Exercise 5 Report

David Malášek
Born: 13 May 2004

November 12, 2025

DVGA12 Programming and Data Structures
Computer Science
Faculty of Health, Science and Technology

I hereby affirm that this report represents my own independent work and that I have neither given nor received unauthorized assistance in it's completion.

Contents

1 Introduction

Exercise 5 is a project focused on implementing a Breakout game clone in Python. According to the project assignment, Breakout game consists of a game board that has a number of bricks in the upper part, a user-controlled bat at the bottom and a bouncing ball that moves around. The objective is to bounce the ball against the bricks to destroy them and stopping the ball from falling below the bottom of the game board with the bat.

The goal of the project is to practice object oriented programming and working with the `pygame` library.

2 Overview

This section presents descriptions of individual classes used in the implementation supported by a class diagram.

2.1 Sprite

This class serves as the superclass for all of the game objects. It defines the basic attributes: `x`, `y`, `width`, `height`, `color` and `rect`, which is an instance of `pygame.Rect` representing the object's position and dimensions. The class also provides the constructor `__init__(x, y, width, height, color)` for initializing mentioned properties. It's methods include `draw(surface)`, which renders the sprite on the specified surface and `update_rect()`, which synchronizes the `rect` object with the current coordinates of the sprite.

2.2 Bat

The `Bat` class represents a paddle, the only object that is movable by the player and it inherits from the superclass `Sprite`. In addition to the inherited attributes, it introduces a specific attribute `speed`, defining the movement speed of the paddle. The constructor `__init__(x, y, width, height, color, speed)` initializes both the inherited and new attributes. The method

`update(screen)` processes keyboard input to control horizontal movement and ensures that the paddle remains within the screen boundaries.

2.3 Ball

This class represents a ball. Besides the inherited attributes, it defines following additional attributes: `radius`, `speed_x`, `speed_y` and an updated `rect` suitable for circular collision detection. The constructor `__init__(x, y, radius, color, speed_x, speed_y)` initializes all of the parameters. The method `update(screen)` handles the movement of the ball and the bouncing logic upon collision with the screen edges, while `draw(surface)` renders the ball.

2.4 Brick

The `Brick` class represents the destructible blocks within the game. It introduces two new attributes, `hits` and `points`, to store the number of collisions required to destroy the brick and the score points awarded upon destruction. The constructor `__init__(x, y, width, height, color, hits=1, points=10)` initializes these parameters. The class includes the method `hit()`, which decrements the number of remaining hits, darkens the color to indicate damage and returns a boolean value indicating whether the brick has been destroyed. The method `draw(surface)` renders the brick.

2.5 Class diagram

Here is a class diagram in UML notation based on the description above.

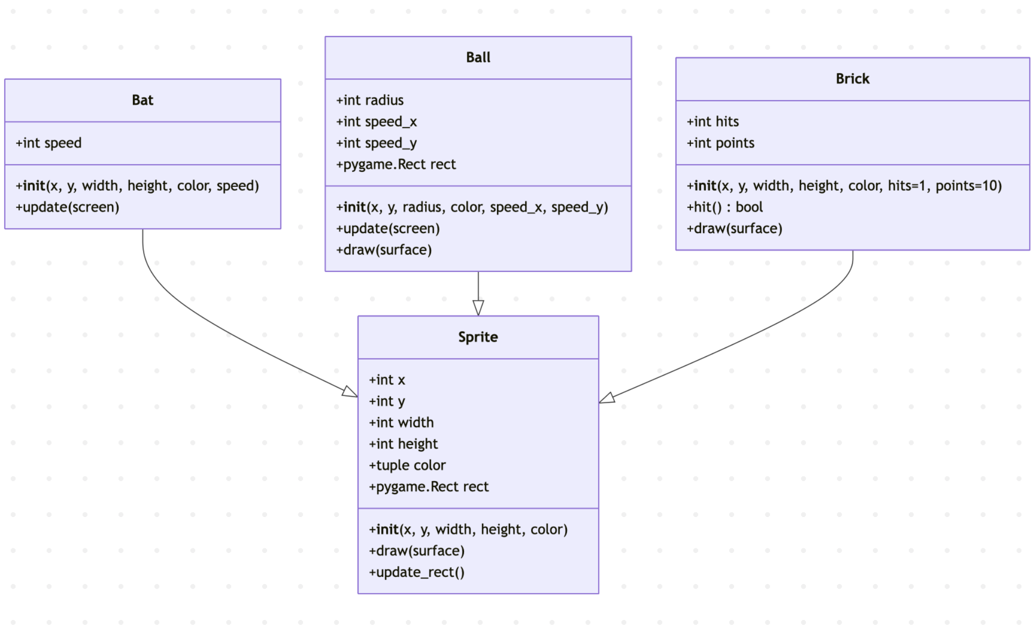

Figure 1: UML diagram

3 Interesting details

3.1 Collision Detection

This project implements collision detection using the `pygame.Rect` class and it's method `colliderect()`, which determines whether two rectangular areas overlap. This technique allows the program to detect interactions between the ball, bat and bricks. Collision detection is a necessary part of the bouncing, scoring and object destruction logic.

4 Spent time

The development of this project required approximately 3 days of work, excluding the time spent on writing the report. While I already had prior experience with the Python programming language, the `pygame` library was new to me. Given this fact, I had to dedicate a portion of the development time to exploring it's documentation, tutorials and examples to understand how the library handles rendering, event management and timing. Eventually, I had to redo some parts of the project, because I discovered a more efficient approach.

5 Lessons learned

Interesting part of the project was designing and implementing the logic that handles the ball's movement. I had to analyze how the horizontal and vertical speed values (`speed_x`, `speed_y`) interact to produce smooth diagonal trajectories and realistic bouncing behavior after collisions.

Additionally, I learned the importance of controlling the game's frame rate to ensure consistent performance across different systems. By setting the frame rate to 60 frames per second, the gameplay became noticeably smoother and more visually stable.