Faculty of Engineering & Applied Science

SOFE 3980U
Software Quality

Assignment 1
Report

Janajan Jeyabalan
100698148

Waterfall Software Process Model

The software process that I have chosen is the waterfall method which is a model that is plan-driven. One that focuses on the specification and development of separate and distinct phases. The phases in the waterfall model in chronological order are requirements analysis, system and software and design, implementation and unit testing, integration and system testing, and essentially operation and maintenance. The waterfall model is the process that I chose to work with when dealing with this project as it works well with the small python game that I developed as the requirements are very clear, defined and understood. In addition, this process works well with my project as it is easy to manage and I like incorporating the fact that in this model all phases are processed and completed one at a time.

1. Changes made to the code: Slowed down the speed of the ball to allow players to have a more competitive game against one another.
2. Changes made to the visual appearance: Made the visuals more clear, bright and visually appealing, to essentially have a more aesthetic look that is easily understood.

Requirements analysis and definition

Use Case 1: Ball Speed

Scenario: A user would want the ability to play the game at a speed in which both players do not lose the game due to the speed of the ball. Players will want to be able to have the ability to react in a quick manner, essentially slowing down the ball speed will help in making a more competitive gaming experience.

Actors: Player A, Player B, PongGame Developers

Acceptance Criteria: Players will be able to move paddles up and down with enough time to allow the ball to hit the paddle and proceed back to the other player's paddle. Essentially, the game becomes a skill of hand speed and eye coordination.

Use Case 2: Visual Appearance

Scenario: A user would want to play a game that is more aesthetically pleasing to the eye. A game that allows users to follow through from left to right as the game progresses and to be visually intrigued.

Actors: Player A, Player B, PongGame Developers

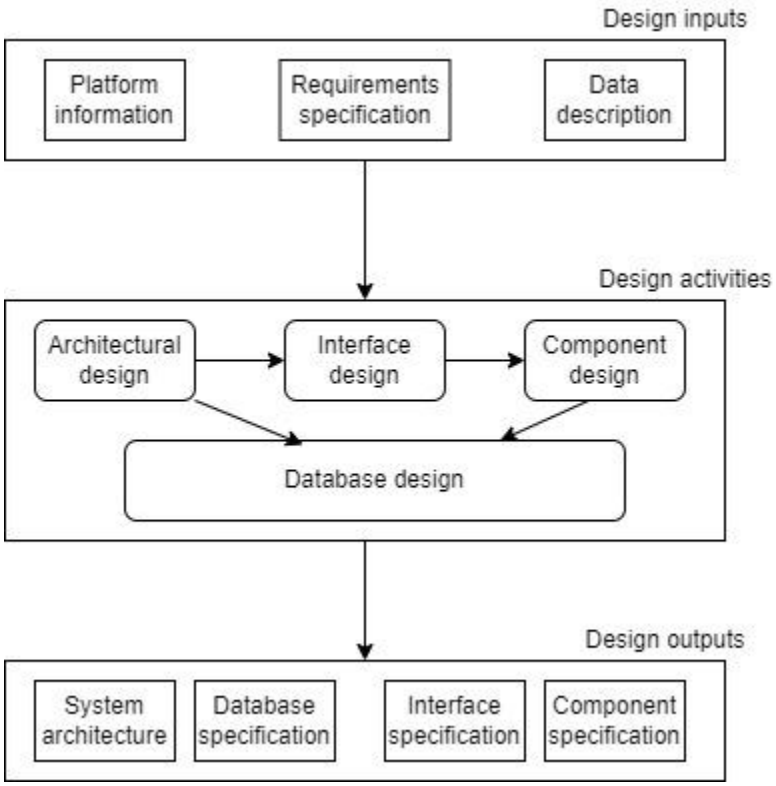
Acceptance Criteria: Players will be able to view a more appealing environment to play in as the colours will indicate each set of attributes in the game very clearly.

Feasibility Study:

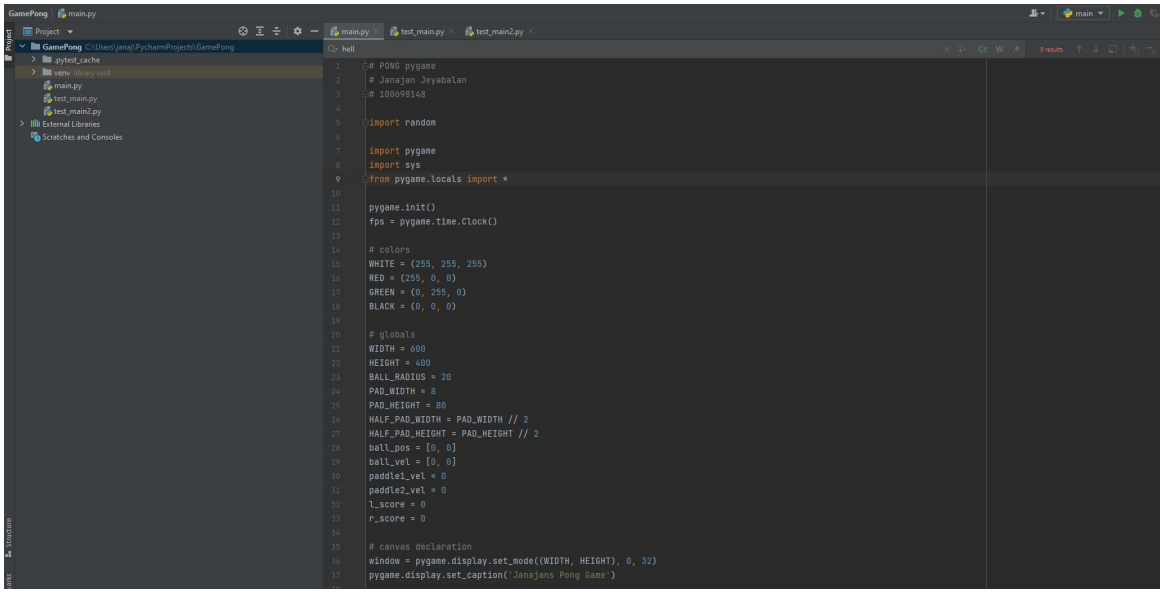
Economic: The organization can complete this application with the given budget.
Legal: The organization can handle this project under any law compliances.
Operation feasibility: The organization can create the operation that is expected by customers.
Technical: The current computer system can support the software.
Schedule: This project can be finished under the given schedule/due date.

System and software design

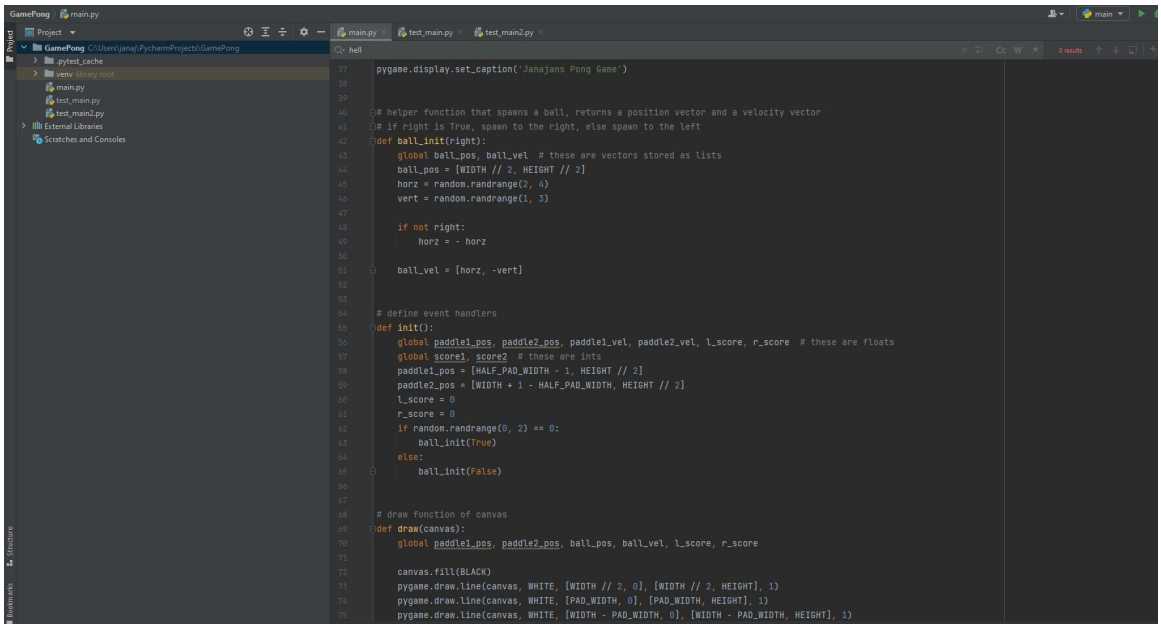
The programming language that will be used to design and develop the architecture of the project will be Python. The integrated development environment (IDE) that will be used for this project is gonna be visual studio code.



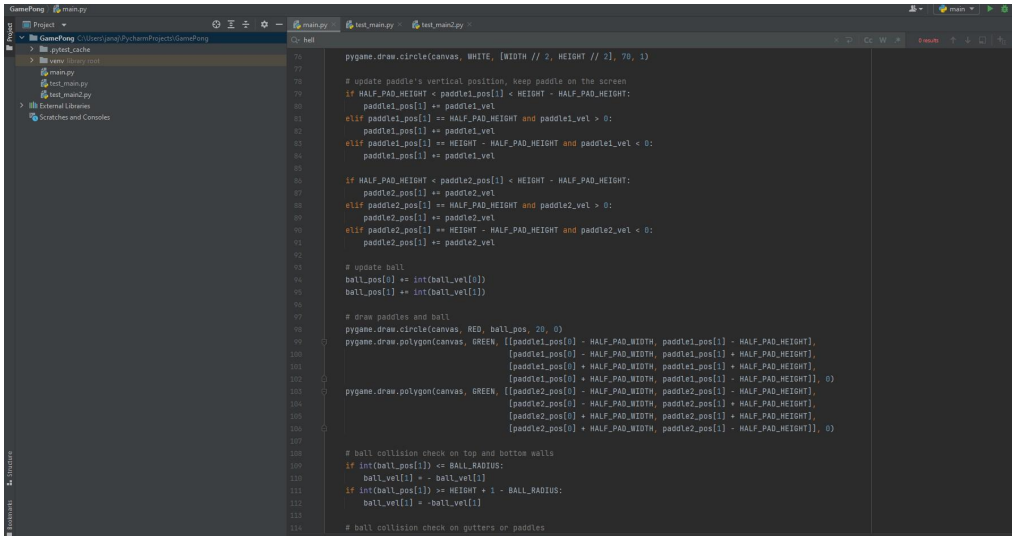
Code:



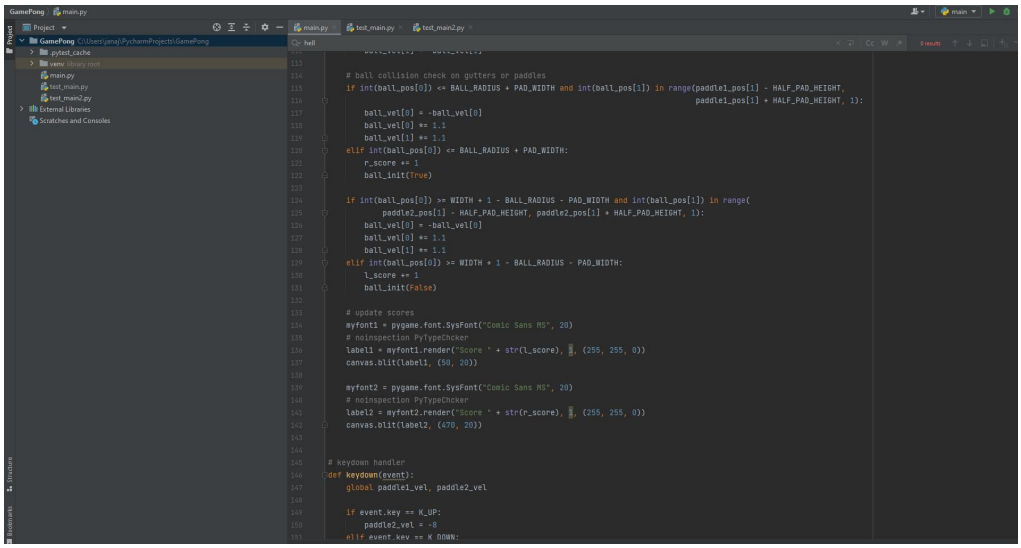
```
# PONG pygame
# Janajan Jeyabalan
# 100698148

import random

import pygame
import sys
from pygame.locals import *

pygame.init()
fps = pygame.time.Clock()

# colors
WHITE = (255, 255, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLACK = (0, 0, 0)

# globals
WIDTH = 600
HEIGHT = 400
BALL_RADIUS = 20
PAD_WIDTH = 8
PAD_HEIGHT = 80
HALF_PAD_WIDTH = PAD_WIDTH // 2
HALF_PAD_HEIGHT = PAD_HEIGHT // 2
ball_pos = [0, 0]
ball_vel = [0, 0]
paddle1_vel = 0
paddle2_vel = 0
l_score = 0
r_score = 0

# canvas declaration
window = pygame.display.set_mode((WIDTH, HEIGHT), 0, 32)
pygame.display.set_caption('Janajans Pong Game')
```



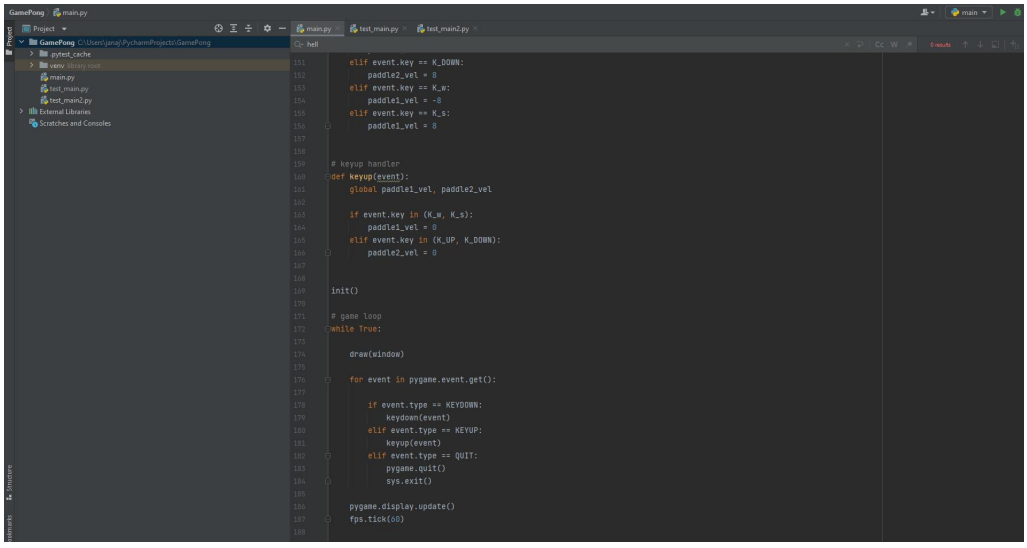
```
pygame.display.set_caption('Janajans Pong Game')


# helper function that spawns a ball, returns a position vector and a velocity vector
# if right is True, spawn to the right, else spawn to the left
def ball_init(right):
    global ball_pos, ball_vel  # these are vectors stored as lists
    ball_pos = [WIDTH // 2, HEIGHT // 2]
    horz = random.randrange(2, 4)
    vert = random.randrange(1, 3)

    if not right:
        horz = - horz

    ball_vel = [horz, -vert]


# define event handlers
def init():
    global paddle1_pos, paddle2_pos, paddle1_vel, paddle2_vel, l_score, r_score  # these are floats
    global score1, score2  # these are ints
    paddle1_pos = [HALF_PAD_WIDTH - 1, HEIGHT // 2]
    paddle2_pos = [WIDTH + 1 - HALF_PAD_WIDTH, HEIGHT // 2]
    l_score = 0
    r_score = 0
    if random.randrange(0, 2) == 0:
        ball_init(True)
    else:
        ball_init(False)


# draw function of canvas
def draw(canvas):
    global paddle1_pos, paddle2_pos, ball_pos, ball_vel, l_score, r_score

    canvas.fill(BLACK)
    pygame.draw.line(canvas, WHITE, [WIDTH // 2, 0], [WIDTH // 2, HEIGHT], 1)
    pygame.draw.line(canvas, WHITE, [PAD_WIDTH, 0], [PAD_WIDTH, HEIGHT], 1)
    pygame.draw.line(canvas, WHITE, [WIDTH - PAD_WIDTH, 0], [WIDTH - PAD_WIDTH, HEIGHT], 1)
```



```
    pygame.draw.circle(canvas, WHITE, [WIDTH // 2, HEIGHT // 2], 70, 1)

    # update paddle's vertical position, keep paddle on the screen
    if HALF_PAD_HEIGHT < paddle1_pos[1] < HEIGHT - HALF_PAD_HEIGHT:
        paddle1_pos[1] += paddle1_vel
    elif paddle1_pos[1] == HALF_PAD_HEIGHT and paddle1_vel > 0:
        paddle1_pos[1] += paddle1_vel
    elif paddle1_pos[1] == HEIGHT - HALF_PAD_HEIGHT and paddle1_vel < 0:
        paddle1_pos[1] += paddle1_vel

    if HALF_PAD_HEIGHT < paddle2_pos[1] < HEIGHT - HALF_PAD_HEIGHT:
        paddle2_pos[1] += paddle2_vel
    elif paddle2_pos[1] == HALF_PAD_HEIGHT and paddle2_vel > 0:
        paddle2_pos[1] += paddle2_vel
    elif paddle2_pos[1] == HEIGHT - HALF_PAD_HEIGHT and paddle2_vel < 0:
        paddle2_pos[1] += paddle2_vel

    # update ball
    ball_pos[0] += int(ball_vel[0])
    ball_pos[1] += int(ball_vel[1])

    # draw paddles and ball
    pygame.draw.circle(canvas, RED, ball_pos, 20, 0)
    pygame.draw.polygon(canvas, GREEN, [[paddle1_pos[0] - HALF_PAD_WIDTH, paddle1_pos[1] - HALF_PAD_HEIGHT],
                                        [paddle1_pos[0] - HALF_PAD_WIDTH, paddle1_pos[1] + HALF_PAD_HEIGHT],
                                        [paddle1_pos[0] + HALF_PAD_WIDTH, paddle1_pos[1] + HALF_PAD_HEIGHT],
                                        [paddle1_pos[0] + HALF_PAD_WIDTH, paddle1_pos[1] - HALF_PAD_HEIGHT]], 0)
    pygame.draw.polygon(canvas, GREEN, [[paddle2_pos[0] - HALF_PAD_WIDTH, paddle2_pos[1] - HALF_PAD_HEIGHT],
                                        [paddle2_pos[0] - HALF_PAD_WIDTH, paddle2_pos[1] + HALF_PAD_HEIGHT],
                                        [paddle2_pos[0] + HALF_PAD_WIDTH, paddle2_pos[1] + HALF_PAD_HEIGHT],
                                        [paddle2_pos[0] + HALF_PAD_WIDTH, paddle2_pos[1] - HALF_PAD_HEIGHT]], 0)

    # ball collision check on top and bottom walls
    if int(ball_pos[1]) <= BALL_RADIUS:
        ball_vel[1] = - ball_vel[1]
    if int(ball_pos[1]) >= HEIGHT + 1 - BALL_RADIUS:
        ball_vel[1] = -ball_vel[1]

    # ball collision check on gutters or paddles
```



```
    # ball collision check on gutters or paddles
    if int(ball_pos[0]) <= BALL_RADIUS + PAD_WIDTH and int(ball_pos[1]) in range(paddle1_pos[1] - HALF_PAD_HEIGHT,
                                                                                 paddle1_pos[1] + HALF_PAD_HEIGHT, 1):
        ball_vel[0] = -ball_vel[0]
        ball_vel[0] *= 1.1
        ball_vel[1] *= 1.1
    elif int(ball_pos[0]) <= BALL_RADIUS + PAD_WIDTH:
        r_score += 1
        ball_init(True)

    if int(ball_pos[0]) >= WIDTH + 1 - BALL_RADIUS - PAD_WIDTH and int(ball_pos[1]) in range(
            paddle2_pos[1] - HALF_PAD_HEIGHT, paddle2_pos[1] + HALF_PAD_HEIGHT, 1):
        ball_vel[0] = -ball_vel[0]
        ball_vel[0] *= 1.1
        ball_vel[1] *= 1.1
    elif int(ball_pos[0]) >= WIDTH + 1 - BALL_RADIUS - PAD_WIDTH:
        l_score += 1
        ball_init(False)

    # update scores
    myfont1 = pygame.font.SysFont("Comic Sans MS", 20)
    # noinspection PyTypeChecker
    label1 = myfont1.render("Score " + str(l_score), 1, (255, 255, 0))
    canvas.blit(label1, (50, 20))

    myfont2 = pygame.font.SysFont("Comic Sans MS", 20)
    # noinspection PyTypeChecker
    label2 = myfont2.render("Score " + str(r_score), 1, (255, 255, 0))
    canvas.blit(label2, (470, 20))


# keydown handler
def keydown(event):
    global paddle1_vel, paddle2_vel

    if event.key == K_UP:
        paddle2_vel = -8
    elif event.key == K_DOWN:
```



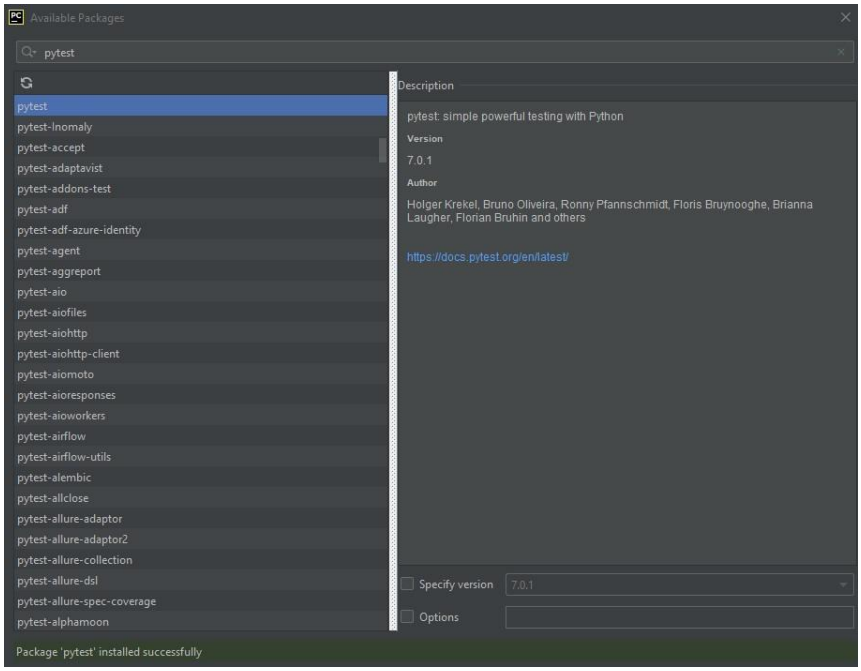
```
        paddle2_vel = 8
    elif event.key == K_w:
        paddle1_vel = -8
    elif event.key == K_s:
        paddle1_vel = 8


# keyup handler
def keyup(event):
    global paddle1_vel, paddle2_vel

    if event.key in (K_w, K_s):
        paddle1_vel = 0
    elif event.key in (K_UP, K_DOWN):
        paddle2_vel = 0


init()

# game loop
while True:

    draw(window)

    for event in pygame.event.get():

        if event.type == KEYDOWN:
            keydown(event)
        elif event.type == KEYUP:
            keyup(event)
        elif event.type == QUIT:
            pygame.quit()
            sys.exit()

    pygame.display.update()
    fps.tick(60)
```

Description of Functionalities:

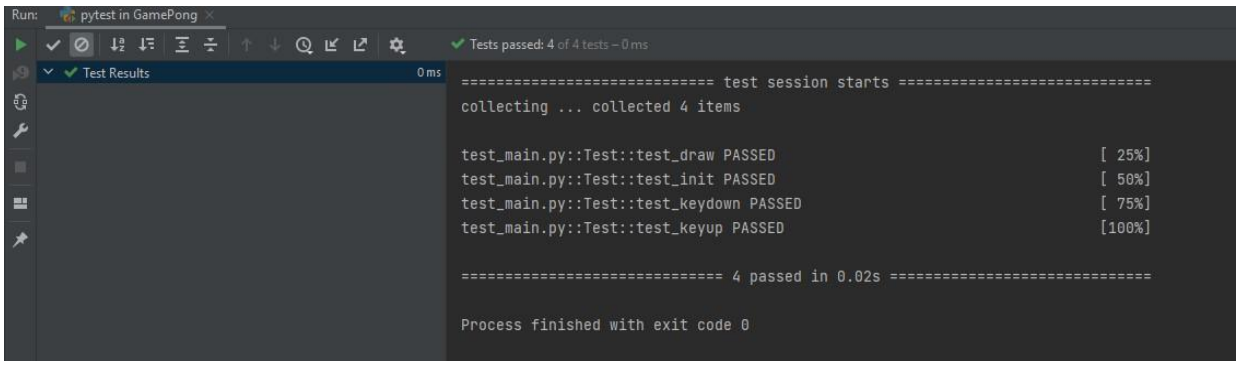
This game is called PongGame, it is a simple, old computer game that is played with two paddles and a ball. It is a game that can be played with two players, and the objective of each player is to control an in-game paddle by moving it up or down to strike the ball when it comes towards the side that they are playing on. Players score a point when the other player can not reach the ball or return it correctly.

Testing:

To test this python game: Pong, pytest was used on pyCharm. The first steps in testing involves downloading and installing packages, in the image below, pytest was downloaded successfully and the first step in starting to test the functions in the game were looking good.



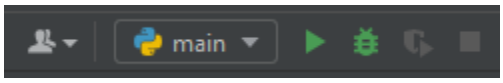
The tests were taken on the functions that defined the event handlers for the game, the draw function of canvas, and the functions that defined the keydown handler and keyup handler.



Essentially, all the tests ended up passing but that wasn't the case to start with. During the automation testing phase, there were a couple challenges that were faced. The first challenge that was faced was selecting the right tool as a lot of testing softwares do not work for python, and even from the ones that work they did not line well with pygame. After going through different testing phases, the testing tool that outshined the other was pytest and unit testing was done with it. Another challenge that was faced although was eventually solved was selecting a proper testing approach from testing with pytest and unit testing and ensuring that all required downloads and installations were equipped to give the best results possible. In essence, the challenges were overcome and all the test cases ended up passing.

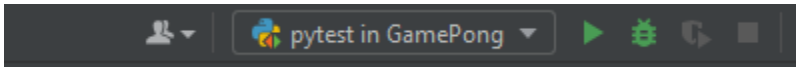
Operation and maintenance:

This software is a Pong Game that is played when you open up the main.py file. Make sure the configuration is set for the main file and then run the software and the game will run.



There will be a score at the top of the screen for both player 1 and player 2 and what I did was I made the wording say: "White Score" and "Green Score" which symbolized the colours of the paddles to make it clear for everyone playing the game to be aware of what the exact score is and how much they have.

To run the software testing using pytest, the pytest package has to be installed in the ide. I used PyCharm to run my game. To run the test, you have to make sure the configuration is set to pytest in GamePong. Then you test the test_main.py file.



References

[1] Pandian, (2017) [Classic Pong Game in Python - using pygame]. https://gist.github.com/vinothpandian/4337527.