CSE104 – Web Programming: Final Project

>BubblesGame<

A web game by Alessandro Massaad

May 2022

Overview

This is a brief overview of how my final project webpage was programmed. The webpage is a game whose goal is to survive as long as possible by killing as many enemies as possible as quickly as possible. Since the rules, mechanics and goald are quite straightforward, I chose a minimalist approach by not explaining them explicitly in the webpage.
The webpage is mainly based on the canvas API on JavaScript, and also uses HTML and some CSS.

Game mechanics

Basics

Canvas is the API on which the game will be programmed. To do so, we create a <canvas> element and link it to the JS file (HTML 32, 37). In the JS file, we start by defining a few variables that will be useful later (JS 3-7) and we create our display() and animate() functions (JS 150 - 160).

Player and projectiles

We first define our Player class with position, color, and radius (JS 26 - 39) modeling the player object, a static disk at the center of the screen. A draw() method is added in order to generate the player when initiating the game. We create our player object (JS 110) at the center of the screen.

Similarly, we define a Projectile class for projectile objects which will be shot by the player and move in a specific direction until they hit enemies. Additionally, this object has a velocity attribute and an update() method in order for projectiles to move.

We add an event listener (JS 230 - 235) for clicks that generates the projectiles. We use trigonometry basics to compute its position and velocity. We add its touchscreen equivalent (JS 236 - 241). We also create a projectiles array (JS 111) to keep track of them.

In the main animation loop, we create a projectile management forEach() loop (JS 170 - 183) over the projectiles array to stop keeping track of far projectiles by removing them when they go past the screen.

Enemies

We create an Enemy class (JS 60 - 79) similar to that of the projectiles. We also create an enemies array (JS 112) to keep track of them.
We create an enemies-generating function spawnEnemies() (JS 126 - 148) using setInterval() to repeatedly generate at a random frequency. The enemy object's attributes are then set in this function: radius is random, color depends on the theme, velocity depends on the angle, and the angle is set depending on position such that the enemy always moves towards the center, using basic trigonometry. The position must be random but only at the sides. To do so, we have 2 cases: either x takes values in {0, canvas.width} while y takes values in [0, canvas.height], or x takes values in [0, canvas.width] whiles y is in {0, canvas.height} at random. Note that we shift the positions by the radius for the enemies to look like they slide into the screen instead of just appearing.

We then create a function to detect enemies' collisions. We write it in an enemy object management forEach() loop (JS 184 - 227) inside the main animation function animate() (JS 156 - 228). To do so, we first implement the projectiles-enemies collisions (JS 194 - 226) in a forEach() loop over the projectiles array. Everytime the distance between an enemy and a projectile is less than 1, we consider it a collision and proceed to shrink the enemy using the gsap module and remove it if it is too small (dead) using a setTimeout() loop for animation purposes (for it not to lag). Note that we can use the online gsap module thanks to the <script> element (HTML 33 - 36). Further note that generating particles and updating the score is done later.
We now add the player-enemies collisions (JS 187 - 193) in an if() statement that works similarly to projectiles-enemies collisions. A collision here will cause the end of the game, trigger cancelAnimationFrame() and let the main menu appear.

Particles effects

We now create a new Particle class (JS 81 - 107). Particles are small disks that will be generated upon enemy-projectile collisions. We give them position, color, radius, and velocity attributes, as well as draw() and update() methods, and create a particles array to keep track of them. Additionally, we set a friction coefficient and an alpha attribute. These will enable us to create a fade effect on the particles. To do so, we add ctx.save() at the beginning of the draw method and ctx.restore() at the end, and between them we set ctx.globalAlpha to the particle's current alpha value. This alpha value will be reduced by 0.01 every time the update() method is called. The friction coefficient is similarly used to reduce the velocity exponentially every time update() is called.
We add a particle generating for() loop (JS 201 - 208) in the projectile-enemy collision loop, that generates particles in random directions upon these collisions.

We add a particle management forEach() loop (JS 161 - 169) over the particles array to remove particles that have faded completely.

Implementing score

We now want to track the score.
We add a #scorecounter element (HTML 13 - 16) that will print the score at the top left of the screen.
We create a score variable (JS 11) set to 0 that we will update depending on the rules we create. This variable will be used to update the innerHTML of the #scorecounter. We create an init() function (JS 114 - 124) that will be called every time the game is restarted, and that sets the score to 0 every time in addition to reinitiating the particles, enemies, and projectiles arrays as well as the player object. We add 100 to the score at every particle-enemy collision and 250 at every kill.

User interface

The user interface consists of a menu with a ‘Start Game!’ button and a ‘Change theme’ button. They are both implemented in the #menu element (HTML 18 - 30).
An event listener (JS 243 - 249) is used when ‘Start Game’ is clicked to initiate the game, and remove the menu by setting its display to ‘none’. The menu will appear again every time the game is over. A #scorediv (HTML 20 - 24) is also added and will be updated at the end of every game to print the score.
A dictionary is created to manage the themes which consist of the colors of the background, the enemies, the player/projectiles, the score display at the top left, and the buttons. The themes button’s event listener (JS 260 - 275) will rotate the themes every time it is triggered. Note that I only had time to implement 2 themes but more designs can be added without completely altering the code.

Note on the transparency of the background: we use rgba to set the background color, with slight transparency to create an effect of trajectory sweeping when the objects move on the screen.

Afterword

There are a few interesting functionalities I was unable to add due to lack of time.
Adding more themes would make it more fun. Also, adding some kind of leaderboard with all the high scores would have been interesting, but that would require using a database and PHP.
Also, I would have liked to fix an issue that my beta testers reported. As you play the game over and over again without reloading the page, the difficulty seems to increase. This might be caused by the random frequency at which enemies are spawned (JS 147) which seems to converge to 0 (hence enemies spawn more quickly).