Arslanbek Zhaparov

Criterion C: Development

Techniques Used:

Snack spawn

```
function generateRandom(){
    snack = {
        x : Math.floor(Math.random()*17+1) * sq,
        y : Math.floor(Math.random()*15+3) * sq
    }
    console.log(snack);
}
```

When doing the function the position x and y of the snack should be randomized, therefore choosing there was a need to use Math random functions. The most important element here is the boundaries of the spawning - they go from 32 to 544 pixels horizontally and from 96 to 544 vertically.

Pause

```
//pause the snake game if the player presses space
function Pause(){
    if (!paused){
        paused = true;
        game = clearInterval(idInterval)
    }else if (paused){
        paused= false;
        idInterval = setInterval(game, 100)
    }
}
```

First of all, to create a pause function there is a need to create a global element interval and a Boolean variable paused that is false in the beginning. When the function is being called it first checks if the passed variable is true or false and unpauses and pauses relatively to the check. In case of pause, it clears the game(clearInterval), but when it is unpaused then the global variable restores the interval(setInterval).

Key down elements

```
window.addEventListener('keydown', function (e){
    var key = e.keyCode;
    if (key === 32){
    Pause();
    }
```

To activate the pause the space key is pressed. The function checks whether or not the key was pressed, then it compares the value of the key to the value of the space bar(32), and if the space bar was pressed it either resumes or stops the game.

Snake Movement

```
    console.log(m);
    if(key == 37 && m != "RIGHT"){
        m = "LEFT";
    }else if(key == 38 && m != "DOWN"){
        m = "UP";
    }else if(key == 39 && m != "LEFT"){
        m = "RIGHT";
    }else if(key == 40 && m != "UP"){
        m = "DOWN";
    }
    console.log(m);
});
```

As the snake moves four directions the function should track it correctly and accurately. It also compares the kye value with the value of the key which is being pressed, however, when the key down is pressed the snake will go down only if it is not moving up.

Movement of the Snake

```
//which direction
if(m == "LEFT") newHeadX -= sq;
if(m == "UP") newHeadY -= sq;
if(m == "RIGHT") newHeadX += sq;
if(m == "DOWN") newHeadY += sq;
```

Although the direction of the snake is known, the way it moves is undefined for the program. Therefore the function connects the directions of the snake and creates a physical action - subtraction or addition of squares to the head of a snake to stimulate the movement. Movement is constant as soon as the key is pressed and the direction is not opposite the snake constantly moves towards that direction.

Overlap

```
function overlap(lx1, ly1, rx1, ry1, lx2, ly2, rx2, ry2) {
    if (lx1 > rx2 || lx2 > rx1){
        return false
    }
    else if (ly1 > ry2 || ly2 > ry1){
        return false
    }else{
        return true
    }
```

The overlap function checks whether the two objects - snake and snack- collide or not. The response of the function is true or false. The function checks it by comparing coordinates of the x and y corners of one square ot another one. If both squares follow the condition then the overlap equals true.

Snake Eating the Snack

```
if(overlap(newHeadX, newHeadY, newHeadX+sq, newHeadY+sq, snack.x, snack.y, snack.x+sq, snack.y+sq)){
    eating = true;
}
else {
    eating = false;
    print();
}
if(eating){
    score++

    generateRandom();
}else{
    //remove the tail as it is moving
    snake.pop();
}
```

The principle that the snake is using is if an overlap function of the snake's head and a snack is true, then the snake eats the snack and the score increases by one. The snake array or tail becomes bigger.

Drawing of the Snake

```
for(var i = 0; i < snake.length; i++){
    ctx.fillStyle = ( i==0 )? "coral":"black";//condition if  fillstyle is green and i == 0 and coral - the snake head is coral the rest is black
    ctx.fillRect(snake[i].x,snake[i].y,sq,sq);

    ctx.strokeStyle = "lightsalmon";
    ctx.strokeRect(snake[i].x,snake[i].y,sq,sq);

}
```

Although the direction of the snake is known, the way it moves should be coded. The function to go through the whole array is used. The function goes through each element of the array or snake and draws it. Furthermore, it draws the head of the snake with a separate color from the tail. The head is filled with coral color, while the rest of the body is black.

Game Over

```
//gameover
if((newHeadX < (0.8*sq)) || (newHeadX > (17 * sq)) || (newHeadY < (3 * sq)) || (newHeadY > (17 *sq))){
    clearInterval(game);
    hidecanvas()
    console.log("clearInterval");

}
```

Game Over function consists of several functions. The boundaries that create a wall for the snake and are 32, 544 horizontally and 96 and 544 vertically, therefore if the snake goes beyond those boundaries, then the interval will be cleared - the game over will happen.