Explaining the code in sections

Section 1

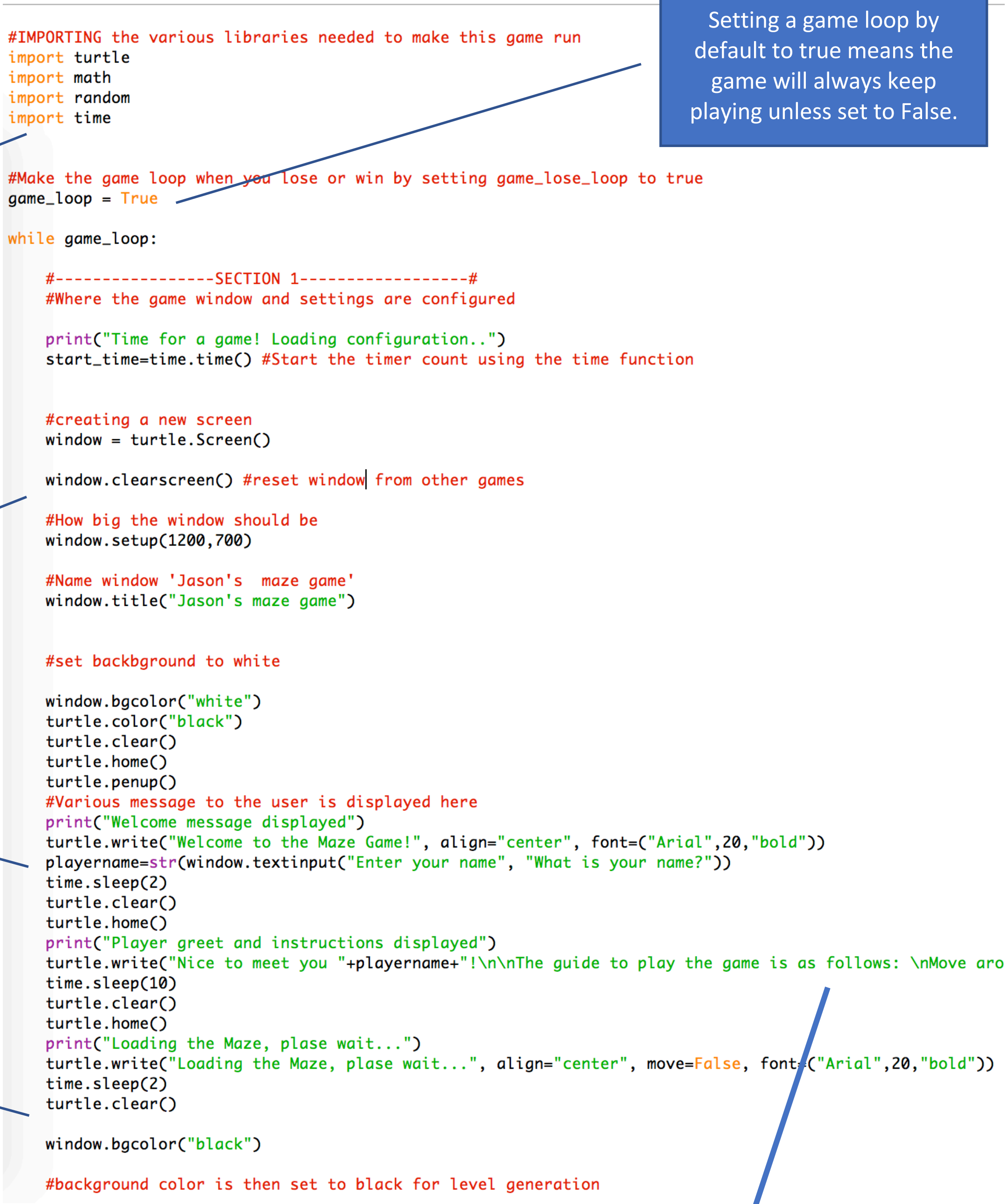
Importing the various libraries needed for the code to function.
Turtle – To draw objects on screen
Math – To calculate distances between a player and object
Random – To randomly choose a level
Time – To calculate how long a user has spent in the game

Setting a game loop by default to true means the game will always keep playing unless set to False.

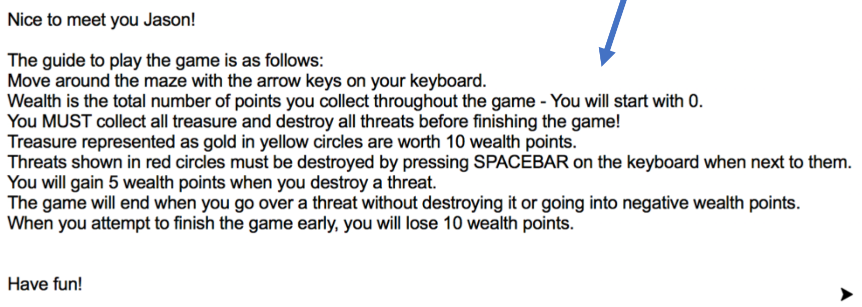
This is the first major section of the code. The window settings are defined and titled as "Jason's maze game". It is then given a white background for various messages to be displayed to the user. The user is also asked for their name.

This is where the various messages are displayed to the user. The welcome message and other messages are displayed in Arial font size 20.

Once these messages are displayed, all writing is cleared and the background color of the window goes to black.



```
#IMPORTING the various libraries needed to make this game run
import turtle
import math
import random
import time


#Make the game loop when you lose or win by setting game_lose_loop to true
game_loop = True

while game_loop:

    #-----------------SECTION 1------------------#
    #Where the game window and settings are configured

    print("Time for a game! Loading configuration..")
    start_time=time.time() #Start the timer count using the time function


    #creating a new screen
    window = turtle.Screen()

    window.clearscreen() #reset window from other games

    #How big the window should be
    window.setup(1200,700)

    #Name window 'Jason's  maze game'
    window.title("Jason's maze game")


    #set backbground to white

    window.bgcolor("white")
    turtle.color("black")
    turtle.clear()
    turtle.home()
    turtle.penup()
    #Various message to the user is displayed here
    print("Welcome message displayed")
    turtle.write("Welcome to the Maze Game!", align="center", font=("Arial",20,"bold"))
    playername=str(window.textinput("Enter your name", "What is your name?"))
    time.sleep(2)
    turtle.clear()
    turtle.home()
    print("Player greet and instructions displayed")
    turtle.write("Nice to meet you "+playername+"!\n\nThe guide to play the game is as follows: \nMove aro
    time.sleep(10)
    turtle.clear()
    turtle.home()
    print("Loading the Maze, plase wait...")
    turtle.write("Loading the Maze, plase wait...", align="center", move=False, font=("Arial",20,"bold"))
    time.sleep(2)
    turtle.clear()

    window.bgcolor("black")

    #background color is then set to black for level generation
```



```
Nice to meet you Jason!

The guide to play the game is as follows:
Move around the maze with the arrow keys on your keyboard.
Wealth is the total number of points you collect throughout the game - You will start with 0.
You MUST collect all treasure and destroy all threats before finishing the game!
Treasure represented as gold in yellow circles are worth 10 wealth points.
Threats shown in red circles must be destroyed by pressing SPACEBAR on the keyboard when next to them.
You will gain 5 wealth points when you destroy a threat.
The game will end when you go over a threat without destroying it or going into negative wealth points.
When you attempt to finish the game early, you will lose 10 wealth points.


Have fun!
```

Section 2 – Defining different classes to be used in the game

The wall class code

The wall class will be drawn by a turtle. It is defined to be a square shape and have a colour that is white. When it is being added to the screen, the pen will be up so that a trail will not be left behind.

```
class Walls(turtle.Turtle):
    def __init__(self): #referring to the object that will be called on
        turtle.Turtle.__init__(self) #initialise pen
        self.shape("square") #shape of the person
        self.color("white") #color of the person
        self.penup() #By default, a turtle leaves a trail behind, we don't want this
        self.speed(1000) #Animation speed
```

The wall class output

The output of the wall class is simply seen below

The player class code

```
class Player(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("blue")
        self.penup()
        self.speed(0)
        self.gold = 0 #defining the gold player has
        self.touchedthreat = 0 #if this is 1 then the game will end and loop
        self.win = 0 #If this turns to 1 then the game will end and loop

    #Defining the movement of the player##
    #Going up is a positive y coorindate
    def go_up(self):
        move_x = player.xcor()
        move_y = player.ycor() + 24
        #if where the player will move to somewhere that is not a wall, then it will allow them to move
        if (move_x, move_y) not in wall_coords:
            self.goto(move_x, move_y) #Y cor is vertical so + is up

    #Going down is a negative y coordinate
    def go_down(self):
        move_x = player.xcor()
        move_y = player.ycor() - 24
        if (move_x, move_y) not in wall_coords:
            self.goto(move_x, move_y)

    #Going left is a negative x coordinate
    def go_left(self):
        move_x = player.xcor() - 24
        move_y = player.ycor()
        if (move_x, move_y) not in wall_coords:
            self.goto(move_x, move_y)

    #Going right is positive x coordinate
    def go_right(self):
        move_x = player.xcor() + 24
        move_y = player.ycor()
        if (move_x, move_y) not in wall_coords:
            self.goto(move_x, move_y)

    #Defining what would count as an object being 'touched' by a played
    def touched_object(self, other): #other would be the object concerned such as threat or treasure
        a = self.xcor()-other.xcor()
        b = self.ycor()-other.ycor()
        distance = math.sqrt((a ** 2) + (b ** 2) )

        #if the distance between the two objects is less than 5, then the object has been 'touched' and returns True
        if distance < 5:
            return True
        else:
            return False

    def near(self, other): #other would be the object concerned such as threat or treasure
        a = self.xcor()-other.xcor()
        b = self.ycor()-other.ycor()
        distance = math.sqrt((a ** 2) + (b ** 2) )

        #if the distance between the two objects is less than 5, then the object has been 'touched' and returns True
        if distance < 26:
            return True
        else:
            return False
```

The player object is defined as being a blue square. It also has the attributes gold, touchedthreat and win.

Each go_x routine has either a x or y coordinate change to the player. These routines will be called upon when the player presses any of the arrow keys which will be seen later.

For both touched_object and near routines they have similar objectives. The maths behind it is that a distance can be calculated when comparing the current player coordinates to the object in question, such as a threat or treasure.

An object is defined as 'touched_object' by the code when the player's distance is less than 5 coordinates away.

An object is defined as 'near' when the player's distance is less than 26 coordinates away.

Outputs of the player will be seen later.

The treasure class code

The treasure class code defines the object and characteristics that a treasure item on screen would have.

```
#Defining a new class for treasure that can be collected in game - it will be circle and gold
class Treasure(turtle.Turtle):
    def __init__(self, x, y): #referring to the object that will be called on + where we want the treasure to appear
        turtle.Turtle.__init__(self) #initialise pen
        self.shape("circle") #shape of the person
        self.color("gold") #color of the person
        self.penup() #By default, a turtle leaves a trail behind, we don't want this
        self.speed(0) #Animation speed
        self.gold = 10 #set the value of the gold
        self.goto(x, y)

    #Destroying a treasure hides its object and places it out of the screen
    def destroy(self):
        self.goto(2000, 2000)
        self.hideturtle()
```

The treasure object is defined as being a gold circle.

The destroy routine within the treasure class is when the treasure would be removed from the players view

The treasure class output

The output of when the treasure class is called is seen below.

The threat class code

The threat class code defines the object and characteristics that a threat item on screen would have.

```
class Threat(turtle.Turtle):
    def __init__(self, x, y): #referring to the object that will be called on + where we want the treasure to appear
        turtle.Turtle.__init__(self) #initialise pen
        self.shape("circle") #shape of the person
        self.color("red") #color of the person
        self.penup() #By default, a turtle leaves a trail behind, we don't want this
        self.speed(0) #Animation speed
        self.gold = 5 #set the value of the gold
        self.goto(x, y)

    #defining destroying the threat
    def destroy_key(self):
        global threatscaught
        if player.near(self):
            self.goto(2000, 2000)
            self.hideturtle()
            player.gold = player.gold + threat.gold
            threats_coords.remove(self)
            print("BOOM! Threat removed! Player wealth points are now: " +(str(player.gold)))
            threatscaught = threatscaught + 1

        else:
            return
```

The treasure object is defined as being a red circle.

The destroy routine within the threat class is when the player is near the threat, presses the spacebar and therefore triggers this routine to occur

The threat class output

The output of when the threat class is called is seen below.

The end class code

The end class code defines the object that would be see by the player. The end class also defines the routine that runs when the game ends and when the player attempts to end the game early.

```
#Defining a new End point class - it will be circle and green
class End(turtle.Turtle):
    def __init__(self, x, y): #referring to the object that will be called on
        turtle.Turtle.__init__(self) #initialise pen
        self.shape("circle") #shape of the person
        self.color("green") #color of the person
        self.penup() #By default, a turtle leaves a trail behind, we don't want this
        self.speed(0) #Animation speed
        self.goto(x, y)

    #Defining the end routine - It will display a message to the player to congratulate them that they have won
    def ended(self):
        turtle.clear()
        turtle.penup()
        turtle.home()
        window.bgcolor("white")
        turtle.color("black")
        print("Displayed congratulations message")
        end_time_minutes=round((time.time()-start_time)/60,1)
        end_time_seconds=round(time.time()-start_time,1)
        turtle.write("Congratulations "+playername+"! You have reached the end!\n\nTotal player wealth points: " +(str(player.gold))+"\nTime
        time.sleep(10)
        turtle.clear()
        turtle.home()
        player.goto(player_coords)
        turtle.clear
        player.win = player.win + 1 #adding 1 to break the loop and start again

    #Defining the routine when the game has not ended, but when player attempts to end it early - it will inform the player what else they ne
    def not_ended(self):
        turtle.clear()
        turtle.penup()
        turtle.home()
        window.bgcolor("white")
        turtle.color("black")
        print("Displayed player attempted to finish early message")
        turtle.write("You cannot finish yet!\nYou have lost 10 wealth points\n\nPlease ensure that you collect all the treasure and destroy a
        player.gold = player.gold - 10
        time.sleep(10)
        turtle.clear()
        turtle.home()
        print("Player wealth points: " +(str(player.gold)))
        turtle.write("Current wealth points: " +(str(player.gold))+"\n\nReturning to start point", align="center", font=("Arial",20,"bold"))
        player.goto(player_coords)
        time.sleep(5)
        turtle.clear()
        window.bgcolor("black")
```

The treasure object is defined as being a green circle.

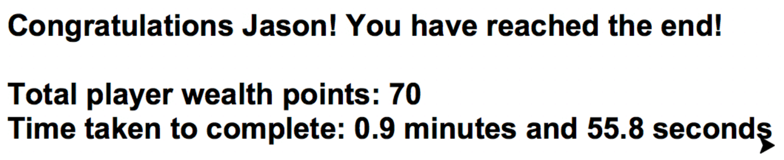
When the game ends, the screen will be cleared and a congratulations message will be displayed (to be shown below)

The player.win status is also updated to 1, which allows the game to end.

When the player tries to end the game early, the screen will turn white and let them know that they can't finish. (to be below)

The end class outputs

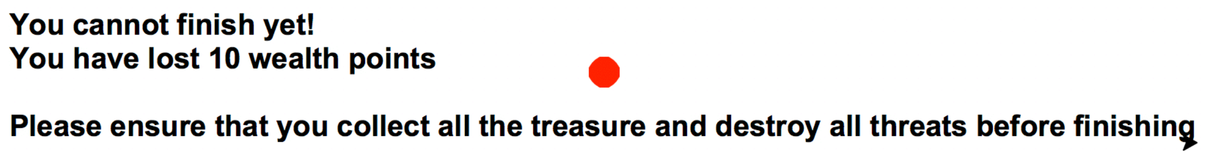
Ending a game completely



Congratulations Jason! You have reached the end!

Total player wealth points: 70
Time taken to complete: 0.9 minutes and 55.8 seconds

Ending a game early

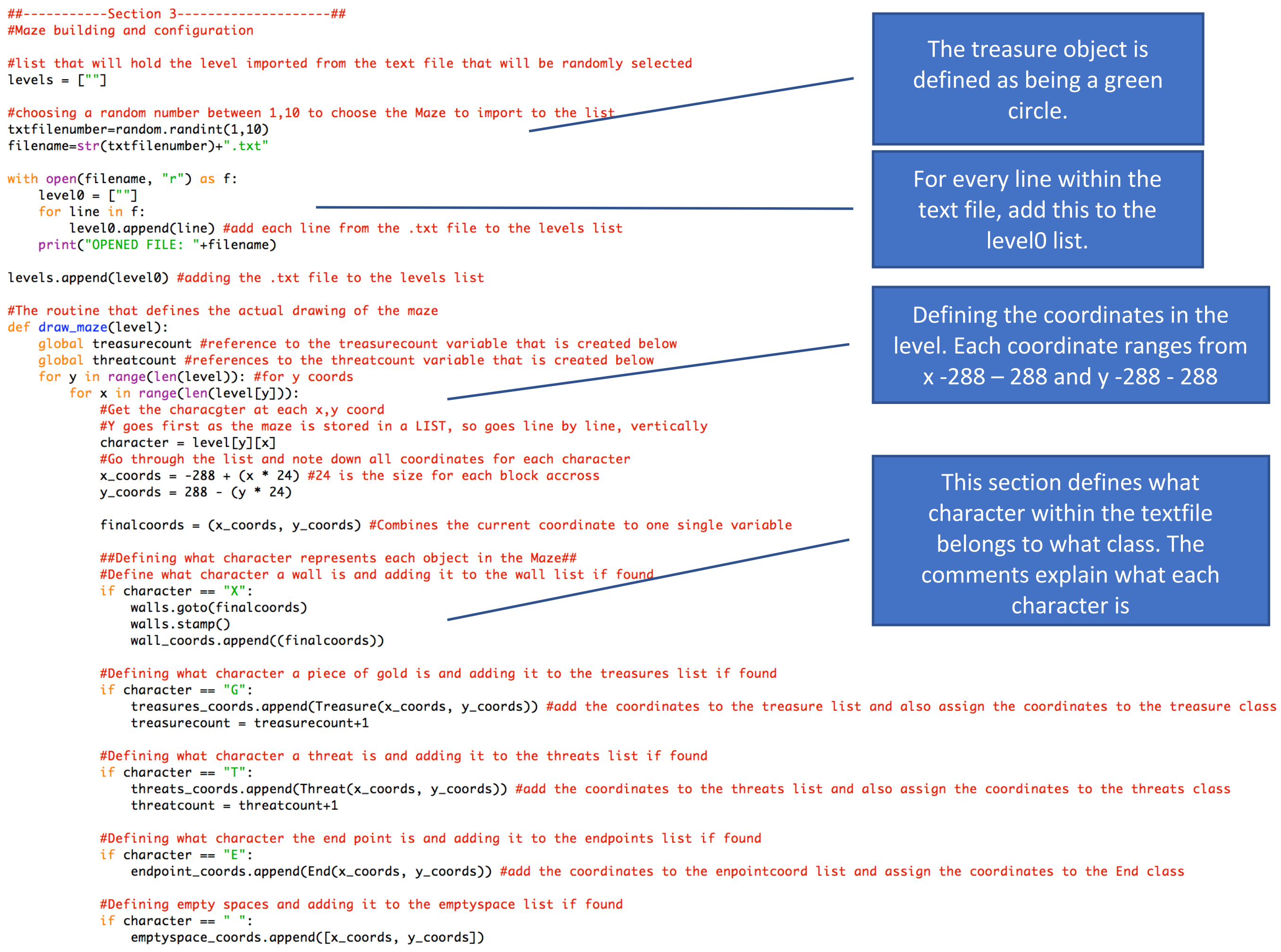


You cannot finish yet!
You have lost 10 wealth points

Please ensure that you collect all the treasure and destroy all threats before finishing

Section 3 – Maze building and configuration

Within this section is where the maze level and definition of objects comes to place. Each level is configured within their own text file and is composed of 25 characters horizontally and vertically. The code then randomly chooses a number between 1 and 10 (which is currently how many levels there are) and it is loaded. Also, each individual character within the text file are defined here.



```python
##-----------Section 3--------------------##
#Maze building and configuration

#list that will hold the level imported from the text file that will be randomly selected
levels = [""]

#choosing a random number between 1,10 to choose the Maze to import to the list
txtfilenumber=random.randint(1,10)
filename=str(txtfilenumber)+".txt"

with open(filename, "r") as f:
    level0 = [""]
    for line in f:
        level0.append(line) #add each line from the .txt file to the levels list
    print("OPENED FILE: "+filename)

levels.append(level0) #adding the .txt file to the levels list

#The routine that defines the actual drawing of the maze
def draw_maze(level):
    global treasurecount #reference to the treasurecount variable that is created below
    global threatcount #references to the threatcount variable that is created below
    for y in range(len(level)): #for y coords
        for x in range(len(level[y])):
            #Get the characgter at each x,y coord
            #Y goes first as the maze is stored in a LIST, so goes line by line, vertically
            character = level[y][x]
            #Go through the list and note down all coordinates for each character
            x_coords = -288 + (x * 24) #24 is the size for each block accross
            y_coords = 288 - (y * 24)

            finalcoords = (x_coords, y_coords) #Combines the current coordinate to one single variable

            ##Defining what character represents each object in the Maze##
            #Define what character a wall is and adding it to the wall list if found
            if character == "X":
                walls.goto(finalcoords)
                walls.stamp()
                wall_coords.append((finalcoords))

            #Defining what character a piece of gold is and adding it to the treasures list if found
            if character == "G":
                treasures_coords.append(Treasure(x_coords, y_coords)) #add the coordinates to the treasure list and also assign the coordinates to the treasure class
                treasurecount = treasurecount+1

            #Defining what character a threat is and adding it to the threats list if found
            if character == "T":
                threats_coords.append(Threat(x_coords, y_coords)) #add the coordinates to the threats list and also assign the coordinates to the threats class
                threatcount = threatcount+1

            #Defining what character the end point is and adding it to the endpoints list if found
            if character == "E":
                endpoint_coords.append(End(x_coords, y_coords)) #add the coordinates to the enpointcoord list and assign the coordinates to the End class

            #Defining empty spaces and adding it to the emptyspace list if found
            if character == " ":
                emptyspace_coords.append([x_coords, y_coords])
```

The treasure object is defined as being a green circle.

For every line within the text file, add this to the level0 list.

Defining the coordinates in the level. Each coordinate ranges from x -288 – 288 and y -288 - 288

This section defines what character within the textfile belongs to what class. The comments explain what each character is

Section 4 – Defining lists and trackers

```
##------------------SECTION 4-------------------##

#Making new variables to call on the classes created earlier
walls = Walls()
player = Player()

#create a list of coordinates for each object defined here: walls, endpoint, treasures, threats, empty spaces
wall_coords = [] #So the player can't walk into walls
endpoint_coords = [] #So the player can go into an endpoint
treasures_coords = [] #So the player can claim treasure
threats_coords = [] #So the player can destroy threats
emptyspace_coords =[] #So the player can randomly spawn in empty spaces

#defining all of the different counts during the game
treasurecount = 0 #Will count how many gold treasures can be collecxted
threatcount = 0 #Will count how many threats there are
treasurecaught = 0 #Will count how much gold a user has taken
threatscaught = 0 #Will count how many threats a user has destroyed


#Setup the level - to draw the maze from the list levels that has been imported from the text file
draw_maze(levels[1])

#Setting and placing the player on the maze once drawn up
player_coords= random.choice(emptyspace_coords) #Choose a random coordinate from the empty space list
player.goto(player_coords) #Then go to the coordinate
print("Player allocated to coords: " +str(player_coords)) #Print the result


#Keyboard controls
window.listen()
window.onkey(player.go_left,"Left") #left arrow
window.onkey(player.go_right,"Right") #right arrow
window.onkey(player.go_up,"Up") #Up key
window.onkey(player.go_down,"Down") #Down key
```

Defining a variable for the walls class and player class so that they can be called on

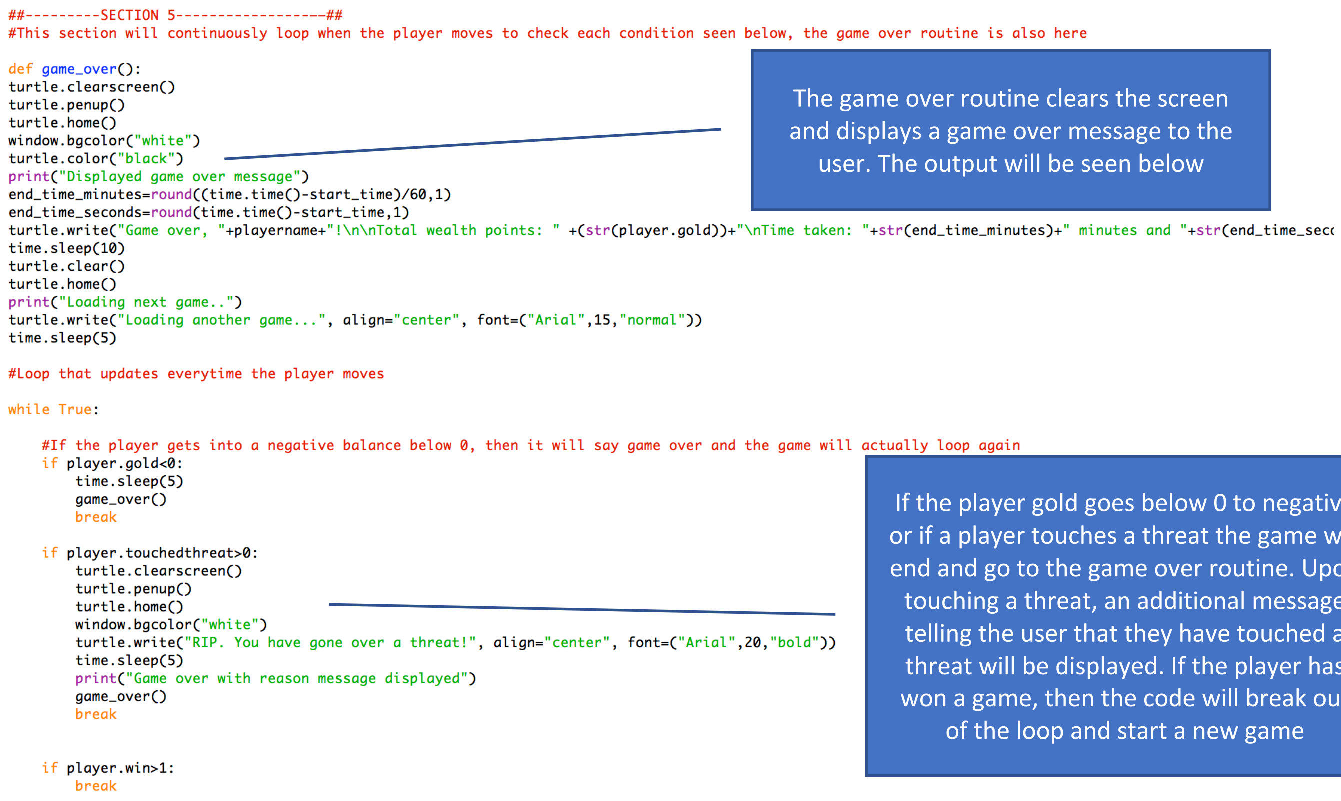
This section defines all of the different variables that will hold key information about the maze.

Set the player to a random coordinate within the maze that has been recorded as empty

Setting the keyboard controls for the player and calling the routine when the key is pressed

Section 5 (part 1) – Checking conditions to win / lose a game

Section 5 contains a continuous loop which checks against certain conditions before deciding to end or continue a game.



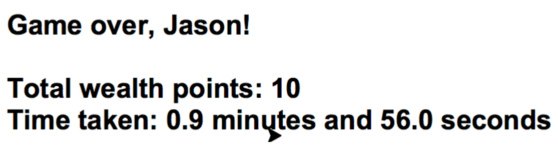
```
##---------SECTION 5-----------------##
#This section will continuously loop when the player moves to check each condition seen below, the game over routine is also here

def game_over():
turtle.clearscreen()
turtle.penup()
turtle.home()
window.bgcolor("white")
turtle.color("black")
print("Displayed game over message")
end_time_minutes=round((time.time()-start_time)/60,1)
end_time_seconds=round(time.time()-start_time,1)
turtle.write("Game over, "+playername+"!\n\nTotal wealth points: " +(str(player.gold))+"\nTime taken: "+str(end_time_minutes)+" minutes and "+str(end_time_secc
time.sleep(10)
turtle.clear()
turtle.home()
print("Loading next game..")
turtle.write("Loading another game...", align="center", font=("Arial",15,"normal"))
time.sleep(5)

#Loop that updates everytime the player moves

while True:

    #If the player gets into a negative balance below 0, then it will say game over and the game will actually loop again
    if player.gold<0:
        time.sleep(5)
        game_over()
        break

    if player.touchedthreat>0:
        turtle.clearscreen()
        turtle.penup()
        turtle.home()
        window.bgcolor("white")
        turtle.write("RIP. You have gone over a threat!", align="center", font=("Arial",20,"bold"))
        time.sleep(5)
        print("Game over with reason message displayed")
        game_over()
        break


    if player.win>1:
        break
```

The game over routine clears the screen and displays a game over message to the user. The output will be seen below

If the player gold goes below 0 to negative or if a player touches a threat the game will end and go to the game over routine. Upon touching a threat, an additional message telling the user that they have touched a threat will be displayed. If the player has won a game, then the code will break out of the loop and start a new game

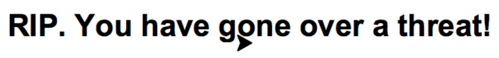
Outputs from section 5 (part 1)

Game over message



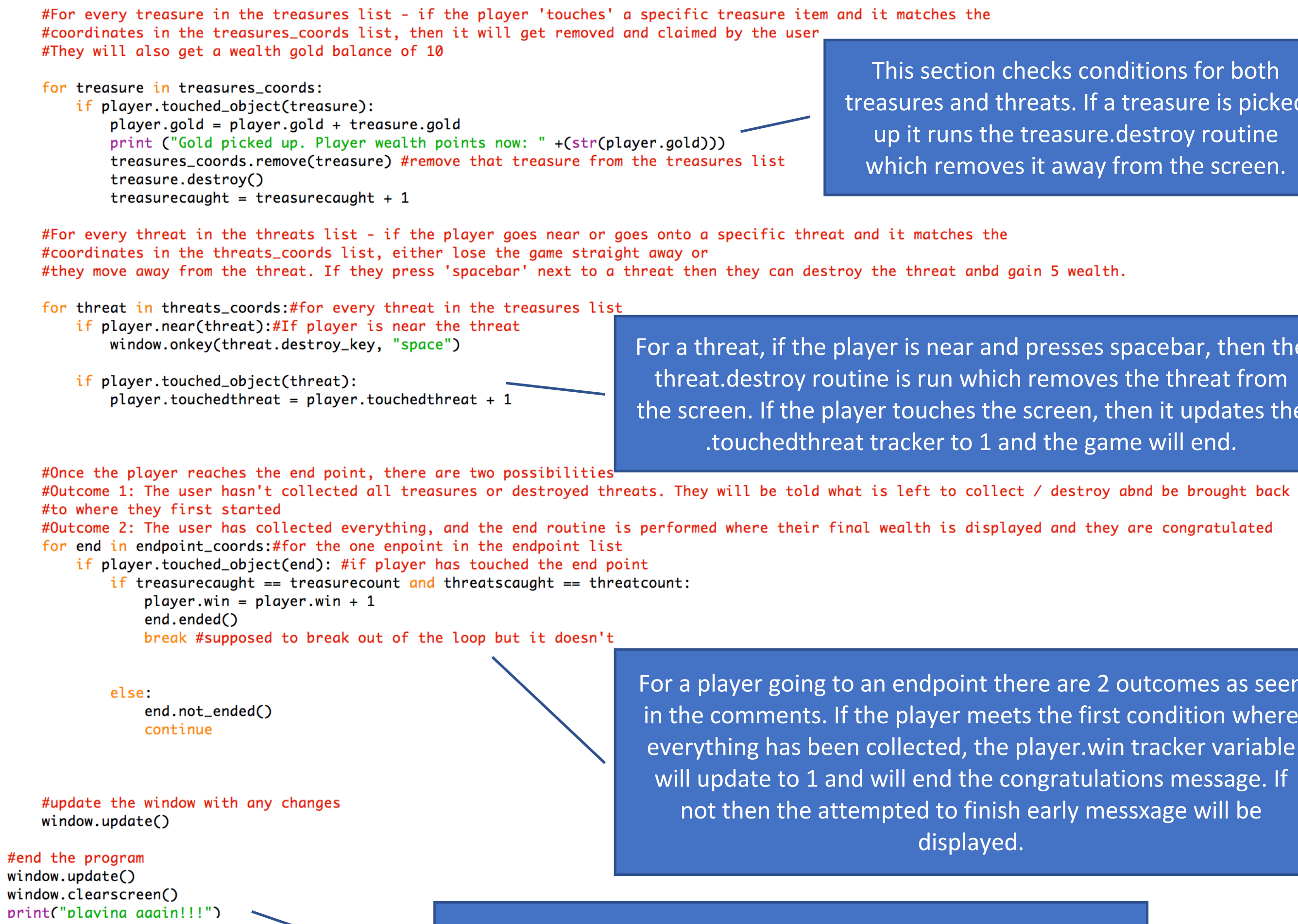
```
Game over, Jason!

Total wealth points: 10
Time taken: 0.9 minutes and 56.0 seconds
```

Going over a threat



```
RIP. You have gone over a threat!
```

Section 5 (part 2) – Checking conditions to win / lose a game



```
    #For every treasure in the treasures list - if the player 'touches' a specific treasure item and it matches the
    #coordinates in the treasures_coords list, then it will get removed and claimed by the user
    #They will also get a wealth gold balance of 10

    for treasure in treasures_coords:
        if player.touched_object(treasure):
            player.gold = player.gold + treasure.gold
            print ("Gold picked up. Player wealth points now: " +(str(player.gold)))
            treasures_coords.remove(treasure) #remove that treasure from the treasures list
            treasure.destroy()
            treasurecaught = treasurecaught + 1

    #For every threat in the threats list - if the player goes near or goes onto a specific threat and it matches the
    #coordinates in the threats_coords list, either lose the game straight away or
    #they move away from the threat. If they press 'spacebar' next to a threat then they can destroy the threat anbd gain 5 wealth.

    for threat in threats_coords:#for every threat in the treasures list
        if player.near(threat):#If player is near the threat
            window.onkey(threat.destroy_key, "space")

        if player.touched_object(threat):
            player.touchedthreat = player.touchedthreat + 1


    #Once the player reaches the end point, there are two possibilities
    #Outcome 1: The user hasn't collected all treasures or destroyed threats. They will be told what is left to collect / destroy abnd be brought back
    #to where they first started
    #Outcome 2: The user has collected everything, and the end routine is performed where their final wealth is displayed and they are congratulated
    for end in endpoint_coords:#for the one enpoint in the endpoint list
        if player.touched_object(end): #if player has touched the end point
            if treasurecaught == treasurecount and threatscaught == threatcount:
                player.win = player.win + 1
                end.ended()
                break #supposed to break out of the loop but it doesn't


            else:
                end.not_ended()
                continue


    #update the window with any changes
    window.update()

#end the program
window.update()
window.clearscreen()
print("playing again!!!")
```

This section checks conditions for both treasures and threats. If a treasure is picked up it runs the treasure.destroy routine which removes it away from the screen.

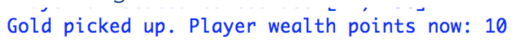
For a threat, if the player is near and presses spacebar, then the threat.destroy routine is run which removes the threat from the screen. If the player touches the screen, then it updates the .touchedthreat tracker to 1 and the game will end.

For a player going to an endpoint there are 2 outcomes as seen in the comments. If the player meets the first condition where everything has been collected, the player.win tracker variable will update to 1 and will end the congratulations message. If not then the attempted to finish early messxage will be displayed.

At the end, the window is updated to reflect changes and when the loop is broken a message saying playing again is displayed in the background python window.

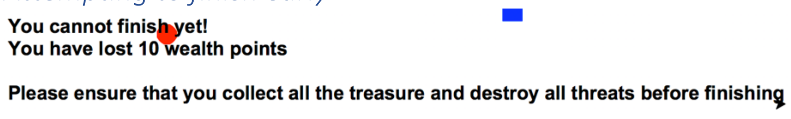
Outputs from section 5 (part 1)

Collecting treasure



```
Gold picked up. Player wealth points now: 10
```

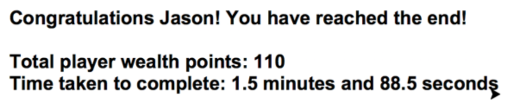
Attempting to finish early



You cannot finish yet!
You have lost 10 wealth points

Please ensure that you collect all the treasure and destroy all threats before finishing

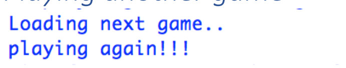
Finishing the game completely



Congratulations Jason! You have reached the end!

Total player wealth points: 110
Time taken to complete: 1.5 minutes and 88.5 seconds

Playing another game



```
Loading next game..
playing again!!!
```

Classes used in the python game

Walls = Defined the characteristics of a wall in the game
Player = Defined the characteristics of a player in the game
Treasure = Defined the characteristics of a treasure in the game
Threat = Defined the characteristics of a threat in the game
End = Defined the end point in the game and what to do when the game was ended early

Routines used in the python game

Player.go_up = Move the player up
Player.go_down = Move the player down
Player.go_left = Move the player left
Player.go_right = Move the player right
Player.touched_object = Outputting TRUE or FALSE if a player has touched an object
Player.near = Outputting TRUE or FALSE if a player is within distance of another object
Player.gold = Contains how much wealth points a player has
Player.touchedthreat = If this turns to 1, the player game is ended instantly
Player.win = If this turns to 1m the player game is ended and congratulations message shown
Threat.destroy_key = When the player presses SPACEBAR near a threat, the threat will be removed and 10 wealth points added to the player
End.ended = Defining the routine for the end message to the player and breaking the loop that checks if conditions have been met to end a game
End.not_ended = Displays the attempted to finish early message to player and reduces player wealth points by 10
Draw_maze = Defining the routine that draws the maze and what characters defined what objects were where

Variables used in the python game

Window = Defines the turtle window in the game
Levels = The list that contains the level that is loaded into the game
Txtfilenumber = Will hold a random number between 1 – 10 for the text file to be opened
Filename = combines the txtfilenumber to .txt to make it into a valid file
Level0 = Defines a list that will be added to the levels list when loading the game
Wall_coords = The list that contains all coordinates of walls in the game
Endpoint_coords = The list that contains the end point within the game
Treasures_coords = The list that contains the coordinates of all treasure in the game
Emptyspace_coords = The list that contains all coordinates of empty space in the game
Treasurecount = Contains how many gold treasures there are
Threatcount = Contains how many threats there are
Treasurecaught = Contains how many items of gold a player has caught
Threatscaught = Contains how many threats a player has destroyed
Player_coords = Contains the randomly selected coordinates from the emptyspace_coords list

Improvements that can be made to the code

1. **Reducing the number of lines within the code**

Currently the total number of lines within the code is 434. There are many lines that could be removed or at least cut down. This could be done by removing some lines which may be duplicate and doing the same thing or placing similar lines together.

2. **Increasing efficiency in the code**

Efficiency within the code could be increased by considering adding repeated lines of code to a routine like most of the code. For example, when the timer was ended, it was duplicated when a game ended or when a person had died. This could be added to a routine next time.

3. **Allowing a player to continue a game or not**

At the end of every game, the player has no control of the starting the game again. It would be nice to allow the player to have the choice to be able to start a new game or not.

4. **Adding messages during the game on screen**

Currently messages such as key events when a player picks up a gold item are displayed within the background window of the game. It would be nice to be able to display this somewhere within the game while the player is running so that they aware of their wealth points and if they die, the reason why.

5. **Adding moving threats**

Threats don't currently move, but if they did it would add more challenging conditions for the player to make the game more interesting

6. **Adding more rooms and not allowing the player to view the whole level**

Currently the game runs and when loaded it shows the player the whole maze. In the future, it would be good to make a version of the game where the player is put into a room, where they cannot move away from unless they choose a certain path. When the player moves, more of the maze and pathways will be revealed to them.