Advanced JavaScript – CA1

OOP Game Development

January 23, 2018

N00143888 https://github.com/aaronoh/AdvJSCA1 Aaron O'Hare

Table of Contents

Functionality

This application is built using JavaScript (ES6), the P5 library and HTML. It is basic recreation of the classic 'Space Invaders' game.

The user must first enter the username and click play. At this point the username is stored in local storage for later use. The user will then be brought into the game, they may use the arrow keys to control the ship and spacebar to fire.

There are 30 yellow invaders each worth one point, these travel the width of the canvas in formation before descending on the y-axis. Every 12 seconds a red invader travels across the width of the canvas, this invader is worth ten points.

The game ends when the user eliminates all of the yellow invaders or the invaders reach the users' ship.

Throughout the game the user is presented with a number of statistics, these include their score, the amount of time that they have been playing, the number of shots they have fired and their percentage accuracy.

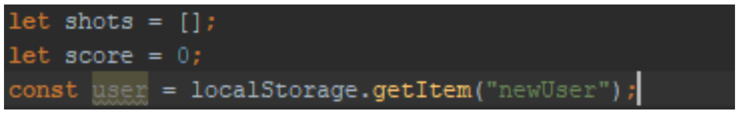
At the end of the game, all of these statistics (along with the users name) are added to the Mongo Database (hosted on MLab). The user is then brought to the end page, here they will see their previous games stats (loaded form local storage) along with the leaderboard, the leaderboard displays the top 5 scores recorded along the users' name and percentage accuracy.

Requirements

The original application was constructed with the aid of a tutorial. This tutorial is referenced below, it did not include any of the database functionality. The finished product of the tutorial was not constructed using let/const/arrow functions,did not span across multiple pages, include any form of username/scoring system, have any form of 'special' enemy, use default parameters, introduce inheritance and only included the first row of 'invaders' (omitting any logic for direction change). I believe I have made sufficient changes to the original tutorial product while also adding all of the features discussed above.

Let and const

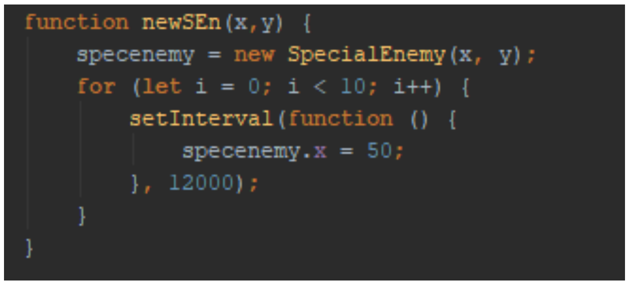
Used consistently throughout the application in order to clearly convey which variables may change and which will not.



```
let shots = [];
let score = 0;
const user = localStorage.getItem("newUser");
```

Function Declarations

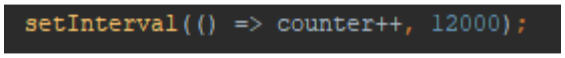
Used consistently throughout the application in order to maintain code readability and to make it easy for changes to be made.



```
function newSEn(x,y) {
    specenemy = new SpecialEnemy(x, y);
    for (let i = 0; i < 10; i++) {
        setInterval(function () {
            specenemy.x = 50;
        }, 12000);
    }
}
```

Arrow functions

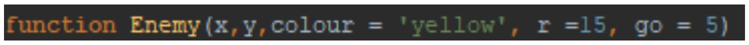
Used where appropriate to shorten the space taken by basic functions. In this instance I have not taken advantage of the fact that they do not have their own 'this'.



```
setInterval(() => counter++, 12000);
```

Default Parameters

Used when constructing the enemy objects. The yellow enemies use the default parameters while the red are set through the function call.



```
function Enemy(x,y,colour = 'yellow', r =15, go = 5)
```

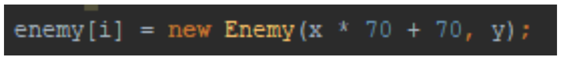
Arrays

The enemies and shots fired are stored in (and removed from) arrays. This allows a shot to be deleted when it leaves the canvas, preserving resources. It also allows the enemies to be placed correctly and to be removed from the canvas when hit.

Both examples shown below are executed within (separate) for loops. In the first example the enemies are created using i as the index, i is iterated up to the defined value (number of enemies), 30.

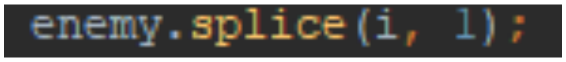
In the second example the for loop works through the array backwards, checking if the shot has been marked for deletion (when a collision is detected the delete (del()) function is called which changes a Boolean value to true, within the for loop discussed above this Boolean value is checked. If it is set to true, the object at index I is removed from the array using splice. Splice requires two arguments, the index of the object and the number of elements to remove (one in this case).

Creating an enemy:



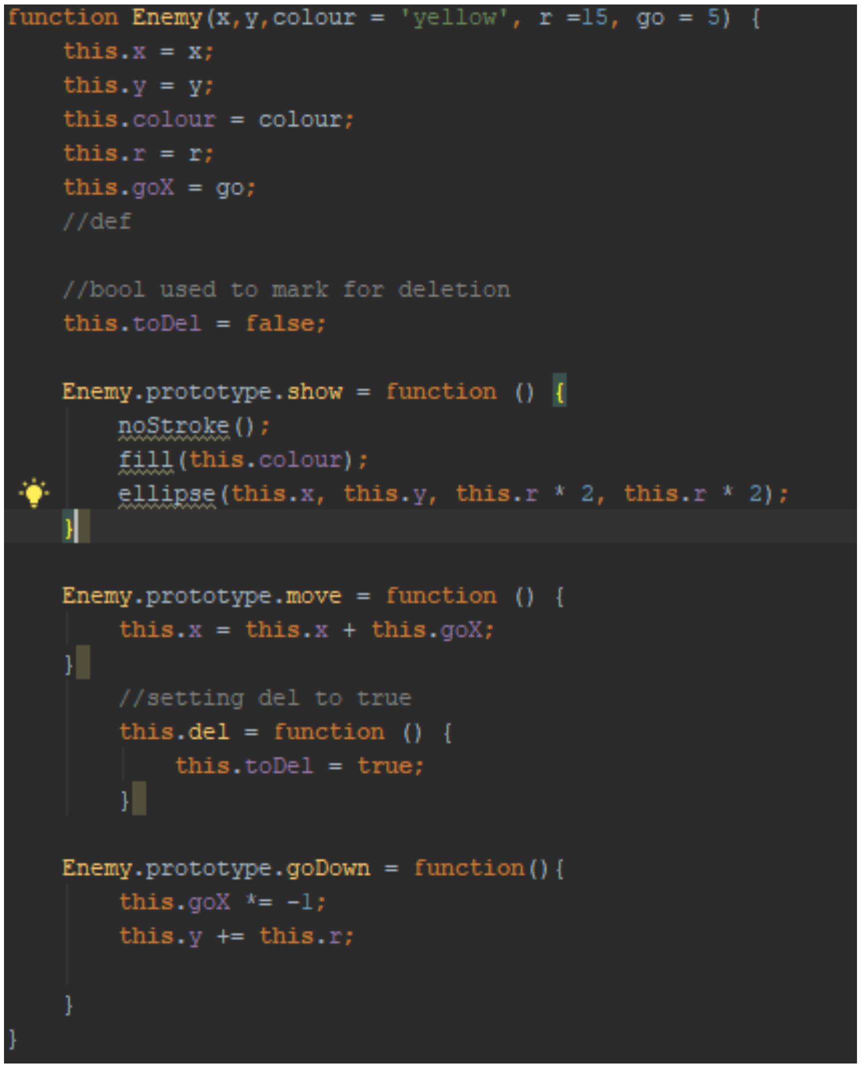
```
enemy[i] = new Enemy(x * 70 + 70, y);
```

Removing a shot from the array:



```
enemy.splice(i, 1);
```

Object Oriented Programming



```
function Enemy(x,y,colour = 'yellow', r =15, go = 5) {
    this.x = x;
    this.y = y;
    this.colour = colour;
    this.r = r;
    this.goX = go;
    //def

    //bool used to mark for deletion
    this.toDel = false;

    Enemy.prototype.show = function () {
        noStroke();
        fill(this.colour);
        ellipse(this.x, this.y, this.r * 2, this.r * 2);
    }

    Enemy.prototype.move = function () {
        this.x = this.x + this.goX;
    }
        //setting del to true
        this.del = function () {
            this.toDel = true;
        }

    Enemy.prototype.goDown = function(){
        this.goX *= -1;
        this.y += this.r;

    }
}
```

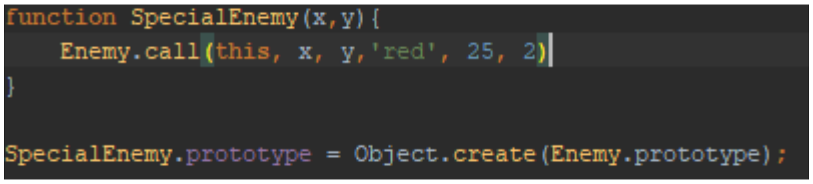
My ship, shots, enemy and special enemy are all objects. These objects are constructed using constructor functions, the enemy/specialEnemy class also makes use of default parameters and prototypal inheritance.

This prevents reputation in code, maintains readability and allows for changes to be made easily.

Prototypal Inheritance

All JavaScript objects have a link to a prototype object, when you are trying to access the properties of an object this search can travel through the porotype chain until a match is found – or it is undefined.

The prototype can be used to inherit properties of objects. This is achieved through creating a new object using the original objects prototype.



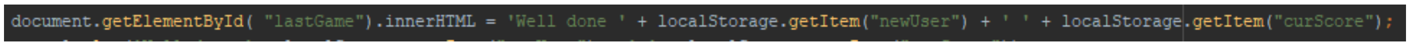
```
function SpecialEnemy(x,y){
    Enemy.call(this, x, y,'red', 25, 2)
}

SpecialEnemy.prototype = Object.create(Enemy.prototype);
```

Accessing and manipulating the DOM

Document Object Model (DOM) allows for the results of the JavaScript code to be shown to the user via HTML. This is used throughout the project to present the user with information.

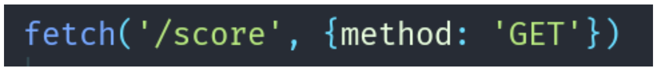
For example, at the end of the game the users statistics are presented to them. This information is retrieved form local storage through JavaScript, the div (lastGame) is found and the inner HTML is set to the data generated.



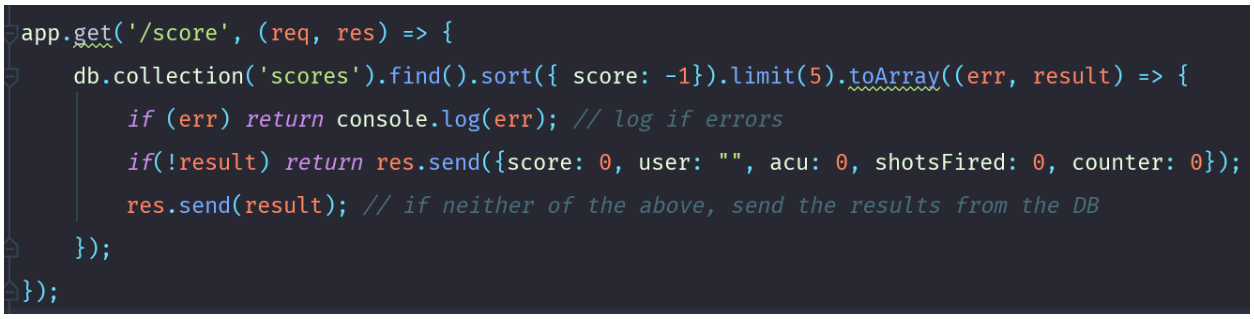
```
document.getElementById( "lastGame").innerHTML = 'Well done ' + localStorage.getItem("newUser") + ' ' + localStorage.getItem("curScore");
```

Asynchronous Requests

An asynchronous request is used to retrieve the high scores from the database at the end of the game.



```
fetch('/score', {method: 'GET'})
```



```
app.get('/score', (req, res) => {
    db.collection('scores').find().sort({ score: -1}).limit(5).toArray((err, result) => {
        if (err) return console.log(err); // log if errors
        if(!result) return res.send({score: 0, user: "", acu: 0, shotsFired: 0, counter: 0});
        res.send(result); // if neither of the above, send the results from the DB
    });
});
```

Reflection

Overall I was happy with how this project progressed. I established what features I wanted to implement early on and I have successfully achieved the original goal. At this stage I would like to improve the UI on the start screen and the appearance of the invaders themselves.

A few additional features that I would have liked to add all revolve around progression – additional rounds, the ability to increase the power of your ship etc. It may also have been nice to introduce a multiplayer aspect to the game.

This project has made me a lot more comfortable with working with objects/constructors. It also introduced me to some features of ES6 (default parameters, let/const, arrow functions). This was also my first time working with inheritance in JavaScript.

References

https://developer.mozilla.org/bm/docs/Web/JavaScript

https://www.w3schools.com/js/

https://www.youtube.com/watch?v=biN3v3ef-Y0

Github

https://github.com/aaronoh/AdvJSCA1