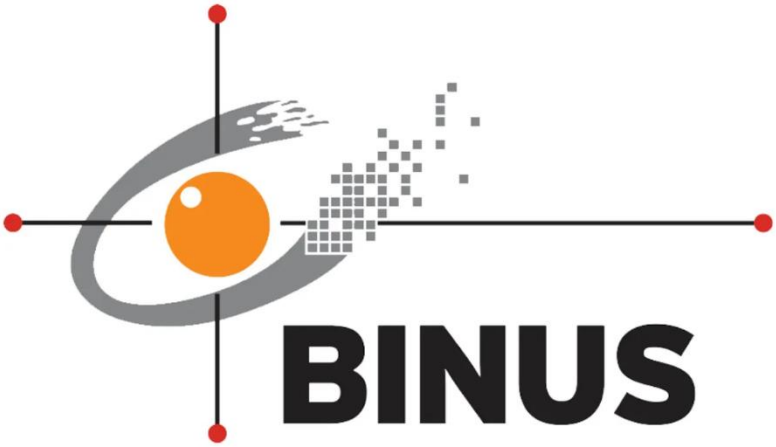
Program Design Methods and Intro to Programming Python Final Project Report



BINUS UNIVERSITY INTERNATIONAL

By:

Michael Christopher

2440047362

Class L1BC

Lecturer:

Ida Bagus Kerthyayana

D5757

Project Specification

What's the project?

In this project, I made a simple game called, "Survival Tile Breaker". It is a game which the player is given with a task to smash wall of bricks by deflecting a ball with the platform in the bottom of the screen. This game is highly inspired by a game called "Brick Breaker" released in 1999 for the BlackBerry phone.

Purpose of the project

In this project, I create a game in Python for the final project and since it is a game, this project purpose is to provide entertainment for the users. Also, this project allows me to learn more and challenge myself about Python in general and how to implement it in various scenarios and problem solving.

What's the vision and mission for this project?

To make a functional simple program that a lot of people will be able to use to entertain themselves. Since this game has a scoring system, the program needs to know how the scoring system work and how to save the high score system.

Solution Design

1. Introduction

As a freshmen in BINUS University International with Computer Science as its major, the Python programming language is the first language to be taught to us as the students of Computer Science. We were taught by our lecturer from the most basic stuff like data types, variables, loops and move to the harder one like functions, Python as an OOP, and classes. After approximately 2 months of learning Python, we were given with a final task to make a Python program of their own ideas with some requirements. Students are free to make anything and use any libraries and module in their program and finally, they need to make a presentation and demonstrate their program to the lecturer in command.

This final project task has been announced since the beginning of our university life so we have a lot of times to think what to make and create the program as we learn through lecture after lecture. I didn't think that much about this project until the end of November and I don't know what to make until the first week of December. At that time, I am still not certain whether I want to make a game or a website because initially I want to make a game but some people said that at that current time, one of the library for making game in Python, PyGame is not that good because it is slow and a little bit outdated. Until I came found the new Python Arcade library where it is a newer and faster library. After I decided to make a game, now it is time for me to decide what kind of game to make and out of the blue I remember this one particular game called "BrickBreaker" in my old BlackBerry phone and I think maybe it is fun to recreate this game in Python and hence "Survival Tile Breaker" game was born.

I officially started doing this project somewhere in the middle of December 2020 and I use Visual Studio Code as my IDE for this project. This project will be uploaded in my GitHub account, https://github.com/EuphosiouX/PDM-FinalProject.

2. Overview

Image 1 - “Survival Tile Breaker” main menu logo

(Logo.png)

“Survival Tile Breaker” is a game here the aim of the game is to destroy the tile by bouncing the ball with a platform in the bottom of the screen. It is designed to be played by people in various age range where the can test their skill and challenge themselves in this game aiming for the highest score. It is also a proper game that provide entertainment.

This game is created in Python 3. 8. 5 3-bit and it uses Arcade 2. 4. 3 external library and modules like random, os, and re.

Image 2 – Python logo

(https://www.cleanpng.com/png-python-programming-language-computer-programming-c-6370006/)

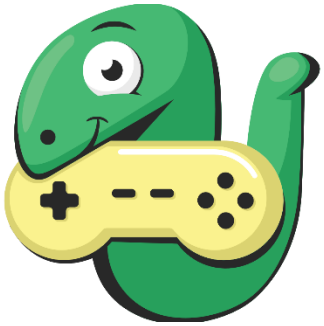


Image 3 – Arcade Library logo

(https://arcade.academy/_images/arcade-logo.svg)

3. Code Flow

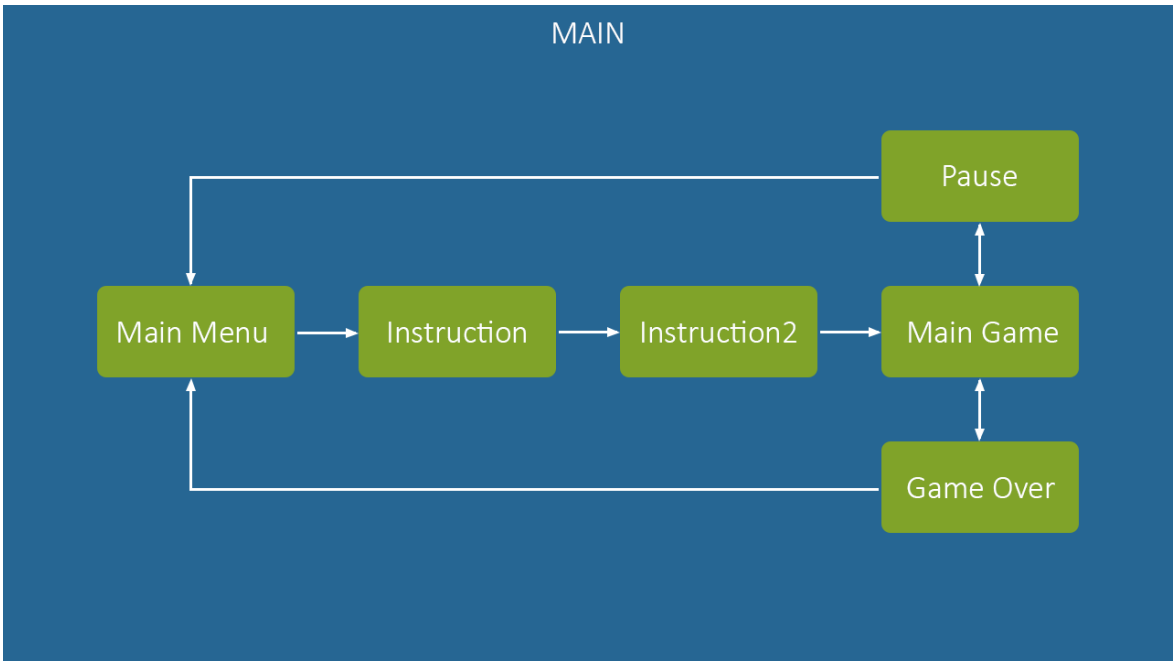


Image 4 – Code flowchart

(CodeFlow.png)

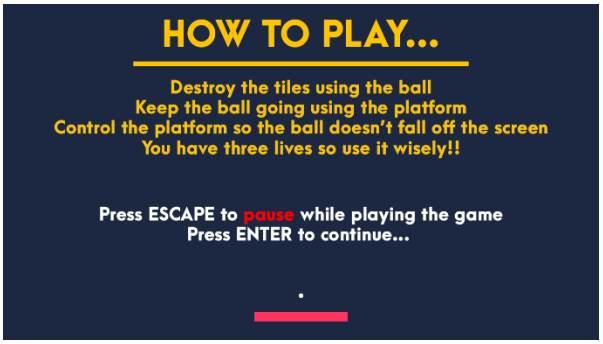
The codes for this program are all inside the main.py file where there a lot of classes inside like MainMenu class, Instruction class, and so on. Each view window has its own class and the way to go to the next view window is by calling the next class.

4. Visual Design

The design for this game is simplistic and minimalistic looking for each view window such as, solid background color, simple tile, platform, and ball design, and minimalistic looking main menu title that could also be considered as a logo.

Image 5 – Main menu preview

(MainMenu.png)



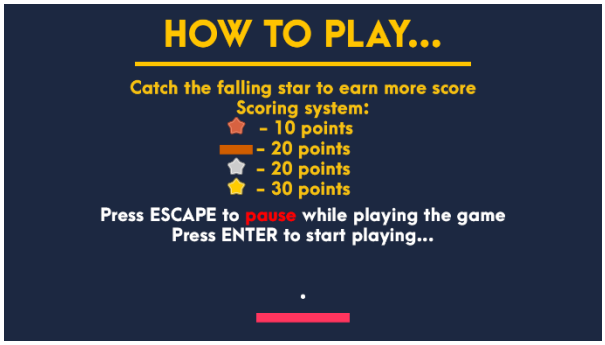


Image 6,7 – Instructions preview

(Instruction.png, Instruction2.png)

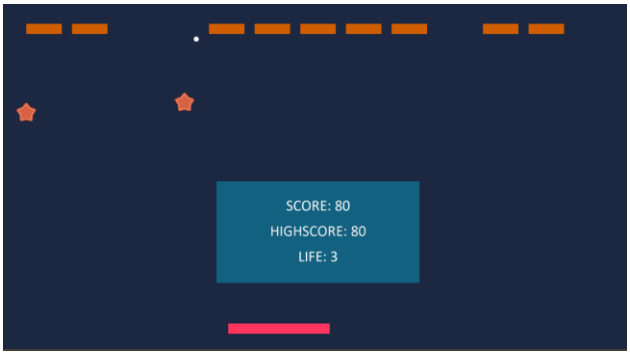


Image 8 – Game preview

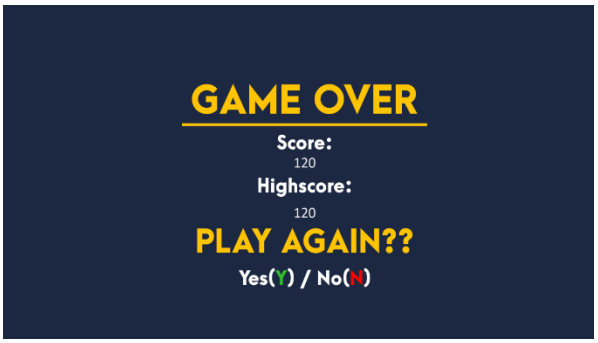


Image 9,10 – pause and game over preview

(Pause.png, GameOver.png)

Implementation

1. Main Setup

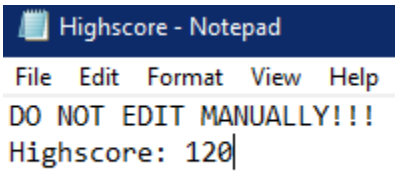
```python
# Required module and library in this code
import arcade
import random
import os
import re

# Setup the app window's size and name
SCREEN_WIDTH = 1280
SCREEN_HEIGHT = 720
SCREEN_TITLE = 'Survival Tile Breaker'

# Variable for the player sprite movement speed
PLAYER_SPEED = 7
```

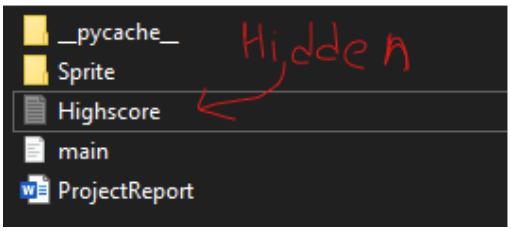
```python
def main():
    """ Main method """
    # Check if Highscore.txt exist or not, create one if it does not
    if os.path.exists("Highscore.txt"):
        # Setup the window attributes and run the program
        window = arcade.Window(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)
        menu = MainMenu()
        window.show_view(menu)
        arcade.run()
    else:
        f = file_opener(mode='w')
        f.write('DO NOT EDIT MANUALLY!!!\nHighscore: 0') # Write the Highscore.txt
        os.system( "attrib +h Highscore.txt") # Makes the file hidden so it is harder t
        print('FIRST LAUNCH SETUP SUCCESSFUL, RUN AGAIN THE PROGRAM TO START PLAYING')

if __name__ == "__main__":
    ''' Call main method'''
    main()
```

First, I made the main setup variables where I set the window's size and the name of the game also the player speed variable that will be used later in the game class. I create main() function to set up the window attributes by calling arcade.Window built in class, show the main menu with window.show_view(menu), and finally run the program.

2. File Opener

```python
def file_opener(file='Highscore.txt', mode='r'):
    '''
    Functions to handle .txt file
    Takes 2 argument (the_file_name, file_handling_mode)
    '''
    f = open(file, mode)
    if mode == 'r': # Return every integers found in the .txt file
        high_score_list = re.findall(r'\d+', f.read())
        high_score_string = "".join(high_score_list)
        high_score = int(high_score_string)
        return high_score
    else:
        return f # Return f itself
```





I created this function to handle a file called “highscore.txt” and the player’s high score is saved in this file. Back to the main() function, there is an if and else there and it is to check whether the player have played this game before or it is their first time playing . So this if checks if “Highscore.txt” exist or not in the player’s folder, if not (which every new player doesn’t have), it will run the program, notice that there is no such file, create the file, and print a sentence in the terminal to tell that the setup is successful and run the game again. After that, I realized that by saving the player’s high score data in a .txt file, it is not save since the player can easily access it to change the high score or even delete the file and that’s why I add os.system(“attrib +h highscore.txt) to make it hidden so it is harder to access manually (It is not hidden in the screenshot because I show all hidden files). Run the program again and now it checks if there is such file yes, so it runs the game this time.

3. Main Menu Class

```
class MainMenu(arcade.View):
    '''
    Class to create the main menu window
    Derived from built in arcade class "View"
    '''
    def on_draw(self):
        '''
        Draw the main menu layout
        '''
        # Start the render
        arcade.start_render()

        # Variable for highscore
        high_score = file_opener()

        arcade.draw_lrwh_rectangle_textured(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT, arcade.load_texture('Sprite/MainMenu.png'))
        # Show the highscore in the main menu
        arcade.draw_text('{}'.format(high_score),
                         SCREEN_WIDTH/2,
                         SCREEN_HEIGHT/4,
                         arcade.color.WHITE,
                         font_size=24,
                         anchor_x='center',
                         anchor_y='center')

    def on_key_press(self, key, key_modifier):
        """
        Called if keys in keyboard are pressed.
        """
        # Go to instruction if ENTER key is pressed
        if key == arcade.key.ENTER:
            instruction = Instruction()
            self.window.show_view(instruction)
```

This is the MainMenu class derived from built in arcade class “View”. This class purpose is to show the main menu in the window (Refer to Image 5). Arcade.draw_lrwh_rectangle_textured is to draw the whole main menu window texture and arcade.draw_text is to show the high score in the main menu. Also, I forgot to mention this before but this game only has keyboard control and no mouse control. So, I add on_key_press() to call if the the key is pressed and call the next class if it is pressed.

4. Instruction Class

```
class Instruction(arcade.View):
    '''
    Class to create the instruction window
    Derived from built in arcade class "View"
    '''
    def on_draw(self):
        '''
        Draw the instruction layout
        '''
        # Start the render
        arcade.start_render()

        arcade.draw_lrwh_rectangle_textured(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT, arcade.load_texture('Sprite/Instruction.png'))

    def on_key_press(self, key, key_modifier):
        """
        Called if keys in keyboard are pressed.
        """
        # Go to instruction2 if ENTER key is pressed
        if key == arcade.key.ENTER:
            instruction2 = Instruction2()
            self.window.show_view(instruction2)
```

```
class Instruction2(arcade.View):
    '''
    Class to create the instruction2 window
    Derived from built in arcade class "View"
    '''
    def on_draw(self):
        '''
        Draw the main menu layout
        '''
        # Start the render
        arcade.start_render()

        arcade.draw_lrwh_rectangle_textured(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT, arcade.load_texture('Sprite/Instruction2.png'))

    def on_key_press(self, key, key_modifier):
        """
        Called if keys in keyboard are pressed.
        """
        # Go to the game if ENTER is pressed
        if key == arcade.key.ENTER:
            game = Game()
            self.window.show_view(game)
```

These are the Instruction class also derived from built in arcade class “View”. This class is to show the instruction page of how to play the game and it is consist of 2 pages (Refer to Image 6,7). On_key_pressed() is also added in these classes.

5. Falling Star Sprite Class

```
class FallingStar(arcade.Sprite):
    '''
    Class to assign the stars sprite behaviour
    Derived from built in arcade class "Sprite"
    '''
    def update(self):
        '''
        Logic of the stars behaviour
        '''
        self.center_y -=2
```

This FallingStar class is to give behaviour to the star sprite later on and the behaviour is on update, move the y-axis position of the star to 2 pixel to the bottom. This class later on together with the Game class will create the star falling from the ceiling animation.

6. Game Class

This class has the longest line of code so it will be explained part by part.

a) Instantiation

```
class Game(arcade.View):
    """
    Main application class
    Class to create the instruction2 window
    Derived from built in arcade class "View"
    """
    def __init__(self):
        '''
        Instantiation
        '''
        super().__init__()
        """
        Setup the game variables
        Setup every sprites and sprite lists
        """
        # Create sprite list
        self.player_list = arcade.SpriteList()
        self.ball_list = arcade.SpriteList()
        self.tiles_list = arcade.SpriteList()
        self.bottom_line_list = arcade.SpriteList()
        self.gold_star_list = arcade.SpriteList()
        self.silver_star_list = arcade.SpriteList()
        self.bronze_star_list = arcade.SpriteList()

        # Create arrow key variable
        self.right = False
        self.left = False
        self.up = False
        self.down = False
```

```
        # Create ball sprite
        self.ball = arcade.Sprite('Sprite/Ball.png')
        self.ball.center_x = SCREEN_WIDTH/2
        self.ball.center_y = 100
        self.ball.change_x = 5
        self.ball.change_y = 5
        self.ball_list.append(self.ball)

        # Create player sprite
        self.player = arcade.Sprite('Sprite/Player.png')
        self.player.center_x = SCREEN_WIDTH/2
        self.player.center_y = 50
        self.player_list.append(self.player)

        # Create bottom line sprite
        self.bottom_line = arcade.Sprite('Sprite/BottomLine.png')
        self.bottom_line.center_x = SCREEN_WIDTH/2
        self.bottom_line.center_y = 5
        self.bottom_line_list.append(self.bottom_line)

        # Other variable
        self.level = 1
        self.score = 0
        self.life = 3
        self.hit_counter = 0

        # Set the game to level 1
        self.level_1()

        # variable for highscore
        self.high_score = file_opener()
```

The first thing to do in the game class is to instantiate the all the attributes that are needed such as, setup the sprite object and characteristic, create the sprite list, create the arrow keys variable, create other variables like score, high score, and life, and set the game to level one.

b) Star Function

```
def gold_star_setup(self):
    '''
    Function to setup the gold star sprite
    '''
    self.gold_star = FallingStar('Sprite/StarGold.png')
    self.gold_star.center_x = random.randrange(SCREEN_WIDTH)
    self.gold_star.center_y = SCREEN_HEIGHT + 30
    self.gold_star_list.append(self.gold_star)

def silver_star_setup(self):
    '''
    Function to setup the silver star sprite
    '''
    self.silver_star = FallingStar('Sprite/StarSilver.png')
    self.silver_star.center_x = random.randrange(SCREEN_WIDTH)
    self.silver_star.center_y = SCREEN_HEIGHT + 30
    self.silver_star_list.append(self.silver_star)
```

```
def bronze_star_setup(self):
    '''
    Function to setup the bronze star sprite
    '''
    self.bronze_star = FallingStar('Sprite/StarBronze.png')
    self.bronze_star.center_x = random.randrange(SCREEN_WIDTH)
    self.bronze_star.center_y = SCREEN_HEIGHT + 30
    self.bronze_star_list.append(self.bronze_star)
```

These star function is to setup the star sprite into the game class so the star positioned outside the window in the ceiling an the x-axis position is randomized by using random.randrange so where the star is spawned is randomized.

c) Star Hit Player Function

```
def star_hit_player(self, star_list, score):
    '''
    Function to create the scoring sytem if the stars are catched by the player
    Takes 3 arguments (self, the_star_list, the_score_of_each_star)
    '''
    star_hit_list = arcade.check_for_collision_with_list(self.player, star_list)
    for star in star_hit_list:
        star.remove_from_sprite_lists()
        self.score += score
```

This function is a function to check if the star hit the player below or not if yes, remove the star and add the score according to the amount of score of each star.

d) Star Off Screen Function

```
def star_off_screen(self, star_list):
    '''
    Function to remove star from the sprite list if it is failed to be catched by the player
    '''
    star_hit_list = arcade.check_for_collision_with_list(self.bottom_line, star_list)
    for star in star_hit_list:
        star.top = self.bottom_line.top
        star.remove_from_sprite_lists()
```

This function is a function to remove all the star that has been spawned but failed to be catched by the player. If this function is deleted, the star that miss the player will keep going down forever and stopped the spawn of the next stars.

e) Level Function

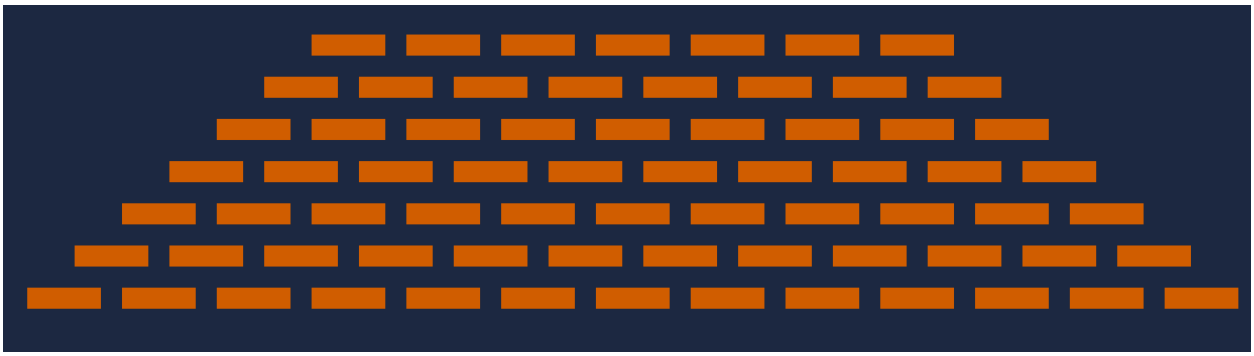
```
def level_1(self):
    '''
    Function to setup the tiles in level 1
    '''
    for x in range (100, SCREEN_WIDTH-90, 90):
        tiles = arcade.Sprite('Sprite/Tiles.png')
        tiles.center_x = x
        tiles.center_y = 640
        self.tiles_list.append(tiles)

def level_2(self):
    '''
    Function to setup the tiles in level 2
    '''
    for x in range(100, SCREEN_WIDTH-90, 90):
        for y in range(400, SCREEN_HEIGHT-40, 40):
            tiles = arcade.Sprite('Sprite/Tiles.png')
            tiles.center_x = x
            tiles.center_y = y
            self.tiles_list.append(tiles)
```

This function is to setup how each level looks like so it place every single tiles into the game window and so far there are total of 5 levels in this game. Below is how the tiles look like in each level.

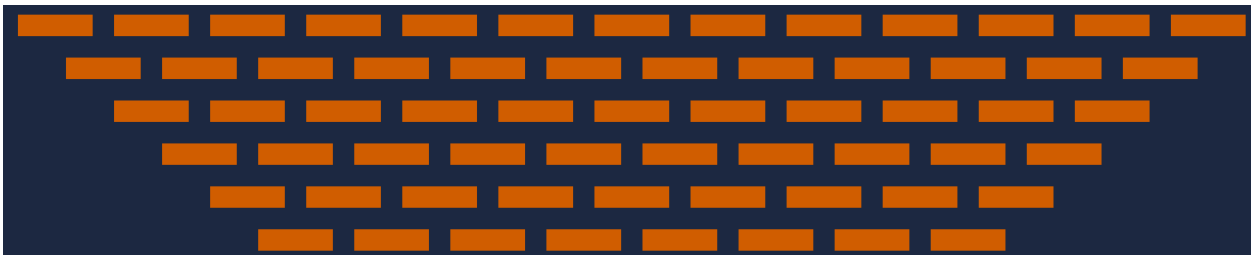
Level 1

Level 2

Level 3



Level 4



Level 5

f) On Show Function

```
def on_show(self):
    '''
    Set background color
    '''
    arcade.set_background_color(arcade.color.YANKEES_BLUE)
```

Function to give set the background color.

g) On Draw function

```
def on_draw(self):
    """
    Draw the game layout
    """
    # Start the render
    arcade.start_render()

    # Draw score box
    arcade.draw_rectangle_filled(SCREEN_WIDTH/2,
                                 SCREEN_HEIGHT/3,
                                 400,
                                 200,
                                 arcade.color.BLUE_SAPPHIRE)

    # Draw score text
    arcade.draw_text('SCORE: {}'.format(self.score),
                     SCREEN_WIDTH/2,
                     SCREEN_HEIGHT/3 + 50,
                     arcade.color.WHITE,
                     font_size=24,
                     anchor_x='center',
                     anchor_y='center')

    # Draw highscore text
    arcade.draw_text('HIGHSCORE: {}'.format(self.high_score),
                     SCREEN_WIDTH/2,
                     SCREEN_HEIGHT/3,
                     arcade.color.WHITE,
                     font_size=24,
                     anchor_x='center',
                     anchor_y='center')
# Draw life text
arcade.draw_text('LIFE: {}'.format(self.life),
                 SCREEN_WIDTH/2,
                 SCREEN_HEIGHT/3 - 50,
                 arcade.color.WHITE,
                 font_size=24,
                 anchor_x='center',
                 anchor_y='center')
```

Function to show the score, high score, and life remaining in the game window.

h) On update

This function update the game every 1/60 second so all of the movement logic is inside this function.

```
def on_update(self, delta_time):
    """
    All of the movement logic
    """
    # Update the sprite list
    self.gold_star_list.update()
    self.silver_star_list.update()
    self.bronze_star_list.update()
```

Update the star list (it updates the Falling Star Sprite Class which is previously mentioned in number 5).

```
# Makes the ball bounce off if it hit the player
if arcade.check_for_collision(self.ball, self.player):
    self.ball.change_y *= -1
    self.hit_counter += 1
```

Check the ball if it hits the player. Change the y direction of the ball if it hits and add 1 to hit counter

```
# change the ball x movement if it hits the left/right side of the tiles
self.ball.center_x += self.ball.change_x
tiles_hit_list = arcade.check_for_collision_with_list(self.ball, self.tiles_list)
for tiles in tiles_hit_list:
    if self.ball.change_x > 0:
        self.ball.right = tiles.left
    elif self.ball.change_x < 0:
        self.ball.left = tiles.right
    tiles.remove_from_sprite_lists()
    self.ball.change_x *= -1
    self.hit_counter += 1
    self.score += 20
```

Make the ball moves in x position and check if the ball hits the tiles. If the ball hit either the left/right side of the tiles, remove the tiles, add hit counter by 1, and add 20 to score.

```
# change the ball y movement if it hits the top/bottom side of the tiles
self.ball.center_y += self.ball.change_y
tiles_hit_list = arcade.check_for_collision_with_list(self.ball, self.tiles_list)
for tiles in tiles_hit_list:
    if self.ball.change_y > 0:
        self.ball.top = tiles.bottom
    elif self.ball.change_y < 0:
        self.ball.bottom = tiles.top
    tiles.remove_from_sprite_lists()
    self.ball.change_y *= -1
    self.hit_counter += 1
    self.score += 20
```

Make the ball moves in y position and check if the ball hits the tiles. If the ball hit either the top/bottom side of the tiles, remove the tiles, add hit counter by 1, and add 20 to score.

```
# Call star_hit_player function
self.star_hit_player(self.gold_star_list, 30)
self.star_hit_player(self.silver_star_list, 20)
self.star_hit_player(self.bronze_star_list, 10)

# Call star_off_screen function
self.star_off_screen(self.gold_star_list)
self.star_off_screen(self.silver_star_list)
self.star_off_screen(self.bronze_star_list)
```

Call the Star Hit Player function (Mentioned in number 6c) for each star with their own scoring system. Then, it calls the Star Off Screen System (Mentioned in number 6d) for each star.

```
# Go to next levels if every tiles in the current level are already destroyed
if len(self.tiles_list) == 0 and self.level == 1:
    self.ball.center_x = self.player.center_x
    self.ball.center_y = 100
    self.level += 1
    self.level_2()
if len(self.tiles_list) == 0 and self.level == 2:
    self.ball.center_x = self.player.center_x
    self.ball.center_y = 100
    self.level += 1
    self.level_3()
if len(self.tiles_list) == 0 and self.level == 3:
    self.ball.center_x = self.player.center_x
    self.ball.center_y = 100
    self.level += 1
    self.level_4()
if len(self.tiles_list) == 0 and self.level == 4:
    self.ball.center_x = self.player.center_x
    self.ball.center_y = 100
    self.level += 1
    self.level_5()
```

Logic for checking if the tiles in the current level are all completely destroyed if yes, reset the ball position, add 1 to level and move on to the next level.

```
# Keep the ball bouncing from the sides and top
if self.ball.center_x < 5 or self.ball.center_x > SCREEN_WIDTH - 5:
    self.ball.change_x *= -1
    self.hit_counter += 1
if self.ball.center_y > SCREEN_HEIGHT - 5:
    self.ball.change_y *= -1
    self.hit_counter += 1
```

Keep the ball bouncing if it hits the sides and top of the screen.

```
# Reset and substract the life if the player failed to keep the ball bouncing
if arcade.check_for_collision(self.ball, self.bottom_line):
    self.ball.center_x = self.player.center_x
    self.ball.center_y = 100
    self.life -= 1
```

Check if the ball hit the bottom of the screen or no if yes, rest the ball position and substract 1 from life.

```
# Game over when life reach zero or final level are finished
game_over = GameOver(self)
if len(self.tiles_list) == 0 and self.level == 5:
    self.window.show_view(game_over)
if self.life < 1:
    self.window.show_view(game_over)
```

Call the Game Over Class (Mentioned in number 7) if the player finished the last level or the player life reach 0.

```
# Move the player if key pressed and keep the player onscreen
if self.right:
    self.player.center_x += PLAYER_SPEED
if self.left:
    self.player.center_x -= PLAYER_SPEED
if self.player.center_x < 100:
    self.left = False
if self.player.center_x > SCREEN_WIDTH - 100:
    self.right = False
```

Logic for the player movement and to keep the player onscreen

```
# Spawn random star from the ceiling every 10 bounces
# 20 bounces is for safety measures because sometimes it counts up to more than 10
if self.hit_counter == 10 or self.hit_counter == 20:
    show_random_star = random.randrange(1, 4)
    if show_random_star == 1:
        self.gold_star_setup()
    if show_random_star == 2:
        self.silver_star_setup()
    if show_random_star == 3:
        self.bronze_star_setup()
    self.hit_counter = 0
```

Logic to spawn random stars by calling star function (Mentioned in number 6b) everytime the ball reach 10 bounce.

```
# Rewrite the Highscore.txt with current score if the current score is higher than the current highscore
f = file_opener(mode='r+')
if self.score > self.high_score:
    self.high_score = self.score
    f.write('DO NOT EDIT MANUALLY!!!\nHighscore: {}'.format(self.high_score))
```

Check if current score is higher that high score or not if yes, change the high score to score.

```
def on_key_press(self, key, key_modifier):
    """
    Called if keys in keyboard are pressed.
    """
    # Set key RIGHT/LEFT pressed to true
    if key == arcade.key.RIGHT:
        self.right = True
    if key == arcade.key.LEFT:
        self.left = True
    # Go to pause if ESCAPE key is pressed
    if key == arcade.key.ESCAPE:
        self.right = False
        self.left = False
        pause = Pause(self)
        self.window.show_view(pause)

def on_key_release(self, key, key_modifier):
    """
    Called whenever the user lets off a previously pressed key.
    """
    # Set key pressed back to false
    if key == arcade.key.RIGHT:
        self.right = False
    if key == arcade.key.LEFT:
        self.left = False
```

To check if there are any key pressed or no and add the logic for each key pressed. It also define what happen if some of the pressed key are released.

7. Pause and Game Over Class

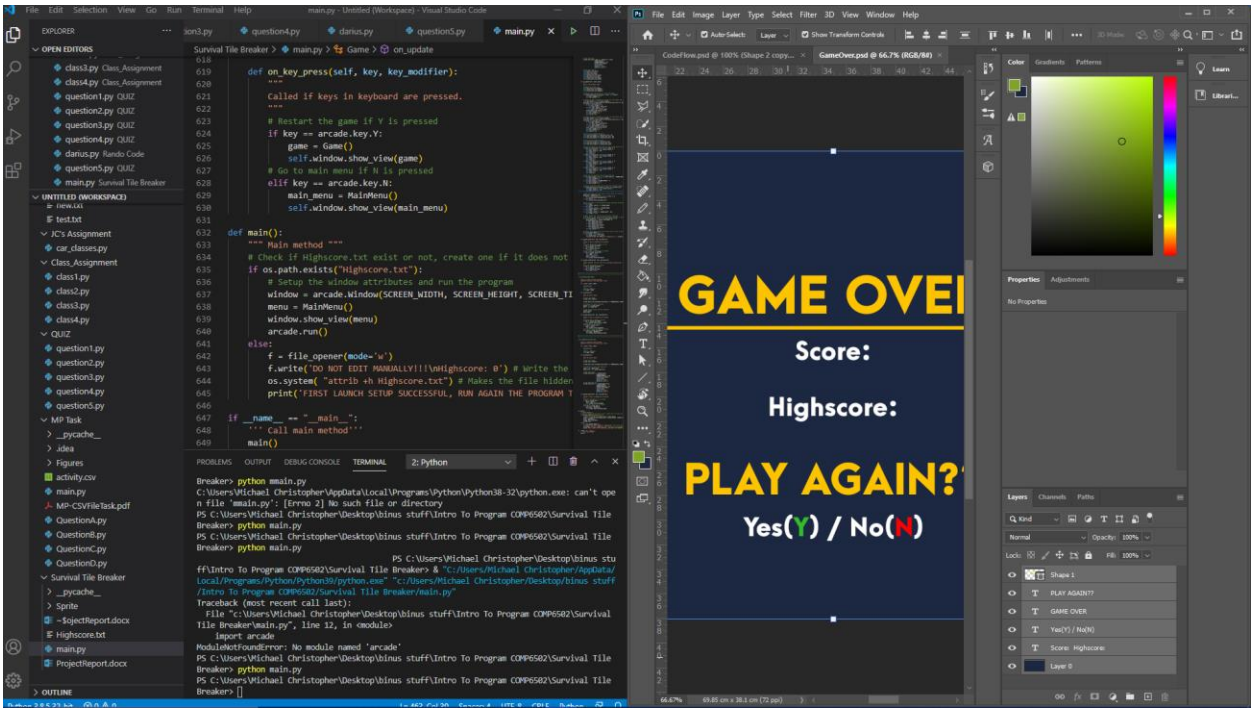
```
class Pause(arcade.View):
    '''
    Class to create the pause window
    Derived from built in arcade class "View"
    '''
    def __init__(self, game):
        '''
        Instantiation
        '''
        super().__init__()
        self.game = game

    def on_draw(self):
        '''
        Draw the pause layout
        '''
        # Start the render
        arcade.start_render()

        arcade.draw_lrwh_rectangle_textured(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT, arcade.load_texture('Sprite/Pause.png'))

        # Variables for player and ball
        player = self.game.player
        ball = self.game.ball

        # Draw current palyer and ball position
        player.draw()
        ball.draw()
```

```
class GameOver(arcade.View):
    '''
    Class to create the pause window
    Derived from built in arcade class "View"
    '''
    def __init__(self, game):
        '''
        Instantiation
        '''
        super().__init__()
        self.game = game

    def on_draw(self):
        '''
        Draw the pause layout
        '''
        # Start the render
        arcade.start_render()

        arcade.draw_lrwh_rectangle_textured(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT, arcade.load_texture('Sprite/GameOver.png'))

        # Variables for score and highscore
        score = self.game.score
        high_score = self.game.high_score

        # Draw score text
        arcade.draw_text('{}'.format(score),
                         SCREEN_WIDTH/2,
                         SCREEN_HEIGHT/3 + 130,
                         arcade.color.WHITE,
```

These 2 class are similar to the previous MainMenu class and Instruction class where it shows the window of pause menu and game over menu (Refer to Image – 9,10). The only difference is both of this class have instantiation.

Evidence Of working Program

Every screenshot above is taken from my IDE and taken by myself. All of the sprite used are created by me using photoshop (Except the star sprite).



Summary and Self Reflection

When I started doing this project, I thought that this is incredibly hard but after learning the libraries from the website, I am finally able to understand how it works and able to implement them in my own program. Overall, this is a really challenging project but in the end, it is pretty fun and this project allows me to know deeper about Python programming language.

Resource and Reference

Resource for star sprite:

https://kenney.nl/assets/tappy-plane

Reference for some of the code and where I learn Arcade library:

https://arcade.academy/examples/index.html

https://www.youtube.com/watch?v=2qP1M1Nf__w&list=PL1P11yPQAo7pPlDlFEaL3IUbcWnnPcALI