5#CHANGES -V8.0#

#loops can now happen if the user wins or loses

#updated messages to user and reworded wealth to wealth points

#changed window size

#added more messages

#iMPORTING the various libraries needed to make this game run

import turtle

import math

import random

import time

##Make the game loop when you loose by setting game\_lose\_loop to true

game\_lose\_loop = True

while game\_lose\_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 around the maze with the arrow keys on your keyboard.\nWealth is the total number of points you collect throughout the game - You will start with 0.\nYou MUST collect all treasure and destroy all threats before finishing the game!\nTreasure represented as gold in yellow circles are worth 10 wealth points.\nThreats shown in red circles must be destroyed by pressing SPACEBAR on the keyboard when next to them.\nYou will gain 5 wealth points when you destroy a threat.\nThe game will end when you go over a threat without destroying it or going into negative wealth points.\nWhen you attempt to finish the game early, you will lose 10 wealth points.\n\n\nHave fun!", align="center", move=True, font=("Arial",15,"normal"))

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

#---------------------SECTION 2-------------------#

#This section contains the classes: Walls, Player, Threat, Treasure and End

#Classes are used throughout the game and are referenced to put objects onto the screen

#These are all components that come togehet to be a part of the Maze

#A class is a definition of an object - defines its behaviour + properties

#Defining a new class called Walls that will become a turtle that draws walls - it will be a swaure and white

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

#Defining a new class for the player that is square and yellow and on screen

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 (move\_x, move\_y) not in wall\_coords: #if where the player will move to is not in the wall coords list, then it will move

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

#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()

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

def playertouched(self):

print("Ouch! Player wealth points reduced by 10")

player.gold = player.gold - 10

print ("Player wealth points are now: " +(str(player.gold)))

return

#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 taken to complete: "+str(end\_time\_minutes)+" minutes and "+str(end\_time\_seconds)+" seconds", align="center", move=True, font=("Arial",20,"bold"))

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 need before they can complete the game

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 all threats before finishing", align="center", move=True, font=("Arial",20,"bold"))

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")

##-----------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

#Definine 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 treasure list and also assign the coordinates to the treasure class

threatcount = threatcount+1

#Defining what character the end point is and adding it to the endpoints list if foun d

if character == "E":

endpoint\_coords.append(End(x\_coords, y\_coords)) #same as above but 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])

#Defining the end of the timer to be calculated below when called

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\_seconds)+" seconds", align="center", font=("Arial",20,"bold"))

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)

##------------------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

##---------SECTION 5----------------––##

#This section will continuously loop when the player moves to check each condition seen below

#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

#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 treasure in the treasures list - if the player 'touches' a specific treasure item and it matches the

#coordinates in the treasures\_coords list, then the player will get harmed and lose 10 wealth until

#they move away from the threat. If they press 'spacebar' 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!!!")