Python Project Fall 2025

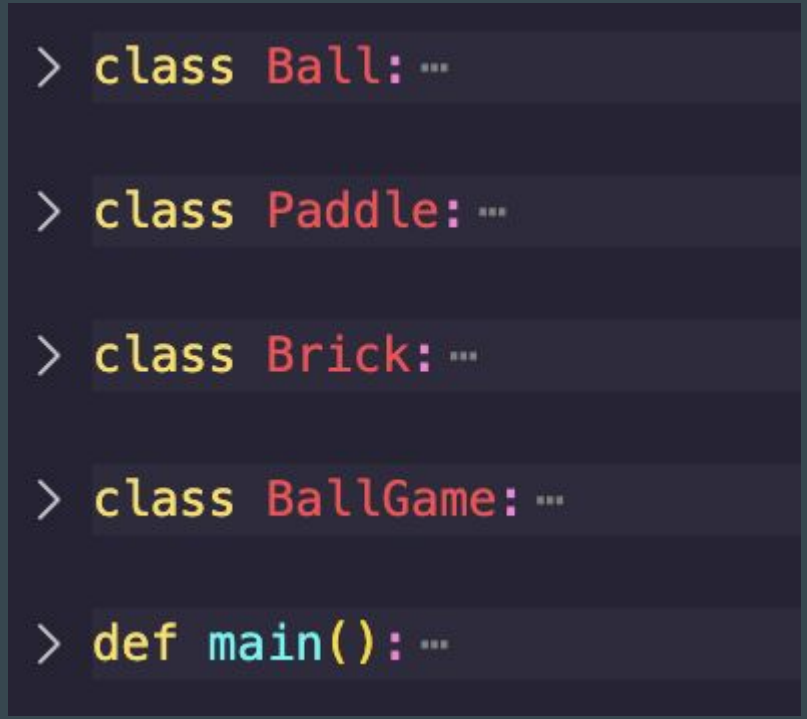
Chase Ripley

Idea

- I wanted to create a simple and fun mini game with live interaction
- Inspired by mini games I’ve played before with a twist
- Used entirely pygame

Overview

- 4 main classes
  - Ball
    - Update position
    - Display
  - Paddle
    - Slide interactively
    - Display
  - Brick
    - Only used for objects
  - BallGame
    - Event loop
    - Generate bricks
    - Check collisions
    - Game over screen



Global Set Up

```python
import pygame
import sys
import random

# initialize pygame
pygame.init()

# screen dimensions
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 700

# high score for the session
GLOBAL_HIGH_SCORE = 0
```

```python
# colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
BLUE = (0, 0, 255)
YELLOW = (255, 255, 0)
ORANGE = (204, 102, 0)

# create screen and game setup
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption("Ball Game")
clock = pygame.time.Clock()
FPS = 60
def main():
    restart = True
    while restart:
        game = BallGame()
        restart = game.run() # runs again if user wants to restart

    pygame.quit()
    sys.exit()
```

Ball Class

```
class Ball:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.radius = 8
        self.vel_x = 0
        self.vel_y = 0
        self.gravity = 0.5
        self.active = True
```

```
def update(self):
    # apply gravity
    self.vel_y += self.gravity

    # update position
    self.x += self.vel_x
    self.y += self.vel_y

    # bounce off walls
    if self.x - self.radius < 0:
        self.x = self.radius
        self.vel_x = -self.vel_x * 0.5
    if self.x + self.radius > SCREEN_WIDTH:
        self.x = SCREEN_WIDTH - self.radius
        self.vel_x = -self.vel_x * 0.5

    # bounce off top
    if self.y - self.radius < 0:
        self.y = self.radius
        self.vel_y = -self.vel_y * 0.5

    # ball falls off bottom (game over for that ball)
    if self.y > SCREEN_HEIGHT:
        self.active = False
```

```
def __init__(self, x, y):
    self.x = x
    self.y = y
    self.width = 110
    self.height = 12
    self.vel_x = 0

def update(self):
    # update position with velocity
    self.x += self.vel_x

    # prevent paddle from leaving screen
    if self.x < 0:
        self.x = 0
    if self.x + self.width > SCREEN_WIDTH:
        self.x = SCREEN_WIDTH - self.width
```

Paddle Class

BallGame Class Pt 1

```python
def __init__(self):
    self.score = 0
    self.game_over = False
    # place paddle centered horizontally using its width
    self.paddle = Paddle(SCREEN_WIDTH // 2 - 55, SCREEN_HEIGHT - 30)
    # put ball above paddle and launch upwards at start
    self.ball = Ball(SCREEN_WIDTH // 2, SCREEN_HEIGHT - 40)
    self.ball.vel_y = -28
    self.bricks = []
    self.generate_random_bricks()
```

```python
def update(self):
    # update game objects and check for game over
    if not self.game_over and self.ball.active:
        self.ball.update()
        self.paddle.update()
        self.check_collisions()
    elif not self.game_over and not self.ball.active:
        global GLOBAL_HIGH_SCORE
        if self.score > GLOBAL_HIGH_SCORE:
            GLOBAL_HIGH_SCORE = self.score
        self.game_over = True
```

```python
def draw(self):
    screen.fill(BLACK)

    # draw game objects
    self.ball.draw(screen)
    self.paddle.draw(screen)
    for brick in self.bricks:
        brick.draw(screen)

    # draw ui
    font = pygame.font.Font(None, 36)
    high_score_text = font.render(f"High Score: {GLOBAL_HIGH_SCORE}", True, WHITE)
    score_text = font.render(f"Score: {self.score}", True, WHITE)
    screen.blit(high_score_text, (10, 10))
    screen.blit(score_text, (10, 40))

    # draw game over screen
    if self.game_over:
        game_over_text = font.render("GAME OVER!", True, RED)
        game_over_rect = game_over_text.get_rect(center=(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2))
        screen.blit(game_over_text, game_over_rect)

        small_font = pygame.font.Font(None, 24)
        high_score_final = small_font.render(f"High Score: {GLOBAL_HIGH_SCORE}", True, WHITE)
        restart_text = small_font.render("Press R to restart", True, WHITE)
        high_score_rect = high_score_final.get_rect(center=(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 40))
        restart_rect = restart_text.get_rect(center=(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 70))
        screen.blit(high_score_final, high_score_rect)
        screen.blit(restart_text, restart_rect)

    pygame.display.flip()
```

BallGame Class Pt 2

```python
def handle_input(self):
    # poll user input events and move paddle or request restart/quit
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            return False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                self.paddle.vel_x = -12
            if event.key == pygame.K_RIGHT:
                self.paddle.vel_x = 12
            if event.key == pygame.K_r and self.game_over:
                return "restart"
        if event.type == pygame.KEYUP:
            if event.key == pygame.K_LEFT or event.key == pygame.K_RIGHT:
                self.paddle.vel_x = 0
    return True
```

```python
def run(self):
    # main loop to run the game. handle input, update, draw at fixed framerate
    running = True
    while running:
        result = self.handle_input()
        if result == False:
            running = False
        elif result == "restart":
            return True

        self.update()
        self.draw()
        clock.tick(FPS)
    return False
```

Note:
BallGame also contains methods for brick generation and collision handling

Let’s play the demo!