Programming Assignment 2

The goal of this assignment is for you to gain more familiarity with the JavaScript programming language. Specifically, I want you to gain experience developing front end JavaScript code.

You and one group member will be tasked with creating a game in JavaScript. The game will be a top down scrolling space shooter. You will earn points for this assignment by implementing the specific features of the game. To assist in programming I have given you a small amount of code to start with. You may choose to not use the provided code. The base code provides examples for creating the game loop, setting up the canvas, handling input, and rendering a simple triangle to the canvas.

A few reminders about how groups work in CS316

- When you decide on a group you must inform me of your decision
- You may not switch groups throughout the semester

Feature Descriptions

Pay careful attention to feature descriptions! You need to make sure you implementation behaves exactly how it is described.

Player Movement

The player should be able to move in eight directions. Assuming you use the default input configuration, the WASD keys will be bound. `W` should move the player up. `D` should move the player right. `S` should move the player down. `A` should move the player left. Then a combination of two keys should move the player diagonally. For example, if the `W` and `D` keys were pressed at the same time the player should move up and to the right at a 45 degree angle (you do not need to rotate the player's sprite).

The player's velocity must always remain constant unless they are not moving. This means that the diagonal movement speed must be equal to the primary movement speed.

Enemies

You are welcome to implement more complex patters than what will be described (depending on the complexity this could earn you extra credit). However, you will earn full credit for implementing the following behavior.

- Enemies should spawn above the top border of the canvas. This means that when an enemy spawns they shouldn't be immediately visible (since they will be off the screen).
- Enemies should spawn at some interval. Example, 10 enemies per second.
- Enemies should move vertically towards the bottom of the canvas. Their velocity can remain constant for their entire lifespan.
- When enemies pass the bottom border of the canvas they should be removed from the entity map.

If you use the starting code there is a variable called `enemy_spawner`. By default it is ignored when the value is null. When assigned a value it is called during the update loop. You do not need to use this variable, it is only there to show you were in the update loop enemies should be spawned. If you want to use the existing variable, assign it equal to an object that has an `update(delta_time)` method.

Collision Handling

To simplify the collision handling process you may assume that all bodies are axis aligned rectangles (even if they are not rendered as rectangles). This means that you can use a simple AABB (Axis-Aligned Bounding Box) collision check between all of the entities. I am not grading how efficient your solution is. An acceptable solution is to have a double for loop where you do an AABB collision check between every body pair.

If you are not familiar with this check, mozilla has documentation showing how it can be done.

The type of collision events to detect are listed below (you will need to add more for later features)

- Player colliding with an enemy
    - the player should take some amount of damage. the amount does not matter
- Enemy colliding with player
    - the enemy should be removed from the entity map

A common bug is checking for collision between a body and itself. So when doing your comparisons be sure to ignore the case when both bodies are the same.

If you use the starting code there is a variable called `collision_handler`. By default it is ignored when the value is null. When assigned a value it is called during the update loop. You do not need to use this variable, it is only there to show you where in the update loop collision handling should be done. If you want to use the existing variable assign it equal to an object that has an `update(delta_time)` method.

Player Combat

You are welcome to implement more complex combat that what will be described (depending on the complexity this could earn you extra credit). However, you will earn full credit for implementing the following behavior.

- When the space bar is held the player fires projectiles. Hint: The input handler already checks for the space bar.
- These projectiles move in a vertical line starting from the position of the player to the top of the canvas.
- There should be some kind of cool down between projectiles. Example, the player can only fire one projectile a second.
- When a projectile hits an enemy the enemy is removed from the world.

Player Stats

You are welcome to implement more complex player statistics than what is described (depending on the complexity this could earn you extra credit). However, you will earn full credit for implementing the following behavior.

You must keep track of the following player statistics

- number of enemies hit by projectile
- how long, in seconds, the player has been alive
- total number of enemies spawned
- a score that should be computed as follows
    - score = Math.floor(30 * num_enemies_hit + seconds_alive)

You must also display this information to the user in some way. How you accomplish this is up to you, but I recommend you render the text to the canvas.

These stats should reset when the player is defeated and the game is restarted.

Rubric

You will submit your program as a zip file in canvas. Part of your grade will include you coming to my office hours to demo the game. If you are not available during my office hours we can schedule a time for you to demo it. The demo should only be a few minutes long. Both members of your team do not need to be present. Points will be assigned as follows.

- total 90 pts: all feature implementations
    - total 10 pts: player movement
        - 8 pts: behaves as specified
        - 2 pts: relevant JSDoc
    - total 25 pts: enemies
        - 23 pts: behaves as specified
        - 2 pts: relevant JSDoc
    - total 20 pts: collision handling
        - 18 pts: behaves as specified
        - 2 pts: relevant JSDoc
    - total 25 pts: player combat
        - 18 pts: behaves as specified
        - 2 pts: relevant JSDoc
    - total 10 pts: player stats
        - 8 pts: behaves as specified
        - 2 pts: relevant JSDoc
- 10 pts: demo

Extra Credit Opportunity

You may earn extra credit by choosing to go beyond the minimum scope of this project and implement additional features. I ask that you discuss you extra credit idea with me so that I can confirm it is sufficient. You are welcome to do whatever you want, but here are a few ideas to get you started.

- high score board
- use images instead of triangles
- different player projectile patterns
- more complex enemy behavior
- projectiles that slow you down