OJ Ruiner

Game Design Document

Ian Kowalski

“OJ Ruiner” will be a 2D game that utilizes python and simple GE to show basic game creation skills, and hopefully be fun.

The game will focus around you guiding a mug around the screen to catch oranges, and avoid spiders. Spiders decrease your score, while oranges increase your score. Each play a noise when making contact with the mug, as well. The mug can move up, down, left, or right. Both spiders and oranges can come from all 4 sides of the screen, and reset to a different origin when exiting the visible screen. Users have a 10 second timer to collect as many oranges as possible.

The start screen displays the highest score from a previous game, to give the player something to aspire to beat. The screen will have a Play button, which plays the game when clicked. The other button is a quit button, which will stop the game entirely.

Upon the timer running out, the player will be returned to the start screen with the score they just achieved displayed. Then, they may choose to play again.

**Instructions Code**:

This section controls game access. Elements include:

btnPlay – the play button

btnQuit- the button to quit the game

instructions – a multilabel containing the games isntrucitons

prevscore – the score of the previous itereation of the gameplay section

response – represents user’s intent

precvscore – records the previous score

Here is the initialization code:

class Instructions(simpleGE.Scene):

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

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

self.prevScore = prevScore

self.setImage("diningRoom.png")

self.response = "Quit"

#self.prevScore = 0

self.directions = simpleGE.MultiLabel()

self.directions.textLines = [

"You live in BSU housing, which you didn't clean",

"Now, spiders are crawling around in the morning",

"Catch oranges in your mug, but not spiders!",

"",

"If there are spiders in my morning OJ",

"My day is ruined and I will cry"]

#self.response = ("Play")

self.directions.center = (320,240)

self.directions.size = (500,250)

self.btnPlay = simpleGE.Button()

self.btnPlay.text = "Play"

self.btnPlay.fgColor = ("Purple")

self.btnPlay.clearBack = True

self.btnPlay.center = (100,400)

self.btnQuit = simpleGE.Button()

self.btnQuit.clearBack = True

self.btnQuit.text = "Quit"

self.btnQuit.fgColor = ("red")

self.btnQuit.center = (540,400)

self.lblScore = simpleGE.Label()

self.lblScore.center = (320,400)

self.lblScore.text = f"Last Score: {self.prevScore}"

self.lblScore.clearBack = True

self.sprites = [self.directions,

self.btnPlay,

self.btnQuit,

self.lblScore]

Here is the event handling code:

def process(self):

if self.btnPlay.clicked:

self.response = "Play"

self.stop()

if self.btnQuit.clicked:

self.response = "Quit"

self.stop()

**The Game Class**

It will be the most important class, and is subclassed from simpleGE.Scene

The Game class has visuals that include:

Apple – an instance of Charlie class

Other Fruit – an instance of Coin class

Blender – an instance of Hurt class

Non- visual assets include:

sndCoin – the coin sound effect representing score

sndHurts- the hurt sound effect representing hitting a blender

**Game Class Components**:

Visuals include a game class as well

**Mug (currently Charlie)**

This is the player controlled spite moving up, down, left, or right, which will increase/decrease score when colliding with orange/spider

Dimensions and size will be determined by the following code:

self.setImage("Charlie.png")

self.setSize(25,25)

self.position = (320,400)

self.moveSpeed = 5

event handling is in the process method, which responds to the players directional input

def process(self):

If user presses the left arrow:

subtract the speed value from the x position

if user presses the right arrow:

add the speed value to the x position

if user presses the up arrow:

subtract the speed value from the y position

if user presses the down arrow:

add the speed value to the y position

**Orange (currently coin):**

The orange move from the side of the screen it originate from (top, bottom, left, or right) , to the opposite side, where it will then reset. Upon reset, it changes origin and speed

class Coin(simpleGE.Sprite): - creating class Coin with a sprite, aka the Orange.

def \_\_init\_\_ with parameters self and scene

super().\_\_init\_\_(scene) so that we can reuse intialized scenes

set the coin image to the image Orange.png

set the coin size to 25,25

set the coins minimum speed to 3

set the coins maximum speed to 8

set the coints minimum negative value to -3 (minimum as in smallest movement distance, in the negative direction)

set the coins maximum negative value to negative 8

self.reset() call function reset

def reset(self):

choose side of the screen coins/hurts come from by selecting a random int # 1 through 4

if # == 1:

self.y = 10

self.x = random.randint(0, self.screenWidth)

self.dy = random.randint(self.minSpeed,self.maxSpeed) (top of screen to bottom)

if # == 2:

self.y = random.randint(0, self.screenHeight)

self.x = 640

self.dx = random.randint(self.maxNegative, self.minNegative) (right of screen to left)

if # == 3:

self.y = 450

self.x = random.randint(0, self.screenWidth)

self.dy = random.randint(self.maxNegative, self.minNegative) (bottom of screen to top)

if # == 4:

self.y = random.randint(0, self.screenHeight)

self.x = 0

self.dx = random.randint(self.minSpeed,self.maxSpeed) (left of screen to right)

def checkBounds(self):

if the bottom of the the coin is greater than the screen height:

reset the coin, giving it new random numbers as well

elif the y value is less than 0:

reset the coin

elif the x value is greaeter than 640:

reset the coin

elif the x value is less than 0:

reset the coin (all 4 collectively reset the coin if it exists the screen that is visible to the play)

**Spider (currently hurts):**

The hurt behaves identically to the orange, but instead removes a point form the players score

class Hurt(simpleGE.Sprite): - creating class Hurt with a sprite, aka the Spider

def \_\_init\_\_ with parameters self and scene

super().\_\_init\_\_(scene) so that we can reuse intialized scenes

set the Spider image (using self) (its the obstacle charlie should avoid)

set the hurts size to 25,25

set the hurts minimum speed to 5

set the hurts maximum speed to 8

set the coints minimum negative value to -3 (minimum as in smallest movement distance, in the negative direction)

set the coins maximum negative value to negative 8

self.reset() call function reset

def reset(self):

choose side of the screen coins/hurts come from by selecting a random int # 1 through 4

if # == 1:

self.y = 10

self.x = random.randint(0, self.screenWidth)

self.dy = random.randint(self.minSpeed,self.maxSpeed) (top of screen to bottom)

if # == 2:

self.y = random.randint(0, self.screenHeight)

self.x = 640

self.dx = random.randint(self.maxNegative, self.minNegative) (right of screen to left)

if # == 3:

self.y = 450

self.x = random.randint(0, self.screenWidth)

self.dy = random.randint(self.maxNegative, self.minNegative) (bottom of screen to top)

if # == 4:

self.y = random.randint(0, self.screenHeight)

self.x = 0

self.dx = random.randint(self.minSpeed,self.maxSpeed) (left of screen to right)

def checkBounds(self):

if the bottom of the the hurt is greater than the screen height:

reset the hurt, giving it new random numbers as well

elif the y value is less than 0:

reset the hurt

elif the x value is greaeter than 640:

reset the hurt

elif the x value is less than 0:

reset the hurt (all 4 collectively reset the hurt if it exists the screen that is visible to the play)

**Main**:

This allows the game to run, and is represented by the following code:

def main():

game = Game()

game.start()

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

main()

**Milestone Plan**:

This is the order of my milestone plan. The order will be gameplay, instructions, and then state management. These will each be completed in order, ideally, one after the other.

1. Game scene with background image

2. Add basic Charlie sprite

3. Add keyboard motion to Charlie

4. Add single coin with reset, falling and boundary behaviors

5. Add collision effect between charlie and coin, sound effect

6. Modify for multiple (ten) coins including collision behavior

7. Add scorekeeping, timing, and appropriate labels

8. Add instructions class and state transition

**Assets**: These are my current assets

Dining Room (self created in Paint)A drawing of chairs and a square picture

Description automatically generated

Spider (self-created in Paint)

A black and red face with two eyes

Description automatically generated

Mug (made in paint)

A blue rectangle with a circle and letters on it

Description automatically generated

Coin.wav

Custom audio by Andy Harris created with jsfxr (in github)

hurtNoise.wav

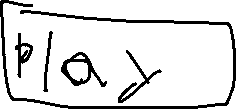
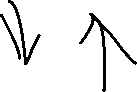
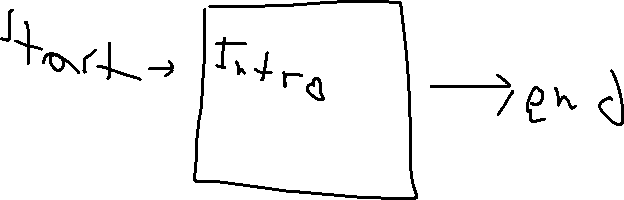
custom audio by Ian Kowalski created with jsfxr (in github

Orange.png (self-created in paint)

An orange oval with a green object in the middle

Description automatically generated

**State Transition Diagram**



The game uses 2-state system, with each state being a simpleGE subclass. Upon starting the game, you are taken to the intro screen with 2 options: play or quit the game. By clicking the play button, the current subclass ends, and it takes users to the gameplay section. Upon selecting the Quit button, the game ends entirely.



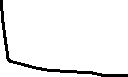
Once time expires, the gameplay section is closed, and the intro screen plays again. However, this time it will display the score the user got during their previous run of the game play section.

**Instructions Screen**

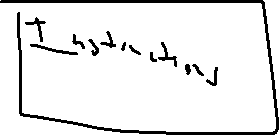
Previous score passed as a parameter

Set Response to Quit and close

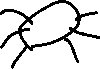
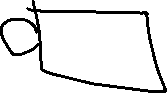
Set Response to Play and close



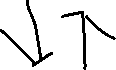
Background = diningRoom.png



**Game Class Diagram**



Upon reaching the side of the screen opposite of origin, resets the oranges or spiders, giving it new speeds and a new origin on the screen



Oranges, coming from all 4 sides of the screen, which the player should make contact with to increase their score

Spider, coming from all 4 sides of the screen, which the user should avoid

Player controlled mug, which moves up, down, left, right, with corresponding keys. The player should move it toward oranges, and away from spiders

Score Counting how many other fruit the user hit

