INDIVIDUAL PROJECT

2.5D PONG

DOMINIC HOLIFIELD

PURDUE ENGR 13300

TABLE OF CONTENTS

INTRODUCTION

This program was written for the Individual Project for ENGR 13300 using knowledge gained throughout the semester. For the project, I chose to use Python as the programming language because of my previous experience as well as highly available libraries. One of these libraries being Pygame which allows the program to display graphics and monitor keyboard and mouse input, perfect for making a game. For the project, I created 2.5D Pong, based off the game pong. My game is like pong in that you move a paddle up and down a side of the screen to hit the ball back to your opponent, based on the sport table tennis or ping pong. How my game differs is that it has part of the 3rd dimension, showing the sides of the objects, their shadows, and bouncing physics for the ball. My game also requires you to hit the ball on the table, contrary to normal pong where the ball bounces off the edges. I chose to create this game as I had previously created pong in Python and had always thought it would be interesting to create a more realistic version. This project was the perfect time to make this idea come to life. I hope that whoever plays the game will enjoy it. The project files can be found here: https://github.com/dholifield/2.5DPong

PROGRAM OVERVIEW

2.5D Pong starts in the main file with the main function where it initializes Pygame, creates a screen, and creates a ball and scoreboard. It then runs the main loop where it first calls the main menu. In the main menu, the user has the option to select one of three options: one player, two player, or zero player. One player is the user against the computer, two player is one user against another on the same computer, and zero player is two computers playing against each other. Once a selection is made, the program goes right into the game where player one serves the ball. Player one uses ‘w’ to move the paddle up and ‘s’ to move the paddle down while player two uses ‘o’ and ‘k’. The user must hit the ball back to the opponent, making sure to hit it on the table. If the ball is hit the upper edge of the paddle, the ball’s velocity decreases in y direction and increases when hit on the lower edge. The paddle also mimics spin by changing the ball speed based on the speed of the paddle on contact. The score is shown at the top of the window and a game is played first to 11 points. Once a winner is determined, the game loop ends, and a winner screen is pulled up with the option to go back to the main menu. Once this button is pressed, the main loop will loop again, bringing the user back to the main menu allowing them to play again. The game is exited when the window is closed.

CLASSES

class Paddle: pg 20

This class is used to create the paddle objects the player or computer will use in the game. There are two subclasses that use Paddles features called playerPaddle and cpuPaddle.

The parent Paddle class includes the functions reset() and render(). The initialization of the object will set the paddles x location on the screen. reset() will reset the y position of the paddle as well as the paddle velocity. render() will render the paddle image to the screen, allowing the player to view it.

The subclass playerPaddle creates a paddle that can be controlled by keyboard input. It has one function named update. Initialization of this object will set the x position as well as the key binds for up and down movement. The update() function updates the position of the paddle based on the current velocity. The velocity is determined by user inputs as well as the borders of the screen.

The subclass cpuPaddle creates a paddle that can be controlled by the computer. Like playerPaddle, it has one function named update. Initialization of this object will set the x position of the paddle as well as set the difficulty of the computer. update() updates the position of the paddle based on the current velocity. Its velocity is determined by the current position of the ball as well as the edge of the screen. Higher difficulties will move faster allowing them to get to the ball sooner and miss less frequently. Support for varying difficulties is not yet in place.

class Ball: pg 22

This class is used to create a ball object that is used during the game. It controls the movement of the ball, detects paddle collisions, detects when a point is scored, and keeps track of the current ball possession. The ball moves in three-dimensional space and is rendered in a way to capture this effect.

The Ball class includes the functions reset(), update(), paddleCollide(), scored(), switchPossession(), and render(). Initialization of a ball object creates a rectangle for the ball and shadow and sets its starting position and velocity.

Reset() resets the balls x, y, and z position and velocity as well as delay to allow the user to recover between points.

Update() updates the position of the ball based on the current velocity. The x component of the velocity is constant in magnitude but flips direction when hit by a paddle. The z component of the velocity is determined by the z acceleration, which is gravity. A sound is played when the ball hits the table. The y component of the velocity is determined by the ball-paddle collision.

PaddleCollide() checks if the ball has collided with any of the paddles. When the ball does collide with a paddle, the x speed is flipped, and the y speed is altered based on the position the ball hits the paddle as well as the speed of the paddle on contact. This function also plays a sound when a ball hits a paddle.

Scored() checks if a ball has been scored or not and returns this value as a Boolean. The ball is scored when

it hits the ground off the edge of the table, or when a player is unable to return the ball.

SwitchPossession() switches the current possession of the ball. This is called in update() and is executed when the ball hits the table. The possession is used to determine who gets the point with scored() returns true.

Render() is the last function in Ball which is simply used to render the ball to the screen. It renders the x position as is but adds a fraction of the z to the y position allowing the player to see the ball bouncing. A shadow is also rendered using only the x and y coordinates of the ball. The y position of the shadow changes if the ball goes off the edge of the table and project onto the ground.

class Scoreboard: pg 25

This class is used to create and object that keeps track of the score as well as determine a winner. Scoreboard's functions include count(), score(), add(), winner(), reset(), and render(). Initialization of a Scoreboard object sets each players' score to 0 as well as set the score to play to.

Count() checks if a point has been scored and the uses add() to reward the point appropriately. This also resets the ball and switches the position as the winner will serve the next point.

Score() returns the score of both players as well as the value of winner() as a tuple which is used set get the score and winner at the end of a match.

Add() adds a point to a player based on the current balls possession. If player one currently has possession

when called, that means they lost the point so one point will be awarded to player two. This is called inside of count().

Winner() returns 1 if player one has won the game or 2 is player two has one. If neither player has won, it simply returns 0. This is called in the game function to determine if there is a winner.

Reset() resets the scores of both players to 0. This is called every time main loops to reset the score between games.

Render() is the last function in Scoreboard which renders the score to the top center of the screen showing player one's score on the left and player two's score on the right of the center.

class Button: pg 26

This class is used to create a button object that can be used in menus to allow user interaction of the program. Three buttons are used in the main menu when selecting between one, two, or zero player modes. Another button is used in the winner menu which will return to the main menu when pressed. This class only has one function called draw(). Initialization of a Button object will set the x and y location of the button, the size of the button, as well as the text you want to be on the button.

Draw() draws the button to the screen and returns a Boolean if the button has been pressed or not. When hovered over, the button pops up and goes down when pressed.

USER-DEFINED FUNCTIONS

def main: pg 15

This is the main function of the program that is the start of the game. When called, it initialized Pygame and creates a window for the game. Ball and Scoreboard objects are then created to be used in the game. A while loop is then run which first calls mainMenu(), then game(), and then winMenu() before looping back to the main menu. This will run indefinitely until the user closes out of the window.

def mainMenu: pg 17

This function if the first screen the user will see. It displays the name of the game at the top center and has three buttons to choose a mode. The possible options are one player where the user plays against the computer, two player where the user plays against another user on the same computer, and zero player where the user watches two computer paddles play against each other. This function will loop until a button is pressed or if the user closes out of the window. It will return a tuple of the two paddles based on the option the user selects.

def game: pg 16

This function is where the gameplay happens. When called, the paddles, ball, and scoreboard are reset from possible previous games. A while loop is then run that first checks if the user has exited the window, then calls

drawBackground() which effectively clears the previous frame. Next the paddles and ball are updated using the update() and paddleCollide() functions and the scoreboard is updated using the count() function. The paddles, ball, and scoreboard are then rendered to the screen using their render() function. A timer then waits for a calculated time to generate more consistent framerates. Lastly, the function checks if there has been a winner and will end the loop if one has been found. This loop will run indefinitely until either the game has finished, or the user closes out of the window. When the loop has been exited, the score is returned which is later used in the winMenu().

def winMenu: pg 19

This function is called when the game has finished. The winner is displayed at the top of the screen with the score in the middle of the screen. A button is on the button center of the screen that will return to the main menu when pressed. This screen will loop until either the user pressed the button to return to the main menu, or the user closes the window. This function returns a Boolean called run that will be false if the button is pressed but true if the window is closed. This is used in main to end the main loop when the window is closed.

def drawShadow: pg 27

This function draws a shadow for the input rectangle. This is used in the Ball.render() function when displaying

the ball shadow, the Paddle.render() function when display the paddle shadow, and the drawBackground() function when drawing the shadow of the table. This is its own function because multiple Pygame function are required to display a rectangle with an alpha value, which reduced the length of the code.

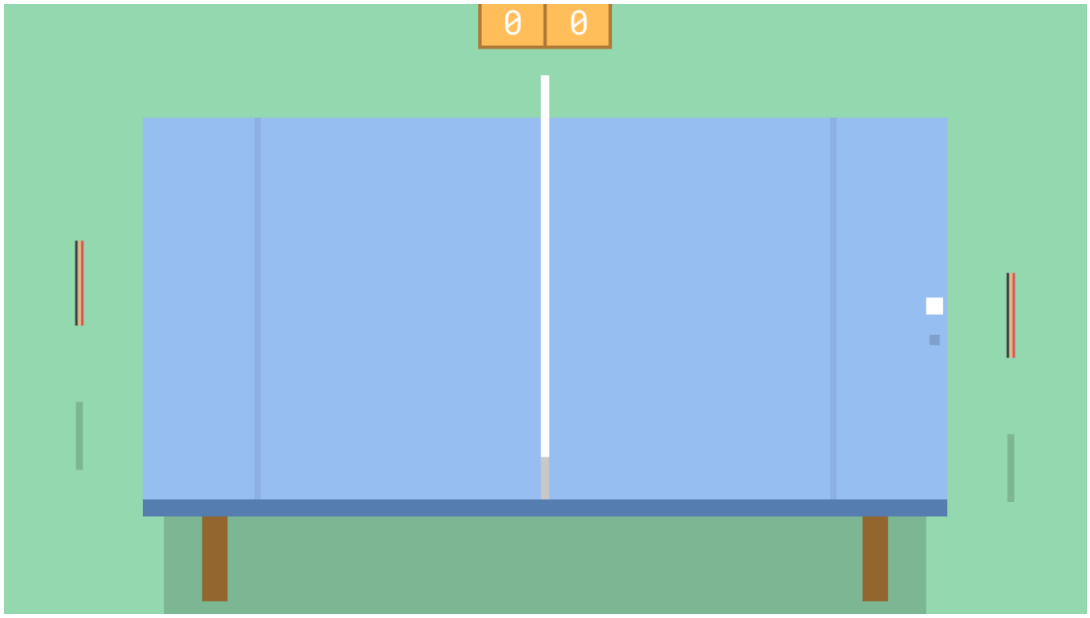
def drawBackground: pg 28

This function draws the background of the game at the beginning of each frame. It starts by filling the screen green, then drawing the two legs of the table. Next it draws the table, its edge, its shadow, and the two lines on the table. Next it draws the net and its edge. Lastly, the scoreboard background is drawn in a wood-like color with a darker border.

USER MANUAL

This program is very intuitive and should not be hard to use after setup. If you are running the python files, you will first need to install the Pygame package. This can be done by typing `python3 -m pip install -U pygame --user` into your command prompt. More information on installation can be found on the Pygame website: https://www.pygame.org/docs/. After you have installed Pygame, the program can be started by simply running main.py. The game will start right up, and you will be seeing the main menu with three options for different modes.

Simply click on the mode you want, and the game will begin.



In one player mode, the ‘w’ key will move the paddle up and the ‘s’ key will move the paddle down. In two player mode, the second player’s key binds are ‘o’ to move the paddle up and ‘k’ to move it down. In zero player mode, the user will watch two computers play against each other. You will play to 11 points, and the game will stop when either player reaches this score.

When the game has finished, you will be shown the winner screen where you can press the main menu button to return to the main menu to play another game. Thanks for trying out my game and I hope you enjoy it!

APPENDIX

main.py

```
1.  import pygame
2.  from time import sleep, time
3.  from pygame.locals import *
4.  from paddle import *
5.  from ball import *
6.  from background import *
7.  from scoreboard import *
8.  from menu import *
9.  from game import *
10. from constants import *
11.
12. # Main function that loops through the menus and game
13. def main():
14.     running = True
15.
16.     # Pygame initialization
17.     pygame.init()
18.     pygame.display.set_caption("2.5D Pong")
19.     screen = pygame.display.set_mode((WIDTH, HEIGHT))
20.
21.     # Initializing objects
22.     ball = Ball()
23.     scoreboard = Scoreboard(11)
24.
25.     # Main loop, will run until you close the window or force stop the program
26.     while running:
27.         # Calls the main menu and gets paddles
28.         paddles = mainMenu(screen, ball)
29.         if paddles[0] != 0:
30.             # Creates Game object with paddles and runs it
31.             winner = game(screen, ball, scoreboard, paddles)
32.
33.             if winner[2] == 0:
34.                 running = False
35.             else:
36.                 sleep(1)
37.                 # Runs win menu when the game is over, loops back to main menu
38.                 running = winMenu(screen, winner)
39.         else:
40.             running = False
41.         #end
42.     #end
43. #end
44.
45. # Runs main code when file is run. Allows main to be accessed elsewhere
46. if __name__ == "__main__":
47.     main()
48. #end
```

game.py

```
1. import pygame
2. from time import sleep, time
3. from pygame.locals import *
4. from paddle import *
5. from ball import *
6. from background import *
7. from scoreboard import *
8. from constants import *
9.
10.# Main loop of game where gameplay happens
11.def game(screen, ball, scoreboard, paddles):
12.    # Initialization of paddles
13.    paddle_one = paddles[0]
14.    paddle_two = paddles[1]
15.
16.    running = True
17.
18.    # Resets ball, paddles, and scoreboard from possible previous games
19.    ball.reset(1)
20.    ball.count = 5
21.    paddle_one.reset()
22.    paddle_two.reset()
23.    scoreboard.reset()
24.
25.    # Main Game loop
26.    while(running):
27.        t0 = time()
28.
29.        # Stop running on exit
30.        for event in pygame.event.get():
31.            if event.type == pygame.QUIT:
32.                running = False
33.        #end
34.
35.        # Clear screen before each frame
36.        drawBackground(screen)
37.
38.        # Update Objects
39.        paddle_one.update()
40.        paddle_two.update()
41.        ball.paddleCollide(paddles)
42.        ball.update()
43.
44.        # Scoring
45.        scoreboard.count(ball)
46.        if ball.count == 1:
47.            paddle_one.reset()
48.            paddle_two.reset()
49.        #end
```

```
50.
51.         # Render objects
52.         paddle_one.render(screen)
53.         paddle_two.render(screen)
54.         ball.render(screen)
55.         scoreboard.render(screen)
56.
57.         # Display frame
58.         display_surface = pygame.display.get_surface()
59.         display_surface.blit(pygame.transform.flip(display_surface, False,
    True), dest=(0, 0))
60.         pygame.display.flip()
61.
62.         t1 = time()
63.         dt = t1 - t0
64.
65.         # Timer for consistant frames per second
66.         if dt < 0.002:
67.             sleep(0.002 - dt)
68.
69.         # Stop running when there is a winner
70.         if scoreboard.winner() != 0:
71.             running = False
72.         #end
73.     #end
74.     return scoreboard.score()
75. #end game
```

menu.py

```
1. import pygame
2. from time import sleep
3. from pygame.locals import *
4. from button import *
5. from paddle import *
6. from constants import *
7.
8. # Main menu where user selects game option
9. def mainMenu(screen, ball):
10.       running = True
11.       mode = 0
12.       difficulty = 2
13.       hover = False
14.
15.       # Creates font for text and sound
16.       font = pygame.font.Font('fonts/MajorMonoDisplay.ttf', 100)
17.       ball_sound = pygame.mixer.Sound('sounds/hit_table.wav')
18.
19.       # Creates three buttons for different play modes
20.       one_player = Button(CENTER_X, 420, BUTTON_SIZE, "one player")
21.       two_player = Button(CENTER_X, 260, BUTTON_SIZE, "two player")
22.       zero_player = Button(CENTER_X, 100, BUTTON_SIZE, "zero player")
23.
24.       # Creates a rectangle for ball sound
```

```
25.      O = Rect(0,0, 58, 73)
26.      O.center = (788, 605)
27.
28.      # Main menu loop
29.      while running:
30.          # Ends loop if you close window
31.          for event in pygame.event.get():
32.              if event.type == pygame.QUIT:
33.                  running = False
34.          #endm
35.
36.          # Fills the screen green and displays the title
37.          screen.fill(GREEN)
38.          title = font.render("2.5D Pong", True, WHITE)
39.          title = pygame.transform.flip(title, False, True)
40.          screen.blit(title, ((WIDTH - title.get_width()) / 2, HEIGHT -
   160))
41.
42.          # Draws the 3 buttons and checking if any are pressed
43.          if one_player.draw(screen):
44.              mode = 1
45.          elif two_player.draw(screen):
46.              mode = 2
47.          elif zero_player.draw(screen):
48.              mode = 3
49.          #end
50.
51.
52.          # Display the frame
53.          display_surface = pygame.display.get_surface()
54.          display_surface.blit(pygame.transform.flip(display_surface,
   False, True), dest=(0, 0))
55.          pygame.display.flip()
56.
57.          # Get posistion of cursor and make a noise if it collides the
   the O
58.          pos = (pygame.mouse.get_pos()[0], HEIGHT -
   pygame.mouse.get_pos()[1])
59.          if O.collidepoint(pos) and not hover:
60.              ball_sound.play()
61.              hover = True
62.          elif not O.collidepoint(pos) and hover:
63.              hover = False
64.          #end
65.
66.          # Timer to prvent unnecessary stress on hardware
67.          sleep(0.01)
68.
69.          # If a button is pressed, end the loop
70.          if mode != 0:
71.              running = False
72.          #end
73.      #end
74.
75.      # Create paddles objects based on user selection
76.      if mode == 2:
77.          paddle_one = playerPaddle(PADDLE_X, pygame.K_w, pygame.K_s)
```

```
78.           paddle_two = playerPaddle(WIDTH - PADDLE_X, pygame.K_o,
   pygame.K_k)
79.       elif mode == 1:
80.           paddle_one = playerPaddle(PADDLE_X, pygame.K_w, pygame.K_s)
81.           paddle_two = cpuPaddle(WIDTH - PADDLE_X, ball, difficulty)
82.       elif mode == 3:
83.           paddle_one = cpuPaddle(PADDLE_X, ball , difficulty)
84.           paddle_two = cpuPaddle(WIDTH - PADDLE_X, ball, difficulty)
85.       else:
86.           paddle_one, paddle_two = 0, 0
87.       #end
88.
89.       # Return the two paddles
90.       return paddle_one, paddle_two
91.   #end mainMenu
92.
93.   # Winner menu when the game has ended
94.   def winMenu(screen, winner):
95.       running = True
96.       run = True
97.
98.       # Creates Font for text
99.       font = pygame.font.Font('fonts/MajorMonoDisplay.ttf', 100)
100.      large_font = pygame.font.Font('fonts/MajorMonoDisplay.ttf', 200)
101.
102.      # Creates button to return to the main menu
103.      main = Button(CENTER_X, 100, BUTTON_SIZE, "main menu")
104.
105.      # Sets text for winner
106.      text = "player one wins!" if winner[2] == 1 else "player two
   wins!"
107.
108.      # Winner Menu Loop
109.      while running:
110.          # Ends loop if you close window
111.          for event in pygame.event.get():
112.              if event.type == pygame.QUIT:
113.                  running = False
114.                  run = False
115.          #end
116.
117.          # Fills the screen green
118.          screen.fill(GREEN)
119.
120.          # Prints out the winner as well as the score
121.          title = font.render(text, True, WHITE)
122.          score = large_font.render(":", True, WHITE)
123.          title = pygame.transform.flip(title, False, True)
124.          score = pygame.transform.flip(score, False, True)
125.          screen.blit(title, ((WIDTH - title.get_width()) / 2, HEIGHT -
   140))
126.          screen.blit(score, ((WIDTH - score.get_width()) / 2, HEIGHT -
   440))
127.
128.          score = large_font.render(str(winner[0]), True, WHITE)
129.          score = pygame.transform.flip(score, False, True)
```

```
130.         screen.blit(score, ((WIDTH - score.get_width()) / 2 - 330,
   HEIGHT - 440))
131.
132.         score = large_font.render(str(winner[1]), True, WHITE)
133.         score = pygame.transform.flip(score, False, True)
134.         screen.blit(score, ((WIDTH - score.get_width()) / 2 + 330,
   HEIGHT - 440))
135.
136.         # Draws button and stops running when pressed
137.         if main.draw(screen):
138.             running = False
139.         #end
140.
141.         # Display the frame
142.         display_surface = pygame.display.get_surface()
143.         display_surface.blit(pygame.transform.flip(display_surface,
   False, True), dest=(0, 0))
144.         pygame.display.flip()
145.
146.         # Timer to prvent unnecessary stress on hardware
147.         sleep(0.01)
148.     #end
149.     return run
150. #end winMenu
```

paddle.py

```
1. import pygame
2. from pygame.locals import *
3. from constants import *
4. from shadow import drawShadow
5.
6. # Paddle Object
7. class Paddle:
8.     # Preset data for paddles
9.     y = CENTER_Y
10.      z = 150
11.      speed = 0
12.
13.      paddle = pygame.image.load('images/paddle.png')
14.      shadow = Rect(0,0, 8, 80)
15.
16.      # Initialization of paddle
17.      def __init__(self, x):
18.          self.x = x
19.      #end
20.
21.      # Resets the paddle position and speed
22.      def reset(self):
23.          self.speed = 0
24.          self.y = CENTER_Y
25.      #end
26.
27.      # Renders the paddle to the screen
28.      def render(self, screen):
```

```
29.             self.shadow.center = (self.x, self.y - 150)
30.             drawShadow(self.shadow, screen)
31.             screen.blit(self.paddle, (self.x - 5, self.y + BALL_Z_START /
    5 - 50))
32.         #end
33.     #end Paddle
34.
35.     # Player Paddle Obejct
36.     class playerPaddle(Paddle):
37.         # Initialization of paddle with keybinds
38.         def __init__(self, x, up_key, down_key):
39.             super().__init__(x)
40.             self.up_key = up_key
41.             self.down_key = down_key
42.         #end
43.
44.         # Updates the position and speed of the paddle
45.         def update(self):
46.             pressed_key = pygame.key.get_pressed()
47.             if pressed_key[self.up_key] and self.y < PADDLE_MAX:
48.                 if self.speed < 0:
49.                     self.speed = 0;
50.                 if self.speed < PADDLE_SPEED:
51.                     self.speed = self.speed + PADDLE_ACCEL
52.             elif pressed_key[self.down_key] and self.y > PADDLE_MIN:
53.                 if self.speed > 0:
54.                     self.speed = 0;
55.                 if self.speed > -PADDLE_SPEED:
56.                     self.speed = self.speed - PADDLE_ACCEL
57.             else:
58.                 if abs(self.speed) >= 0.1:
59.                     self.speed = self.speed - (abs(self.speed) /
    self.speed) * PADDLE_DECEL
60.                 else:
61.                     self.speed = 0
62.             self.y = self.y + self.speed
63.         #end
64.     #end playerPaddle
65.
66.     # Computer Paddle Object
67.     class cpuPaddle(Paddle):
68.         # Initialization of paddle with difficulty
69.         def __init__(self, x, ball, difficulty):
70.             super().__init__(x)
71.             self.ball = ball
72.             self.difficulty = difficulty
73.
74.             if difficulty == 1:
75.                 self.max_speed = 1
76.                 self.accel = 0.01
77.                 self.decel = 0.03
78.             elif difficulty == 2:
79.                 self.max_speed = 1.5
80.                 self.accel = 0.02
81.                 self.decel = 0.10
82.             elif difficulty == 3:
83.                 # Expert difficulty
```

```
84.                 pass
85.             #end
86.         #end
87.
88.         # Updates the position and speed of the paddle
89.         def update(self):
90.             diff = self.y - self.ball.y
91.             direction = self.ball.x_speed * (self.x - self.ball.x)
92.             if direction > 0 and self.ball.count == 0:
93.                 if diff < 0:
94.                     if self.speed < 0 and self.difficulty > 2:
95.                         self.speed = 0;
96.                     if self.speed < self.max_speed:
97.                         self.speed = self.speed + self.accel
98.                 elif diff > 0:
99.                     if self.speed > 0 and self.difficulty > 2:
100.                        self.speed = 0;
101.                    if self.speed > -self.max_speed:
102.                        self.speed = self.speed - self.accel
103.                else:
104.                    if abs(self.speed) >= 0.1:
105.                        self.speed = self.speed - (abs(self.speed) /
    self.speed) * self.decel
106.                    else:
107.                        self.speed = 0
108.            else:
109.                if abs(self.speed) >= 0.1:
110.                    self.speed = self.speed - (abs(self.speed) /
    self.speed) * self.decel
111.                else:
112.                    self.speed = 0
113.            #end
114.            self.y = self.y + self.speed
115.        #end
116. #end cpuPaddle
```

ball.py

```
1. import pygame
2. from time import sleep, time
3. from random import seed, random
4. from pygame.locals import *
5. from constants import *
6. from shadow import drawShadow
7. pygame.mixer.init()
8. miss_sound = pygame.mixer.Sound('sounds/hit_miss.wav')
9. paddle_sound = pygame.mixer.Sound('sounds/hit_table.wav')
10.  table_sound = pygame.mixer.Sound('sounds/hit_miss.wav')
11.
12.  # Ball Object
13.  class Ball:
14.      # Preset values for a ball
15.      ball = Rect((0,0),BALL_SIZE)
16.      shadow = Rect(0,0, 15, 15)
17.
```

```
18.      x = BALL_X_START
19.      y = CENTER_Y
20.
21.      ball.center = (x, y)
22.      count = 1
23.      seed(time)
24.
25.      # Initialization of ball
26.      def __init__(self):
27.          self.reset(1)
28.      #end
29.
30.      # Reset ball position and speed
31.      def reset(self, player):
32.          self.x = BALL_X_START
33.          self.y = CENTER_Y
34.          self.z = BALL_Z_START
35.          self.sy = self.y
36.
37.          self.x_speed = BALL_X_SPEED
38.          if player == 2:
39.              self.x = WIDTH - BALL_X_START
40.              self.x_speed = -BALL_X_SPEED
41.          self.y_speed = random() / 2 - 0.25
42.          self.z_speed = BALL_Z_START_SPEED
43.          self.z_accel = BALL_Z_ACCEL
44.
45.          self.possession = player
46.
47.          self.count = 250
48.      #end
49.
50.      # Updates ball position
51.      def update(self):
52.          if self.count <= 0:
53.              self.z_speed = self.z_speed + self.z_accel
54.              self.z = self.z + self.z_speed
55.              if self.z <= 0:
56.                  self.z = 0
57.                  self.z_speed = BALL_Z_SPEED
58.                  self.switchPossession()
59.                  table_sound.play()
60.
61.              self.x = self.x + self.x_speed
62.              self.y = self.y + self.y_speed
63.
64.              diff_x = CENTER_X - self.x
65.              self.sy = self.y
66.              if abs(diff_x) > TABLE_LENGTH / 2:
67.                  self.sy = self.sy - 150
68.          #end
69.
70.          # Play sound if ball is hit or missed
71.          if self.count == 1:
72.              paddle_sound.play()
73.          elif self.count == 249:
74.              miss_sound.play()
```

```
75.       #end
76.
77.       # Alter x and y speeds of ball when in contact with a paddle
78.       def paddleCollide(self, paddles):
79.           for paddle in paddles:
80.               x_diff = self.x - paddle.x
81.               if abs(x_diff) <= (PADDLE_WIDTH + self.ball.width) / 2:
82.                   y_diff = self.y - paddle.y
83.                   if abs(y_diff) <= (PADDLE_HEIGHT + self.ball.height) /
    2:
84.                       self.y_speed = self.y_speed + (y_diff / 50) -
    paddle.speed / 2
85.                       self.x_speed = BALL_X_SPEED if x_diff > 0 else -
    BALL_X_SPEED
86.                       paddle_sound.play()
87.                   #end
88.               #end
89.           #end
90.       #end
91.
92.       # Detects when a point is scored
93.       def scored(self):
94.           scored = False
95.           # If ball falls off side of table
96.           if self.z < BALL_Z_SPEED:
97.               diff = CENTER_Y - self.y
98.               if abs(diff) > (TABLE_WIDTH + self.ball.width) / 2:
99.                   scored = True
100.          # If ball goes off back of table
101.          if self.x < 0 or self.x > WIDTH:
102.              scored = True
103.          return scored
104.      #end
105.
106.      # Switch possession on contact with table
107.      def switchPossession(self):
108.          self.possession = 1 - self.possession + 2
109.      #end
110.
111.      # Renders the ball to the screen
112.      def render(self, screen):
113.          if self.count <= 0:
114.              self.ball.center = (self.x, self.y + (self.z) / 5)
115.
116.              size = 20 - self.z / 25
117.              self.shadow.size = (size, size)
118.              self.shadow.center = (self.x, self.sy)
119.          else:
120.              self.count = self.count - 1
121.          if abs(CENTER_X - self.x) > TABLE_LENGTH / 2 or abs(CENTER_Y -
    self.y) < TABLE_WIDTH / 2:
122.              drawShadow(self.shadow, screen)
123.          pygame.draw.rect(screen, WHITE, self.ball)
124.      #end
125. #end Ball
```

scoreboard.py

```
1. import pygame
2. from pygame.locals import *
3. from constants import *
4.
5. # Scoreboard Object
6. class Scoreboard:
7.     # Preset values for a scoreboard
8.     score_one = 0
9.     score_two = 0
10.
11.      # Initialization of scoreboard with max score
12.      def __init__(self, max):
13.          self.max = max
14.      #end
15.
16.      # Counts each time a ball is scored
17.      def count(self, ball):
18.          if ball.scored():
19.              self.add(ball.possession)
20.              ball.switchPossession()
21.              ball.reset(ball.possession)
22.      #end
23.
24.      # Returns the score
25.      def score(self):
26.          return self.score_one, self.score_two, self.winner()
27.      #end
28.
29.      # Adds a point to a player
30.      def add(self, player):
31.          if player == 2:
32.              self.score_one = self.score_one + 1
33.          elif player == 1:
34.              self.score_two = self.score_two + 1
35.      #end
36.
37.      # Returns 1 or 2 when a player has won, otherwise 0
38.      def winner(self):
39.          winner = 0
40.
41.          if self.score_one == self.max:
42.              winner = 1
43.          elif self.score_two == self.max:
44.              winner = 2
45.          return winner
46.      #end
47.
48.      # Resets the score
49.      def reset(self):
50.          self.score_one = 0
```

```
51.          self.score_two = 0
52.      #end
53.
54.      # Renders the score to the screen
55.      def render(self, screen):
56.          font = pygame.font.Font('fonts/RobotoMono.ttf', 40)
57.
58.          player_one = font.render(str(self.score_one), True, WHITE)
59.          player_two = font.render(str(self.score_two), True, WHITE)
60.          player_one = pygame.transform.flip(player_one, False, True)
61.          player_two = pygame.transform.flip(player_two, False, True)
62.
63.          x1 = len(str(abs(self.score_one))) - 1
64.          x2 = len(str(abs(self.score_two))) - 1
65.          screen.blit(player_one, (CENTER_X - 50 - x1 * 14, HEIGHT -
   48))
66.          screen.blit(player_two, (CENTER_X + 28 - x2 * 14, HEIGHT -
   48))
67.      #end
68.  #end Scoreboard
```

button.py

```
1. import pygame
2. from pygame.locals import *
3. from constants import *
4.
5. # Button Object
6. class Button():
7.     # Preset values for a button
8.     button_col = WHITE
9.     hover_col = GRAY
10.      text_col = GRAY
11.      click_col = LIGHT_GRAY
12.
13.      clicked = False
14.      space = 0
15.
16.      # Initialization of button with location, size, and text
17.      def __init__(self, x, y, size, text):
18.          self.x = x
19.          self.y = y
20.          self.text = text
21.          self.size = size
22.      #end
23.
24.      # Draws button and returns true if button is pressed
25.      def draw(self, screen):
26.          action = False
27.
28.          pos = (pygame.mouse.get_pos()[0], HEIGHT -
   pygame.mouse.get_pos()[1])
29.
30.          # Creates rectangles for button, collision, and shaddow
31.          button_rect = Rect((0,0), self.size)
```

```
32.           collision = Rect(0,0, self.size[0], self.size[1] + self.space)
33.           shaddow = Rect((0,0), self.size)
34.
35.           button_rect.center = (self.x, self.y + self.space)
36.           collision.center = (self.x , self.y + self.space / 2)
37.           shaddow.center = (self.x, self.y)
38.
39.           # Checks if button is hovered over or pressed
40.           if collision.collidepoint(pos):
41.               if pygame.mouse.get_pressed()[0] == 1:
42.                   self.clicked = True
43.                   self.space = 4
44.               elif pygame.mouse.get_pressed()[0] == 0 and self.clicked
   == True:
45.                   self.clicked = False
46.                   action = True
47.               else:
48.                   self.space = 8
49.           else:
50.               self.space = 0
51.               self.clicked = False
52.           #end
53.
54.           # Draws button, shaddow, and text
55.           pygame.draw.rect(screen, self.hover_col, shaddow)
56.           pygame.draw.rect(screen, self.button_col, button_rect)
57.
58.           font = pygame.font.Font('fonts/MajorMonoDisplay.ttf', 35)
59.           text_img = font.render(self.text, True, self.text_col)
60.           text_img = pygame.transform.flip(text_img, False, True)
61.           text_len = text_img.get_width()
62.           screen.blit(text_img, (self.x - text_len / 2, self.y - 17 +
   self.space))
63.
64.           return action
65.       #end
66.   #end Button
```

shadow.py

```
1. import pygame
2. from pygame.locals import *
3. from constants import *
4.
5. # Draws the shadow of the ball
6. def drawShadow(shadow, screen):
7.     shape_surf = pygame.Surface(pygame.Rect(shadow).size,
   pygame.SRCALPHA)
8.     pygame.draw.rect(shape_surf, SHADOW, shape_surf.get_rect())
9.     screen.blit(shape_surf, shadow)
10.  #end
```

background.py

```
1. import pygame
2. from pygame.locals import *
3. from constants import *
4. from shadow import drawShadow
5.
6. # Rectangles that makeup the table and net
7. table = Rect(0,0, TABLE_LENGTH, TABLE_WIDTH)
8. table_edge = Rect(0,0, TABLE_LENGTH, TABLE_HEIGHT)
9. table_shadow = Rect(0,0, TABLE_LENGTH - 50, TABLE_SHADOW_WIDTH)
10.  leg = Rect(0,0, 30, LEG_HEIGHT)
11.  line = Rect(0,0, 7, TABLE_WIDTH)
12.  net = Rect(0,0, NET_LENGTH, TABLE_WIDTH)
13.  net_shadow = Rect(0,0, NET_LENGTH, NET_HEIGHT)
14.
15.  # Rectangles for the scoreboard and border
16.  scoreboard = Rect(0,0, SCOREBOARD_WIDTH, SCOREBOARD_HEIGHT)
17.  border = Rect(0,0, SCOREBOARD_WIDTH + 2 * SCOREBOARD_BORDER,
    SCOREBOARD_HEIGHT + SCOREBOARD_BORDER)
18.  scoreboard_line = Rect(0,0, SCOREBOARD_BORDER, SCOREBOARD_HEIGHT +
    SCOREBOARD_BORDER)
19.
20.  # Renders the background including the ground, the table, and the
    scoreboard background
21.  def drawBackground(screen):
22.      # Background
23.      screen.fill(GREEN)
24.
25.      # Table Legs
26.      leg.center = (250, (HEIGHT - TABLE_WIDTH - LEG_HEIGHT) / 2 -
    TABLE_HEIGHT)
27.      pygame.draw.rect(screen, BROWN, leg)
28.      leg.centerx = WIDTH - 250
29.      pygame.draw.rect(screen, BROWN, leg)
30.
31.      # Table
32.      table.center = CENTER
33.      pygame.draw.rect(screen, LIGHT_BLUE, table)
34.      table_edge.center = (CENTER_X, (HEIGHT - TABLE_WIDTH -
    TABLE_HEIGHT) / 2)
35.      pygame.draw.rect(screen, DARK_BLUE, table_edge)
36.      table_shadow.center = (CENTER_X, CENTER_Y - TABLE_WIDTH / 2 -
    TABLE_HEIGHT - TABLE_SHADOW_WIDTH / 2)
37.      drawShadow(table_shadow, screen)
38.
39.      # Table Lines
40.      line.center = (300, CENTER_Y)
41.      pygame.draw.rect(screen, BLUE, line)
42.      line.centerx = WIDTH - 300
43.      pygame.draw.rect(screen, BLUE, line)
44.
45.      # Net
46.      net.center = (CENTER_X, CENTER_Y + NET_HEIGHT)
47.      net_shadow.centerx = CENTER_X
```

```
48.       net_shadow.y = CENTER_Y - TABLE_WIDTH / 2
49.       pygame.draw.rect(screen, WHITE, net)
50.       pygame.draw.rect(screen, LIGHT_GRAY, net_shadow)
51.
52.       # Scoreboard Background
53.       scoreboard.center = (CENTER_X, HEIGHT - SCOREBOARD_HEIGHT / 2)
54.       border.center = (CENTER_X, HEIGHT - (SCOREBOARD_HEIGHT +
   SCOREBOARD_BORDER) / 2)
55.       scoreboard_line.center = (CENTER_X, HEIGHT - SCOREBOARD_HEIGHT /
   2)
56.       pygame.draw.rect(screen, BROWN, border)
57.       pygame.draw.rect(screen, ORANGE, scoreboard)
58.       pygame.draw.rect(screen, BROWN, scoreboard_line)
59.   #end
```

constants.py

```
1. # Colors
2. WHITE = (255, 255, 255)
3. SHADOW = (0, 0, 0, 40)
4. LIGHT_GRAY = (200, 200, 200)
5. GRAY = (150, 150, 150)
6. DARK_GRAY = (100, 100, 100)
7. LIGHT_BLUE = (150, 190, 240)
8. BLUE = (140, 175, 230)
9. DARK_BLUE = (85, 125, 175)
10.  GREEN = (147, 216, 175)
11.  ORANGE = (255, 190, 90)
12.  BROWN = (175, 122, 56)
13.  RED = (255, 133, 109)
14.  YELLOW = (252, 255, 132)
15.
16.  # Screen Size
17.  WIDTH = 1280
18.  HEIGHT = 720
19.  CENTER_X = WIDTH / 2
20.  CENTER_Y = HEIGHT / 2
21.  CENTER = (CENTER_X, CENTER_Y)
22.
23.  # Paddles
24.  PADDLE_X = 90
25.  PADDLE_WIDTH = 10
26.  PADDLE_HEIGHT = 100
27.
28.  PADDLE_SPEED = 3
29.  PADDLE_ACCEL = 0.05
30.  PADDLE_DECEL = 0.2
31.  PADDLE_MIN = 0
32.  PADDLE_MAX = HEIGHT
33.
34.  # Ball
35.  BALL_SIZE = (20, 20)
36.  BALL_X_START = PADDLE_X + 20
37.  BALL_Z_START = 150
38.
```

```
39.  BALL_X_SPEED = 5 # 5
40.  BALL_Z_START_SPEED = 3.691
41.  BALL_Z_SPEED = 5.35 # 5.425
42.  BALL_Z_ACCEL = -0.05 # -0.05
43.
44.  # Table and Net
45.  TABLE_LENGTH = 950
46.  TABLE_WIDTH = 450 #520
47.  TABLE_HEIGHT = 20
48.  TABLE_SHADOW_WIDTH = 125
49.  NET_LENGTH = 10
50.  NET_HEIGHT = 50
51.  LEG_HEIGHT = 100
52.
53.  # Scoreboard
54.  SCOREBOARD_HEIGHT = 50
55.  SCOREBOARD_WIDTH = 150
56.  SCOREBOARD_BORDER = 4
57.
58.  # Button
59.  BUTTON_SIZE = (320, 100)
```