GitHub repository URL

<https://github.com/saif3234/Block-Breaker-X.git>

import pygame

import random

import sys

# Initialize pygame

pygame.init()

# Screen dimensions

WIDTH, HEIGHT = 800, 600

# Colours

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

BLUE = (0, 0, 255)

YELLOW = (255, 255, 0)

# Initialize screen

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Block Breaker X")

# Clock for controlling frame rate

clock = pygame.time.Clock()

# Paddle

class Paddle:

    def \_\_init\_\_(self):

        self.width = 100

        self.height = 20

        self.x = (WIDTH - self.width) // 2

        self.y = HEIGHT - 30

        self.speed = 7

        self.color = BLUE

        self.lives = 3

        self.items = []

    def move(self, keys):

        if keys[pygame.K\_LEFT] and self.x > 0:

            self.x -= self.speed

        if keys[pygame.K\_RIGHT] and self.x < WIDTH - self.width:

            self.x += self.speed

    def draw(self):

        pygame.draw.rect(screen, self.color, (self.x, self.y, self.width, self.height))

    def collect\_item(self, item):

        self.items.append(item)

        if item.type == "longer\_paddle":

            self.width += 20

        elif item.type == "faster\_paddle":

            self.speed += 2

# Ball

class Ball:

    def \_\_init\_\_(self):

        self.radius = 10

        self.x = WIDTH // 2

        self.y = HEIGHT // 2

        self.dx = random.choice([-4, 4])

        self.dy = -4

        self.color = RED

    def move(self):

        self.x += self.dx

        self.y += self.dy

        # Bounces on the walls

        if self.x <= 0 or self.x >= WIDTH:

            self.dx \*= -1

        if self.y <= 0:

            self.dy \*= -1

    def draw(self):

        pygame.draw.circle(screen, self.color, (self.x, self.y), self.radius)

# Block class

class Block:

    def \_\_init\_\_(self, x, y, width, height):

        self.x = x

        self.y = y

        self.width = width

        self.height = height

        self.color = GREEN

        self.destroyed = False

    def draw(self):

        if not self.destroyed:

            pygame.draw.rect(screen, self.color, (self.x, self.y, self.width, self.height))

# Upgrade class

class Upgrade:

    def \_\_init\_\_(self, x, y, type):

        self.x = x

        self.y = y

        self.width = 20

        self.height = 20

        self.color = YELLOW

        self.type = type

        self.active = True

    def move(self):

        self.y += 3

        if self.y > HEIGHT:

            self.active = False

    def draw(self):

        if self.active:

            pygame.draw.rect(screen, self.color, (self.x, self.y, self.width, self.height))

# Generate blocks

def create\_blocks(rows, cols):

    blocks = []

    block\_width = WIDTH // cols

    block\_height = 30

    for row in range(rows):

        for col in range(cols):

            block = Block(col \* block\_width, row \* block\_height, block\_width - 5, block\_height - 5)

            blocks.append(block)

    return blocks

# Main game loop

def main():

    paddle = Paddle()

    ball = Ball()

    blocks = create\_blocks(5, 10)

    upgrades = []

    running = True

    font = pygame.font.Font(None, 36)

    while running:

        screen.fill(BLACK)

        # Handle events

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                running = False

        keys = pygame.key.get\_pressed()

        paddle.move(keys)

        # Update ball

        ball.move()

        # Collision with paddle

        if (paddle.y < ball.y + ball.radius < paddle.y + paddle.height and

                paddle.x < ball.x < paddle.x + paddle.width):

            ball.dy \*= -1

        # Collision with blocks

        for block in blocks:

            if not block.destroyed and block.x < ball.x < block.x + block.width and block.y < ball.y < block.y + block.height:

                block.destroyed = True

                ball.dy \*= -1

                if random.random() < 0.3:  # 30% chance to drop an upgrade

                    upgrade\_type = random.choice(["longer\_paddle", "faster\_paddle"])

                    upgrades.append(Upgrade(block.x + block.width // 2, block.y + block.height // 2, upgrade\_type))

        # Update upgrades

        for upgrade in upgrades:

            if upgrade.active:

                upgrade.move()

                if (paddle.y < upgrade.y + upgrade.height < paddle.y + paddle.height and

                        paddle.x < upgrade.x < paddle.x + paddle.width):

                    paddle.collect\_item(upgrade)

                    upgrade.active = False

        # Check if ball falls off screen

        if ball.y > HEIGHT:

            paddle.lives -= 1

            ball.x, ball.y = WIDTH // 2, HEIGHT // 2

        # Draw everything

        paddle.draw()

        ball.draw()

        for block in blocks:

            block.draw()

        for upgrade in upgrades:

            upgrade.draw()

        # Draw lives and items

        lives\_text = font.render(f"Lives: {paddle.lives}", True, WHITE)

        screen.blit(lives\_text, (10, 10))

        items\_text = font.render(f"Items: {len(paddle.items)}", True, WHITE)

        screen.blit(items\_text, (10, 40))

        # End game if lives run out

        if paddle.lives <= 0:

            running = False

        pygame.display.flip()

        clock.tick(60)

    pygame.quit()

    sys.exit()

if \_\_name\_\_ == "\_\_main\_\_":

    main()