CS371 Team Project—OOP Shooter Game

Due on 12/1/2023

For this project, you're tasked with creating a mobile shooter game app, building upon the OOP shooter demo we discussed in class. Essential tools for this project include the Composer and Physics libraries, along with Lua's OOP principles. Begin by downloading the necessary project files.

You are allowed to alter the OOP shooter for this project and also download and update '1_OOPShooter.zip', 'OOPShooterWithKirby.zip' and 'Scrolling_demo.zip' from our class's Canvas Modules.

Basic Requirement:

Configure the game for iPhone X, using a resolution of 640x1163, and ensure the status bar is excluded. Adjust the sizes of the game objects to be scaled and visually appropriate for this specified resolution.

The app should be oriented in landscape mode.

This game app is structured around two distinct scenes: 1) Title and 2) Game.

1. Title Scene:

- Display a [Start] button that users can tap to initiate the game.
- Provide a list of your team members' names.

2. Game Scene:

Horizontal Parallax Scrolling:

- Initiate parallax scrolling using two background images provided in the package: farback.gif and starfield.png. Download them from Canvas Modules.
- starfield.png should scroll faster than farback.gif.
- As the game progresses, the background images should scroll horizontally. You have the option to utilize Scrolling_demo.zip available in the Files section of Canvas, or you can code a better scrolling function.

Game objects:

- Player Character (PC) Specifications:
    - The PC does not require a sprite, but it should be easily distinguishable from the background.
    - **Lua OOP is not necessary for implementing the Player character.**
    - The PC should be capable of spawning projectiles. (You can reuse the code from the demo)

    - The PC has a hit point (HP) value of 5, which will decrease upon collision with Enemy objects or Enemy bullets.
    - When the PC's HP reaches zero, a 'game over' message should be displayed.
    - The PC can only move horizontally. Users can manage this movement via a touch-enabled interface (control box; the gray regions in Figure 1 & 2). The necessary code for this feature has been implemented in the demo and is available for reuse.

- Implementation of Two Enemy Types (Enemy1.lua & Enemy2.lua):
    - There is no need for sprites for enemies, but each enemy should be clearly distinct from the background and another enemy.
    - Two distinct types of Enemies are required. Each type should inherit variables and methods from the base Enemy class. However, you will need to create two subclasses and adapt or override certain methods for each subclass as described below.
    - The enemy types and their corresponding hit points (HP) are:
        - Enemy 1: 2 HP
        - Enemy 2: 3 HP
    - When an Enemy's HP drops to zero, that Enemy object should be removed from both the screen and memory.
    - If any Enemy object moves off-screen, it must also be removed from memory.
    - The two enemies should spawn randomly at regular intervals, with an equal 50% chance for each. It is possible for multiple Enemies to appear simultaneously.
    - Each Enemy type will have distinct spawn() and move() methods. These methods must be overridden based on the Enemy type:
        - Enemy 1 (Enemy1.lua):
            - spawn(): Create a distinct shape for Enemy 1.
            - move(): The starting y-position should be randomly assigned, while the starting x-position should be off-screen, specifically on the right side. This type of Enemy will move as illustrated in Figure 1.
        - Enemy 2 (Enemy2.lua):
            - spawn(): Create a distinct shape for Enemy 2.
            - move(): Similarly, The starting y-position should be randomly assigned, while the starting x-position should be off-screen, specifically on the right side. The Enemy2 should move towards the PC object. This type of Enemy will move as illustrated in Figure 2.
    - The speed of both enemies should be adjusted to ensure that the gameplay is sufficiently challenging.

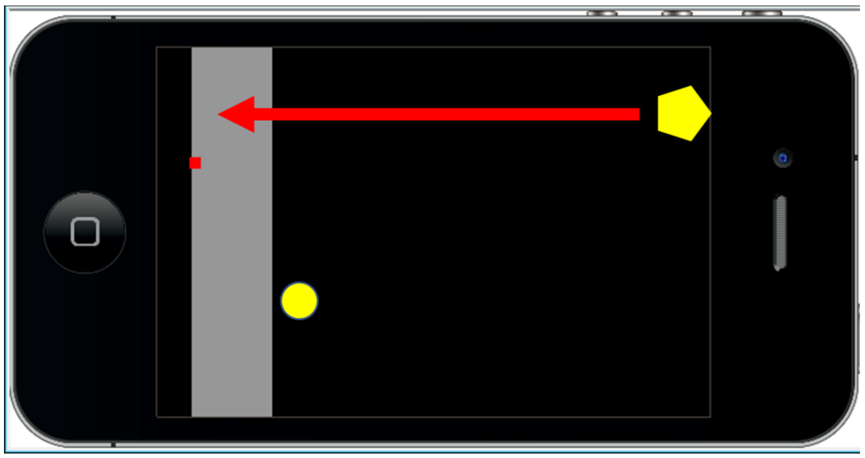
Figure 1. Enemy1's movement. You don't need to follow these shapes.

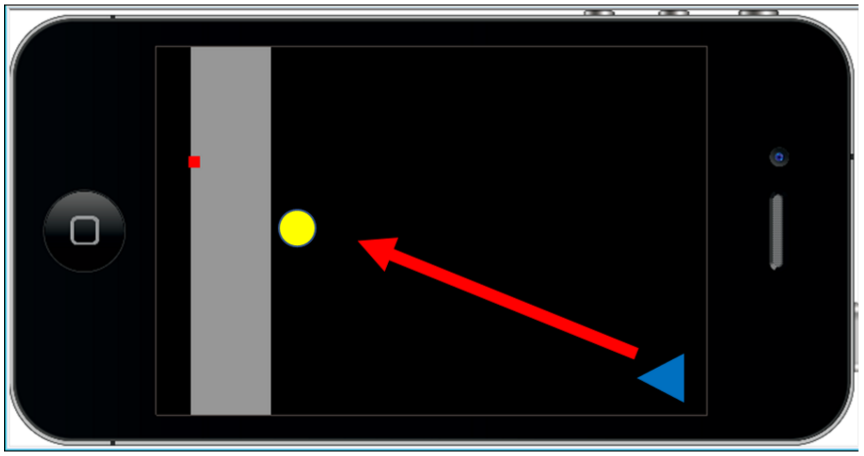
Figure 2. Enemy2's movement. You don't need to follow these shapes.

- Boss Implementation (Boss.lua):
  - Utilize your 'Keen Bayonet' from Assignment 2 as the boss character.
  - Similar to its implementation in Assignment 2, the Keen Bayonet keeps moving to random positions.
  - After 2 minutes of play, cease the spawning of the enemies, and the Keen Bayonet should appear as Boss.
  - Boss should spawn some projectiles/bullets toward PC.
  - The Boss should extend the functionality of the Enemy class (Enemy.lua). You will need to develop two subclasses for the Boss, adapting or overriding certain methods as required.
  - Assign a hit point (HP) value of 30 to the Boss.
- Collision Handling:
  - Manage collision events between the following entities (Collision Filter usage is not mandatory):

    - Between PC's bullet and Enemy (and Boss) (decrease Enemy's HP).
    - Between PC and Enemy (decrease PC’s HP).
    - Between PC and Boss
    - Between PC and Boss’s bullet
- Sound Effects:
    - Implement distinctive sound effects for each type of collision event.
- HUD (Heads-Up Display):
    - Display the score on the screen.
        - Each destroyed enemy increases the score by 100.
        - Destroying the boss should boost the score by 10,000.
    - Display PC’s HP.
- Winning the Game and Game Over:
    - If the PC survives after the boss fight, display a 'Congratulations' message and return to the start screen if the user taps on the screen.
    - If the PC’s HP falls to 0, show a 'Game Over' message and return to the start screen if the user taps on the screen.

Project Report:

- List all team members' names and their specific contributions to the game.
- Include source codes.
- Provide multiple screenshots of the app.

Instructions for submission and notes regarding grading are the same as the previous HW.