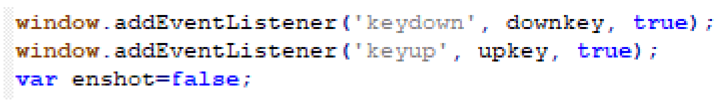
Report

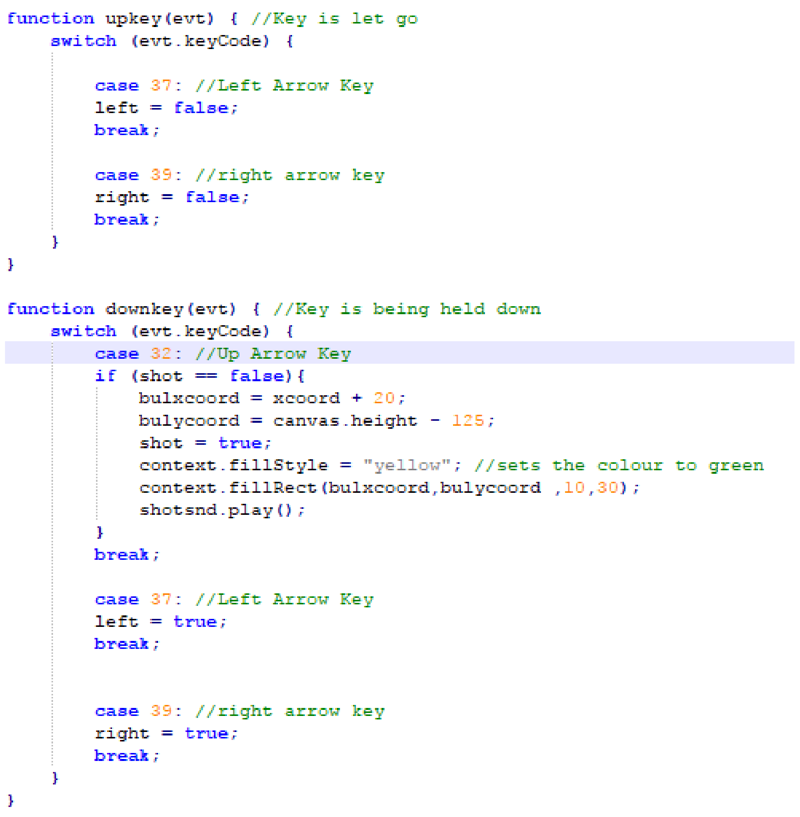
The following report will detail how space invaders [2] was recreated in JavaScript for the multimedia design and application second assignment. Within the game, the player controls a spaceship, able to move left or right and shoot upwards. The enemy aliens move their way towards the bottom of the screen while randomly shooting. There are also barriers which both the player and enemies cannot shoot through and need to destroy before being able to shoot through the location, and the player loses once all their lives are depleted.



```
window.addEventListener('keydown', downkey, true);
window.addEventListener('keyup', upkey, true);
var enshot=false;
```

Figure 1: Screenshot of code using Event Listeners

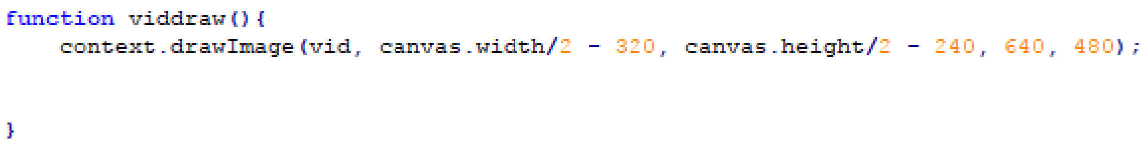
There were various JavaScript functions used when developing this game, such as Event Listeners which were used to detect when the player holds down a key and when they let it go, as seen in figure 1. The JavaScript function called "requestAnimationFrame" to automatically call a function the next available frame, which allows the function to automatically go along with the browser's framerate and if the window is inactive or hidden, then it won't be called until the window is deemed active again by the browser. Said function was used with a custom function called "update" to tie animations to the framerate.



```
function upkey(evt) { //Key is let go
    switch (evt.keyCode) {

        case 37: //Left Arrow Key
        left = false;
        break;

        case 39: //right arrow key
        right = false;
        break;
    }
}

function downkey(evt) { //Key is being held down
    switch (evt.keyCode) {
        case 32: //Up Arrow Key
        if (shot == false){
            bulxcoord = xcoord + 20;
            bulycoord = canvas.height - 125;
            shot = true;
            context.fillStyle = "yellow"; //sets the colour to green
            context.fillRect(bulxcoord,bulycoord ,10,30);
            shotsnd.play();
        }
        break;

        case 37: //Left Arrow Key
        left = true;
        break;


        case 39: //right arrow key
        right = true;
        break;
    }
}
```

Figure 2: Player controls using keycode

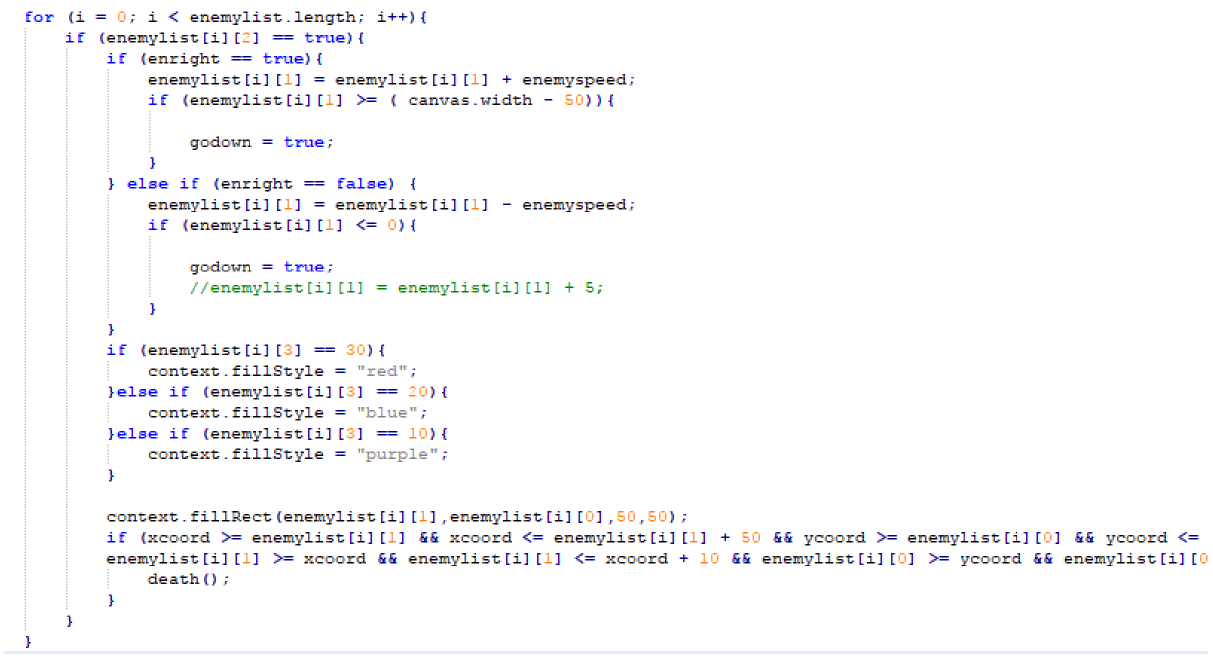
As seen in figure 2, switch case statements were used in conjunction with the evt.keyCode function to determine which keyboard key was used by the player and acting accordingly, such as making the variable “right” equal true when the right arrow key is held down. The evt input is taken from the Event Listeners mentioned previously. The player is then moved in the update function depending on if the variables right or left are true, moving them accordingly until they reach the boundaries of the canvas at which it stops moving them towards that direction. Collision is detected by having each object’s positions tied to variables which by using an if statement to check of their locations collide then the according action is taken. The player has a life system which is depleted if hit by an enemy shot or if enemies reach a certain screen point.



```
function viddraw(){
    context.drawImage(vid, canvas.width/2 - 320, canvas.height/2 - 240, 640, 480);


}
```

Figure 3: function displaying video

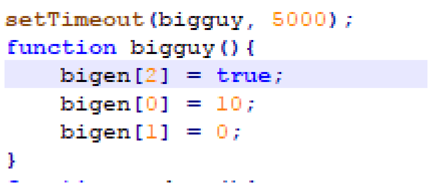
Using the function drawImage to draw a frame of the video and calling it each time the update function is called and the player’s lives are 0, it allows JavaScript to play the video on top of the canvas. The video is played out when the player gets a game over by losing all of their lives. A scoring system has also been implemented which awards the player an amount of points depending on the enemy that they destroy, ranging between 10, 20, and 30 for each of the normal enemies and 300 if the orange alien that spawns at certain intervals are taken down.



```
for (i = 0; i < enemylist.length; i++){
    if (enemylist[i][2] == true){
        if (enright == true){
            enemylist[i][1] = enemylist[i][1] + enemyspeed;
            if (enemylist[i][1] >= ( canvas.width - 50)){

                godown = true;
            }
        } else if (enright == false) {
            enemylist[i][1] = enemylist[i][1] - enemyspeed;
            if (enemylist[i][1] <= 0){

                godown = true;
                //enemylist[i][1] = enemylist[i][1] + 5;
            }
        }
        if (enemylist[i][3] == 30){
            context.fillStyle = "red";
        }else if (enemylist[i][3] == 20){
            context.fillStyle = "blue";
        }else if (enemylist[i][3] == 10){
            context.fillStyle = "purple";
        }

        context.fillRect(enemylist[i][1],enemylist[i][0],50,50);
        if (xcoord >= enemylist[i][1] && xcoord <= enemylist[i][1] + 50 && ycoord >= enemylist[i][0] && ycoord <=
        enemylist[i][1] >= xcoord && enemylist[i][1] <= xcoord + 10 && enemylist[i][0] >= ycoord && enemylist[i][0
            death();
        }
    }
}
```

Figure 4: enemy movement

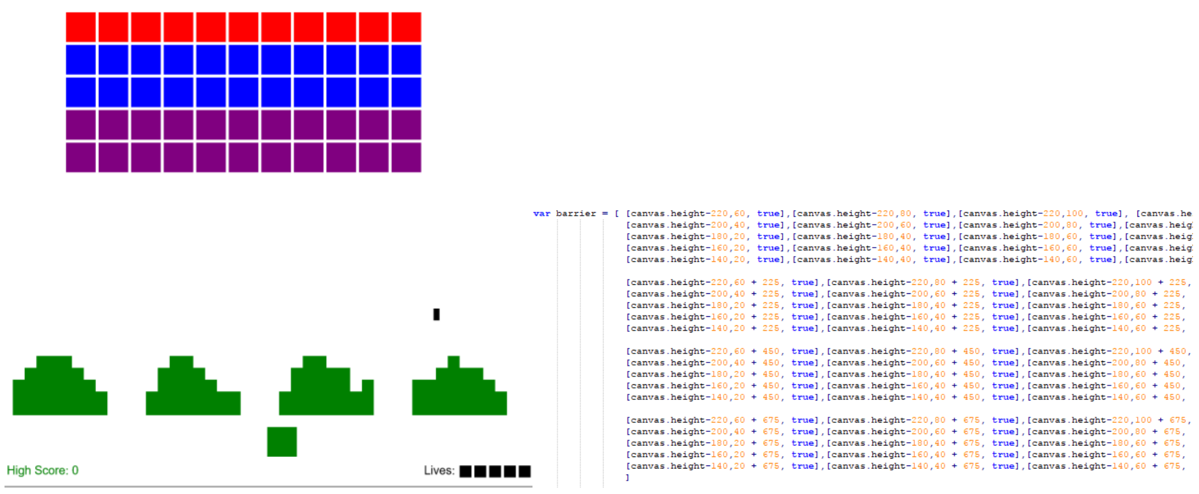
Enemies are implemented using a 2-dimensional array which holds arrays representing each of the enemies in the game. Each of the arrays holds the x and y coordinates, a Boolean determining if the enemy is alive or not and a number representing the score awarded when destroyed by the player. The enemies move to the right or left together, until they hit the edge of the canvas and then move down by a small amount and change direction, starting to move again to the left or right until hitting the edge again and so on. While their movement is simple, it was a bit complex to implement. As seen in figure 4, this was implemented using a for loop on the array 2-dimensional array holding the enemies, which first of all checks if the enemy is alive through its Boolean. If the enemy is alive, it will then move it right or left according to the current motion of movement, and once it has moved it will check if it reached the edge of the canvas, at which it will set a variable Boolean called "godown" as true. Depending on the score of the enemy, it will be coloured differently, and then it is drawn on the canvas. It also checks if the enemy's coordinates are in the same location as the player and calls the death method if so. The enemies downward movement is controlled by the godown Boolean and an if statement which checks it. If godown is true, it will move each of the enemies down a bit, redrawing them on the canvas and at the end of the statement, if the enemies y coordinates are the same or greater then the player's y coordinates, then the player loses a life and the game is reset.



```
setTimeout(bigguy, 5000);
function bigguy(){
    bigen[2] = true;
    bigen[0] = 10;
    bigen[1] = 0;
}
```

Figure 5: Orange alien spawning function

Additional functionality includes having a special orange enemy being spawned on a specific interval of 5 seconds, as seen in the bigguy function in figure 5, which awards 300 points if destroyed. Even if it is spawned using a setTimeout function, it is still animated in the update function to prevent it going through the screen while the window is inactive or hidden. Additionally, as mentioned previously, different enemies award different amounts of points and are colour coded depending on the amount of points they give, the 10 point enemies are purple, the 20 point enemies are blue and the 30 point enemies are red. The enemies also randomly shoot a single bullet from one of the lowest alive enemy of each row, which was achieved by creating another list of enemies which checks if the enemy is the lowest enemy in a row and if it is alive. The enemy is chosen randomly from that list using the Math.Random() function to get a decimal number between 0 and 1, multiplying it by the list's length – 1 and then using the Math.Floor function to round it to the closest whole number. Also, enemies will speed up depending on how many enemies are still alive, where it goes twice as fast when there are less then 15 enemies, four times as fast if there are less then 3 enemies alive and six times as fast when there is only one enemy left alive.

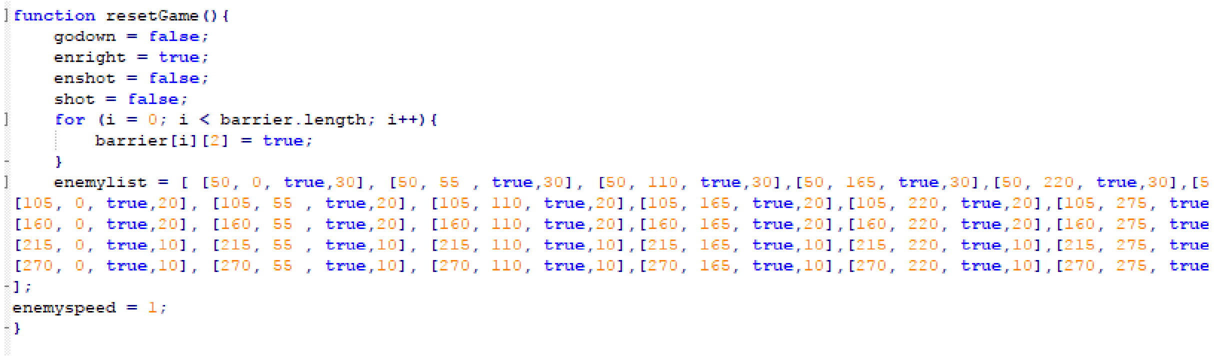


```
var barrier = [ [canvas.height-220,60, true],[canvas.height-220,80, true],[canvas.height-220,100, true], [canvas.he
                [canvas.height-200,40, true],[canvas.height-200,60, true],[canvas.height-200,80, true],[canvas.heig
                [canvas.height-180,20, true],[canvas.height-180,40, true],[canvas.height-180,60, true],[canvas.heig
                [canvas.height-160,20, true],[canvas.height-160,40, true],[canvas.height-160,60, true],[canvas.heig
                [canvas.height-140,20, true],[canvas.height-140,40, true],[canvas.height-140,60, true],[canvas.heig

                [canvas.height-220,60 + 225, true],[canvas.height-220,80 + 225, true],[canvas.height-220,100 + 225,
                [canvas.height-200,40 + 225, true],[canvas.height-200,60 + 225, true],[canvas.height-200,80 + 225,
                [canvas.height-180,20 + 225, true],[canvas.height-180,40 + 225, true],[canvas.height-180,60 + 225,
                [canvas.height-160,20 + 225, true],[canvas.height-160,40 + 225, true],[canvas.height-160,60 + 225,
                [canvas.height-140,20 + 225, true],[canvas.height-140,40 + 225, true],[canvas.height-140,60 + 225,

                [canvas.height-220,60 + 450, true],[canvas.height-220,80 + 450, true],[canvas.height-220,100 + 450,
                [canvas.height-200,40 + 450, true],[canvas.height-200,60 + 450, true],[canvas.height-200,80 + 450,
                [canvas.height-180,20 + 450, true],[canvas.height-180,40 + 450, true],[canvas.height-180,60 + 450,
                [canvas.height-160,20 + 450, true],[canvas.height-160,40 + 450, true],[canvas.height-160,60 + 450,
                [canvas.height-140,20 + 450, true],[canvas.height-140,40 + 450, true],[canvas.height-140,60 + 450,

                [canvas.height-220,60 + 675, true],[canvas.height-220,80 + 675, true],[canvas.height-220,100 + 675,
                [canvas.height-200,40 + 675, true],[canvas.height-200,60 + 675, true],[canvas.height-200,80 + 675,
                [canvas.height-180,20 + 675, true],[canvas.height-180,40 + 675, true],[canvas.height-180,60 + 675,
                [canvas.height-160,20 + 675, true],[canvas.height-160,40 + 675, true],[canvas.height-160,60 + 675,
                [canvas.height-140,20 + 675, true],[canvas.height-140,40 + 675, true],[canvas.height-140,60 + 675,
                ]
```

Figure 6: On the left, screenshot of barriers being destroyed block by block. On the right, 2D array holding the barriers.

Another additional functionality which was implemented was the barriers from space invaders [2] which can be destroyed block by block, as seen in figure 6 on the left. This was done so by creating a

2-dimensional array which holds each block compromising the barrier, represented by an array holding its x and y coordinates and a Boolean which determines if the block is destroyed or not, as seen on the right of figure 6. Then in the update method, a for loop goes through the barrier array, checking each array within it if the Boolean equals true or not, where if true it draws it on the canvas and if not it doesn't. Within the collision detection with the barriers also checks if the barrier block is alive and if so if the enemy or player shot is within the same location, where if it is the barrier block is destroyed.



```
function resetGame(){
    godown = false;
    enright = true;
    enshot = false;
    shot = false;
    for (i = 0; i < barrier.length; i++){
        barrier[i][2] = true;
    }
    enemylist = [ [50, 0, true,30], [50, 55 , true,30], [50, 110, true,30],[50, 165, true,30],[50, 220, true,30],[5
[105, 0, true,20], [105, 55 , true,20], [105, 110, true,20],[105, 165, true,20],[105, 220, true,20],[105, 275, true
[160, 0, true,20], [160, 55 , true,20], [160, 110, true,20],[160, 165, true,20],[160, 220, true,20],[160, 275, true
[215, 0, true,10], [215, 55 , true,10], [215, 110, true,10],[215, 165, true,10],[215, 220, true,10],[215, 275, true
[270, 0, true,10], [270, 55 , true,10], [270, 110, true,10],[270, 165, true,10],[270, 220, true,10],[270, 275, true
];
enemyspeed = 1;
}
```

Figure 7: reset game function

Finally, when all enemies in the game are defeated, a function named "resetGame" is called which resets all important variables to their default state, resets all barriers and enemies to their starting state which allows the player to keep playing and reaching higher scores, as seen in figure 7. Also, When the player reaches a game over, after the video finishes playing, the page is automatically reloaded to allow the player to restart the game.

Therefore, basic functionality of the game includes player controlled movement and shooting, collision detection between objects, boundaries limiting object movements, a life system which is depleted whenever the player is hit by enemies' shots or if enemies reach a certain height, a video is played when the player hits game over, and a scoring system awarding the player when enemies are destroyed. Additional functionality of the game included colour coded enemies awarding different amounts of points, increasing difficulty depending on number of enemies alive, barriers destructible block by block, additional enemy spawning on intervals and controlled independently from the other enemies, the lowest alive enemies shooting and enemies & barriers resetting when all enemies are defeated to allow the player to keep getting a higher score.

References

[1] Video used in game taken from: https://www.youtube.com/watch?v=vxJ01puNw4E [Last accessed 11/05/2017]

[2] Space Invaders [Online], Wikipedia, May 2017. Available at: https://en.wikipedia.org/wiki/Space_Invaders [Last accessed 11/05/2017]