

**Project: Simple Game**

**Performed by**: Ruskulbek Makpal

**Checked by:** Shoiynbek Temirlan

**Almaty, 2024**

**“Simple Jump Game” Project**

**Introduction**

This project is a simple arcade game built with JavaScript, where the player controls a character that must jump to avoid moving obstacles. The game uses HTML, CSS, and JavaScript to create dynamic interaction and an engaging user experience.

**1. Project Description**

The "Simple Jump Game" is a game where the player controls a character using the **Space** key. The player must avoid colliding with moving obstacles that appear on the screen. The game features dynamic difficulty: as the score increases, the speed of obstacles increases, and the player needs to press the **Space** key at the right time to jump.

**Key Features:**

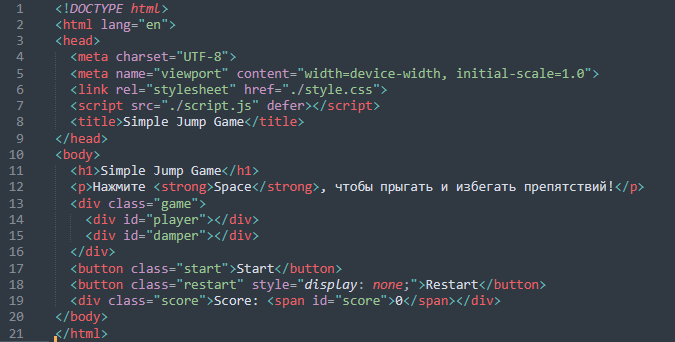
* **Character control** using the **Space** key.
* **Dynamic difficulty**: obstacle speed increases as the score goes up.
* **High score table**: displaying the best scores.
* **Game restart**: allows the player to restart the game after it ends.

**2. Technologies Used in the Project**

1. **HTML**: Structure of the web page.
2. **CSS**: Styling and design of game elements.
3. **JavaScript**: Game logic, event handling, and user interaction.

**3. Detailed Code Description**

**3.1 HTML Structure**



**Explanation:**

* <h1>: Displays the game title at the top of the page.
* <p>: Provides instructions to the user on how to play the game (press the Space key to jump).
* <div class="game">: A container that holds the game elements (player and obstacles).
* <div id="player">: The player character (Mario, in this case), which can be controlled by the user.
* <div id="damper">: The moving obstacle (represented by an image of a turtle).
* <button class="start">: The "Start" button that begins the game.
* <button class="restart">: The "Restart" button that appears when the game ends (hidden initially).
* <div class="score">: Displays the current score.

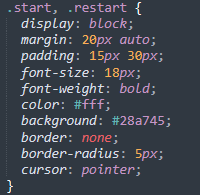
The <script> and <link> tags load the external JavaScript and CSS files, respectively

* **HTML** defines the structure of the page, including elements for the game (player character, obstacles), control buttons (Start, Restart), and score display.

2. **CSS Styling**

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание



Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

**Explanation:**

* **body**: Sets the background color and image, as well as overall padding and font settings for the entire page.
* **.game**: Creates a container for the game area, with a fixed width and height, and an overflow property to ensure elements don't spill out of the box. It also adds rounded corners and a shadow for a 3D effect.
* **#player**: Defines the size and position of the player character (Mario), and applies a background image for the character.
* **#damper**: Defines the size and position of the moving obstacle (turtle), and applies a background image.
* **.start, .restart**: Styles for the "Start" and "Restart" buttons, giving them padding, colors, and hover effects.
* **.score**: Styles the score display, making it bold and centered at the top of the screen.
* **@keyframes**: Defines animations for the obstacle and the player character. The **moveDamper** animation makes the obstacle move from right to left, while the **jumpPlayer** animation makes the player character "jump."

3. **JavaScript Game Logic**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Explanation:**

* **DOM Elements**: The script begins by selecting key elements from the HTML document using **getElementById** and **querySelector** to store them in variables. These include the player character, obstacle, buttons, and score display.
* **Game State Variables**:
  + **isGameRunning**: A boolean flag to check if the game is running.
  + **gameInterval**: A variable to store the interval for the game loop.
  + **score**: Stores the current score.
  + **speed**: Controls the speed of the obstacle’s movement.
  + **highScores**: An array to store high scores.
* **Sound Effects**: Loads the audio files for jump and game over sounds.

3.1 **Jumping Logic**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**activeJump():** This function handles the jump action. If the game is running and the player is not already in the "active" (jumping) state, it adds the active class to the player, triggering the jump animation. The jumpSound.play() plays the jump sound, and after 500ms, the player’s active class is removed, completing the jump.

**3.2 Game Start and End**



**startGame():** This function initializes the game. It sets the game state to isGameRunning = true, resets the score and speed, and starts the obstacle animation. The setInterval() function is used to create a game loop that checks for collisions, updates the score, and increases the difficulty (speed) every 100ms.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

**endGame(): E**nds the game, clears the game loop, removes the obstacle animation, adds the score to the high scores array, sorts the scores, and displays an alert with the player’s score and high scores. The gameOverSound.play() is triggered, and the Restart button is shown.

**restartGame():** Restarts the game by resetting the obstacle speed and calling startGame() again.

**3.4 Event Listeners**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**keydown Event:** Listens for the Space key press to trigger the jump action.

**Button Events:** Attaches event listeners to the Start and Restart buttons to start and restart the game.

**4. User Guide**

**How to Start:**

1. Press the "Start" button to begin the game.
2. Use the Space key to make the character jump.

**How to Avoid Collisions:**

* Jump to avoid colliding with the moving obstacles. If the obstacle comes too close, the game will end.

**How to Restart the Game:**

* After the game ends, press the "Restart" button to start the game again.

**4. Conclusion**

This project demonstrates the use of fundamental JavaScript concepts like DOM manipulation, event handling, and animations to create an interactive and engaging game. It includes handling dynamic content updates, responding to user input, and creating a game loop to manage the game’s logic. The project also provides a simple but effective example of how to manage the game state and improve user experience with sound effects, animations, and a high score system.

**You can see the codes here:** [[github.com](https://github.com/rysk1m/jumpgame.git)](https://github.com/rysk1m/jumpgame.git)