CS25320 Assignment Report

Contents

1. Overview

The game I chose as my starting point was falling_stuff.html [1]. I saw it as a good base for an old school side-scrolling game, similar to a much more basic version of Thunder Force for the Sega Mega Drive, with some of the basic functionality already in place. For the type of game I wanted to make, the player would only need to be able to move along one axis, so I didn't need the full mouse functionality that collision2.html provided. I also knew I could repurpose the falling boxes to work as enemies, that could collide with the player to damage their health and that the player could shoot to score points.

To add more mechanics to the game I created powerup objects that the player can pickup to alter the game's properties, such as a shield that will absorb damage the player would normally receive, or the mining laser which augments the player's weapon to fire one large continuous shot. In addition to these powerups, the game also has a chance of spawning health pickups that the player can collect to regain lost health.

The session ends when the player's health reaches zero, at which point a game over screen is triggered, where the player is greeted with their final score and prompted to enter their name in a textbox below to save their score in the leaderboard.

2. Process

It is crucial that before I could start to manipulate any of the provided code, I had to familiarize myself with the provided code by combing through it and making sure I understand what parts accomplish what, and how they all fit together. This is important to ensure I do not end up writing any unnecessary code or try implement any solutions that will end up clashing with the current foundations.

I also made sure to get well acquainted with the list of functional requirements provided to us within the brief, to make sure I was hitting the right goals along the way.

Throughout the project version control was something I paid close attention to, as in previous modules I have made mistakes that have broken parts of the code without realizing until later on, causing lots of avoidable stress. The method I chose for version control was to just make copies of the files each time significant changes had to be made or added, and increment the number at the end of the file name. I would also write the biggest changes from version to version in the annotations at the head of the code, so I would not forget my goals between work sessions.

3. Development and Deployment Environment

To develop my game I primarily used Visual Studio Code (VSC). I chose VSC because it is the IDE that I am most familiar with, and has great support for HTML, CSS, JavaScript and PHP, which can all be further improved with the use of extensions. To test the game, I used the extension “Live Server” for VSC. This allows the user to create a local development server for testing webpages, and saves the user time by automatically refreshing the page when any changes are saved to the code.

Before uploading the files to public_html I also ensured that my University account had PHP permissions enabled in the account settings, as the game would be inaccessible without the correct read and write permissions.

4. Design

As the planned game was more complex than the original “falling_stuff” [1] game I had to add and alter attributes of most of the objects in the game. I added health, score and state variables to the Box constructor. The health variable is to be constantly read to display the player’s health points on the heads-up-display (HUD), and the value incremented if a player touches a health pickup, or decreased when the player is hit by a falling box. Score was also necessary to calculate how much score each hit falling box is worth. The value of the score is based on the size of the box. Any object that is going to require collision detection is also given a state variable which is a Boolean that will toggle true if a collision is detected, marking it for deletion from the array.

Instead of having the bullet be the same class as the falling boxes but inverted to travel the opposite direction, I instead made the bullets their own object so it is easier to compare the two when it came time to implement the collision detection. This meant that code could be easily repurposed down the line when it came to added powerup drops.

There are three different collision detection functions that check the positions of the boxes, bullets and powerups. All of these functions are called at different times in the game loop, and code located within the game loop is executed if collision is detected.

The powerups are spawned using a similar method to the “maybe_add_random_falling_box” function provided, except a random number is generated which is used as the index to access the “pickup_types” and randomly selected a type of powerup. The properties of the powerup are different depending on which type is selected.

5. Implementation

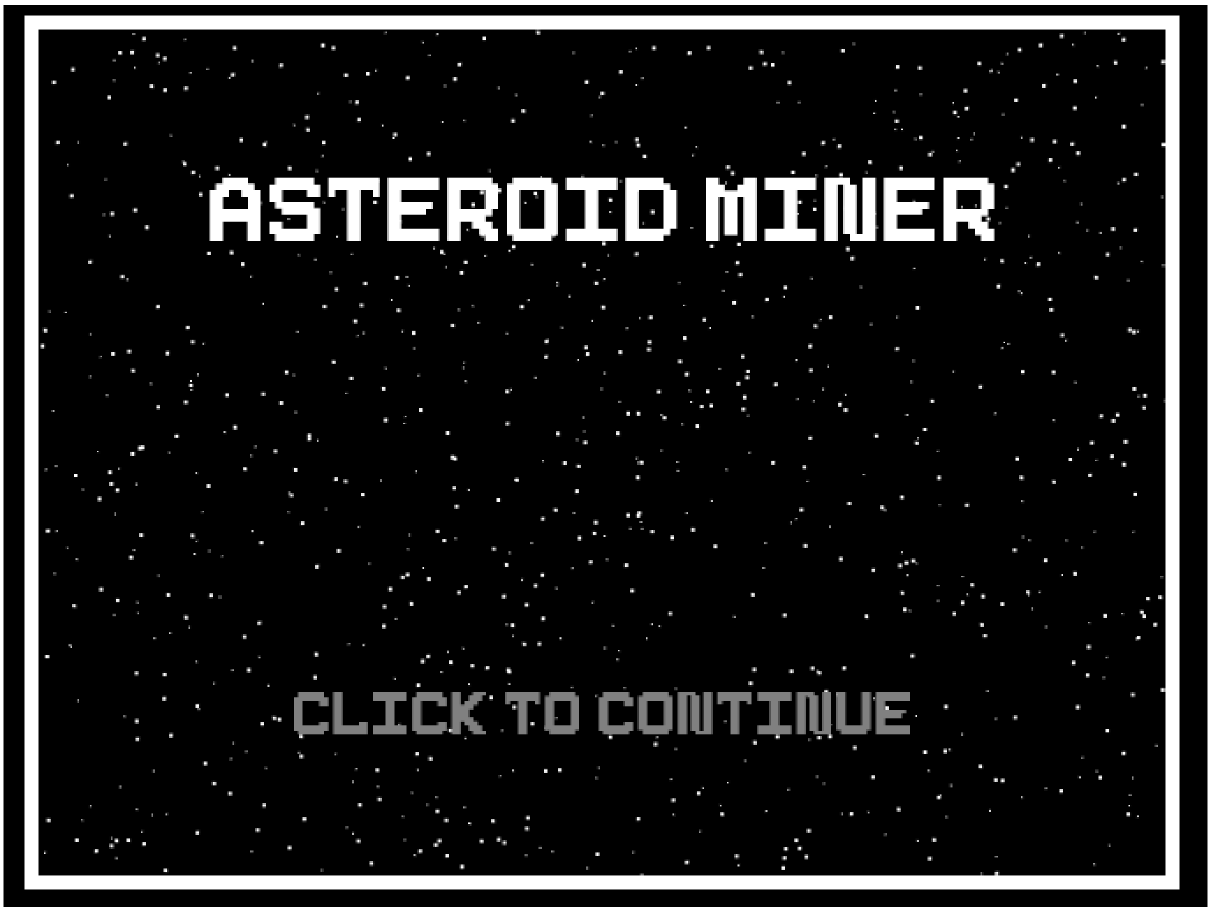


Figure 1: Splash screen

The splash screen is the first screen the user will see when they load the webpage. The stars are randomly drawn each time. When the user clicks the canvas they will be taken to the main menu.

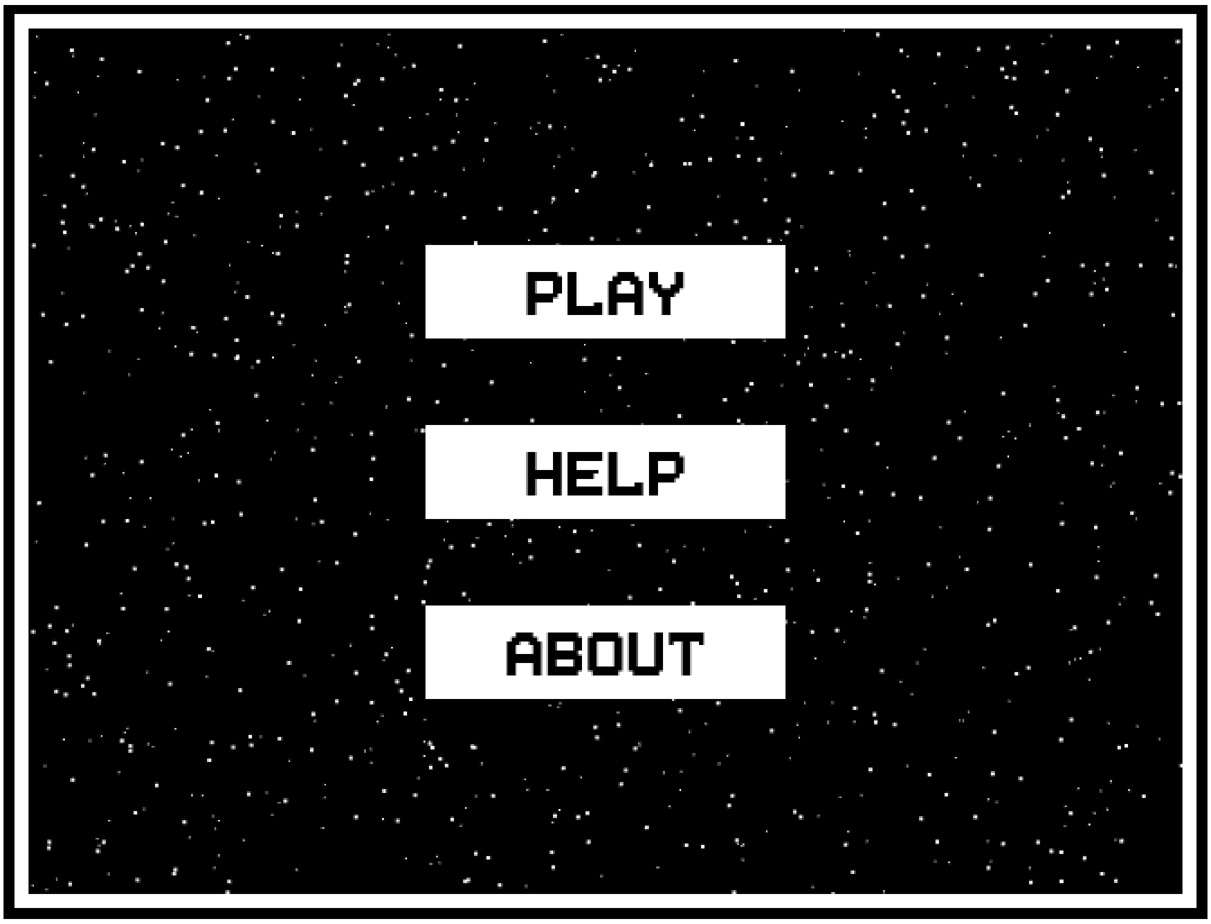
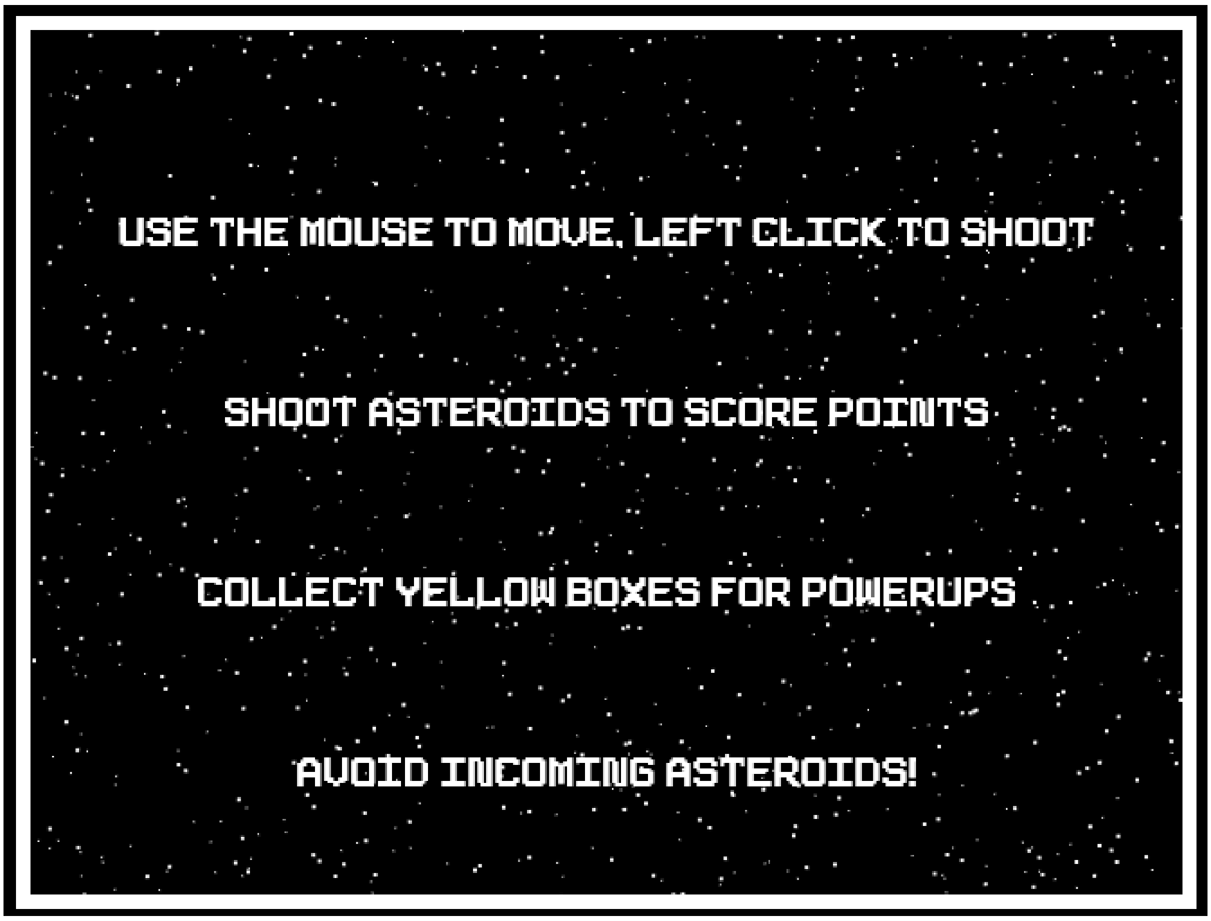


Figure 2: Main menu

This is the main menu. I'm happy with the design visually, but the buttons on the canvas have no working functionality currently, so the menus must be navigated using the HTML buttons located above the canvas.


USE THE MOUSE TO MOVE, LEFT CLICK TO SHOOT

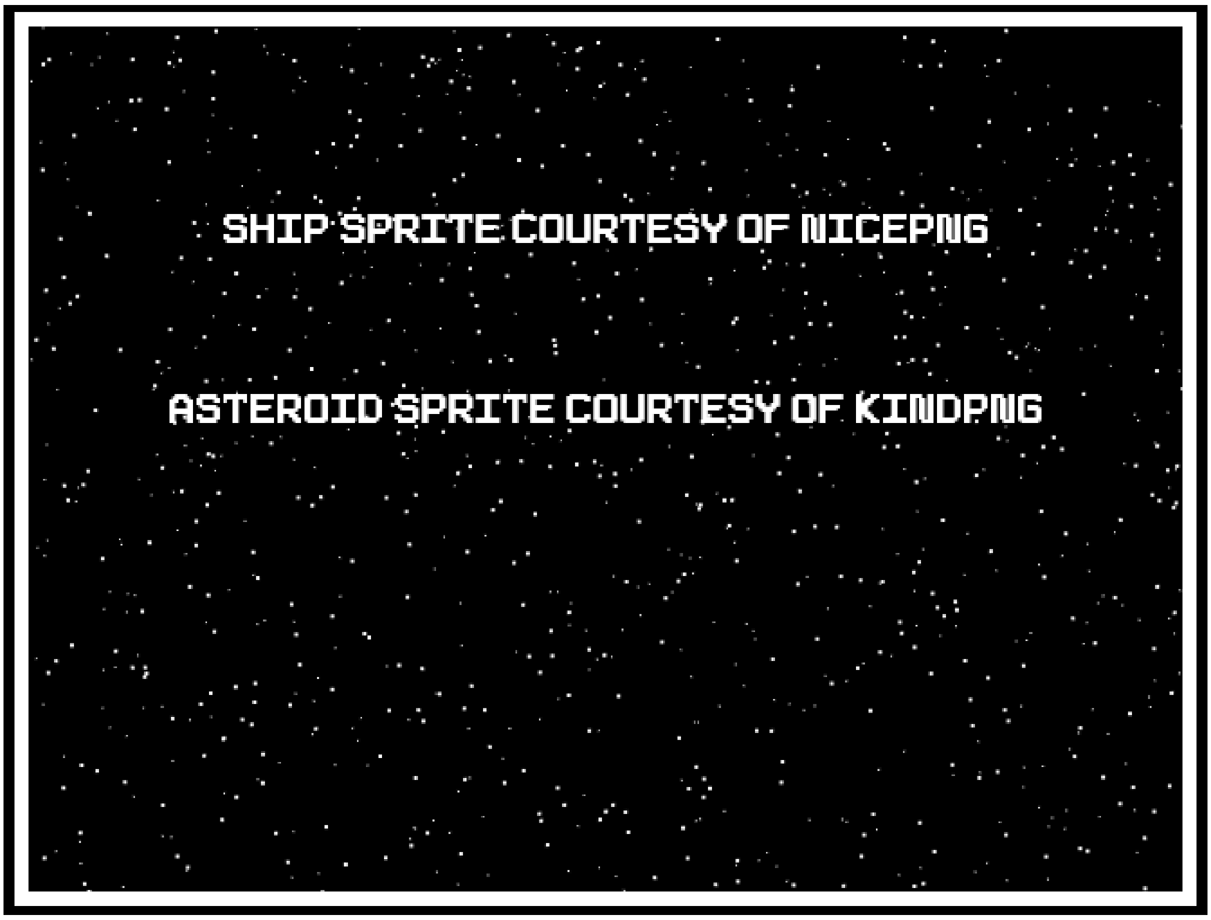
SHOOT ASTEROIDS TO SCORE POINTS

COLLECT YELLOW BOXES FOR POWERUPS

AVOID INCOMING ASTEROIDS!

Figure 3: Help page

The above screenshot shows the help page, which contains instructions on how to play the game.



SHIP SPRITE COURTESY OF NICEPNG

ASTEROID SPRITE COURTESY OF KINDPNG

Figure 4: About page

The about page contains references for the sprites that have been used in the project.

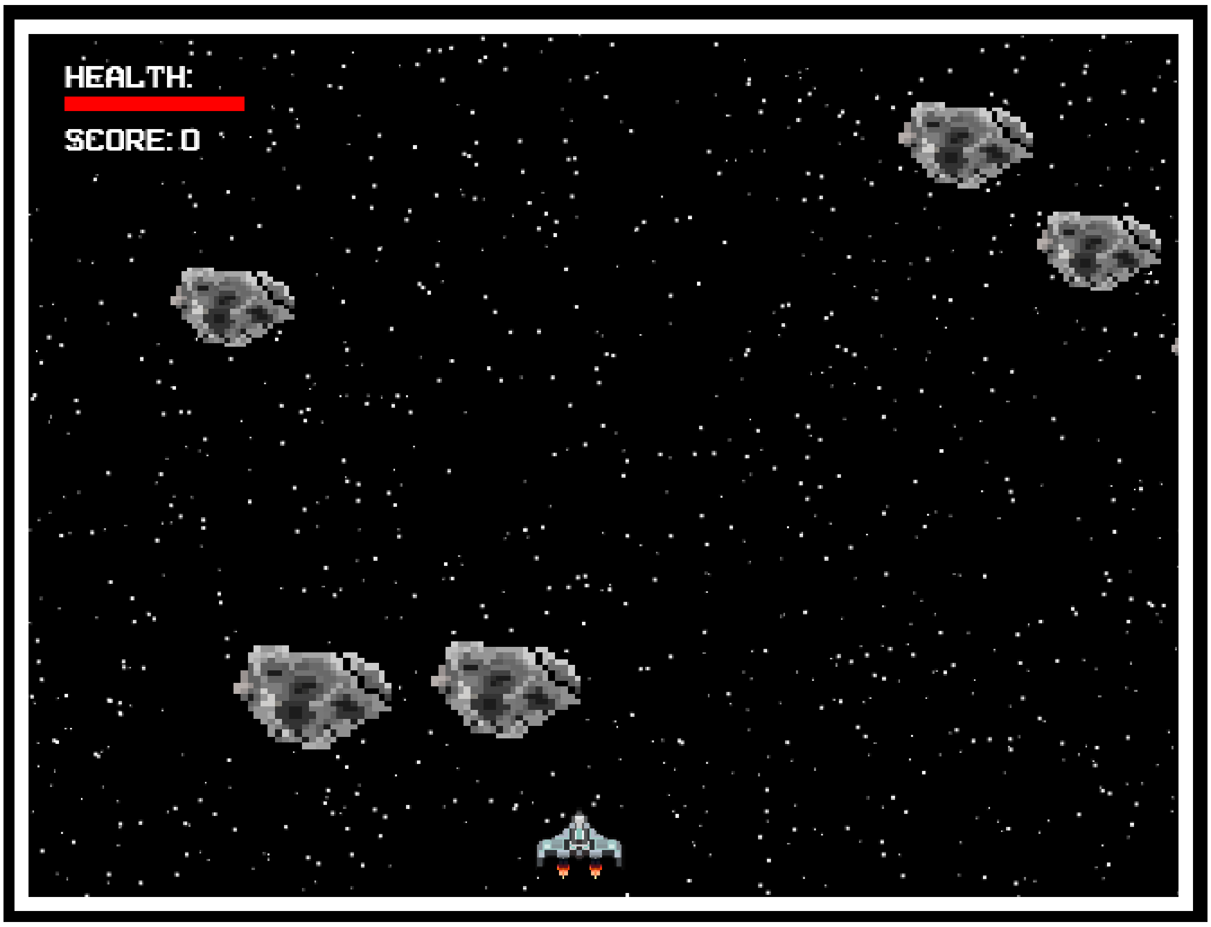


Figure 5: Game in action

This screenshot shows the game running. The asteroids and powerups appear just above the canvas and come into view before dropping down to give the player a chance to collide.

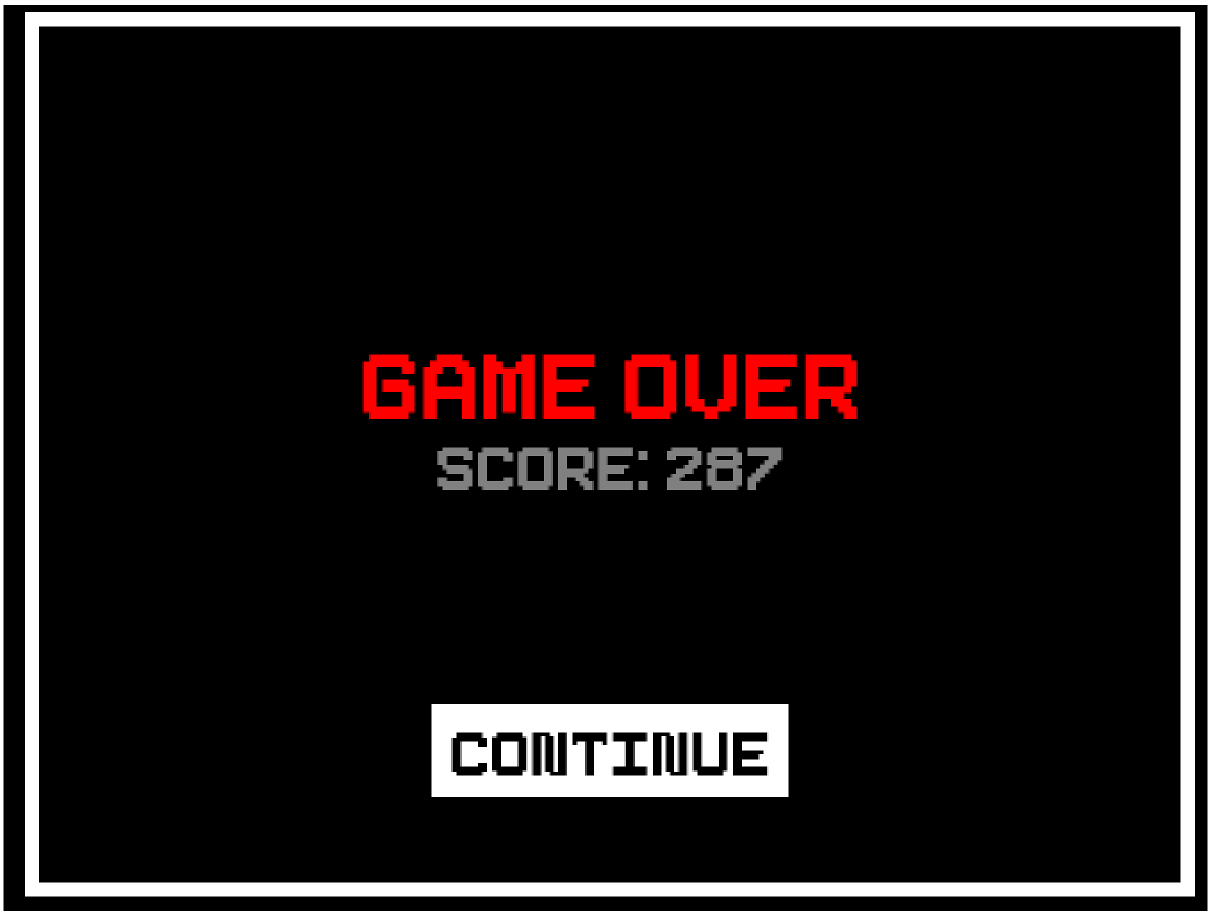


Figure 6: Game over state

The game over screen that appears once the player's health reaches 0. The final score is calculated and displayed.

6. Testing and Deployment

Every time I made any sort of functionality change to the code, I ran the game to see if it worked as intended. I tested to see if the game worked under multiple condition types, including trying to get hit at each possible angle, and shooting as many bullets as possible to see if that would affect performance.

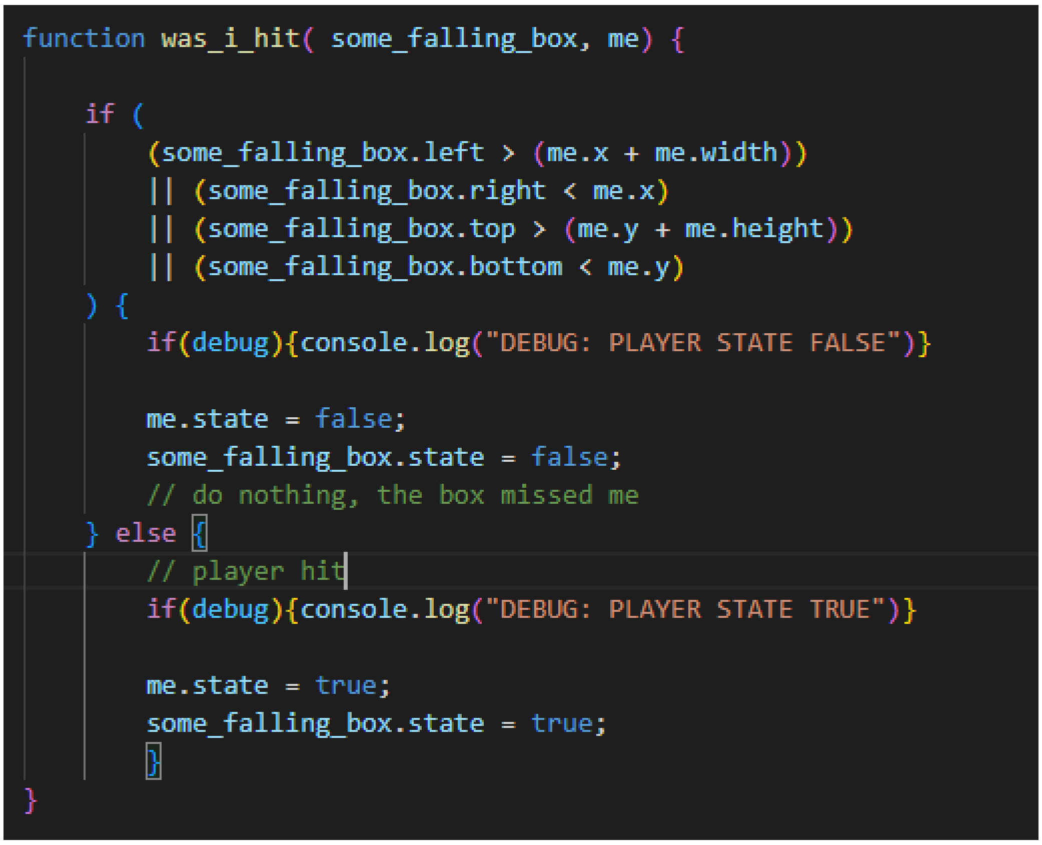
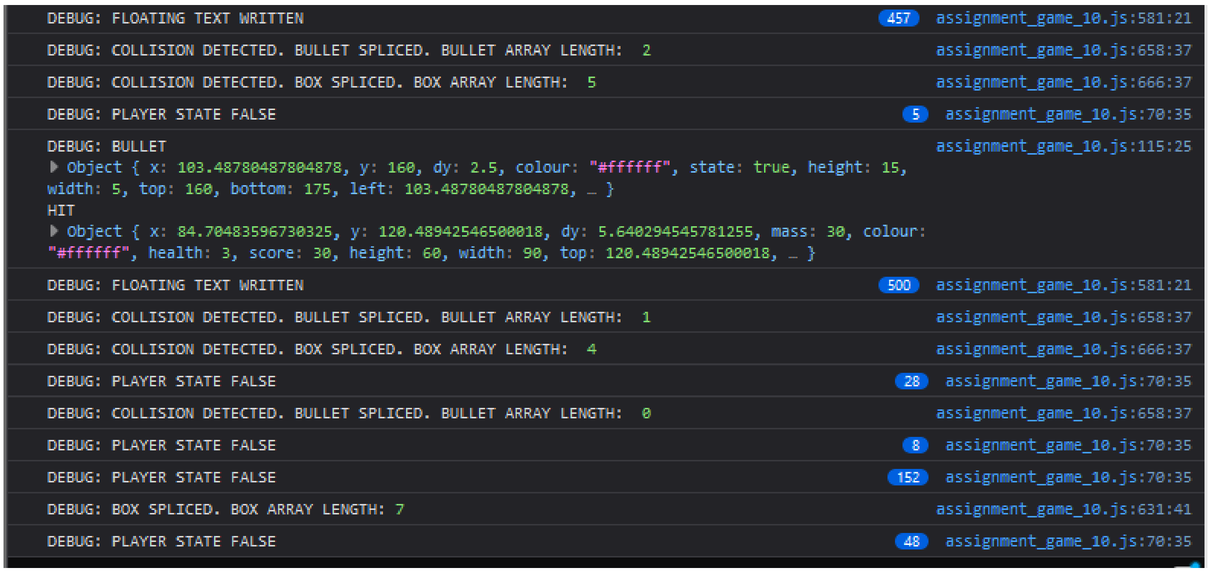

```
function was_i_hit( some_falling_box, me) {

    if (
        (some_falling_box.left > (me.x + me.width))
        || (some_falling_box.right < me.x)
        || (some_falling_box.top > (me.y + me.height))
        || (some_falling_box.bottom < me.y)
    ) {
        if(debug){console.log("DEBUG: PLAYER STATE FALSE")}

        me.state = false;
        some_falling_box.state = false;
        // do nothing, the box missed me
    } else {
        // player hit
        if(debug){console.log("DEBUG: PLAYER STATE TRUE")}

        me.state = true;
        some_falling_box.state = true;
        }
}
```

Figure 7: Debug command in hit detection code

To further help with the testing and debugging process, I also added a debug constant at the head of the code which can be used to toggle various debug console log commands that help keep track of functions being entered and data being passed. This was particularly useful when trying to get the collision detection to work.



```
DEBUG: FLOATING TEXT WRITTEN                                  457 assignment_game_10.js:581:21
DEBUG: COLLISION DETECTED. BULLET SPLICED. BULLET ARRAY LENGTH:  2  assignment_game_10.js:658:37
DEBUG: COLLISION DETECTED. BOX SPLICED. BOX ARRAY LENGTH:  5      assignment_game_10.js:666:37
DEBUG: PLAYER STATE FALSE                                       5 assignment_game_10.js:70:35
DEBUG: BULLET                                                     assignment_game_10.js:115:25
▶ Object { x: 103.48780487804878, y: 160, dy: 2.5, colour: "#ffffff", state: true, height: 15,
width: 5, top: 160, bottom: 175, left: 103.48780487804878, … }
HIT
▶ Object { x: 84.70483596730325, y: 120.48942546500018, dy: 5.640294545781255, mass: 30, colour:
"#ffffff", health: 3, score: 30, height: 60, width: 90, top: 120.48942546500018, … }
DEBUG: FLOATING TEXT WRITTEN                                  500 assignment_game_10.js:581:21
DEBUG: COLLISION DETECTED. BULLET SPLICED. BULLET ARRAY LENGTH:  1  assignment_game_10.js:658:37
DEBUG: COLLISION DETECTED. BOX SPLICED. BOX ARRAY LENGTH:  4      assignment_game_10.js:666:37
DEBUG: PLAYER STATE FALSE                                      28 assignment_game_10.js:70:35
DEBUG: COLLISION DETECTED. BULLET SPLICED. BULLET ARRAY LENGTH:  0  assignment_game_10.js:658:37
DEBUG: PLAYER STATE FALSE                                       8 assignment_game_10.js:70:35
DEBUG: PLAYER STATE FALSE                                     152 assignment_game_10.js:70:35
DEBUG: BOX SPLICED. BOX ARRAY LENGTH: 7                           assignment_game_10.js:631:41
DEBUG: PLAYER STATE FALSE                                      48 assignment_game_10.js:70:35
```

Figure 8: Browser console as game is running

7. Reflection

Overall I am quite happy with the basic functionality of the game. The asteroids and bullets mostly behave as intended. The player takes damage when appropriate, and the correct score is calculated per hit.

One thing I struggled with is the core game_loop() function. Each frame the game_loop() runs through multiple for-loops to determine whether a hit has connected, however I could not find a way to correctly nest each for-loop within each other so they all have access to the required index number at the same time. This means that, for example, the bullets and player will not animate unless the array of falling boxes is greater than zero.

Powerup collision detection is also bugged and I could not find the cause. The code is using the same base hit detection system that the game uses to determine whether the player has collided with an asteroid, which should mostly work without much tweaking but the powerups are marked for deletion too soon.

Menu functionality was a surprisingly difficult task. I thought this would be one of the easier aspects of the game to program but I ended up having to settle for HTML buttons as it was easier to get those to a basic functioning state.

A feature I would love to have implemented would be fully animated sprites. As it stands, the movement of the static sprites is a bit boring to look at. I think full animation would have boosted the presentation of the game considerably.

I also couldn’t get the leaderboard to function as intended in time, with there being only the functionality provided in worksheet 5 behind the leaderboard still.

8. References

[1] *falling_stuff*. Retrieved from **1https://blackboard.aber.ac.uk/bbcswebdav/pid-2341333-dt-content-rid-7728775_1/xid-7728775_1**

[2] *ship sprite.* Retrieved from **https://www.nicepng.com/downpng/u2q8a9y3a9r5i1r5_vector-spaces-ship-8-bit-spaceship-sprite/**

[3] *asteroid sprite.* Retrieved from **https://www.kindpng.com/imgv/wmmowx_pixel-art-asteroid-sprite-hd-png-download/**