**INTRODUCTION**

‘Fruitz by god’ is our version of the game fruit ninja originally created by halfbrick. In the given timeframe of 60 seconds, the target is to slice as many fruits as possible. Our game has different wallpapers and two different knives to choose from and make the game more interactice and fun to play. All it takes to play the game is a little concentration and decent hand eye coordination. No special skills are required.

**SYSTEM REQUIREMENTS**

The system should be able to support python version 2.7.4 and the pygame module should also be downloaded.

**MODULES**

We used the following modules in our game

-Future

-Pygame

-Math

-Random

-time

-datetime

**SOURCE CODE**

from \_\_future\_\_ import division

from pygame import \*

from math import \*

from random import randint, choice, uniform

from time import localtime as local

from time import time as clock

import datetime

#Initializations

init()

mixer.init()

font.init()

#Time object that helps set FPS

fps = time.Clock()

#Creating the display surface, setting the caption and icon

width, height = 1280,720

screen = display.set\_mode((width, height))

display.set\_caption("Fruitz by god")

icon = transform.scale(image.load("Other Images/Icon.png"), (64, 64))

display.set\_icon(icon)

#Opening file for writing high scores

highFile = open("Files/High Scores.txt")

oldHS = int(highFile.read()); highscore = oldHS

#Variable for the current screen

mode = "Splash Screen"

'''Importing and setting up all gameplay images'''

apple\_main = transform.scale(image.load("Fruit Images/apple.png"), (128, 128)).convert()

apple\_bit1 = transform.scale(image.load("Fruit Images/applebit1.png"), (36, 48)).convert()

apple\_bit2 = transform.scale(image.load("Fruit Images/applebit2.png"), (44, 68)).convert()

banana\_main = transform.scale(image.load("Fruit Images/banana.png"), (128, 128)).convert()

banana\_bit1 = transform.scale(image.load("Fruit Images/bananabit1.png"), (56, 56)).convert()

banana\_bit2 = transform.scale(image.load("Fruit Images/bananabit2.png"), (92, 62)).convert()

coconut\_main = transform.scale(image.load("Fruit Images/coconut.png"), (128, 128)).convert()

coconut\_bit1 = transform.scale(image.load("Fruit Images/coconutbit1.png"), (74, 102)).convert()

coconut\_bit2 = transform.scale(image.load("Fruit Images/coconutbit2.png"), (64, 100)).convert()

lemon\_main = transform.scale(image.load("Fruit Images/lemon.png"), (128, 128)).convert()

lemon\_bit1 = transform.scale(image.load("Fruit Images/lemonbit1.png"), (76, 70)).convert()

lemon\_bit2 = transform.scale(image.load("Fruit Images/lemonbit2.png"), (74, 84)).convert()

pear\_main = transform.scale(image.load("Fruit Images/pear.png"), (128, 128)).convert()

pear\_bit1 = transform.scale(image.load("Fruit Images/pearbit1.png"), (56, 66)).convert()

pear\_bit2 = transform.scale(image.load("Fruit Images/pearbit2.png"), (36, 68)).convert()

watermelon\_main = transform.scale(image.load("Fruit Images/watermelon.png"), (128, 128)).convert()

watermelon\_bit1 = transform.scale(image.load("Fruit Images/watermelonbit1.png"), (84, 98)).convert()

watermelon\_bit2 = transform.scale(image.load("Fruit Images/watermelonbit2.png"), (128, 132)).convert()

#Correlating the fruit types with their respective images in dictionaries

Fruit\_Images = {"apple": apple\_main, "banana": banana\_main, "coconut": coconut\_main, "lemon": lemon\_main, \

"pear": pear\_main, "watermelon": watermelon\_main}

Bit\_Images1 = {"apple": apple\_bit1, "banana": banana\_bit1, "coconut": coconut\_bit1, "lemon": lemon\_bit1, \

"pear": pear\_bit1, "watermelon": watermelon\_bit1}

Bit\_Images2 = {"apple": apple\_bit2, "banana": banana\_bit2, "coconut": coconut\_bit2, "lemon": lemon\_bit2, \

"pear": pear\_bit2, "watermelon": watermelon\_bit2}

imgs = [apple\_main, apple\_bit1, apple\_bit2, banana\_main, banana\_bit1, banana\_bit2,

coconut\_main, coconut\_bit1, coconut\_bit2, lemon\_main, lemon\_bit1, lemon\_bit2,

pear\_main, pear\_bit1, pear\_bit2, watermelon\_main, watermelon\_bit1, watermelon\_bit2]

for img in imgs:

img.set\_colorkey((0, 0, 0))

if imgs.index(img) % 3 == 0:

img.set\_alpha(250)

else:

img.set\_alpha(125)

wall1 = image.load("Wallpapers/Wallpaper1.png").convert()

wall1.set\_alpha(255)

wall1List = [(0, 0, 0), (235, 61, 0), (0, 0, 0), (247, 237, 0)]

wall2 = image.load("Wallpapers/Wallpaper2.png").convert()

wall2.set\_alpha(150)

wall2List = [(0, 179, 224), (247, 237, 0), (163, 232, 255), (188, 226, 158)]

wall3 = image.load("Wallpapers/Wallpaper5.png").convert()

wall3.set\_alpha(255)

wall3List = [(255, 250, 80), (255, 255, 255), (98, 222, 253), (244, 101, 36)]

wall4 = image.load("Wallpapers/Wallpaper4.png").convert()

wall4.set\_alpha(255)

wall4List = [(247, 237, 0), (0, 0, 0), (245, 182, 203), (98, 222, 253)]

wallpaper = wall3

wallList = wall3List

#Importing the images for the knife blades in the game

kitchenblade = transform.scale(image.load("Other Images/KitchenBlade.png"), (128, 128))

fightingblade = transform.scale(image.load("Other Images/FightingBlade.png"), (128, 128))

blade = "kitchen"

blades = {"kitchen": kitchenblade, "fighting": fightingblade}

#Indexes of the respective colours (score, HS, time, FPS)

colS = 0

colHS = 1

colT = 2

colFPS = 3

#Loading of other screens in game (credits, options, pause menu, etc.)

load\_screen = image.load("Screens/PythonLoader.png").convert()

splash\_screen = image.load("Screens/Loading Screen.png").convert()

credits\_screen = image.load("Screens/Untitled.png").convert()

options\_screen = image.load("Screens/Options.png").convert()

customize\_screen = image.load("Screens/Options\_Customize.png").convert()

gameover\_screen = image.load("Screens/Game Over.png").convert()

paused\_screen = image.load("Screens/Paused.png").convert()

'''Importing Sound Effects and Background Music'''

#Sound effects

Press = mixer.Sound("Sounds/Press.wav"); Press.set\_volume(0.5)

Tick = mixer.Sound("Sounds/Tick.wav"); Tick.set\_volume(0.5)

Punch = mixer.Sound("Sounds/Punch.wav"); Punch.set\_volume(0.5)

Splash = mixer.Sound("Sounds/Splash.wav"); Splash.set\_volume(0.5)

Woosh = mixer.Sound("Sounds/Woosh.wav"); Woosh.set\_volume(0.5)

#Background music

song1 = mixer.Sound("Sounds/Pompeii.wav"); song1.set\_volume(0.25)

song2 = mixer.Sound("Sounds/All I Do Is Win.wav"); song2.set\_volume(0.25)

song3 = mixer.Sound("Sounds/Animals.wav"); song3.set\_volume(0.25)

song4 = mixer.Sound("Sounds/Wipe Out.wav"); song4.set\_volume(0.25)

song = song3

'''Importing fonts'''

smallFont = font.Font("Fonts/Avenir.ttc", 18)

medFont = font.Font("Fonts/Avenir.ttc", 40)

largeFont = font.Font("Fonts/Avenir.ttc", 60)

'''Function definitions'''

def mp():

return mouse.get\_pos()

def lclick():

if mouse.get\_pressed()[0] == 1:

return True

else:

return False

def collide(col\_type, basepoint, movepoint, len1, len2 = None):

if col\_type == "circle":

bx, by = basepoint

mx, my = movepoint

if ((bx-mx)\*\*2 + (by-my)\*\*2)\*\*0.5 <= len1 and lclick() == True:

return True

else:

return False

elif col\_type == "rect":

bx, by = basepoint

mx, my = movepoint

collideRect = Rect (bx-len1//2,by-len2//2,len1, len2)

if collideRect.collidepoint((mx,my)) and lclick() == True:

return True

else:

return False

def randfruit():

fruits = ["apple", "banana", "coconut", "lemon", "pear", "watermelon"]

return choice(fruits)

'''Setting up object classes'''

CollisionRadius = {"apple": 35, "banana": 64, "coconut": 40, "lemon": 50, "pear": 40, "watermelon": 50}

#List of the options of x starting values

x\_choices = [x for x in range (0,width//4+1,2)] + [x for x in range (width//4, 3\*width//4 + 1)] + [x for x in range (3\*width//4, width+1,40)]

#Class Fruit: Contains all Fruits that have not yet been sliced, while they are on the screen

class Fruit:

def \_\_init\_\_ (self, fruit\_type, loading = False):

self.x, self.y = choice(x\_choices), height + 1

self.fruit\_type = fruit\_type

self.dist = abs(self.x - width//2)

self.vx = self.dist//70

self.vy = uniform (7,8)

self.angle = 0

#1 means moving from left to right, -1 means moving from right to left

if self.x > width//2:

self.direction = -1

elif 0 <= self.x <= width//4:

self.direction = 1

else:

self.direction = choice ([1, 1, 1, 1, 1, -1, 1, -1, 1, 1, 1, -1, 1, 1, 1, 1, 1, -1])

if self.fruit\_type == "banana":

self.collide\_type = "rect"

else:

self.collide\_type = "circle"

self.draw\_image = transform.rotate(Fruit\_Images[self.fruit\_type], self.angle)

self.dx, self.dy = int(self.x - (self.draw\_image.get\_width()//2)), int(self.y - (self.draw\_image.get\_height()//2))

self.loading = loading

def updatePos(self):

if self.loading == False:

self.vy -= 0.07

self.x += int(self.vx\*self.direction)

self.y -= int(self.vy)

#Adds by self.direction so that the fruits rotate the same way they're moving

self.angle += self.direction\*choice([1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 3])

self.draw\_image = transform.rotate(Fruit\_Images[self.fruit\_type], self.angle)

#By creating dx and dy, x and y are at the center of the fruit image and we can draw circles for collision

self.dx, self.dy = int(self.x - (self.draw\_image.get\_width()//2)), int(self.y - (self.draw\_image.get\_height()//2))

def checkCollide(self):

if self.fruit\_type != "banana":

if collide(self.collide\_type, (self.x, self.y), mp(), CollisionRadius[self.fruit\_type]) == True:

return True

else:

return False

else:

if collide(self.collide\_type, (self.x, self.y), mp(), self.draw\_image.get\_width(), self.draw\_image.get\_height()) == True:

return True

else:

return False #True means colliding, False means not colliding

def drawFruit(self, surf):

surf.blit(self.draw\_image, (self.dx, self.dy))

#Makes the corresponding sound if the fruit is sliced ("punch" for bananas, "woosh" otherwise)

def makeSound(self):

if self.fruit\_type == "banana":

Punch.play()

else:

Woosh.play()

air = []

cut = []

for i in range (choice([1, 1, 2, 2, 2, 3, 3, 4, 5])):

air.append(Fruit(randfruit()))

class Bits:

def \_\_init\_\_ (self, bit\_type, startx, starty, ang, vy):

self.bit\_type = bit\_type

self.x1, self.x2, self.y = startx, startx, starty

self.angle = ang

self.vx = uniform (3, 4)

self.vy = vy

self.draw\_image1 = transform.rotate (Bit\_Images1[self.bit\_type], self.angle)

self.draw\_image2 = transform.rotate (Bit\_Images2[self.bit\_type], self.angle)

def updatePos(self):

self.vy += 0.3

self.x1 -= self.vx

self.x2 += self.vx

self.y += self.vy

self.draw\_image1 = transform.rotate (Bit\_Images1[self.bit\_type], self.angle)

self.draw\_image2 = transform.rotate (Bit\_Images2[self.bit\_type], self.angle)

def drawBits(self, surf):

surf.blit(self.draw\_image1, (int(self.x1), int(self.y)))

surf.blit(self.draw\_image2, (int(self.x2), int(self.y)))

class Button:

def \_\_init\_\_ (self, rect, up, down, click = None):

self.rect = rect

self.up = up

self.down = down

if click != None:

self.click = click

#If the mouse is simply over the button, this returns true to draw the "Down" image

def checkHover(self):

if self.rect.collidepoint(mp()):

return True

else:

return False

#If the button is being clicked, this returns True

def checkClicked(self):

if self.checkHover() == True and lclick():

return True

else:

return False

def drawButton(self, surf):

surf.blit(self.up, (self.rect.x, self.rect.y))

if self.checkHover() == True:

surf.blit(self.down, (self.rect.x, self.rect.y))

elif hasattr(self, "click") and self.checkClicked() == True:

surf.blit(self.click, (self.rect.x, self.rect.y))

class Blade:

def \_\_init\_\_(self, blade\_type):

self.blade\_type = blade\_type

self.original\_image = blades[self.blade\_type]

def updateBlade(self, pos):

x = pos[0]

if x >= width//2:

self.draw\_image = transform.rotate(self.original\_image, 45)

else:

self.draw\_image = transform.rotate(self.original\_image, 45)

self.draw\_image = transform.flip(self.draw\_image, True, False)

return self.draw\_image

def drawBlade(self, surf, pos):

x = pos[0]

if x < width//2:

surf.blit(self.updateBlade(pos), (pos[0] - self.updateBlade(pos).get\_width()//2 - 50, pos[1] - self.updateBlade(pos).get\_height()//2 + 50))

else:

surf.blit(self.updateBlade(pos), (pos[0] - 40, pos[1] - self.updateBlade(pos).get\_height()//2 + 50))

'''Creating buttons'''

Wall1\_Button = Button(Rect (138, 396, 467, 72), image.load("Buttons/Wall1Up.png").convert(), \

image.load("Buttons/Wall1Down.png").convert(), image.load("Buttons/Wall1Click.png").convert())

Wall2\_Button = Button(Rect (675, 396, 467, 72), image.load("Buttons/Wall2Up.png").convert(), \

image.load("Buttons/Wall2Down.png").convert(), image.load("Buttons/Wall2Click.png").convert())

Wall3\_Button = Button(Rect (138, 492, 467, 72), image.load("Buttons/Wall3Up.png").convert(), \

image.load("Buttons/Wall3Down.png").convert(), image.load("Buttons/Wall3Click.png").convert())

Wall4\_Button = Button(Rect (675, 492, 467, 72), image.load("Buttons/Wall4Up.png").convert(), \

image.load("Buttons/Wall4Down.png").convert(), image.load("Buttons/Wall4Click.png").convert())

Kitchen\_Button = Button(Rect (138, 630, 467, 72), image.load("Buttons/KitchenUp.png").convert(), \

image.load("Buttons/KitchenDown.png").convert())

Fighting\_Button = Button(Rect (675, 630, 467, 72), image.load("Buttons/FightingUp.png").convert(), \

image.load("Buttons/FightingDown.png").convert())

Done\_Button = Button(Rect (50, 50, 150, 50), image.load("Buttons/DoneUp.png").convert(), \

image.load("Buttons/DoneDown.png").convert())

Pause\_Button = Button(Rect (50, 650, 150, 50), image.load("Buttons/PauseUp.png").convert(), \

image.load("Buttons/PauseDown.png").convert())

Customize\_Button = Button(Rect (406, 431, 467, 72), image.load("Buttons/CustomizeUp.png").convert(), \

image.load("Buttons/CustomizeDown.png").convert())

Sound\_Button = Button(Rect (406, 551, 467, 72), image.load("Buttons/SoundUp.png").convert(), \

image.load("Buttons/SoundDown.png").convert())

PlayAgain\_Button = Button(Rect (787, 428, 467, 72), image.load("Buttons/PlayAgainUp.png").convert(), \

image.load("Buttons/PlayAgainDown.png").convert())

MainMenu\_Button = Button(Rect (787, 526, 467, 72), image.load("Buttons/MainMenuUp.png").convert(), \

image.load("Buttons/MainMenuDown.png").convert())

Quit\_Button = Button(Rect (787, 624, 467, 72), image.load("Buttons/QuitUp.png").convert(), \

image.load("Buttons/QuitDown.png").convert())

'''Creating blade'''

blade = Blade("kitchen")

'''Defining the functions for each screen'''

def splashScreen():

global oldSec, oldMin, startSec, secLeft, mode, game

splashChoice = None

drawSplashBits = False

#For these three fruit objects, "loading" is set to True: we don't want the x and y values to update

splashFruits = [Fruit("banana", True), Fruit("apple", True), Fruit("watermelon", True)]

for fruit in splashFruits:

fruit.x = 653 + 225\*splashFruits.index(fruit) + 64

fruit.y = 394 + 64

fruit.vx, fruit.vy = 0, 0

modes = {"banana": "Credits", "apple": "Play", "watermelon": "Options"}

running = True

mouse.set\_visible(False)

while running:

for evt in event.get():

if evt.type == QUIT:

return "exit"

for fruit in splashFruits:

fruit.updatePos()

if drawSplashBits == True:

splashChoice.updatePos()

#If the bits fall off the screen, the mode is updated

if drawSplashBits == True:

if splashChoice.y > height:

mode = modes[splashChoice.bit\_type]

if mode == "Play":

oldSec = local(clock())[5]

oldMin = local(clock())[4]

secLeft = 60

air = []

cut = []

for i in range (choice([1, 1, 2, 2, 2, 3, 3, 4, 5])):

air.append(Fruit(randfruit()))

song.play()

running = False

drawBits = False

#Checking for collisions

for fruit in splashFruits:

if fruit.checkCollide() == True:

fruit.makeSound()

splashChoice = Bits(fruit.fruit\_type, fruit.x, fruit.y, fruit.angle, uniform(0.5, 1.5))

del splashFruits[splashFruits.index(fruit)]

drawSplashBits = True

screen.blit(splash\_screen, (0,0))

for fruit in splashFruits:

fruit.drawFruit(screen)

if drawSplashBits == True:

if splashChoice.y < height:

splashChoice.drawBits(screen)

blade.drawBlade(screen, mp())

display.flip()

fps.tick(60)

return mode

#Credits menu, accessed by swiping the banana on the splash screen

def creditsMenu():

mouse.set\_visible(True)

running = True

while running:

for evt in event.get():

if evt.type == QUIT:

return "exit"

if Done\_Button.checkClicked() == True:

running = False

screen.blit(credits\_screen, (0, 0))

Done\_Button.drawButton(screen)

display.flip()

return "Splash Screen"

#Pause menu, accessed while playing the game

def pauseScreen():

global mode, song, secLeft, pauseTime, score

mouse.set\_visible(True)

mixer.pause()

running = True

while running:

for evt in event.get():

if evt.type == QUIT:

return "exit"

if Done\_Button.checkClicked() == True:

running = False

if MainMenu\_Button.checkClicked() == True:

score = 0

return "Splash Screen"

if Quit\_Button.checkClicked() == True:

return "exit"

screen.blit(paused\_screen, (0, 0))

Done\_Button.drawButton(screen)

MainMenu\_Button.drawButton(screen)

Quit\_Button.drawButton(screen)

display.flip()

mixer.unpause()

secLeft = pauseTime

return "Play"

def optionsScreen():

global wallpaper, wallList, blade, song

buttons = [Wall1\_Button, Wall2\_Button, Wall3\_Button, Wall4\_Button]

wallpapers = [wall1, wall2, wall3, wall4]

lists = [wall1List, wall2List, wall3List, wall4List]

songs = [song1, song2, song3, song4]

mouse.set\_visible(True)

running = True

while running:

for evt in event.get():

if evt.type == QUIT:

return "exit"

for button in buttons:

if button.checkClicked() == True:

wallpaper = wallpapers[buttons.index(button)]

wallList = lists[buttons.index(button)]

song = songs[buttons.index(button)]

if Kitchen\_Button.checkClicked() == True:

blade = Blade("kitchen")

if Fighting\_Button.checkClicked() == True:

blade = Blade("fighting")

if Done\_Button.checkClicked() == True:

running = False

screen.blit(customize\_screen, (0, 0))

for button in buttons:

if wallpaper == wallpapers[buttons.index(button)]:

screen.blit(button.down, (button.rect.x, button.rect.y))

else:

button.drawButton(screen)

if blade.blade\_type == "kitchen":

screen.blit(Kitchen\_Button.down, (Kitchen\_Button.rect.x, Kitchen\_Button.rect.y))

else:

Kitchen\_Button.drawButton(screen)

if blade.blade\_type == "fighting":

screen.blit(Fighting\_Button.down, (Fighting\_Button.rect.x, Fighting\_Button.rect.y))

else:

Fighting\_Button.drawButton(screen)

Done\_Button.drawButton(screen)

display.flip()

return "Splash Screen"

def afterScreen():

global score, oldSec, oldMin, secLeft, air, cut, wallList, oldScore

mouse.set\_visible(True)

startCount = 0

running = True

while running:

for evt in event.get():

if evt.type == QUIT:

return "exit"

startCount += 1

if startCount >= 120:

if PlayAgain\_Button.checkClicked() == True:

oldSec = local(clock())[5]

oldMin = local(clock())[4]

air = []

cut = []

for i in range (choice([1, 1, 2, 2, 2, 3, 3, 4, 5])):

air.append(Fruit(randfruit()))

song.play()

return "Play"

if MainMenu\_Button.checkClicked() == True:

return "Splash Screen"

if Quit\_Button.checkClicked() == True:

return "exit"

screen.blit(gameover\_screen, (0, 0))

scoreBlit = largeFont.render("Score: " + str(oldScore), True, wallList[colS])

screen.blit(scoreBlit, (950, 350))

PlayAgain\_Button.drawButton(screen)

MainMenu\_Button.drawButton(screen)

Quit\_Button.drawButton(screen)

display.flip()

#Function for the actual game

score = 0

def main():

count = False

counter = 0

counter2 = 0

global mode, song, secLeft, oldSec, oldMin, pauseTime, air, cut, score, blade, oldScore

mouse.set\_visible(False)

running = True

while running:

for evt in event.get():

if evt.type == QUIT:

return "exit"

'Updating fruit positions'

for fruit in air:

fruit.updatePos()

'Updating bit positions'

for bit in cut:

bit.updatePos()

'Interactions'

#Instead, I created two "if" options: if the current second value doesn't equal the previous recorded second value OR the current second value equals

#the previous recorded second value however the minutes are different, then the seconds left counter should be decreased

if local(clock())[5] != oldSec or (local(clock())[4] != oldMin and local(clock())[5] == oldSec):

secLeft -= 1

oldSec = local(clock())[5]

oldMin = local(clock())[4]

if secLeft <= 0:

endScore = score

running = False

if count == True:

counter += 1

if count == False:

counter = 0

if counter >= choice ([10, 20, 20, 30, 30, 30, 50, 50, 100, 100, 100, 200, 200, 300]):

for i in range (choice([1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 5])):

air.append(Fruit(randfruit()))

count = False

#this randomly adds more fruit to the "air" list after an interval of time

counter2 += 1

if counter2 >= choice ([100, 100, 100, 200, 200, 300, 300, 400]):

for i in range (choice([1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 5])):

air.append(Fruit(randfruit()))

counter2 = 0

#Determines what the current high score is

if oldHS > score:

highscore = oldHS

else:

highscore = score

'Checking for collision'

for fruit in air:

if fruit.checkCollide() == True:

fruit.makeSound()

if blade.blade\_type == "kitchen":

point = choice ([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 10])

score += point

else:

point = choice ([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 10]) #The fighting blade gives a higher change of getting 10 points on a slice (user doesn't need to know)

score += point

del air[air.index(fruit)]

cut.append(Bits(fruit.fruit\_type, fruit.x, fruit.y, fruit.angle, fruit.vy))

count = True

'Checking if buttons are clicked'

if Pause\_Button.checkClicked() == True:

pauseTime = secLeft

return "Pause"

'Deleting fruits from air list if they are off the screen'

for fruit in air:

if fruit.y > height:

del air[air.index(fruit)]

'''Drawing'''

screen.blit(wallpaper, (0, 0))

for fruit in air:

fruit.drawFruit(screen)

for bit in cut:

bit.drawBits(screen)

Pause\_Button.drawButton(screen)

blade.drawBlade(screen, mp())

if secLeft % 10 == 0 or secLeft <= 9:

timeCol = (255, 23, 34)

else:

timeCol = wallList[colT]

secBlit = medFont.render(str(secLeft), True, timeCol)

screen.blit(secBlit, (width//2 - (secBlit.get\_width()//2), 15))

#Current Score

scoreBlit = largeFont.render(str(score), True, wallList[colS])

screen.blit(scoreBlit, (50, 15))

#High Score

highBlit = smallFont.render("High Score: " + str(highscore), True, wallList[colHS])

screen.blit(highBlit, (50, 88))

#FPS

fpsBlit = smallFont.render("FPS: " + str(fps.get\_fps())[0:2], True, wallList[colFPS])

screen.blit(fpsBlit, (1190, 15))

display.flip()

mouse.set\_visible(True)

mixer.stop()

oldScore = score

score = 0

secLeft = 60

return "After"

#Two second loading screen at the start of the program, I added this simply for effect

#If the user presses enter or space it is skipped (like in a real game)

def loadingScreen():

loadCount = 0

running = True

while running:

for evt in event.get():

if evt.type == QUIT:

quit()

raise SystemExit

keys = key.get\_pressed()

screen.blit(load\_screen, (0, 0))

if keys[K\_RETURN] or keys[K\_SPACE]:

running = False

loadCount += 1

if loadCount >= 120:

running = False

display.flip()

fps.tick(60)

return None

loadingScreen()

#Main game loop

while mode != "exit":

if mode == "Splash Screen":

mode = splashScreen()

elif mode == "Play":

mode = main()

elif mode == "Pause":

mode = pauseScreen()

elif mode == "Credits":

mode = creditsMenu()

elif mode == "Options":

mode = optionsScreen()

elif mode == "Options\_Customize":

mode = customizeScreen()

elif mode == "After":

mode = afterScreen()

#Writing the new high score to file if it is higher than the current high score

if highscore > oldHS:

highFile = open("Untitled.txt", "w")

highFile.write(str(highscore))

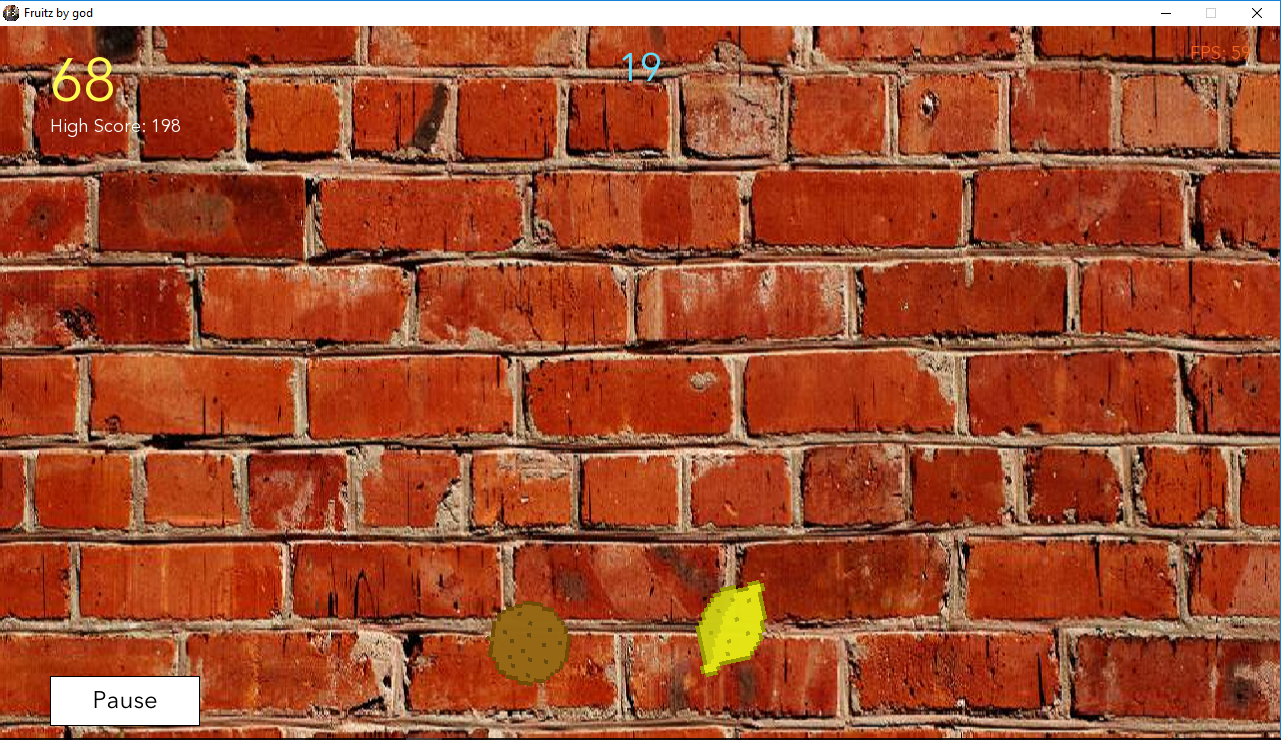
font.quit()

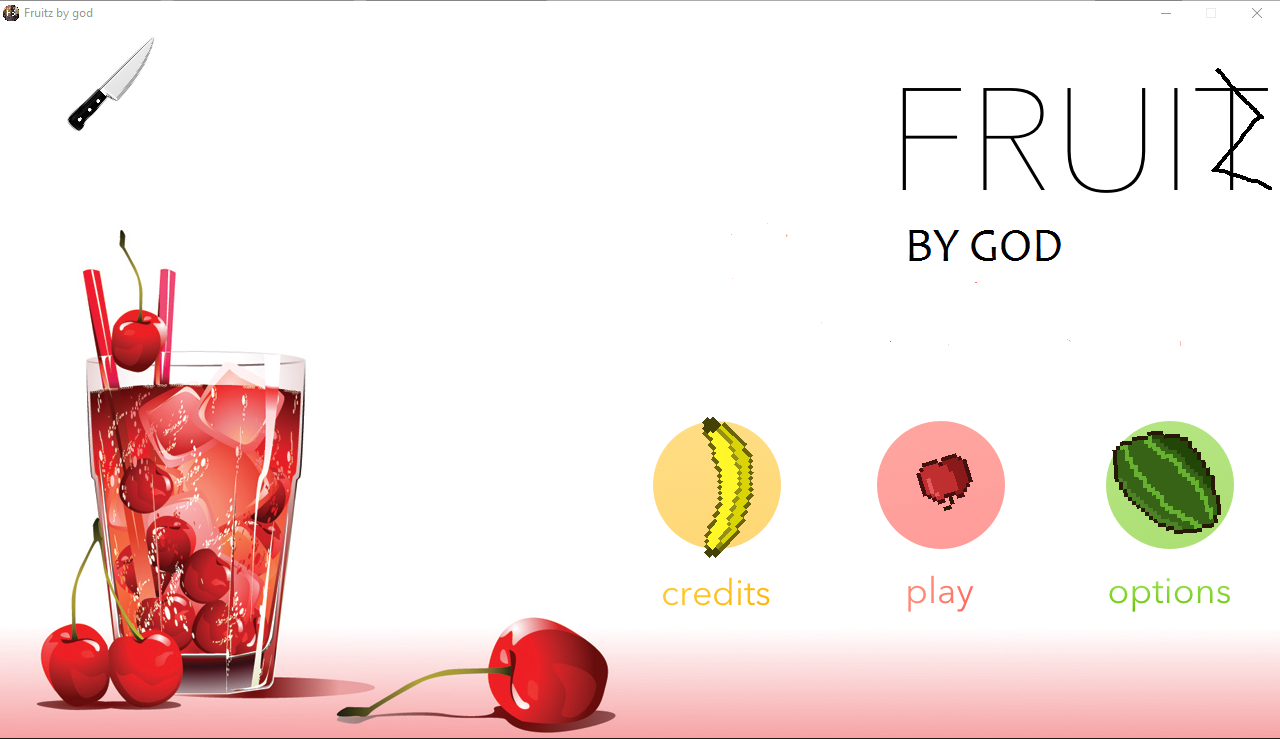
mixer.quit()

quit()

**SCREENSHOTS**

****

****

****

**BIBLOGRAPHY**

http://www.pygame.org/docs/ref/surface.html

<https://en.wikipedia.org/wiki/Fruit_Ninja>

http://thepythongamebook.com/en:pygame:step004

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