Pong for Pi

CSE 321 Term Project
University at Buffalo
Christian Wilson and Stephanie Paquin

1.1 Problem Statement and Solution

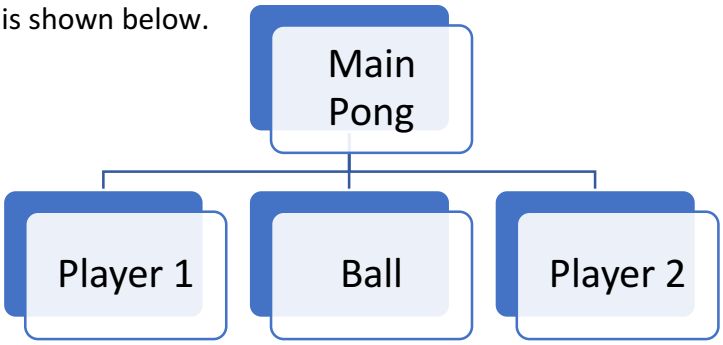
Everyone gets bored from time to time, but sometimes there is nothing you can do to entertain yourself? Imagine this, you and a colleague are sitting around the office and have nothing to do. You've already finished all your work, yet your boss won't sign off on either of you going home early. What can you do? One could read a book, or watch some silly video online, or maybe just chill at your desk with a cup of coffee, and for some people that may be enough. This isn't necessarily enough for everyone, especially for people who like to keep their minds busy and challenged. Office boredom is a real problem, so what can one do? With this in mind, we decided to create Pong for Pi. A classic game with the twist of no one needing to download it. Pong for Pi runs on an embedded system, a Raspberry Pi, that allows for ease of use for anyone who could want to play. This allows simple plug-and-play functionality that makes it easy for use anywhere you are.

1.2 Motivation

With both of us having experience with office boredom, we were both extremely motivated to come up with something as a solution. We had an interest in programming a stand-alone game that can be easily and quickly played by two people. When we started to brainstorm what game, we should make, ideas like the snake game and the impossible game were first on our minds. We quickly realized that snake lacks competition and the impossible game would be too frustrating for an office setting. Desiring to make a game that was both fun and had a competitive factor, we eventually settled on the classic game of Pong. In making Pong, we allow for two players instead of just one, and while it is not the most complicated of games, it is still fun! So, in conclusion, our goal for this project was to make a simple to run, simple to play, fun and competitive game for use in any setting in which a HDMI monitor and a keyboard can be found.

1.3 Design and Architecture

During the design process, we decided that we would accomplish this game through the use of three Python classes; one for player 1, one for player 2, and one for the ball. All of these classes would then be used in the main portion of our code to run the game. A diagram to show how each class is related is shown below.



This diagram shows that the main pong game is dependent on the player 1, player 2, and ball class. The ball class also uses the two instances of player 1 and player 2 that are made in the beginning, but the ball class isn't dependent on the classes themselves.

With this design, we were able to successfully make the classic game pong for two people to play. This program was put on a Raspberry Pi and is run through the terminal for ease of access and use. With this game being designed with Pi in mind, it is a versatile form of entertainment, since all you need to play is an HDMI screen, mouse, keyboard, and one simple command on how to start the game.

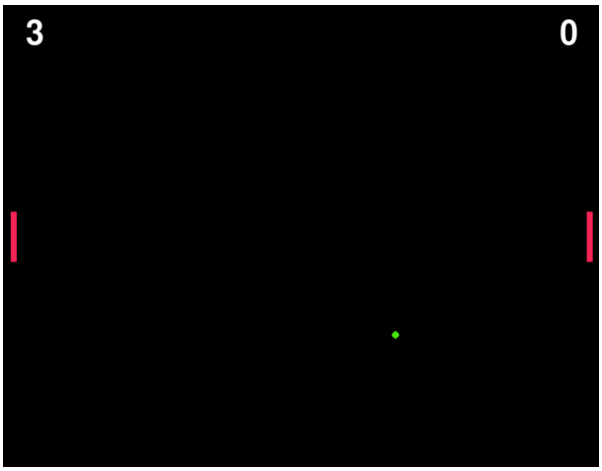
1.4 Technical Description

When we originally thought about how to make pong, our first thought was to use C++ since it is the language we are both most comfortable with. We found a library called allegro 4.4.2 to use, and we did make a fully functional pong in Microsoft's Visual Studio IDE, after it had been modified with the allegro library. This version of pong used a pong class and three files, pong.h, pong.cpp, and main.cpp. The challenge that neither if us intended was getting that game to successfully run on the Raspberry Pi. After several days of trying, we both decided that rewriting our pong game in Python was going to be the solution we needed to use. Since we already had a functional pong in C++, we were completely aware of what we needed to do and the logic we would use, we just needed to research how to accomplish this in Python.

This picture shows the initial startup of pong. Player 1 is the paddle on the left and Player 2 is the paddle on the right. Each players score is in their respective corners, and the keys that are needed to play are displayed on the bottom of the screen. Once space is pressed, the ball will appear and all components will be able to move.

Our search for an allegro for Python lead us to use pygame. Pygame has an abundance of documentation that we were able to read and familiarize ourselves with in order to create pong. With a basic form of what was needed to get an initial black window to pop up on startup, we then started to add the necessary pieces that we would need to build pong.

Each player class is initialized with several properties: an x and y location, a paddle height and width, and a score. The x and y location are necessary for us to move the paddle and also for collision checking with the ball. The move function of the player class first checks if the paddle is hitting the top or bottom border. If an attempt is made to move the paddle off the screen, which no one intentionally does yet happen often, then it is moved back to the amount of the border, 0 and the height of the window minus the height of the paddle as an offset. The second part of move is to check what key is pressed and then moves the paddle accordingly. Another function inside the player class is the draw function. This is done through the use of pygame.draw.rect and uses the players attributes to draw it. The last portion of the player class is the drawScore function. This pygame.font, font.render, and screen.blit to add the score in the upper corner of the players respective side. This then checks if the players score is 5, and if it is then a message is printed to the screen saying which player won and the keys to either restart or quit. There is logic for waiting until a key is pressed and functions called accordingly based on what key you pressed.



The ball class has an x and y location and an x and y move factor. Similar to the player classes, the ball class uses pygame.draw.circle to draw the ball to the screen. The move function of this class is different than the player classes in that the ball class needs to check for collisions with the walls and paddles. Right when move is called, the x move factor is added to the x location, and similar for the y factor and y location. Then move checks if the ball hit the top or bottom wall and changes the balls direction if it has by multiplying by -1. If the ball has hit the left wall, player two gets 1 added to their score, the ball is reinitialized, and begins moving towards player 1 first. If the ball has hit the right wall, player one gets 1 added to their score, the ball is reinitialized, and begins moving towards player 2 first. The last part of the move function is the check if the ball hit the paddle. If the ball did hit the paddle, it flips directions and starts to move towards the other player.

To start the game, we use pygame.init(), along with an instance of each player and ball to be used throughout the entirety of the game. There is a start function that waits for the space key to be pressed. Once it is pressed, the game function is called that runs a continuous loop of the players and ball being redrawn and moved. When the game is complete and someone had won, they can quit or restart. If the player chooses to restart, the start function is called once again which will then initialize everything again, resetting everything back to its original state.

1.5 User's Manual

This game is easy to setup and play, just follow these few basic steps!

1. Gather Raspberry Pi with power cord, mouse, keyboard, HDMI monitor, and HDMI cord
2. Plug all necessary cords in before the power cord, then plug the power cord in
3. Once Pi is on, double click on the Terminal from the Pi Desktop
4. Type the command "python3 pong.py" without quotation marks and hit enter
5. Game will prompt you with the rest, have fun!

1.6 Presentation Details

For our presentation, we have taken a short video going over our setup and the game. We show the Raspberry Pi and how to hook it up to the appropriate displays. Then we show how to run and play the game on the Pi quickly and easily. Finally, we play the game to demonstrate how it works.

1.7 Programmer's Manual

Since Python spacing is crucial, this code may look somewhat strange in this report. It is all well tested in Pycharm and on the Raspberry Pi through the terminal to ensure correct functionality.

```
# Top left corner of screen is (0,0)

import pygame

class Player1():
    # has x location, y location, score, width, height
    def __init__(self):
        self.x_loc = 12
        self.score = 0
        self.width = 8
        self.height = 64
        self.y_loc = WINDOW_HEIGHT / 2 - (self.height / 2)  # for offset so
it's in center

    def drawPlayer(self):
        pygame.draw.rect(screen, (245, 37, 85), (self.x_loc, self.y_loc,
self.width, self.height))

    def move(self):
        # check if player is at border of frame
        if self.y_loc <= 0:
            self.y_loc = 0
        elif self.y_loc >= WINDOW_HEIGHT - self.height:
            self.y_loc = WINDOW_HEIGHT - self.height

        # check what key player pressed
        keyPressed = pygame.key.get_pressed()
        if keyPressed[pygame.K_s]:      # wants to move down
            self.y_loc = self.y_loc + MOVE
        elif keyPressed[pygame.K_w]:    # wants to move up
            self.y_loc = self.y_loc - MOVE

    def drawScore(self):
```

```
        font = pygame.font.Font(None, 64)
        p1_blit = font.render(str(self.score), 1, (255, 255, 255))
        screen.blit(p1_blit, (32, 20))
        if self.score == 5:
            screen.fill((0, 0, 0))
            font = pygame.font.Font(None, 64)
            font2 = pygame.font.Font(None, 36)
            blit = font.render("Player 1 Wins!", 1, (255, 255, 255))
            blit2 = font2.render("  Press q to quit  ", 1, (255, 255, 255))
            blit3 = font2.render("Press r to restart ", 1, (255, 255, 255))
            screen.blit(blit, (WINDOW_WIDTH / 2 - 150, WINDOW_HEIGHT / 2 -
20))
            screen.blit(blit2, (WINDOW_WIDTH / 2 - 100, WINDOW_HEIGHT - 70))
            screen.blit(blit3, (WINDOW_WIDTH / 2 - 100, WINDOW_HEIGHT - 40))
            pygame.display.flip()
            event = pygame.event.wait()
            if event.type == pygame.QUIT:
                pygame.quit()
                exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_q:
                    exit()                      # pressed q key to quit
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_r:
                    start()                     # pressed r key to restart

class Player2():
    # has x location, y location, score, width, height
    def __init__(self):
        self.score = 0
        self.width = 8
        self.height = 64
        self.x_loc = WINDOW_WIDTH - 12 - self.width
        self.y_loc = WINDOW_HEIGHT / 2 - (self.height / 2)  # for offset so
it's in center

    def drawPlayer(self):
        pygame.draw.rect(screen, (245, 37, 85), (self.x_loc, self.y_loc,
self.width, self.height))

    def move(self):
        # check if player is at border of frame
        if self.y_loc <= 0:
            self.y_loc = 0
        elif self.y_loc >= WINDOW_HEIGHT - self.height:
            self.y_loc = WINDOW_HEIGHT - self.height

        # check what key player pressed
        keyPressed = pygame.key.get_pressed()
        if keyPressed[pygame.K_DOWN]:
            self.y_loc = self.y_loc + MOVE
        elif keyPressed[pygame.K_UP]:
            self.y_loc = self.y_loc - MOVE

    def drawScore(self):
        font = pygame.font.Font(None, 64)
        p2_blit = font.render(str(self.score), 1, (255, 255, 255))
        screen.blit(p2_blit, (WINDOW_WIDTH - 32 - 24, 20))  # 32 to match, 24
```

```
because it looks closest
        if self.score == 5:
            screen.fill((0, 0, 0))
            font = pygame.font.Font(None, 64)
            font2 = pygame.font.Font(None, 36)
            blit = font.render("Player 2 Wins!", 1, (255, 255, 255))
            blit2 = font2.render("  Press q to quit  ", 1, (255, 255, 255))
            blit3 = font2.render("Press r to restart ", 1, (255, 255, 255))
            screen.blit(blit, (WINDOW_WIDTH / 2 - 150, WINDOW_HEIGHT / 2 -
20))
            screen.blit(blit2, (WINDOW_WIDTH / 2 - 100, WINDOW_HEIGHT - 70))
            screen.blit(blit3, (WINDOW_WIDTH / 2 - 100, WINDOW_HEIGHT - 40))
            pygame.display.flip()
            event = pygame.event.wait()
            if event.type == pygame.QUIT:
                pygame.quit()
                exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_q:
                    exit()                      # pressed q key to quit
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_r:
                    start()                     # pressed r key to restart


class Ball():
    # has x location, y location, x movement, y movement
    def __init__(self):
        self.x_loc = WINDOW_WIDTH / 2
        self.y_loc = WINDOW_HEIGHT / 2
        self.x_move = MOVE
        self.y_move = MOVE

    def move(self):
        # moves right when called, then does all collision checks
        self.x_loc = self.x_loc + self.x_move
        self.y_loc = self.y_loc + self.y_move

        # check of ball hit bottom wall
        if self.y_loc >= WINDOW_HEIGHT:
            self.y_move = -1 * self.y_move

        # check if ball hit top wall
        if self.y_loc <= 0:
            self.y_move = -1 * self.y_move

        # check if ball hit left wall - P2 point
        if self.x_loc <= 0:
            player2.score = player2.score + 1
            self.__init__()                     # reinitialize ball to start
in middle
            self.x_move = self.x_move * - 1     # change ball direction to go
towards P1

        # check if ball hit right wall - P1 point
        if self.x_loc >= WINDOW_WIDTH:
            player1.score = player1.score + 1
            self.__init__()                      # reinitialize ball to start
```

```
in middle

        # check if ball hits P1
        for n in range(-5, player1.height):
            if self.y_loc == n + player1.y_loc:                 # ball y ==
player y + increment
                if self.x_loc <= player1.x_loc + player1.width:
                    self.x_move = self.x_move * - 1             # change ball
direction
                    break
            n = n + 1

        # check if ball hits P2
        for n in range(-5, player2.height):
            if self.y_loc == n + player2.y_loc:                 # ball y ==
player y + increment
                if self.x_loc >= player2.x_loc - player2.width:
                    self.x_move = self.x_move * - 1             # change ball
direction
                    break
            n = n + 1

    def drawBall(self):
        pygame.draw.circle(screen, (74, 235, 12), (int(self.x_loc),
int(self.y_loc)), 5)

def game():
    done = False
    while not done:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                done = True
                exit()

        # move then clear screen then redraw in new spot
        player1.move()
        player2.move()
        ball.move()

        screen.fill((0, 0, 0))

        ball.drawBall()
        player1.drawPlayer()
        player1.drawScore()
        player2.drawPlayer()
        player2.drawScore()

        pygame.display.flip()


WINDOW_WIDTH = 800
WINDOW_HEIGHT = 600
MOVE = 5    # move speed for everything for easy changing
screen = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
pygame.init()
done = False

player1 = Player1()
```

```
player2 = Player2()
ball = Ball()

def start():
    # reinitialization for restart
    player1.__init__()
    player2.__init__()
    ball.__init__()
    while True:
        screen.fill((0, 0, 0))
        player1.drawPlayer()
        player2.drawPlayer()
        player1.drawScore()
        player2.drawScore()

        font = pygame.font.Font(None, 64)
        font2 = pygame.font.Font(None, 36)
        blit = font.render(" Press Space to Start ", 1, (255, 255, 255))
        blit2 = font2.render("  Player 1 keys w & s  ", 1, (255, 255, 255))
        blit3 = font2.render("Player 2 keys up & down", 1, (255, 255, 255))
        screen.blit(blit, (WINDOW_WIDTH / 2 - 220, WINDOW_HEIGHT / 2 - 20))
        screen.blit(blit2, (WINDOW_WIDTH / 2 - 125, WINDOW_HEIGHT - 70))
        screen.blit(blit3, (WINDOW_WIDTH / 2 - 146, WINDOW_HEIGHT - 40))
        pygame.display.flip()

        event = pygame.event.wait()
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                game()

start()
```