Daniel Kolodziej

ITMD 413 Final Project

Abstract:

The final project I have chosen to create was a game that I call “Paper, Scissors, Smash!”. I took inspiration from various arcade games such as Space-War as the basis and modified/changed the rules and functionality to create the game I made. The objective of the game is as the player, ‘Scissors’, you must catch and ‘cut the paper objects’ to collect points. The player has 3 lives and loses one for every ‘enemy or rock object’ they collide into. The game tracks and displays the players current score and lives which resets upon hitting a game over state and the user agrees to restart, otherwise program is terminated. The program uses the turtle module for all the drawn displays as well as controls(arrow keys) for the game which are displayed in the splash-screen (up =accelerate,down=decelerate,left/right=rotate). The pygame module is used in order to play sounds during the course of the game without interruption, in this case, a different sound for each crash, as well as starting music upon launch. Attached with the main py program file are various images(gif) and sound(wav/mp3) files that are used to run the program. The program takes an object oriented approach and makes use of inheritance to allow for easier implementation of code to each sprite object that is used in the program(player/paper/rock/particles).

Resources/Sources:

This program functions with the use of module turtle as the main builder. It also uses modules pygame for sound/music, tkinter messagebox to display a restart message to the user, math to calculate collision, and random to generate random spawn locations for sprites. The following sources were used to further understand the modules:

<http://christianthompson.com/node/47>

<https://docs.python.org/2/library/turtle.html>

<https://www.pygame.org/wiki/tutorials>

<https://www.pygame.org/docs/ref/mixer.html>

<https://www.tutorialspoint.com/python3/tk_messagebox.htm>

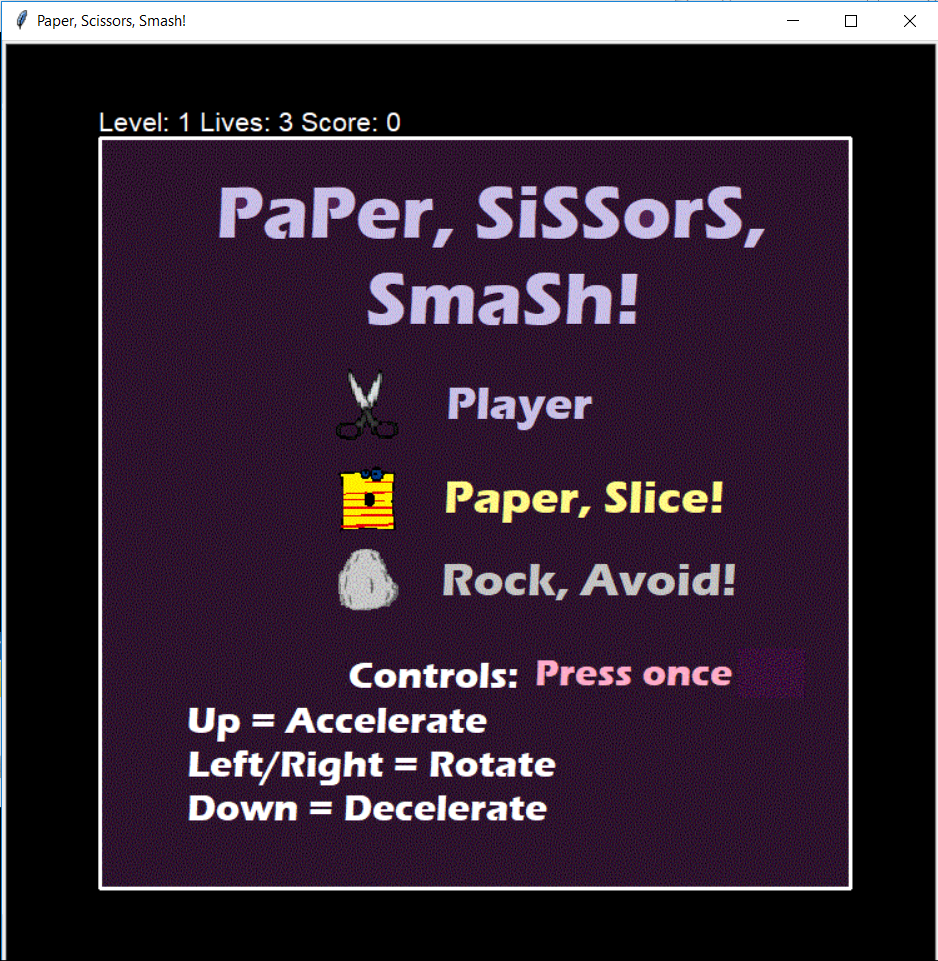
<http://www.jesshamrick.com/2011/05/18/an-introduction-to-classes-and-inheritance-in-python/>

Challenges:

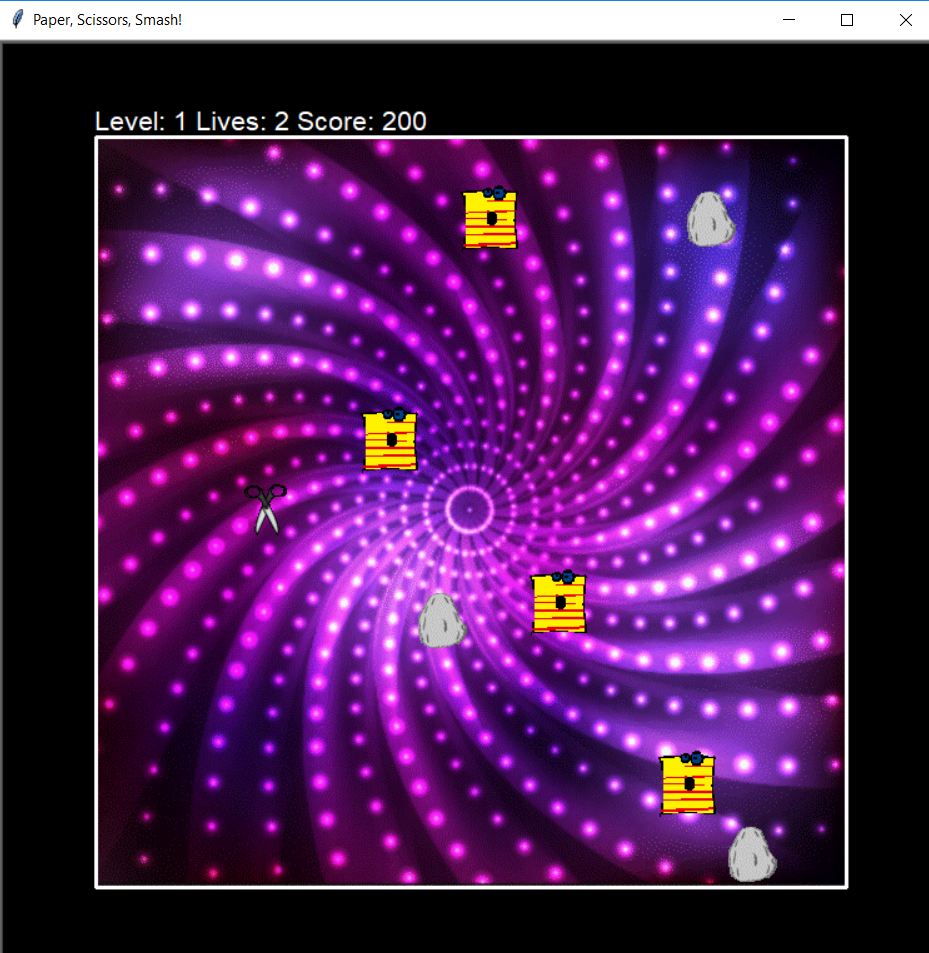
Originally the plan was to incorporate use of sqlite and further tkinter use in order to display a login screen which would allow the user to enter their usernames to play with and then be tracked with the associated score they receive by the end of the game. This hiscore was to be visible with a button on the login page. Unfortunately I was unable to incorporate those functions to work properly so those accessory features were scrapped for the time present and further development was focused on making sure the game itself was fully functional.

Screenshots:

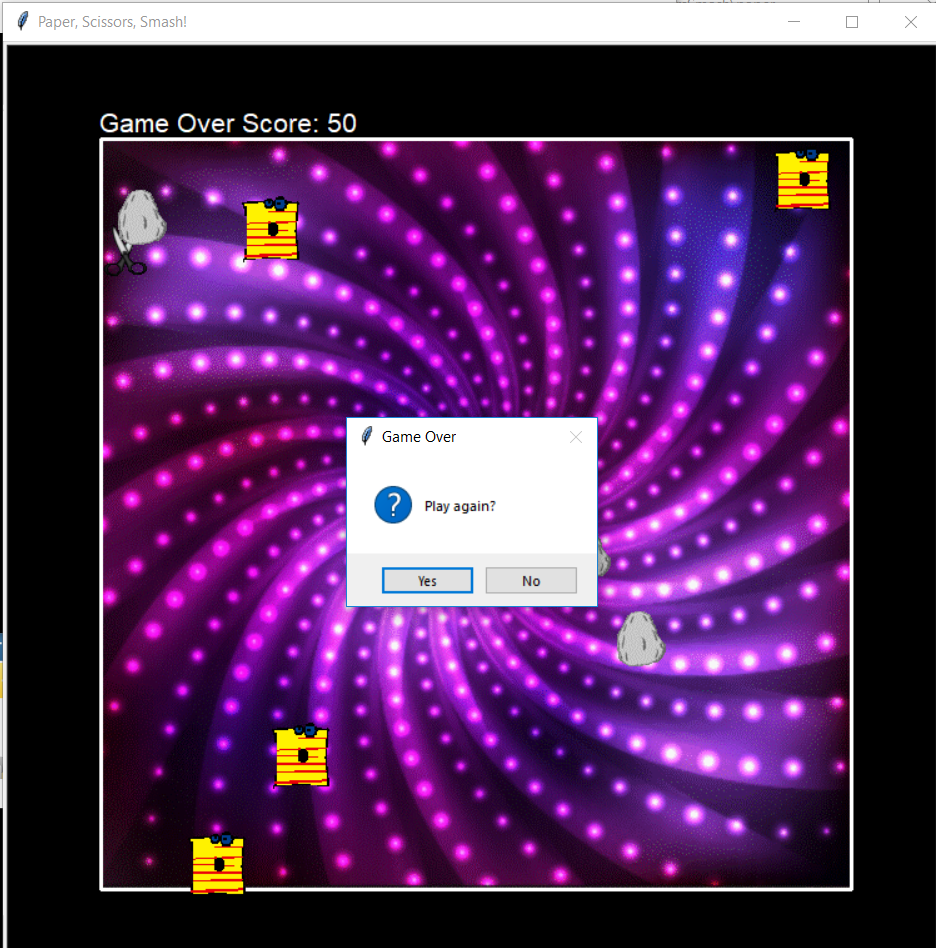
-initial display-splashscreen



-after splash, game state in playing



-upon losing and given restart message



Source code:

#Daniel Kolodziej

#4/25/18

#ITMD 413 Final Project

import os

import random

import time

import math

from tkinter import messagebox

import turtle

import pygame

#vars to store sound/music

file = 'zap.wav'

file2 = 'paperRip.wav'

file3 = 'gameMusic.mp3'

#play pregame music with pygame

pygame.mixer.pre\_init(44100, 16, 2, 4096) #frequency, size, channels, buffersize

pygame.init()

pygame.mixer.init()

pygame.mixer.music.load(file3)

pygame.mixer.music.play()

#Change the window size

turtle.setup(750, 750)

turtle.fd(0) #just in case for MacOSX

#Set the animations speed to the maximum

turtle.speed(0)

#Change the background color

turtle.bgcolor("black")

#Change the window title

turtle.title("Paper, Scissors, Smash!")

#Change the background image to starting splash screen

turtle.bgpic("my-splash-screen.gif")

#Hide the default turtle

turtle.ht()

#This saves memory

turtle.setundobuffer(1)

#This speeds up drawing

turtle.tracer(0)

#Register sprite shapes

turtle.register\_shape("transparent\_paper.gif")

turtle.register\_shape("transparent\_stone.gif")

turtle.register\_shape("transparent\_scissors.gif")

turtle.register\_shape("transparent\_scissors45.gif")

turtle.register\_shape("transparent\_scissors90.gif")

turtle.register\_shape("transparent\_scissors135.gif")

turtle.register\_shape("transparent\_scissors180.gif")

turtle.register\_shape("transparent\_scissors225.gif")

turtle.register\_shape("transparent\_scissors270.gif")

turtle.register\_shape("transparent\_scissors315.gif")

#main parent class of game objects

class Sprite(turtle.Turtle):

def \_\_init\_\_(self, spriteshape, color, startx, starty):

turtle.Turtle.\_\_init\_\_(self, shape = spriteshape)

self.speed(0)

self.penup()

self.color(color)

self.fd(0)

self.goto(startx, starty)

self.speed = 1

def move(self):

self.fd(self.speed)

#Boundary border detection to rotate and bounce off

if self.xcor() > 280:

self.setx(280)

self.rt(60)

if self.xcor() < -280:

self.setx(-280)

self.rt(60)

if self.ycor() > 280:

self.sety(280)

self.rt(60)

if self.ycor() < -280:

self.sety(-280)

self.rt(60)

#method to check if a sprite has collided with another object

def is\_collision(self, other):

if (self.xcor() >= (other.xcor() - 20)) and \

(self.xcor() <= (other.xcor() + 20)) and \

(self.ycor() >= (other.ycor() - 20)) and \

(self.ycor() <= (other.ycor() + 20)):

return True

else:

return False

#child class of Sprite

class Player(Sprite):

def \_\_init\_\_(self, spriteshape, color, startx, starty):

Sprite.\_\_init\_\_(self, spriteshape, color, startx, starty)

self.shapesize(stretch\_wid=0.6, stretch\_len=1.1, outline=None)

self.speed = 2

self.lives = 3

self.headPosition = 90

#methods for controls' actions

def turn\_left(self):

self.lt(45)

self.headPosition -= 45

if self.headPosition < 0:

self.headPosition = 360

print(self.headPosition)

def turn\_right(self):

self.rt(45)

self.headPosition +=45

if self.headPosition > 360:

self.headPosition = 0

print(self.headPosition)

def accelerate(self):

self.speed += 1

def decelerate(self):

self.speed -= 1

#'''

#override move method to changer for player class

def move(self):

self.fd(self.speed)

#Boundary border detection to rotate and bounce off

if self.xcor() > 280:

self.setx(280)

self.rt(45)

self.headPosition += 45

if self.headPosition > 360:

self.headPosition = 0

print(self.headPosition)

if self.xcor() < -280:

self.setx(-280)

self.rt(45)

self.headPosition += 45

if self.headPosition > 360:

self.headPosition = 0

print(self.headPosition)

if self.ycor() > 280:

self.sety(280)

self.rt(45)

self.headPosition += 45

if self.headPosition > 360:

self.headPosition = 0

print(self.headPosition)

if self.ycor() < -280:

self.sety(-280)

self.rt(45)

self.headPosition += 45

if self.headPosition > 360:

self.headPosition = 0

print(self.headPosition)

#'''

#child class of Sprite class

class Paper(Sprite):

def \_\_init\_\_(self, spriteshape, color, startx, starty):

Sprite.\_\_init\_\_(self, spriteshape, color, startx, starty)

self.speed = 5 #speed at which object will move

self.setheading(random.randint(0,360)) #to move in diff direction

#child class of Sprite class

class Rock(Sprite):

def \_\_init\_\_(self, spriteshape, color, startx, starty):

Sprite.\_\_init\_\_(self, spriteshape, color, startx, starty)

self.speed = 7

self.setheading(random.randint(0,360))

#override move method to changer for rock class

def move(self):

self.fd(self.speed)

#Boundary border detection

if self.xcor() > 280:

self.setx(280)

self.lt(60)

if self.xcor() < -280:

self.setx(-280)

self.lt(60)

if self.ycor() > 280:

self.sety(280)

self.lt(60)

if self.ycor() < -280:

self.sety(-280)

self.lt(60)

#child class of Sprite class

class Particle(Sprite):

def \_\_init\_\_(self, spriteshape, color, startx, starty):

Sprite.\_\_init\_\_(self, spriteshape, color, startx, starty)

self.shapesize(stretch\_wid=0.1, stretch\_len=0.1, outline=None)

self.goto(-1000,-1000)

self.frame = 0

#starting position for particles to form

def explode(self, startx, starty):

self.goto(startx,starty)

self.setheading(random.randint(0,360))

self.frame = 1

#particles move to simulate explosion

def move(self):

if self.frame > 0:

self.fd(10)

self.frame += 1

if self.frame > 15:

self.frame = 0

self.goto(-1000, -1000)

#game class for game state rules

class Game():

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

self.level = 1

self.score = 0

self.state = "splash"

self.pen = turtle.Turtle()

self.lives = 3

self.count = 0

def draw\_border(self):

#Draw Border

self.pen.speed(0)

self.pen.color("white")

self.pen.pensize(3)

self.pen.penup()

self.pen.goto(-300, 300)

self.pen.pendown()

for side in range(4):

self.pen.fd(600)

self.pen.rt(90)

self.pen.ht()

self.pen.penup()

def show\_status(self):

self.pen.undo()

#displays top message according to amount of player lives

if game.lives > 0:

msg = "Level: %s Lives: %s Score: %s " %(self.level, self.lives, self.score)

else:

msg = "Game Over Score: %s" %(self.score)

self.pen.penup()

self.pen.goto(-300, 300)

self.pen.write(msg, font=("Arial", 16, "normal"))

#displays initial splash art screen, waits 5 seconds and changes game state

def show\_splash(self):

turtle.bgpic("my-splash-screen.gif")

turtle.update()

time.sleep(6)

turtle.bgpic("newbackground.gif")

self.state = "setup"

def set\_state(self, state):

states = ["splash", "setup", "playing", "restart", "gameover"]

if state in states:

self.state = state

else:

state = "splash"

#Create game object

game = Game()

#Draw the game border

game.draw\_border()

#Show the game status

game.show\_status()

if game.state == "splash":

game.show\_splash()

if game.state == "setup":

#Create my sprites

player = Player("transparent\_scissors90.gif", "white", 0, 0) #registered shape

#Keyboard bindings to methods

turtle.onkey(player.turn\_left, "Left")

turtle.onkey(player.turn\_right, "Right")

turtle.onkey(player.accelerate, "Up")

turtle.onkey(player.decelerate, "Down")

turtle.listen()

#papers sprite list to populate with paper objects

papers =[]

for i in range(4):

x = random.randint(-200, 200)

y = random.randint(-200, 200)

papers.append(Paper("transparent\_paper.gif", "red", x, y))

#rocks sprite list to populate with rock objects

rocks =[]

for i in range(3):

x = random.randint(-200, 200)

y = random.randint(-200, 200)

rocks.append(Rock("transparent\_stone.gif", "blue", x, y))

#particles sprite list to populate with particle object

particles = []

for i in range(20):

particles.append(Particle("circle", "gray", 0, 0))

game.state = "playing"

#Main game loop

while True:

#reset trackers when game restarts

if game.state == "restart":

game.lives = 3

game.score = 0

player.speed = 0

player.goto(0,0)

player.setheading(0)

game.count = 0

player.headPosition = 90

game.show\_status()

#spawn sprites in random location

for paper in papers:

paper.goto(random.randint(-200, 200), random.randint(-200, 200))

for rock in rocks:

rock.goto(random.randint(-200, 200), random.randint(-200, 200))

game.state = "playing"

if game.state == "playing":

turtle.update()

time.sleep(0.02)

#allows player to move with set key bindings

player.move()

if player.headPosition == 90:

player.shape("transparent\_scissors90.gif")

elif player.headPosition == 45:

player.shape("transparent\_scissors45.gif")

elif player.headPosition == 135:

player.shape("transparent\_scissors135.gif")

elif player.headPosition == 225:

player.shape("transparent\_scissors225.gif")

elif player.headPosition == 315:

player.shape("transparent\_scissors315.gif")

elif player.headPosition == 180:

player.shape("transparent\_scissors180.gif")

elif player.headPosition == 270:

player.shape("transparent\_scissors270.gif")

elif player.headPosition == 360:

player.shape("transparent\_scissors.gif")

elif player.headPosition == 0:

player.shape("transparent\_scissors.gif")

elif player.headPosition < 0:

player.headPosition == 360

elif player.headPosition > 360:

player.headPosition == 0

#makes all paper sprite move on own

for paper in papers:

paper.move()

#Check for a collision with the player

if player.is\_collision(paper):

#Play rip sound with pygame

pygame.init()

pygame.mixer.init()

pygame.mixer.music.load(file2)

pygame.mixer.music.play()

x = random.randint(-250, 250)

y = random.randint(-250, 250)

paper.goto(x, y)

game.score += 100

game.count += 1

#tracks "kills" in order to add bonus lives

if game.count == 3:

game.lives += 1

game.count = 0

game.show\_status()

#makes all rock sprite move on own

for rock in rocks:

rock.move()

#Check for a collision between the missile and the ally

if player.is\_collision(rock):

#Play explosion sound with pygame

pygame.init()

pygame.mixer.init()

pygame.mixer.music.load(file)

pygame.mixer.music.play()

for particle in particles:

particle.explode(rock.xcor(), rock.ycor())

x = random.randint(-250, 250)

y = random.randint(-250, 250)

rock.goto(x, y)

#Decrease the score

game.score -= 50

game.count -= 1

#reset bonus lives count if wrong crash

if game.count < 0:

game.count = 0

game.lives -= 1

if game.lives < 1:

game.state = "gameover"

game.show\_status()

#makes all particle sprites move on own

for particle in particles:

particle.move()

#checks if game state is gameoever in which player is given option to play again

if game.state == "gameover":

for i in range(360):

player.rt(1)

if messagebox.askyesno("Game Over", "Play again?") == True:

game.state = "restart"

else:

exit()