited'

#calls the 'create\_paths' function which returns the 'maze' array

maze = create\_paths(maze, MAZE\_HEIGHT, MAZE\_WIDTH)

#set up a FOR loop for the rows of the maze

for i in range(0, MAZE\_HEIGHT):

for j in range(0, MAZE\_WIDTH): #set up a nested FOR loop for the maze columns

if (maze[i][j] == 'None'): #use an IF statement to check if an item in 'maze' is 'None'

maze[i][j] = False #if it is 'None' sets it to False for the walls of the maze

#sets the entrance and exit of the maze

maze[0][1] = 'None'

maze[MAZE\_WIDTH-1][MAZE\_WIDTH-2] = 'None'

#set up a FOR loop for the rows of the maze

for x in range(0, MAZE\_HEIGHT):

for y in range(0, MAZE\_WIDTH): #set up a nested FOR loop for the maze columns

X = (MAZE\_CONSTANT\*x)+(midX-((MAZE\_CONSTANT\*MAZE\_WIDTH) / 2)) #set the X coordinate to be 25x bigger than maze index

Y = (MAZE\_CONSTANT\*y)+(midY-(MAZE\_CONSTANT\*MAZE\_HEIGHT)) #set the Y coordinate to be 25x bigger than maze index

if maze[x][y] == False: #check if maze index is a wall

wall = Wall([X,Y]) #set up a wall sprite with X and Y as the location

walls.add(wall) #add it to the 'walls' group

all\_sprites.add(wall) #add it to the 'all\_sprites' group

elif maze[x][y] == True: #check if maze index is a cell

cell = Cell([X,Y]) #set up a cell sprite with X and Y as the location

cells.add(cell) #add it to the 'cells' group

all\_sprites.add(cell) #add it to the 'all\_sprites' group

else: #otherwise:

empty = Empty([X,Y]) #set up an empty sprite with X and Y as the location

all\_sprites.add(empty) #adds it to the 'all\_sprites' group

if x == (MAZE\_WIDTH-1): #if it is the exit of the maze

keys = pygame.sprite.Group() #create a sprite group for keys

key = Key("Key.png", [X,Y]) #set up a key sprite with X and Y as the location

all\_sprites.add(key) #adds it to the 'all\_sprites' group

keys.add(key) #adds it to the 'keys' group

return keys, visited #returns the 'keys' group to the 'pygame\_window' class and also visited 2D array

#Class for a Tkinter window called 'QuestionWindow'

class QuestionWindow(Tk):

def \_\_init\_\_(self, questions, score):

super().\_\_init\_\_() #uses the super construcor for 'Tk'

#set the window attributes up

self.geometry('2000x500')

self.title('Questions')

self.configure(bg="#12AAE6")

#set the question up for the main label

self.question = questions[0]

thisQuestion = self.question[0]

#set the multi-choice answers for the buttons

answer1 = self.question[1]

answer2 = self.question[2]

answer3 = self.question[3]

answer4 = self.question[4]

#set some variables

correctAnswer = answer1

self.Score = StringVar()

self.movesLeft = 0

self.score = score

self.status = "Incorrect!"

#set the question text label up

label1 = Label(self, text=thisQuestion, bg="#12AAE6")

label1.config(font=('Helvetica bold',20))

label1.grid(row=0,columnspan=2) #arrange using the grid method in row 0

#set up the buttons for the answer choices

number1 = Button(self, text=answer1, bg="#12AAE6", command=lambda: self.ansPressed(answer1, correctAnswer))

number1.grid(row=1, column=0) #arrange using the grid method in row 1, col 0

number2 = Button(self, text=answer2, bg="#12AAE6", command=lambda: self.ansPressed(answer2, correctAnswer))

number2.grid(row=1, column=1) #arrange using the grid method in row 1, col 1

number3 = Button(self, text=answer3, bg="#12AAE6", command=lambda: self.ansPressed(answer3, correctAnswer))

number3.grid(row=2, column=0) #arrange using the grid method in row 2, col 0

number4 = Button(self, text=answer4, bg="#12AAE6", command=lambda: self.ansPressed(answer4, correctAnswer))

number4.grid(row=2, column=1) #arrange using the grid method in row 2, col 1

#set the score box as a disabled entry that only the program can update

self.scoreBox = Entry(self, bg="#12AAE6", textvariable=self.Score)

self.scoreBox.config(state= "disabled")

self.scoreBox.grid(row=3, column=1) #arrange this using grid method in row 3, col 1

self.Score.set(str(self.score))

#set the score text label up

label2 = Label(self, text="Score:", bg="#12AAE6")

label2.config(font=('Helvetica bold',10))

label2.grid(row=3, column=0) #arrange using the grid method in row 3, col 0

def ansPressed(self, answer, correctAnswer):

if answer == correctAnswer: #check if the answer the user chose is correct

self.status = "Correct!" #if it is, set status to 'Correct!'

self.score += 1 #increment the score by 1

else: #otherwise

self.status = "Incorrect!" #if it is, set status to 'Incorrect!'

self.score -= 1 #decrement the score by 1

self.Score.set(str(self.score)) #set the 'Score' text variable to be the attribute 'currentScore'

self.destroy()

def return\_info(self):

self.currentScore = self.score

if self.status == "Correct!":

self.movesLeft = 5

else:

self.movesLeft = 1

return self.currentScore, self.movesLeft

class QuestionSelect\_screen(screens):

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

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

#initiate pygame

pygame.init()

#Calculate where the middle of the screen is

self.\_\_winX = self.\_winXY[0]

self.\_\_winY = self.\_winXY[1]

self.\_\_midX = self.\_\_winX / 2

self.\_\_midY = self.\_\_winY / 2

#instantiate sprites

self.\_\_locX1 = self.\_\_midX

self.\_\_locY1 = 150

self.\_\_locY2 = self.\_\_midY - 200

self.\_\_locY3 = self.\_\_midY - 50

self.\_\_locY4 = self.\_\_midY + 100

self.\_\_locY5 = self.\_\_midY + 250

self.\_\_locY6 = self.\_\_winY - 100

self.\_\_SyQD = Text1((self.\_\_locX1,self.\_\_locY1),"Select your Question Difficulty")

self.\_\_GCSE = Button1((self.\_\_locX1,self.\_\_locY2),"GCSE")

self.\_\_Easy = Button2((self.\_\_locX1,self.\_\_locY3),"Easy")

self.\_\_Medium = Button3((self.\_\_locX1,self.\_\_locY4),"Medium")

self.\_\_Hard = Button4((self.\_\_locX1,self.\_\_locY5),"Hard")

self.\_\_NS = Button5((self.\_\_locX1,self.\_\_locY6),"Ninson's specials")

#add sprites to all\_sprites

self.\_all\_sprites.add(self.\_\_SyQD)

self.\_all\_sprites.add(self.\_\_GCSE)

self.\_all\_sprites.add(self.\_\_Easy)

self.\_all\_sprites.add(self.\_\_Medium)

self.\_all\_sprites.add(self.\_\_Hard)

self.\_all\_sprites.add(self.\_\_NS)

def display(self, qs\_diff):

#get location for buttons where mouse could be clicked

loc1 = self.\_\_locX1 - 100

loc2 = self.\_\_locX1 + 100

loc3 = self.\_\_locY5 - 50

loc4 = self.\_\_locY5 + 50

loc5 = self.\_\_locY2 - 50

loc6 = self.\_\_locY2 + 50

loc7 = self.\_\_locY3 - 50

loc8 = self.\_\_locY3 + 50

loc9 = self.\_\_locY4 - 50

loc10 = self.\_\_locY4 + 50

loc11 = self.\_\_locY6 - 50

loc12 = self.\_\_locY6 + 50

run = True

#initiate a while loop until run is no longer True

while run:

#initiate a for loop to detect events

for event in pygame.event.get():

#allows the user to quit the game

if event.type == pygame.QUIT:

run = False #sets run to False which breaks the loop

if event.type == pygame.KEYDOWN: #checks if a keyboard key is pressed

if event.key == pygame.K\_f: #if the key is f

pygame.display.toggle\_fullscreen() #toggles fullscreen mode of the pygame window

if event.type == pygame.MOUSEBUTTONDOWN: #checks if the mouse is clicked

if loc1 <= mouse[0] <= loc2 and loc5 <= mouse[1] <= loc6:

qs\_diff = "GCSE"

game.run\_main(qs\_diff)

if loc1 <= mouse[0] <= loc2 and loc7 <= mouse[1] <= loc8:

qs\_diff = "Easy"

game.run\_main(qs\_diff)

if loc1 <= mouse[0] <= loc2 and loc9 <= mouse[1] <= loc10:

qs\_diff = "Medium"

game.run\_main(qs\_diff)

if loc1 <= mouse[0] <= loc2 and loc3 <= mouse[1] <= loc4:

qs\_diff = "Hard"

game.run\_main(qs\_diff)

if loc1 <= mouse[0] <= loc2 and loc11 <= mouse[1] <= loc12:

qs\_diff = "NS"

game.run\_main(qs\_diff)

#fills the pygame window with white

self.\_win.fill((0,0,0))

#stores the position of the mouse

mouse = pygame.mouse.get\_pos()

for sprite in self.\_all\_sprites: #set a for loop to cycle through the whole list of sprites

self.\_win.blit(sprite.surf, sprite.rect) #blit each sprite on to the screen

pygame.display.update() #update the display with all sprites blitted on

#detects if run is set to False

if not run:

sleep(2) #waits for 2 seconds

pygame.quit() #closes the pygame window

class high\_scores\_screen2(Toplevel):

def \_\_init\_\_(self, parent, score, name, movesBonus, questionsBonus, livesBonus):

super().\_\_init\_\_(parent) #use the super constructor of hss1

high\_scores = HighScores() #instantiate the 'HighScores' class

scoreValid = None #set up the varaible to check if the score is valid as 'None'

fileIn = open('high\_scores.txt', 'r') #open the file that will be passed into the 'HighScores' class

high\_scores.get\_scores(fileIn) #get the scores from the file

fileIn.close() #close the file that was passed in

if high\_scores.check\_score(score): #checks the score and if it returns True then...

scoreValid = True #sets ScoreValid to True

high\_scores.add\_score(name, score) #adds the score

sortedNames = high\_scores.sortNames() #sorts the names dictionary and returns it as an array of tuples

sortedNames = sortedNames[::-1] #reverse it so it's in desending order

fileOut = open('high\_scores.txt', 'w') #opens the file which will be written to

high\_scores.saveScore(fileOut, sortedNames) #saves the name and score to the file

fileOut.close() #closes the file that was written to

else: #otherwise...

scoreValid = False #sets scoreValid to False

#set the window attributes up

self.geometry('1650x1000')

self.title('Maths Maze Navigator')

self.configure(bg="#12AAE6")

#set the main text label up and arrange using the grid method in row 0

label1 = Label(self, text="High Scores", bg="#12AAE6")

label1.config(font=('Helvetica bold',40))

label1.grid(row=0,columnspan=2)

#set the textBox up which will display the highscores and arrange using grid method in row 1

self.textBox = Text(self, background="#12AAE6")

self.textBox.grid(row=1, columnspan=2)

if scoreValid: #checks if the score is valid

self.textBox.delete("1.0", END) #if it is clears the textBox

else: #otherwise...

sortedNames = high\_scores.sortNames() #gets the sortedNames array

sortedNames = sortedNames[::-1] #reverse it so it's in desending order

self.textBox.delete("1.0", END) #clears the textBox anyway

for num in range(1, len(sortedNames)+1): #uses a FOR loop to iterate over over the sortedNames array

nameScore = sortedNames[num-1] #get the current name and score and stores it

name = nameScore[0] #gets the name seperately

score = nameScore[1] #gets the score seperately

text = str(num)+"."+name+": "+str(score)+"\n" #creates the text which will be inserted into the textBox

self.textBox.insert(END, text) #inserts the text into the textBox

#set up a button which closes the window

Button(self,text='More Info',bg="#12AAE6",command=lambda: self.displayInfo(movesBonus, questionsBonus, livesBonus)).grid(row=2, column=0)

Button(self,text='Close',bg="#12AAE6",command=self.destroy).grid(row=2, column=1)

def displayInfo(self, movesBonus, questionsBonus, livesBonus):

text1 = f"Moves bonus: {movesBonus} \n"

text2 = f"Questions bonus: {questionsBonus} \n"

text3 = f"Lives bonus: {livesBonus} \n"

text4 = "Completion bonus: 500 \n"

self.textBox.insert(END, "\n")

self.textBox.insert(END, text1) #inserts the text into the textBox

self.textBox.insert(END, text2) #inserts the text into the textBox

self.textBox.insert(END, text3) #inserts the text into the textBox

self.textBox.insert(END, text4) #inserts the text into the textBox

class high\_scores\_screen1(Tk):

def \_\_init\_\_(self, score, movesBonus, questionsBonus, livesBonus):

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

#set the window attributes up

self.geometry('1650x1000')

self.title('Maths Maze Navigator')

self.configure(bg="#12AAE6")

text = f"Your score is: {score}"

#set the main text label up and arrange using the grid method in row 0

label1 = Label(self, text=text, bg="#12AAE6")

label1.config(font=('Helvetica bold',40))

label1.grid(row=0,columnspan=2)

#set up the secondary text label and arrange using the grid method in row 1, column 1

label2 = Label(self, text="Enter name here:", bg="#12AAE6")

label2.config(font=('Helvetica Bold',20))

label2.grid(row=1,column=0)

#set up the entry where the user will enter their name

self.entry1 = Entry(self, bg="#12AAE6")

self.entry1.grid(row=1, column=1) #arrange using grid method in row 1, column 1

#set up the textBox and arrange using grid method in row 2

self.textBox = Text(self, background="#12AAE6")

self.textBox.grid(row=2, columnspan=2)

#set up a button which opens the new window for the high scores

Button(self,text='Submit score',bg="#12AAE6",command=lambda: self.open\_window(score, movesBonus, questionsBonus, livesBonus)).grid(row=3, column=0)

#set up a button which closes the window

Button(self,text='Close',bg="#12AAE6",command=self.destroy).grid(row=3, column=1)

#the method to open the new window and check entry

def open\_window(self, score, movesBonus, questionsBonus, livesBonus):

name = self.entry1.get() #get the name from the entry

if len(name) > 10 or len(name) < 2: #check if the name is valid

#if it isn't, insert a suitable error message into the textBox

text = "Error: Name has to be between 2 and 10 characters long!"

self.textBox.delete("1.0", END)

self.textBox.insert("1.0", text)

else: #otherwise...

hss2 = high\_scores\_screen2(self, score, name, movesBonus, questionsBonus, livesBonus) #calls the Tkinter window 'high\_scores\_screen2'

hss2.grab\_set() #makes it so that events can run in the tkinter window

class save\_screen(screens):

def \_\_init\_\_(self, maze, CurrentScore, lives, moves, Time):

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

#initiate pygame

pygame.init()

#instantiate sprites

self.SaveScreen = SS\_image()

#add sprites to all\_sprites

self.\_all\_sprites.add(self.SaveScreen)

#instantiate the 'GameSave' class

self.\_\_gameSave = GameSave(maze, CurrentScore, lives, moves, Time)

def save(self, coords):

file = open("mazeInfo.txt", 'w')

self.\_\_gameSave.writeToFile(file, coords)

file.close()

def display(self, qs\_diff):

run = True

#initiate a while loop until run is no longer True

while run:

#initiate a for loop to detect events

for event in pygame.event.get():

#allows the user to quit the game

if event.type == pygame.QUIT:

run = False #sets run to False which breaks the loop

if event.type == pygame.KEYDOWN: #checks if a keyboard key is pressed

if event.key == pygame.K\_f: #if the key is f

pygame.display.toggle\_fullscreen() #toggles fullscreen mode of the pygame window

if event.type == pygame.MOUSEBUTTONDOWN:

game.run\_main(qs\_diff)

#fills the pygame window with white

self.\_win.fill((0,0,0))

for sprite in self.\_all\_sprites: #set a for loop to cycle through the whole list of sprites

self.\_win.blit(sprite.image, sprite.rect) #blit each sprite on to the screen

pygame.display.update() #update the display with all sprites blitted on

#detects if run is set to False

if not run:

sleep(2) #waits for 2 seconds

pygame.quit() #closes the pygame window

class load\_screen(screens):

def \_\_init\_\_(self, maze, CurrentScore, lives, moves, Time, playerPos, midX, midY):

#set up attributes using passed parameters

self.\_\_savedMaze = maze

self.\_\_CurrentScore = CurrentScore

self.\_\_lives = lives

self.\_\_moves = moves

self.\_\_Time = Time

self.\_\_playerPos = playerPos

self.\_\_midX = midX

self.\_\_midY = midY

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

#initiate pygame

pygame.init()

#instantiate sprites

self.LoadScreen = LS\_image()

#add sprites to all\_sprites

self.\_all\_sprites.add(self.LoadScreen)

def display(self, qs\_diff):

run = True

#initiate a while loop until run is no longer True

while run:

#initiate a for loop to detect events

for event in pygame.event.get():

#allows the user to quit the game

if event.type == pygame.QUIT:

run = False #sets run to False which breaks the loop

if event.type == pygame.KEYDOWN: #checks if a keyboard key is pressed

if event.key == pygame.K\_f: #if the key is f

pygame.display.toggle\_fullscreen() #toggles fullscreen mode of the pygame window

if event.type == pygame.MOUSEBUTTONDOWN:

game.run\_main(qs\_diff)

#fills the pygame window with white

self.\_win.fill((0,0,0))

for sprite in self.\_all\_sprites: #set a for loop to cycle through the whole list of sprites

self.\_win.blit(sprite.image, sprite.rect) #blit each sprite on to the screen

pygame.display.update() #update the display with all sprites blitted on

#detects if run is set to False

if not run:

sleep(2) #waits for 2 seconds

pygame.quit() #closes the pygame window

def loadMaze(self, maze, midX, midY):

all\_sprites = pygame.sprite.Group()

walls = pygame.sprite.Group()

cells = pygame.sprite.Group()

#set up a FOR loop for the rows of the maze

for x in range(0, MAZE\_HEIGHT):

for y in range(0, MAZE\_WIDTH): #set up a nested FOR loop for the maze columns

X = (MAZE\_CONSTANT\*x)+(midX-((MAZE\_CONSTANT\*MAZE\_WIDTH) / 2)) #set the X coordinate to be 25x bigger than maze index

Y = (MAZE\_CONSTANT\*y)+(midY-(MAZE\_CONSTANT\*MAZE\_HEIGHT)) #set the Y coordinate to be 25x bigger than maze index

if maze[x][y] == False: #check if maze index is a wall

wall = Wall([X,Y]) #set up a wall sprite with X and Y as the location

walls.add(wall) #add it to the 'walls' group

all\_sprites.add(wall) #add it to the 'all\_sprites' group

elif maze[x][y] == True: #check if maze index is a cell

cell = Cell([X,Y]) #set up a cell sprite with X and Y as the location

cells.add(cell) #add it to the 'cells' group

all\_sprites.add(cell) #add it to the 'all\_sprites' group

else: #otherwise:

empty = Empty([X,Y]) #set up an empty sprite with X and Y as the location

all\_sprites.add(empty) #adds it to the 'all\_sprites' group

return walls, cells, all\_sprites

def loadGame(self, all\_sprites, MAZE\_HEIGHT, MAZE\_WIDTH):

self.\_new\_sprites = all\_sprites

pixelX = (MAZE\_CONSTANT\*int(self.\_\_playerPos[1]))+(self.\_\_midX-((MAZE\_CONSTANT\*MAZE\_WIDTH) / 2))

pixelY = (MAZE\_CONSTANT\*int(self.\_\_playerPos[4]))+(self.\_\_midY-(MAZE\_CONSTANT\*MAZE\_HEIGHT))

player = Player("Player.png",[pixelX, pixelY])

self.\_\_locX1 = self.\_\_midX - 250

self.\_\_locX2 = self.\_\_midX - 100

self.\_\_locX3 = self.\_\_midX + 100

self.\_\_locX4 = self.\_\_midX + 250

self.\_\_locY = self.\_\_midY + (self.\_\_midY / 2)

self.\_\_scoreBoard = Scoreboard(15,"white",100,150,self.\_\_locX1,self.\_\_locY)

self.\_\_livesCounter = LivesC(15,"white",100,150,self.\_\_locX2,self.\_\_locY)

self.\_\_livesCounter.set(self.\_\_lives,15,"white",100,150)

self.\_\_movesCounter = MovesC(15,"white",100,150,self.\_\_locX3,self.\_\_locY)

self.\_\_movesCounter.update(self.\_\_moves,15,"white",100,150)

self.\_\_timer = Timer(15,"white",100,150,self.\_\_locX4,self.\_\_locY)

self.\_\_timer.update(self.\_\_Time,15,"white",100,150)

self.\_new\_sprites.add(player)

self.\_new\_sprites.add(self.\_\_scoreBoard)

self.\_new\_sprites.add(self.\_\_livesCounter)

self.\_new\_sprites.add(self.\_\_movesCounter)

self.\_new\_sprites.add(self.\_\_timer)

return self.\_new\_sprites, player, self.\_\_moves, self.\_\_Time

class win\_screen(screens):

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

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

#initiate pygame

pygame.init()

#instantiate sprites

self.WinScreen = winScreen()

#add sprites to all\_sprites

self.\_all\_sprites.add(self.WinScreen)

def display(self, currentScore, movesBonus, questionsBonus, livesBonus):

run = True

#initiate a while loop until run is no longer True

while run:

#initiate a for loop to detect events

for event in pygame.event.get():

#allows the user to quit the game

if event.type == pygame.QUIT:

run = False #sets run to False which breaks the loop

if event.type == pygame.KEYDOWN: #checks if a keyboard key is pressed

if event.key == pygame.K\_f: #if the key is f

pygame.display.toggle\_fullscreen() #toggles fullscreen mode of the pygame window

if event.type == pygame.MOUSEBUTTONDOWN:

hss1 = high\_scores\_screen1(currentScore, movesBonus, questionsBonus, livesBonus) #calls the Tkinter window 'high\_scores\_screen1'

hss1.mainloop() #makes it so that events can run in the tkinter window

#fills the pygame window with white

self.\_win.fill((0,0,0))

for sprite in self.\_all\_sprites: #set a for loop to cycle through the whole list of sprites

self.\_win.blit(sprite.image, sprite.rect) #blit each sprite on to the screen

pygame.display.update() #update the display with all sprites blitted on

#detects if run is set to False

if not run:

sleep(2) #waits for 2 seconds

pygame.quit() #closes the pygame window

class lose\_screen(screens):

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

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

#initiate pygame

pygame.init()

#instantiate sprites

self.LoseScreen = loseScreen()

#add sprites to all\_sprites

self.\_all\_sprites.add(self.LoseScreen)

def display(self):

run = True

#initiate a while loop until run is no longer True

while run:

#initiate a for loop to detect events

for event in pygame.event.get():

#allows the user to quit the game

if event.type == pygame.QUIT:

run = False #sets run to False which breaks the loop

if event.type == pygame.KEYDOWN: #checks if a keyboard key is pressed

if event.key == pygame.K\_f: #if the key is f

pygame.display.toggle\_fullscreen() #toggles fullscreen mode of the pygame window

if event.type == pygame.MOUSEBUTTONDOWN:

run = False

#fills the pygame window with white

self.\_win.fill((0,0,0))

for sprite in self.\_all\_sprites: #set a for loop to cycle through the whole list of sprites

self.\_win.blit(sprite.image, sprite.rect) #blit each sprite on to the screen

pygame.display.update() #update the display with all sprites blitted on

#detects if run is set to False

if not run:

sleep(2) #waits for 2 seconds

pygame.quit() #closes the pygame window

class controls\_screen(screens):

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

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

#initiate pygame

pygame.init()

#instantiate sprites

self.\_\_controls\_image = controlsImage()

#add sprites to all\_sprites

self.\_all\_sprites.add(self.\_\_controls\_image)

def display(self):

run = True

#initiate a while loop until run is no longer True

while run:

#initiate a for loop to detect events

for event in pygame.event.get():

#allows the user to quit the game

if event.type == pygame.QUIT:

run = False #sets run to False which breaks the loop

if event.type == pygame.KEYDOWN: #checks if a keyboard key is pressed

if event.key == pygame.K\_f: #if the key is f

pygame.display.toggle\_fullscreen() #toggles fullscreen mode of the pygame window

if event.type == pygame.MOUSEBUTTONDOWN:

game.run\_main()

#fills the pygame window with white

self.\_win.fill((0,0,0))

for sprite in self.\_all\_sprites: #set a for loop to cycle through the whole list of sprites

self.\_win.blit(sprite.image, sprite.rect) #blit each sprite on to the screen

pygame.display.update() #update the display with all sprites blitted on

#detects if run is set to False

if not run:

sleep(2) #waits for 2 seconds

pygame.quit() #closes the pygame window

class instructions\_screen(screens):

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

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

#initiate pygame

pygame.init()

#instantiate sprites

self.\_\_instructions\_image = instructionsImage()

#add sprites to all\_sprites

self.\_all\_sprites.add(self.\_\_instructions\_image)

def display(self):

run = True

#initiate a while loop until run is no longer True

while run:

#initiate a for loop to detect events

for event in pygame.event.get():

#allows the user to quit the game

if event.type == pygame.QUIT:

run = False #sets run to False which breaks the loop

if event.type == pygame.KEYDOWN: #checks if a keyboard key is pressed

if event.key == pygame.K\_f: #if the key is f

pygame.display.toggle\_fullscreen() #toggles fullscreen mode of the pygame window

if event.type == pygame.MOUSEBUTTONDOWN:

game.run\_main()

#fills the pygame window with white

self.\_win.fill((0,0,0))

for sprite in self.\_all\_sprites: #set a for loop to cycle through the whole list of sprites

self.\_win.blit(sprite.image, sprite.rect) #blit each sprite on to the screen

pygame.display.update() #update the display with all sprites blitted on

#detects if run is set to False

if not run:

sleep(2) #waits for 2 seconds

pygame.quit() #closes the pygame window

class start\_screen(screens):

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

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

#initiate pygame

pygame.init()

#Calculate where the middle of the screen is

self.\_\_winX = self.\_winXY[0]

self.\_\_winY = self.\_winXY[1]

self.\_\_midX = self.\_\_winX / 2

self.\_\_midY = self.\_\_winY / 2

#instantiate sprites

self.\_\_locX1 = self.\_\_midX

self.\_\_locX2 = 400

self.\_\_locX3 = self.\_\_midX - 200

self.\_\_locX4 = self.\_\_midX + 200

self.\_\_locX5 = self.\_\_winX - 400

self.\_\_locY1 = self.\_\_midY - (self.\_\_midY / 2)

self.\_\_locY2 = self.\_\_midY + (self.\_\_midY / 2)

self.\_\_title = Title((self.\_\_locX1,self.\_\_locY1),"Welcome to Maths Maze Navigator")

self.\_\_controls = Controls((self.\_\_locX2,self.\_\_locY2),"Controls")

self.\_\_play = Play((self.\_\_locX3,self.\_\_locY2),"Play game")

self.\_\_instructions = Instructions((self.\_\_locX4,self.\_\_locY2),"How to play")

self.\_\_quit = Quit((self.\_\_locX5,self.\_\_locY2),"Quit!")

#add sprites to all\_sprites

self.\_all\_sprites.add(self.\_\_title)

self.\_all\_sprites.add(self.\_\_controls)

self.\_all\_sprites.add(self.\_\_play)

self.\_all\_sprites.add(self.\_\_instructions)

self.\_all\_sprites.add(self.\_\_quit)

def display(self):

#get location for buttons where mouse could be clicked

loc1 = self.\_\_locX2 - 100

loc2 = self.\_\_locX2 + 100

loc3 = self.\_\_locY2 - 50

loc4 = self.\_\_locY2 + 50

loc5 = self.\_\_locX3 - 100

loc6 = self.\_\_locX3 + 100

loc7 = self.\_\_locX4 - 100

loc8 = self.\_\_locX4 + 100

loc9 = self.\_\_locX5 - 100

loc0 = self.\_\_locX5 + 100

run = True

#initiate a while loop until run is no longer True

while run:

#initiate a for loop to detect events

for event in pygame.event.get():

#allows the user to quit the game

if event.type == pygame.QUIT:

run = False #sets run to False which breaks the loop

if event.type == pygame.KEYDOWN: #checks if a keyboard key is pressed

if event.key == pygame.K\_f: #if the key is f

pygame.display.toggle\_fullscreen() #toggles fullscreen mode of the pygame window

if event.type == pygame.MOUSEBUTTONDOWN: #checks if the mouse is clicked

if loc1 <= mouse[0] <= loc2 and loc3 <= mouse[1] <= loc4:

game.run\_controls()

if loc5 <= mouse[0] <= loc6 and loc3 <= mouse[1] <= loc4:

game.run\_qss()

if loc7 <= mouse[0] <= loc8 and loc3 <= mouse[1] <= loc4:

game.run\_instructions()

if loc9 <= mouse[0] <= loc0 and loc3 <= mouse[1] <= loc4:

run = False

#fills the pygame window with white

self.\_win.fill((0,0,0))

#stores the position of the mouse

mouse = pygame.mouse.get\_pos()

for sprite in self.\_all\_sprites: #set a for loop to cycle through the whole list of sprites

self.\_win.blit(sprite.surf, sprite.rect) #blit each sprite on to the screen

pygame.display.update() #update the display with all sprites blitted on

#detects if run is set to False

if not run:

sleep(2) #waits for 2 seconds

pygame.quit() #closes the pygame window

class pygame\_window(screens):

def \_\_init\_\_(self, MAZE\_HEIGHT, MAZE\_WIDTH, maze):

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

#assign parameters to variables

self.\_\_maze\_height = MAZE\_HEIGHT

self.\_\_maze\_width = MAZE\_WIDTH

self.\_\_maze = maze

#initiate pygame

pygame.init()

#Calculate where the middle of the screen is

self.\_\_winX = self.\_winXY[0]

self.\_\_winY = self.\_winXY[1]

self.\_\_midX = self.\_\_winX / 2

self.\_\_midY = self.\_\_winY / 2

#instantiate sprites

self.\_\_player = Player("Player.png",[800,275])

self.\_\_locX1 = self.\_\_midX - 250

self.\_\_locX2 = self.\_\_midX - 100

self.\_\_locX3 = self.\_\_midX + 100

self.\_\_locX4 = self.\_\_midX + 250

self.\_\_locY = self.\_\_midY + (self.\_\_midY / 2)

self.\_\_locY1 = self.\_\_midY - (self.\_\_midY / 1.5)

self.\_\_scoreBoard = Scoreboard(15,"white",100,150,self.\_\_locX1,self.\_\_locY)

self.\_\_livesCounter = LivesC(15,"white",100,150,self.\_\_locX2,self.\_\_locY)

self.\_\_movesCounter = MovesC(15,"white",100,150,self.\_\_locX3,self.\_\_locY)

self.\_\_timer = Timer(15,"white",100,150,self.\_\_locX4,self.\_\_locY)

self.\_\_saveButton = SaveButton((self.\_\_locX3,self.\_\_locY1))

self.\_\_loadButton = LoadButton((self.\_\_locX4,self.\_\_locY1))

#Set variables for Scoreboard, Lives counter, Moves counter and Timer

self.\_\_CurrentScore = 0

self.\_\_lives = 3

self.\_\_moves = 0

self.\_\_movesLeft = 5

self.\_\_Time = 0

self.\_\_coords = [0, 1]

self.\_\_file = None

#instantiate the 'GameSave' class

self.\_\_gameSave = GameSave(maze, self.\_\_CurrentScore, self.\_\_lives, self.\_\_moves, self.\_\_Time)

#create sprite groups

self.\_\_walls = pygame.sprite.Group()

self.\_\_cells = pygame.sprite.Group()

#add sprites to all\_sprites

self.\_all\_sprites.add(self.\_\_scoreBoard)

self.\_all\_sprites.add(self.\_\_livesCounter)

self.\_all\_sprites.add(self.\_\_movesCounter)

self.\_all\_sprites.add(self.\_\_timer)

def display(self, qs\_diff):

#call the function to generate the maze and 'catch' the 2 variables it returns

keys, visited = generate\_maze(self.\_\_maze\_height, self.\_\_maze\_width, self.\_\_maze, self.\_\_walls,

self.\_\_cells, self.\_all\_sprites, self.\_\_midX, self.\_\_midY)

#choose which file to get questions from

if qs\_diff == "GCSE":

self.\_\_file = open("GCSE\_qs.txt", mode='r')

elif qs\_diff == "Easy":

self.\_\_file = open("Easy\_qs.txt", mode='r')

elif qs\_diff == "Medium":

self.\_\_file = open("Medium\_qs.txt", mode='r')

elif qs\_diff == "Hard":

self.\_\_file = open("Hard\_qs.txt", mode='r')

elif qs\_diff == "NS":

self.\_\_file = open("NS\_qs.txt", mode='r')

else:

print("An error has occured!")

#get location for buttons where mouse could be clicked

loc1 = self.\_\_locX3 - 50

loc2 = self.\_\_locX3 + 50

loc3 = self.\_\_locX4 - 50

loc4 = self.\_\_locX4 + 50

loc5 = self.\_\_locY1 - 50

loc6 = self.\_\_locY1 - 20

self.score = 0

questions = get\_questions(self.\_\_file)

self.calls = 0

self.crashes = 0

self.devMode = False

run = True

#initiate a while loop until run is no longer True

while run:

#initiate a for loop to detect events

for event in pygame.event.get():

#allows the user to quit the game

if event.type == pygame.QUIT:

run = False #sets run to False which breaks the loop

if event.type == pygame.KEYDOWN: #checks if a keyboard key is pressed

if event.key == pygame.K\_f: #if the key is f

pygame.display.toggle\_fullscreen()

if event.key == pygame.K\_l:

self.devMode = True

if event.type == pygame.MOUSEBUTTONUP: #checks if a keyboard key is pressed

pygame.time.set\_timer(event, 1000) #sets a timer for 1 second

self.\_\_Time += 1 #increments the 'Time' variable by 1

if event.type == pygame.MOUSEBUTTONDOWN:

if loc1 <= mouse[0] <= loc2 and loc5 <= mouse[1] <= loc6:

#save the game and call the 'save\_screen' display method after instantiating it

SS = save\_screen(self.\_\_maze, self.\_\_CurrentScore, self.\_\_lives, self.\_\_moves, self.\_\_Time)

SS.save(self.\_\_coords)

SS.display(qs\_diff)

if loc3 <= mouse[0] <= loc4 and loc5 <= mouse[1] <= loc6:

#load the game and call the 'load\_screen' display method after instantiating it

file = open("mazeInfo.txt", 'r')

maze, CurrentScore, lives, moves, Time, playerPos = self.\_\_gameSave.loadMaze(file, self.\_\_maze\_height, self.\_\_maze\_width)

LS = load\_screen(maze, CurrentScore, lives, moves, Time, playerPos, self.\_\_midX, self.\_\_midY)

self.\_\_walls, self.\_\_cells, self.\_all\_sprites = LS.loadMaze(self.\_\_maze, self.\_\_midX, self.\_\_midY)

self.\_all\_sprites, self.\_\_player, self.\_\_moves, self.\_\_Time = LS.loadGame(self.\_all\_sprites, self.\_\_maze\_height, self.\_\_maze\_width)

LS.display(qs\_diff)

#calls the maths question if 'movesLeft' is 0 or less and 'calls' is 10 or less

if self.\_\_movesLeft <= 0:

if self.calls < 10:

self.open\_question(questions)

self.calls += 1 #increment calls by 1

else: #otherwise adds 100 moves as last question has been reached

self.\_\_movesLeft += 100

#fills the pygame window with white

self.\_win.fill((0,0,0))

#stores the key that the user presses

pressed\_keys = pygame.key.get\_pressed()

#stores the position of the mouse

mouse = pygame.mouse.get\_pos()

for sprite in self.\_all\_sprites: #set a FOR loop to cycle through the whole list of sprites

self.\_win.blit(sprite.surf, sprite.rect) #blit each sprite on to the screen

self.\_win.blit(self.\_\_saveButton.image, self.\_\_saveButton.rect)

self.\_win.blit(self.\_\_loadButton.image, self.\_\_loadButton.rect)

self.\_win.blit(self.\_\_player.surf, self.\_\_player.rect)

pygame.display.update() #update the display with all sprites blitted on

#calls the move method of the player with the key that the user presses as the parameter

self.\_\_moves, self.\_\_movesLeft = self.\_\_player.move(pressed\_keys, self.\_\_moves, self.\_\_movesLeft)

self.\_\_movesCounter.update(self.\_\_movesLeft,15,"white",100,150) #calls the update method of the 'movesCounter'

self.\_\_timer.update(self.\_\_Time,15,"white",100,150) #update the timer

#checks if the player collides with any of the walls

if pygame.sprite.spritecollideany(self.\_\_player, self.\_\_walls):

if not self.devMode:

self.\_\_player.kill() #if it does then despawn the player

self.\_\_player = Player("Player.png",[800,275]) #re-instantiate the player

self.\_win.blit(self.\_\_player.surf, self.\_\_player.rect) #respawn the player

self.crashes += 1 #increments 'crashes' by 1

if self.crashes % 5 == 0: #checks if the crashes is a multiple of 5

#calls the update method of the 'livesCounter'

self.\_\_lives = self.\_\_livesCounter.update(self.\_\_lives,15,"white",100,150)

#checks if the lives has gone below 0

if self.\_\_lives < 0:

game.run\_lose() #if it has then runs the lose end screen

#store the cell that the player collides with as 'thisCell'

thisCell = pygame.sprite.spritecollideany(self.\_\_player, self.\_\_cells)

#run a FOR loop through all the cells

for cell in self.\_\_cells:

if thisCell == cell: #check if the current cell is 'thisCell'

#respawn the player on top of the cell and update 'coords'

cellLoc = cell.getLocation()

player = Player("Player.png",cellLoc)

self.\_\_coords = self.\_\_gameSave.calcPlayerPos(cellLoc, MAZE\_HEIGHT, MAZE\_WIDTH, self.\_\_midX, self.\_\_midY)

X = self.\_\_coords[0]

Y = self.\_\_coords[1]

visited[Y][X] = True

self.\_win.blit(player.surf, player.rect)

pygame.display.update() #updates the display

#checks if the player collides with the keys

if pygame.sprite.spritecollideany(self.\_\_player, keys):

self.movesBonus = 1500 - (MAZE\_CONSTANT \* self.\_\_moves)

self.questionsBonus = self.score \* MAZE\_CONSTANT

self.livesBonus = self.\_\_lives \* MAZE\_CONSTANT

self.\_\_CurrentScore = 500 + self.movesBonus + self.questionsBonus + self.livesBonus #if it does calculate the score

self.\_\_scoreBoard.update(self.\_\_CurrentScore,15,"white",100,150) #calls the update method of the 'scoreBoard'

sleep(2) #sleeps the program for 2 seconds

game.run\_win(self.\_\_CurrentScore, self.movesBonus, self.questionsBonus, self.livesBonus) #run the win screen from its display method in game

#detects if run is set to False

if not run:

sleep(2) #waits for 2 seconds

pygame.quit() #closes the pygame window

def open\_question(self, questions):

window = QuestionWindow(questions, self.score) #if it is, calls the window for next question

window.mainloop() #allows events to happen within the window object

question = questions[0] #get the first question from the array

questions.remove(question) #remove the previous question from the questions array

self.score, self.\_\_movesLeft = window.return\_info() #gets the new score and movesLeft from 'window'

#set up the game class

class Game(object):

#set the constructor method with 3 parameters

def \_\_init\_\_(self, MAZE\_HEIGHT, MAZE\_WIDTH, maze, qs\_diff):

self.\_\_qs\_diff = qs\_diff

#set the 'display' attributes to the various classes

self.display1 = start\_screen()

self.display2 = pygame\_window(MAZE\_HEIGHT, MAZE\_WIDTH, maze)

self.display3 = controls\_screen()

self.display4 = instructions\_screen()

self.display5 = win\_screen()

self.display6 = lose\_screen()

self.display7 = QuestionSelect\_screen()

#set up the run\_start method which will call the display method of the 'start\_screen' class

def run\_start(self):

self.display1.display() #call the display method of the class assigned to 'display1'

#set up the run\_controls method which will call the display method of the 'controls\_screen' class

def run\_controls(self):

self.display3.display() #call the display method of the class assigned to 'display3'

#set up the run\_instructions method which will call the display method of the 'instructions\_screen' class

def run\_instructions(self):

self.display4.display() #call the display method of the class assigned to 'display4'

#set up the run\_main method which will call the display method of the 'pygame\_window' class

def run\_main(self, qs\_diff):

self.display2.display(qs\_diff) #call the display method of the class assigned to 'display2'

#set up the run\_win method which will call the display method of the 'win\_screen' class

def run\_win(self, currentScore, movesBonus, questionsBonus, livesBonus):

self.display5.display(currentScore, movesBonus, questionsBonus, livesBonus) #call the display method of the class assigned to 'display5'

#set up the run\_lose method which will call the display method of the 'lose\_screen' class

def run\_lose(self):

self.display6.display() #call the display method of the class assigned to 'display6'

#set up the run\_qss method which will call the display method of the 'QuestionSelect\_screen' class

def run\_qss(self):

self.display7.display(self.\_\_qs\_diff) #call the display method of the class assigned to 'display7'

#set up an if statement for the main part of the program

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

qs\_diff = ""

game = Game(MAZE\_HEIGHT, MAZE\_WIDTH, maze, qs\_diff) #instantiate the game class

game.run\_start() #call the run method of the game class

#Maths quiz game

from tkinter import \*

from random import \*

def get\_questions():

questions = []

file = open("maths\_qs.txt", "r")

for line in file:

newLine = line.strip("\n").split(",")

questions.append(newLine)

return questions

class Results(Toplevel):

def \_\_init\_\_(self, parent, score):

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

self.geometry('410x200')

self.title('Results')

self.configure(bg="#12AAE6")

label1 = Label(self, text="The results are in", bg="#12AAE6")

label1.config(font=('Helvetica bold',40))

label1.grid(row=0,columnspan=2)

thisText = f"You scored: {score} / 10"

label2 = Label(self, text=thisText, bg="#12AAE6")

label2.config(font=('Helvetica bold',25))

label2.grid(row=1, columnspan=2)

class Window(Toplevel):

def \_\_init\_\_(self, parent, calls, questions):

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

self.geometry('2000x500')

self.title('Questions')

self.configure(bg="#12AAE6")

self.question = questions[0]

thisQuestion = self.question[0]

answer1 = self.question[1]

answer2 = self.question[2]

answer3 = self.question[3]

answer4 = self.question[4]

correctAnswer = answer1

self.Score = StringVar()

label1 = Label(self, text=thisQuestion, bg="#12AAE6")

label1.config(font=('Helvetica bold',20))

label1.grid(row=0,columnspan=2)

number1 = Button(self, text=answer1, bg="#12AAE6", command=lambda: self.ansPressed(answer1, correctAnswer))

number1.grid(row=1, column=0)

number2 = Button(self, text=answer2, bg="#12AAE6", command=lambda: self.ansPressed(answer2, correctAnswer))

number2.grid(row=1, column=1)

number3 = Button(self, text=answer3, bg="#12AAE6", command=lambda: self.ansPressed(answer3, correctAnswer))

number3.grid(row=2, column=0)

number4 = Button(self, text=answer4, bg="#12AAE6", command=lambda: self.ansPressed(answer4, correctAnswer))

number4.grid(row=2, column=1)

self.scoreBox = Entry(self, bg="#12AAE6", textvariable=self.Score)

self.scoreBox.config(state= "disabled")

self.scoreBox.grid(row=3, column=1)

label2 = Label(self, text="Score:", bg="#12AAE6")

label2.config(font=('Helvetica bold',10))

label2.grid(row=3, column=0)

Button(self,text="Next Question",bg="#12AAE6",command=lambda: self.open\_window(score, questions)).grid(row=4, column=0)

Button(self,text='Close',bg="#12AAE6",command=self.destroy).grid(row=4, column=1)

def ansPressed(self, answer, correctAnswer):

global score

if answer == correctAnswer:

print("Correct!")

score += 1

else:

print("Incorrect!")

score -= 1

self.Score.set(str(score))

def open\_window(self, score, questions):

global calls

questions.remove(self.question)

calls += 1

self.destroy()

if calls < 10:

window = Window(app, calls, questions)

window.grab\_set()

else:

results = Results(app, score)

results.grab\_set()

class App(Tk):

def \_\_init\_\_(self, score, calls):

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

self.geometry('825x500')

self.title('Maths Maze Navigator')

self.configure(bg="#12AAE6")

label1 = Label(self, text="Welcome to Maths Maze Navigator", bg="#12AAE6")

label1.config(font=('Helvetica bold',40))

label1.grid(row=0,columnspan=2)

Button(self,text='Open questions',bg="#12AAE6",command=self.open\_window).grid(row=1, column=0)

Button(self,text='Close',bg="#12AAE6",command=self.destroy).grid(row=1, column=1)

def open\_window(self):

calls = 0

questions = get\_questions()

window = Window(self, calls, questions)

window.grab\_set()

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

score = 0

calls = 0

app = App(score, calls)

app.mainloop()

#Maze objects

import pygame

#set up the 'Wall' class

class Wall(pygame.sprite.Sprite):

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

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

#set up the sprite's surf

self.surf = pygame.Surface((25, 25))

self.surf.fill((255,255,255))

#set up the player as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#set up the 'Cell' class

class Cell(pygame.sprite.Sprite):

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

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

#set up the sprite's surf

self.surf = pygame.Surface((25, 25))

self.surf.fill((0,0,255))

self.\_\_location = location

#set up the player as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(self.\_\_location))

def getLocation(self): #set up the function to get the location

return self.\_\_location #return the location to the main program

#set up the 'Empty' class

class Empty(pygame.sprite.Sprite):

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

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

#set up the sprite's surf

self.surf = pygame.Surface((25, 25))

self.surf.fill((0,0,0))

#set up the player as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

class Key(pygame.sprite.Sprite):

def \_\_init\_\_(self,image\_file,location):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

#set up the sprite's surf

self.surf = pygame.Surface((25, 25))

self.surf.fill((0,0,0))

#set the sprite's image

self.image = pygame.image.load(image\_file).convert()

#set up the key as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#Other objects

import pygame

class controlsImage(pygame.sprite.Sprite):

#define the constructor method of the class

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

super(controlsImage, self).\_\_init\_\_() #Call the parent class (Sprite) constructor

#set up the sprite's surf

self.surf = pygame.Surface((1920,1080))

self.surf.fill((255,255,255))

#set up the controlsImage as a rectangle in the middle of the screen

self.image = pygame.image.load('Controls.png').convert()

self.rect = self.surf.get\_rect(center=(960,540))

class instructionsImage(pygame.sprite.Sprite):

#define the constructor method of the class

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

super(instructionsImage, self).\_\_init\_\_() #Call the parent class (Sprite) constructor

#set up the sprite's surf

self.surf = pygame.Surface((1920,1080))

self.surf.fill((255,255,255))

#set up the instructionsImage as a rectangle in the middle of the screen

self.image = pygame.image.load('Instructions.png').convert()

self.rect = self.surf.get\_rect(center=(960,540))

class winScreen(pygame.sprite.Sprite):

#define the constructor method of the class

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

super(winScreen, self).\_\_init\_\_() #Call the parent class (Sprite) constructor

#set up the sprite's surf

self.surf = pygame.Surface((1920,1080))

self.surf.fill((255,255,255))

#set up the winScreen as a rectangle in the middle of the screen

self.image = pygame.image.load('Win\_Screen.png').convert()

self.rect = self.surf.get\_rect(center=(960,540))

class loseScreen(pygame.sprite.Sprite):

#define the constructor method of the class

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

super(loseScreen, self).\_\_init\_\_() #Call the parent class (Sprite) constructor

#set up the sprite's surf

self.surf = pygame.Surface((1920,1080))

self.surf.fill((255,255,255))

#set up the loseScreen as a rectangle in the middle of the screen

self.image = pygame.image.load('Lose\_Screen.png').convert()

self.rect = self.surf.get\_rect(center=(960,540))

class SS\_image(pygame.sprite.Sprite):

#define the constructor method of the class

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

super(SS\_image, self).\_\_init\_\_() #Call the parent class (Sprite) constructor

#set up the sprite's surf

self.surf = pygame.Surface((1920,1080))

self.surf.fill((255,255,255))

#set up the SS\_image as a rectangle in the middle of the screen

self.image = pygame.image.load('Save\_Screen.png').convert()

self.rect = self.surf.get\_rect(center=(960,540))

class LS\_image(pygame.sprite.Sprite):

#define the constructor method of the class

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

super(LS\_image, self).\_\_init\_\_() #Call the parent class (Sprite) constructor

#set up the sprite's surf

self.surf = pygame.Surface((1920,1080))

self.surf.fill((255,255,255))

#set up the LS\_image as a rectangle in the middle of the screen

self.image = pygame.image.load('Load\_Screen.png').convert()

self.rect = self.surf.get\_rect(center=(960,540))

#Player class

import pygame

#Importing the commands of they keys that the user will press

from pygame.locals import (

RLEACCEL, #for image loading

K\_UP, #for moving up

K\_DOWN, #for moving down

K\_LEFT, #for moving left

K\_RIGHT, #for moving right

K\_w, #for moving up

K\_s, #for moving down

K\_a, #for moving left

K\_d, ##for moving right

)

class Player(pygame.sprite.Sprite):

def \_\_init\_\_(self, image\_file, location):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

#set up the sprite's surf

self.surf = pygame.image.load(image\_file).convert()

self.surf.set\_colorkey((0, 0, 255), RLEACCEL)

#set up the player as a rectangle in the specified part of the screen

self.rect = self.surf.get\_rect(center=(location))

def move(self, pressed\_keys, moves):

#if up or w is pressed:

if pressed\_keys[K\_UP] or pressed\_keys[K\_w]:

self.rect.move\_ip(0, -10) #moves player up by 10

moves += 1 #increments moves by 1

#if down or s is pressed:

if pressed\_keys[K\_DOWN] or pressed\_keys[K\_s]:

self.rect.move\_ip(0, 10) #moves player down by 10

moves += 1 #increments moves by 1

#if left or a is pressed:

if pressed\_keys[K\_LEFT] or pressed\_keys[K\_a]:

self.rect.move\_ip(-10, 0) #moves player left by 10

moves += 1 #increments moves by 1

#if right or d is pressed:

if pressed\_keys[K\_RIGHT] or pressed\_keys[K\_d]:

self.rect.move\_ip(10, 0) #moves player right by 10

moves += 1 #increments moves by 1

#following 8 lines stop player from moving off screen

if self.rect.left < 0:

self.rect.left = 0

if self.rect.right > 1200:

self.rect.right = 1200

if self.rect.top <= 0:

self.rect.top = 0

if self.rect.bottom >= 600:

self.rect.bottom = 600

return moves

#Question screen objects

import pygame

class Text1(pygame.sprite.Sprite):

def \_\_init\_\_(self,location,text):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

self.font = pygame.font.SysFont("Arial", 50) #Set up the font for the text

self.textSurf = self.font.render(text, 1, "white") #set up the text with 'text' as the string

#set up the sprite's surf

self.surf = pygame.Surface((1000, 200))

self.surf.fill((0,0,0))

#set up the sprite as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [1000/2 - W/2, 200/2 - H/2])

class Button1(pygame.sprite.Sprite):

def \_\_init\_\_(self,location,text):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

self.font = pygame.font.SysFont("Arial", 35) #Set up the font for the text

self.textSurf = self.font.render(text, 1, "white") #set up the text with 'text' as the string

#set up the sprite's surf

self.surf = pygame.Surface((200, 100))

self.surf.fill((100,69,69))

#set up the sprite as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [200/2 - W/2, 100/2 - H/2])

class Button2(pygame.sprite.Sprite):

def \_\_init\_\_(self,location,text):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

self.font = pygame.font.SysFont("Arial", 35) #Set up the font for the text

self.textSurf = self.font.render(text, 1, "white") #set up the text with 'text' as the string

#set up the sprite's surf

self.surf = pygame.Surface((200, 100))

self.surf.fill((69,100,69))

#set up the sprite as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [200/2 - W/2, 100/2 - H/2])

class Button3(pygame.sprite.Sprite):

def \_\_init\_\_(self,location,text):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

self.font = pygame.font.SysFont("Arial", 35) #Set up the font for the text

self.textSurf = self.font.render(text, 1, "white") #set up the text with 'text' as the string

#set up the sprite's surf

self.surf = pygame.Surface((200, 100))

self.surf.fill((100,100,69))

#set up the sprite as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [200/2 - W/2, 100/2 - H/2])

class Button4(pygame.sprite.Sprite):

def \_\_init\_\_(self,location,text):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

self.font = pygame.font.SysFont("Arial", 35) #Set up the font for the text

self.textSurf = self.font.render(text, 1, "white") #set up the text with 'text' as the string

#set up the sprite's surf

self.surf = pygame.Surface((200, 100))

self.surf.fill((69,69,0))

#set up the sprite as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [200/2 - W/2, 100/2 - H/2])

class Button5(pygame.sprite.Sprite):

def \_\_init\_\_(self,location,text):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

self.font = pygame.font.SysFont("Arial", 30) #Set up the font for the text

self.textSurf = self.font.render(text, 1, "white") #set up the text with 'text' as the string

#set up the sprite's surf

self.surf = pygame.Surface((200, 100))

self.surf.fill((255,69,69))

#set up the sprite as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [200/2 - W/2, 100/2 - H/2])

#Starting objects

import pygame

class Title(pygame.sprite.Sprite):

def \_\_init\_\_(self,location,text):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

self.font = pygame.font.SysFont("Arial", 75) #Set up the font for the text

self.textSurf = self.font.render(text, 1, "white") #set up the text with 'text' as the string

#set up the sprite's surf

self.surf = pygame.Surface((1000, 500))

self.surf.fill((0,0,0))

#set up the player as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [1000/2 - W/2, 500/2 - H/2])

class Controls(pygame.sprite.Sprite):

def \_\_init\_\_(self,location,text):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

self.font = pygame.font.SysFont("Arial", 40) #Set up the font for the text

self.textSurf = self.font.render(text, 1, "white") #set up the text with 'text' as the string

#set up the sprite's surf

self.surf = pygame.Surface((200, 100))

self.surf.fill((100,69,69))

#set up the player as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [200/2 - W/2, 100/2 - H/2])

class Play(pygame.sprite.Sprite):

def \_\_init\_\_(self,location,text):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

self.font = pygame.font.SysFont("Arial", 30) #Set up the font for the text

self.textSurf = self.font.render(text, 1, "white") #set up the text with 'text' as the string

#set up the sprite's surf

self.surf = pygame.Surface((200, 100))

self.surf.fill((69,100,69))

#set up the player as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [200/2 - W/2, 100/2 - H/2])

class Instructions(pygame.sprite.Sprite):

def \_\_init\_\_(self,location,text):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

self.font = pygame.font.SysFont("Arial", 30) #Set up the font for the text

self.textSurf = self.font.render(text, 1, "white") #set up the text with 'text' as the string

#set up the sprite's surf

self.surf = pygame.Surface((200, 100))

self.surf.fill((100,100,69))

#set up the player as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [200/2 - W/2, 100/2 - H/2])

class Quit(pygame.sprite.Sprite):

def \_\_init\_\_(self,location,text):

# Call the parent class (Sprite) constructor

pygame.sprite.Sprite.\_\_init\_\_(self)

self.font = pygame.font.SysFont("Arial", 40) #Set up the font for the text

self.textSurf = self.font.render(text, 1, "white") #set up the text with 'text' as the string

#set up the sprite's surf

self.surf = pygame.Surface((200, 100))

self.surf.fill((69,69,0))

#set up the player as a rectangle as the center set to the location variable

self.rect = self.surf.get\_rect(center=(location))

#get the width and length of the text object

W = self.textSurf.get\_width()

H = self.textSurf.get\_height()

#blit the text object on to the center of the sprite

self.surf.blit(self.textSurf, [200/2 - W/2, 100/2 - H/2])