Algorithm and Programming (COMP6047001) Final Project Documentation

Christoffer Raffaelo Wijaya: 2602177051

Brick-Breaker Project

Table of Contents

Overview

The video game that Ali Asari made, which was preloaded at blackberry and released in 1999. It is not as exciting as other RPG or FPS game but it is an old game that i've plates since I was borned. In my era, the blackberry has only half screen and the other half is the keyboard. So basically, the game is about the player deflecting a bouncing ball with a paddle in order to break down a brick wall. The paddle can be moved horizontally and is moved with the touch of a finger, the computer mouse, or the trackwheel on a BlackBerry. A life is lost if the ball lands at the bottom of the screen, the player starts out with three lives. Because of the original Brick Breaker has a lot of levels, I only make one level so the game is not that hard. Once you move to the next level, it would be hard to break the brick.Brick Breaker comes in a variety of variations, some of which let you launch fiery fireballs or use more than one ball if the player is stuck.

Conception

Originally, I actually aimed to recreate a car game in pygame considering that I have played car games since i was little. However, with a few errors that I will go over later and a few time constraint issues, I thought that the implementation of cars, rivals for textures seemed too ambitious especially for my first semester. Nevertheless, Pygame has several features that can help with the Brick-Breaker ,such as the use of shapes and number of the bricks that could be break by the ball. By using that, I could create the ball and the brick which was to break with the use of a combination of lines and rectangles. I started my production on the 28th of November but I changed my topics due to the error that occurred in my visual studio code.

Executions

Modules:

- Pygame
  - Used different kinds of function in the program that is made in windows

- Math
  - It has always been present and is included in the standard Python release and the feature has something to do with the mathematical operations

- Sys
  - It is to exit the brick breaker game

Files: (main.py) because I only make one level

The window game which was made 600 by 800 with one entire screen that had the brick 10 times 3. It was made to be break and it is not a brick that can be hit once and broken. To make it harder, I make the brick, which has to be break twice in order for the brick to be broken. For each player that plays brick breaker game, they are given 3 lives and once the live is over, that means the game is over and you should restart again.

The player can also move the bar to left and right using the right or left chevron button on the motherboard. Although brick breaker is known one of the oldest game, it is also known as one of the most interesting game which I first try before I am into RPG, FPS and MOBA games.

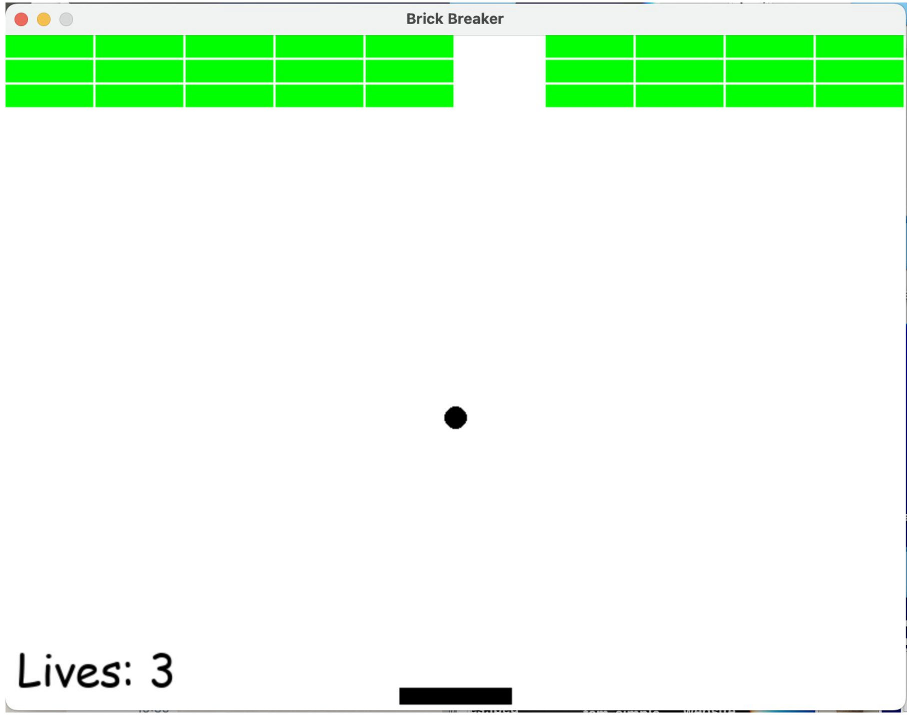


Figure 1

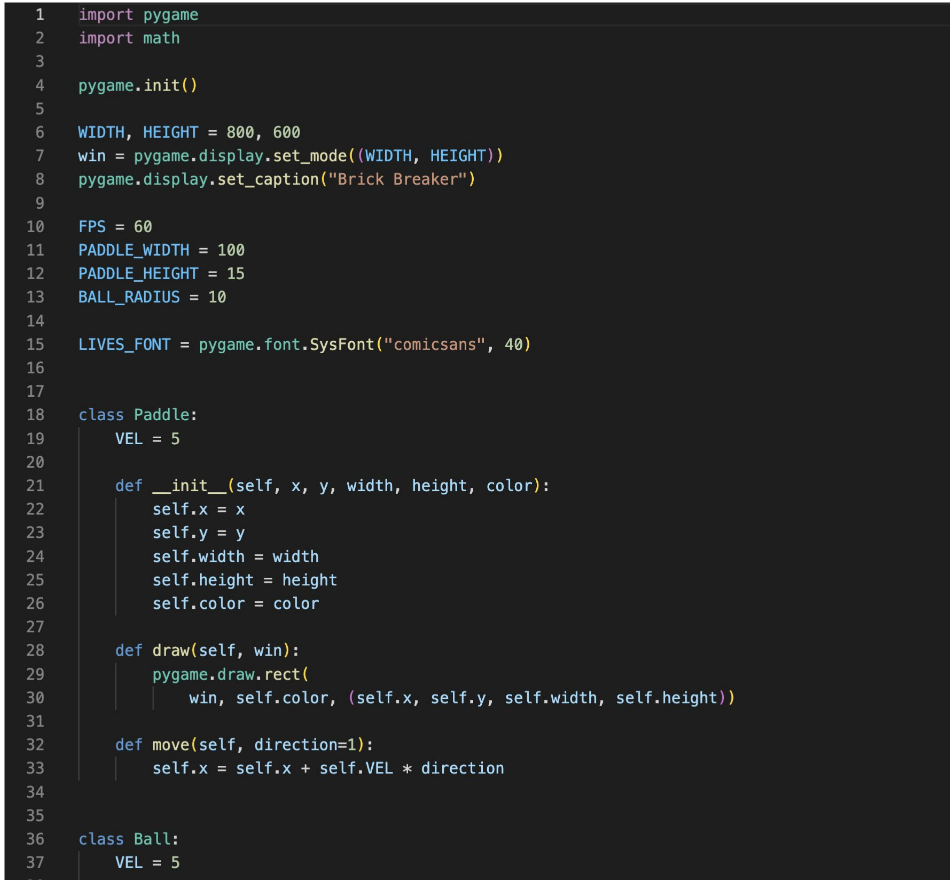
Brick Breaker Algorithm and Code

This code is a basic implementation of the game "Brick Breaker" using the Pygame library in Python. It contains several classes and functions that work together to create the game.

First, the code initializes Pygame and sets the display size, caption, and FPS. It uses the pygame.init() function to initialize the Pygame library, and the pygame.display.set_mode() function to set the display size to 800x600 pixels. It also uses the pygame.display.set_caption() function to set the caption of the game window to "Brick Breaker".

Then, the code defines three classes: Paddle, Ball, and Brick. Each class has its own properties, methods, and behavior.

The Paddle class has a VEL class variable, which is set to 5, and an init method that is used to initialize the object. This method takes in the x and y position, width, height, and color of the paddle as arguments. It also has a draw method, which is used to draw the paddle on the screen, and a move method which is used to move the paddle in the specified direction.



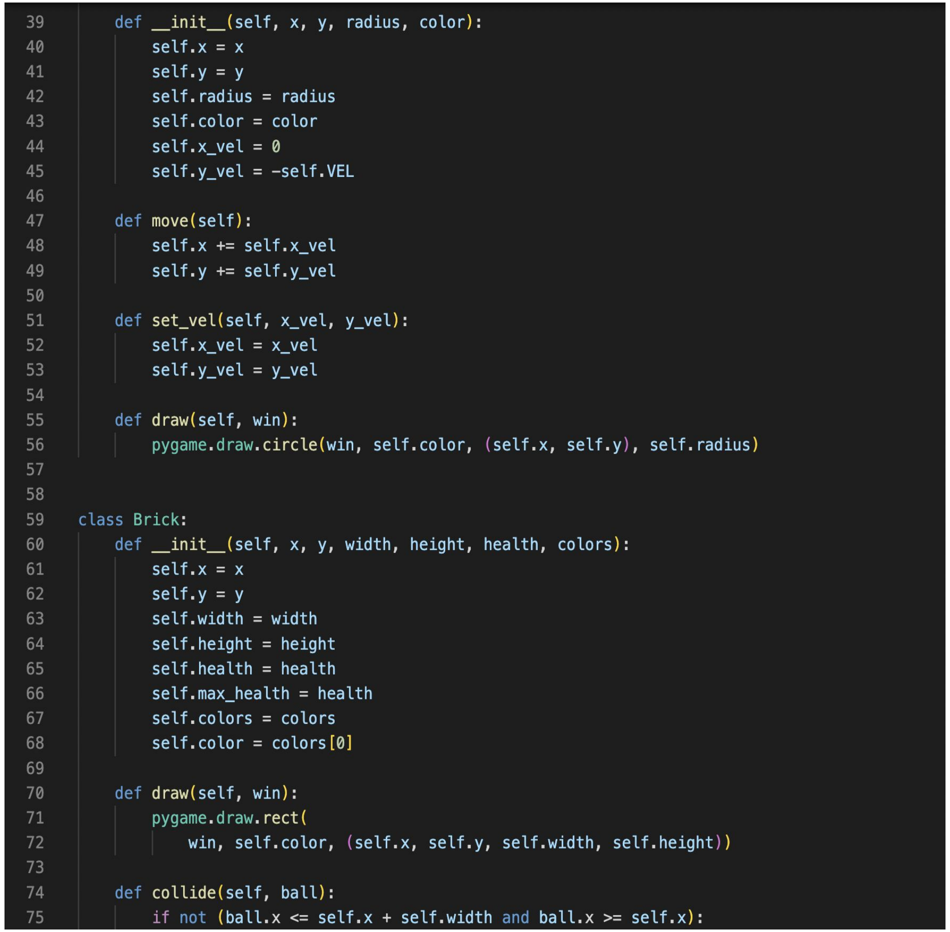
```python
import pygame
import math

pygame.init()

WIDTH, HEIGHT = 800, 600
win = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Brick Breaker")

FPS = 60
PADDLE_WIDTH = 100
PADDLE_HEIGHT = 15
BALL_RADIUS = 10

LIVES_FONT = pygame.font.SysFont("comicsans", 40)


class Paddle:
    VEL = 5

    def __init__(self, x, y, width, height, color):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.color = color

    def draw(self, win):
        pygame.draw.rect(
            win, self.color, (self.x, self.y, self.width, self.height))

    def move(self, direction=1):
        self.x = self.x + self.VEL * direction


class Ball:
    VEL = 5
```

The Ball class has a VEL class variable, which is set to 5, and an init method that is used to initialize the object. This method takes in the x and y position, radius, and color of the ball as arguments. It also has a move method that updates the ball's position based on its current velocity, a set_vel method that sets the velocity of the ball, and a draw method that is used to draw the ball on the screen.

The Brick class has an init method that takes in the x and y position, width, height, health, and colors of the brick as arguments. It has a draw method that is used to draw the brick on the screen, a collide method that detects when the ball collides with the brick, and a hit method that decrements the brick's health and changes its color based on its remaining health.
It also has a static method interpolate that takes in the color_a, color_b and the t as argument and it is used to interpolate the color of the brick based on the health.

The code also defines a draw function that takes in the window, paddle, ball, bricks, and lives as arguments. This function is used to update the display with the current state of the game. It uses the fill() function to fill the background with white color, the draw() method of the Paddle and Ball class to draw the paddle and ball on the screen, and a for loop to iterate through the list of bricks and draw each brick on the screen. It also uses the pygame.font.SysFont() function to create a font object for the text that displays the number of lives, the render() method of the font object to render the text, and the blit() function to draw the text on the screen. Finally, it uses the pygame.display.update() function to update the display.

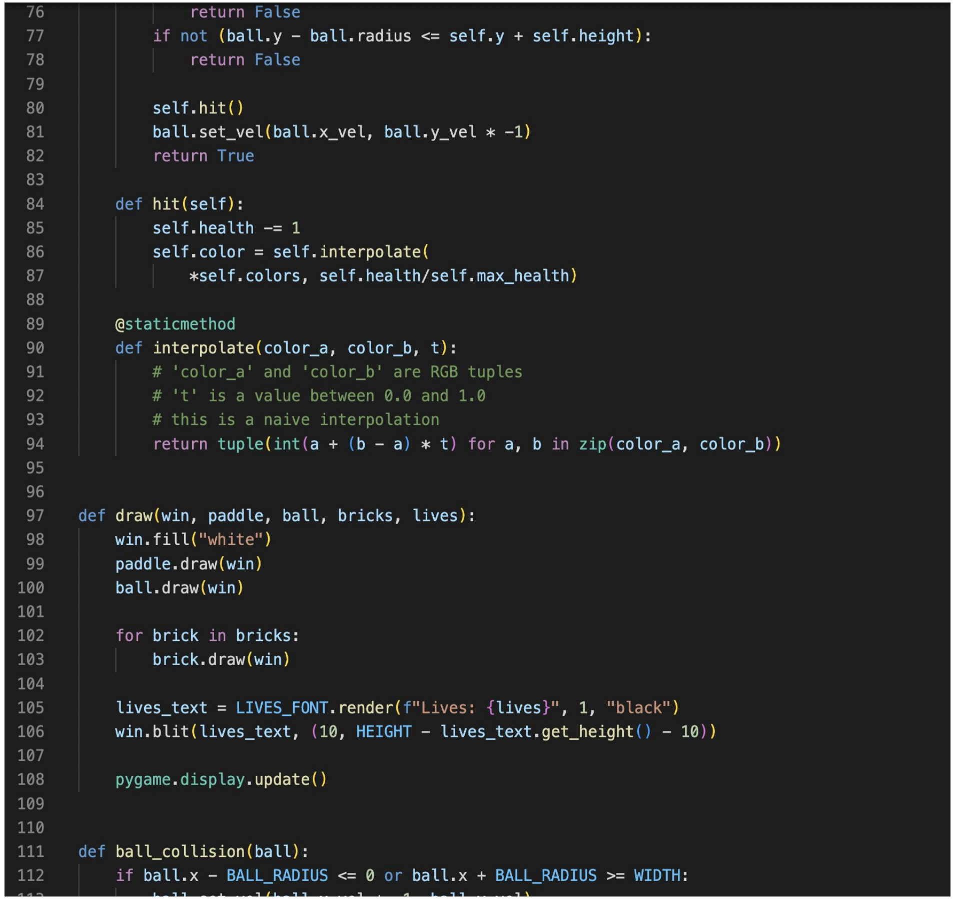
The code also defines a ball_collision function that takes in the ball as an argument. This function is used to handle the ball's movement and collision with the walls and bricks. It checks if the ball has collided with the left or right wall, and if so, it changes the x velocity of the ball. It also checks if the ball has collided with the top or



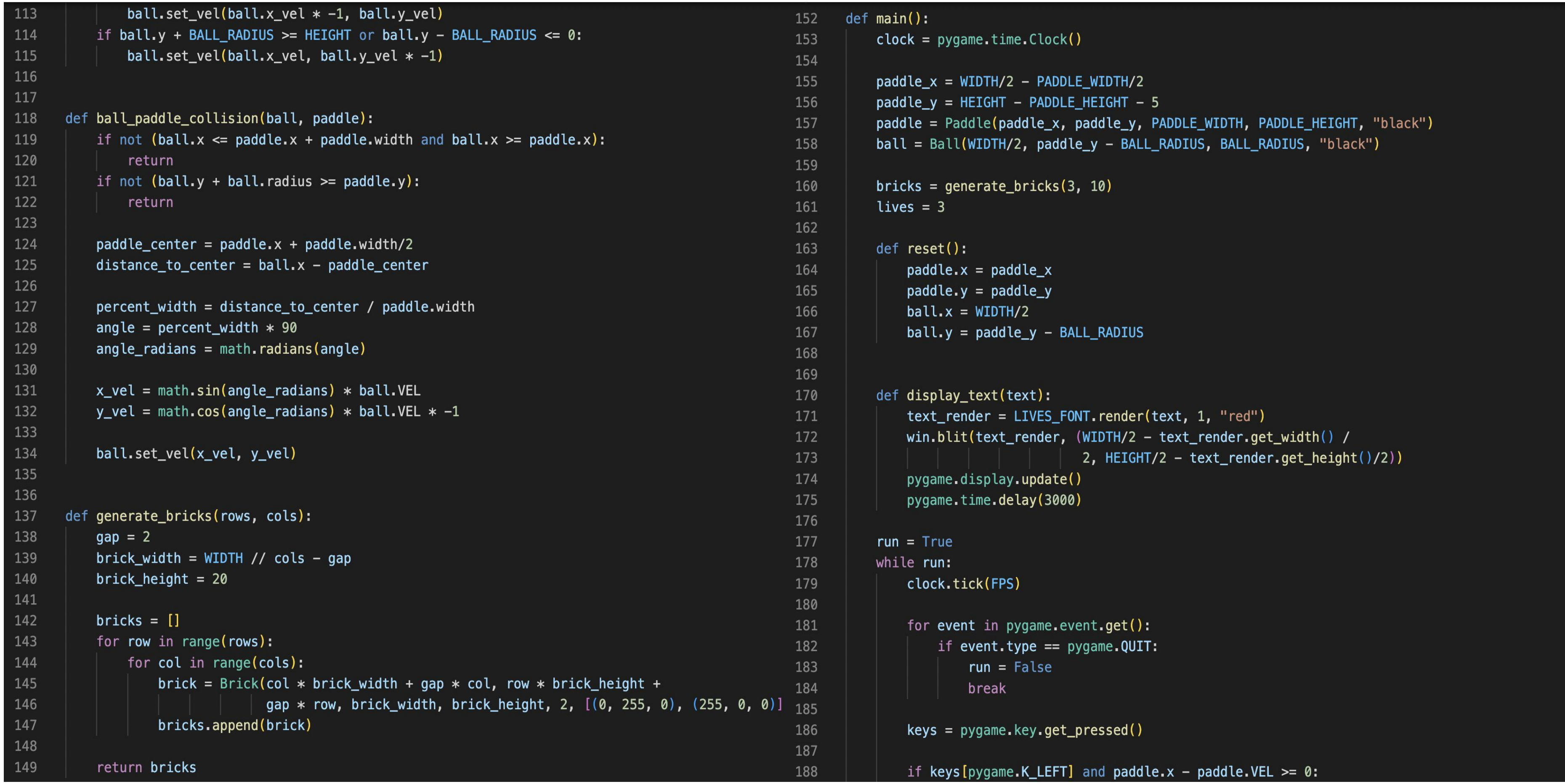
```python
    def __init__(self, x, y, radius, color):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.x_vel = 0
        self.y_vel = -self.VEL

    def move(self):
        self.x += self.x_vel
        self.y += self.y_vel

    def set_vel(self, x_vel, y_vel):
        self.x_vel = x_vel
        self.y_vel = y_vel

    def draw(self, win):
        pygame.draw.circle(win, self.color, (self.x, self.y), self.radius)


class Brick:
    def __init__(self, x, y, width, height, health, colors):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.health = health
        self.max_health = health
        self.colors = colors
        self.color = colors[0]

    def draw(self, win):
        pygame.draw.rect(
            win, self.color, (self.x, self.y, self.width, self.height))

    def collide(self, ball):
        if not (ball.x <= self.x + self.width and ball.x >= self.x):
```

The code also defines a draw function that takes in the window, paddle, ball, bricks, and lives as arguments. This function is used to update the display with the current state of the game. It uses the fill() function to fill the background with white color, the draw() method of the Paddle and Ball class to draw the paddle and ball on the screen, and a for loop to iterate through the list of bricks and draw each brick on the screen. It also uses the pygame.font.SysFont() function to create a font object for the text that displays the number of lives, the render() method of the font object to render the text, and the blit() function to draw the text on the screen. Finally, it uses the pygame.display.update() function to update the display.

The code also defines a ball_collision function that takes in the ball as an argument. This function is used to handle the ball's movement and collision with the walls and bricks. It checks if the ball has collided with the left or right wall, and if so, it changes the x velocity of the ball. It also checks if the ball has collided with the top or

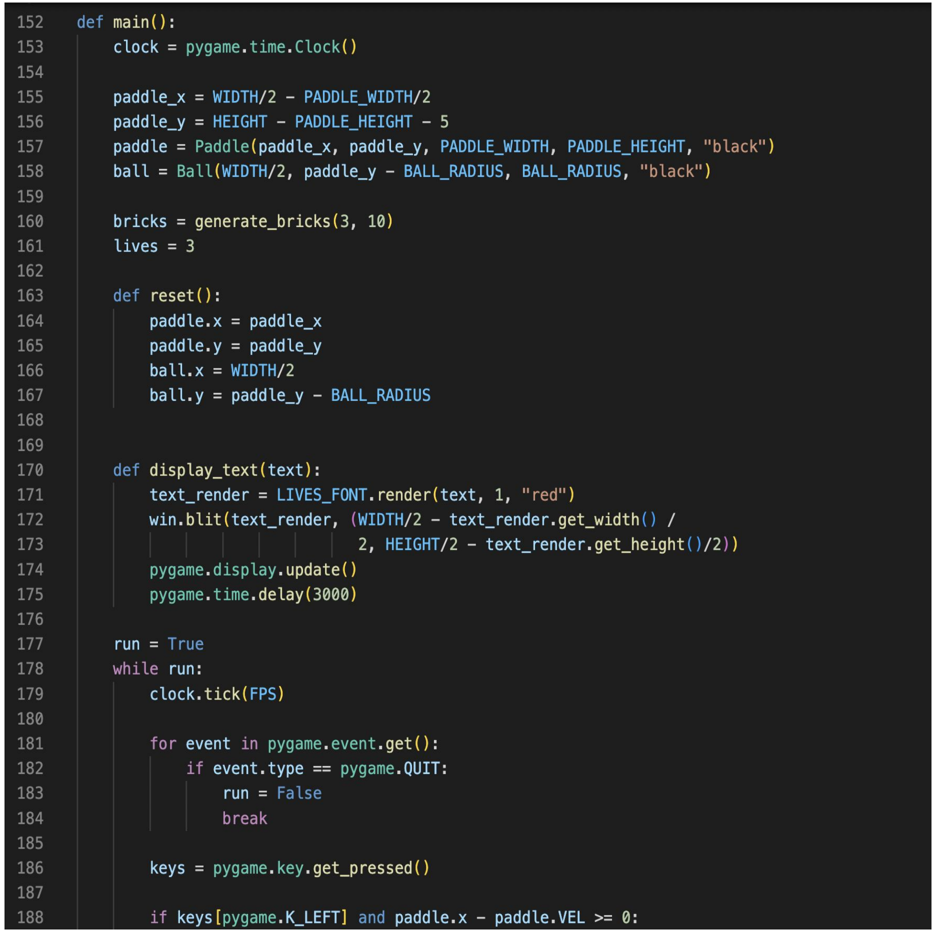


```python
            return False
        if not (ball.y - ball.radius <= self.y + self.height):
            return False

        self.hit()
        ball.set_vel(ball.x_vel, ball.y_vel * -1)
        return True

    def hit(self):
        self.health -= 1
        self.color = self.interpolate(
            *self.colors, self.health/self.max_health)

    @staticmethod
    def interpolate(color_a, color_b, t):
        # 'color_a' and 'color_b' are RGB tuples
        # 't' is a value between 0.0 and 1.0
        # this is a naive interpolation
        return tuple(int(a + (b - a) * t) for a, b in zip(color_a, color_b))


def draw(win, paddle, ball, bricks, lives):
    win.fill("white")
    paddle.draw(win)
    ball.draw(win)

    for brick in bricks:
        brick.draw(win)

    lives_text = LIVES_FONT.render(f"Lives: {lives}", 1, "black")
    win.blit(lives_text, (10, HEIGHT - lives_text.get_height() - 10))

    pygame.display.update()


def ball_collision(ball):
    if ball.x - BALL_RADIUS <= 0 or ball.x + BALL_RADIUS >= WIDTH:
```

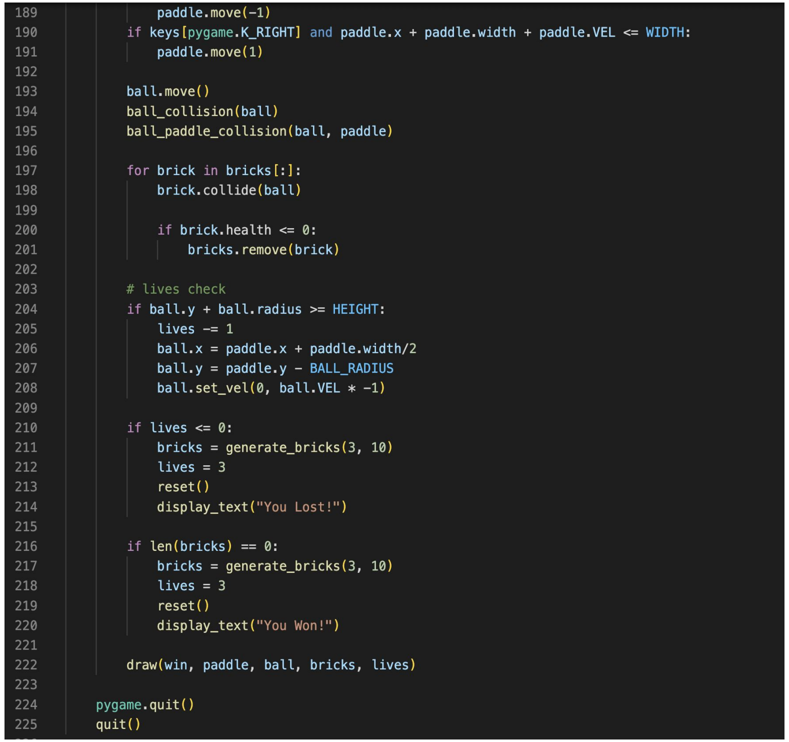


```python
        ball.set_vel(ball.x_vel * -1, ball.y_vel)
    if ball.y + BALL_RADIUS >= HEIGHT or ball.y - BALL_RADIUS <= 0:
        ball.set_vel(ball.x_vel, ball.y_vel * -1)


def ball_paddle_collision(ball, paddle):
    if not (ball.x <= paddle.x + paddle.width and ball.x >= paddle.x):
        return
    if not (ball.y + ball.radius >= paddle.y):
        return

    paddle_center = paddle.x + paddle.width/2
    distance_to_center = ball.x - paddle_center

    percent_width = distance_to_center / paddle.width
    angle = percent_width * 90
    angle_radians = math.radians(angle)

    x_vel = math.sin(angle_radians) * ball.VEL
    y_vel = math.cos(angle_radians) * ball.VEL * -1

    ball.set_vel(x_vel, y_vel)


def generate_bricks(rows, cols):
    gap = 2
    brick_width = WIDTH // cols - gap
    brick_height = 20

    bricks = []
    for row in range(rows):
        for col in range(cols):
            brick = Brick(col * brick_width + gap * col, row * brick_height +
                          gap * row, brick_width, brick_height, 2, [(0, 255, 0), (255, 0, 0)])
            bricks.append(brick)

    return bricks
```

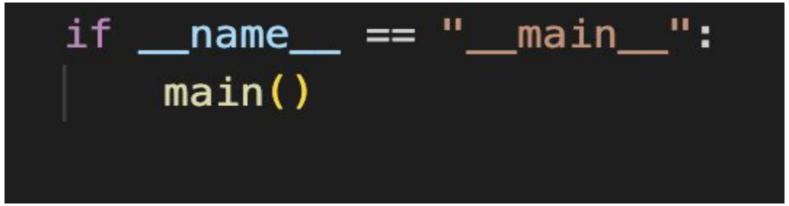
```python
def main():
    clock = pygame.time.Clock()

    paddle_x = WIDTH/2 - PADDLE_WIDTH/2
    paddle_y = HEIGHT - PADDLE_HEIGHT - 5
    paddle = Paddle(paddle_x, paddle_y, PADDLE_WIDTH, PADDLE_HEIGHT, "black")
    ball = Ball(WIDTH/2, paddle_y - BALL_RADIUS, BALL_RADIUS, "black")

    bricks = generate_bricks(3, 10)
    lives = 3

    def reset():
        paddle.x = paddle_x
        paddle.y = paddle_y
        ball.x = WIDTH/2
        ball.y = paddle_y - BALL_RADIUS


    def display_text(text):
        text_render = LIVES_FONT.render(text, 1, "red")
        win.blit(text_render, (WIDTH/2 - text_render.get_width() /
                               2, HEIGHT/2 - text_render.get_height()/2))
        pygame.display.update()
        pygame.time.delay(3000)

    run = True
    while run:
        clock.tick(FPS)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
                break

        keys = pygame.key.get_pressed()

        if keys[pygame.K_LEFT] and paddle.x - paddle.VEL >= 0:
```



```python
def main():
    clock = pygame.time.Clock()

    paddle_x = WIDTH/2 - PADDLE_WIDTH/2
    paddle_y = HEIGHT - PADDLE_HEIGHT - 5
    paddle = Paddle(paddle_x, paddle_y, PADDLE_WIDTH, PADDLE_HEIGHT, "black")
    ball = Ball(WIDTH/2, paddle_y - BALL_RADIUS, BALL_RADIUS, "black")

    bricks = generate_bricks(3, 10)
    lives = 3

    def reset():
        paddle.x = paddle_x
        paddle.y = paddle_y
        ball.x = WIDTH/2
        ball.y = paddle_y - BALL_RADIUS


    def display_text(text):
        text_render = LIVES_FONT.render(text, 1, "red")
        win.blit(text_render, (WIDTH/2 - text_render.get_width() /
                               2, HEIGHT/2 - text_render.get_height()/2))
        pygame.display.update()
        pygame.time.delay(3000)

    run = True
    while run:
        clock.tick(FPS)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
                break

        keys = pygame.key.get_pressed()

        if keys[pygame.K_LEFT] and paddle.x - paddle.VEL >= 0:
```



```python
            paddle.move(-1)
        if keys[pygame.K_RIGHT] and paddle.x + paddle.width + paddle.VEL <= WIDTH:
            paddle.move(1)

        ball.move()
        ball_collision(ball)
        ball_paddle_collision(ball, paddle)

        for brick in bricks[:]:
            brick.collide(ball)

            if brick.health <= 0:
                bricks.remove(brick)

        # lives check
        if ball.y + ball.radius >= HEIGHT:
            lives -= 1
            ball.x = paddle.x + paddle.width/2
            ball.y = paddle.y - BALL_RADIUS
            ball.set_vel(0, ball.VEL * -1)

        if lives <= 0:
            bricks = generate_bricks(3, 10)
            lives = 3
            reset()
            display_text("You Lost!")

        if len(bricks) == 0:
            bricks = generate_bricks(3, 10)
            lives = 3
            reset()
            display_text("You Won!")

        draw(win, paddle, ball, bricks, lives)

    pygame.quit()
    quit()
```



```python
if __name__ == "__main__":
    main()
```

Reflection

Making pygame is not a easy kind of project for me in the final project because i just learn pygame this semester and since this is the first semester, maybe the brick breaker game that i made is just a simple game which is famous in the old eras and my brick breaker game only can go right and left. With this project, I learned a lot about pygame and it is really fun if we can make our own game in the future. I hope I can make more difficult games than this. The problems that I occurred at first that the code is not working but now it is working well. The buttons that we can use at the game were only right and left chevron button.