Car Dash

Author: Zhifan Li
UCID: 30089428

GAME OVERVIEW:

Target Platform: This game is only designed for laptops. With two players.

Game Genre: Car racing, collision avoidance, 2 players, retro game.

Games Objective:

The primary goal of users is to avoid colliding with the obstacles.
Whoever has the highest score will win.
As the game goes on, the speed of the cars will increase.

Rules of the Game:

Players need to click the Start button to begin the game.
Player 1 needs to use a, d on the keyboard to control the red car to move left or right to avoid colliding with the obstacles.
As the game goes on, the speed of the car will increase, the high probability the cars will collide with obstacles.
If an obstacle is avoided successfully, the player will get 1 score.
Player 2 needs to use the left arrow, and right arrow on the keyboard to control the red car to move left or right to avoid colliding with the obstacles.
Whoever runs into an obstacle will stop playing, if both players run into obstacles, the game will end. Whoever has the highest score wins the game.

Game Mechanics: Describe the primary actions the players w
ill perform.

Player 1: a: move left, d: move right.
Player 2: leftArrow: move left, rightArrow: move right.
Start button: Start the game for both players.

UI DESIGN

Include the details outlined below.
Layout and Structure:

Layout:

1. Logo and Icon

2. Background image
3. Button to start the game
4. Game container that has two game windows
5. Two game windows representing two players
- Each window has a car, two moving obstacles
- Each window has a scoreboard
6. Copyright and author at the bottom

UI Design:

Two game windows, one on the left, and one on the right.
Each game window has a car and two obstacles they need to avoid.
Each game window has a scoreboard beside it.
There is a button. If you click start, the button will disappear, and the game will start.

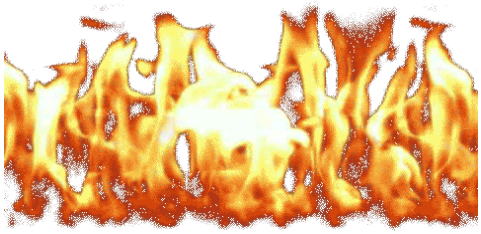
Visual Elements:

Cars

Obstacles

logo of the game:

fire flame:



FUNCTIONALITY DESIGN

The functionality of your game should include the following features to be implemented in vanilla JavaScript:

1) Custom Animations

The user will move either the red car or the blue car by using the keyboard.

We're using CSS transitions to smoothly animate the movement of the red and blue cars when the arrow keys are pressed. Here's how it works:

```
document.addEventListener('keydown', (event) => {
  const redCar = document.querySelector('.red-car');
  const blueCar = document.querySelector('.blue-car');

  if (event.key === 'ArrowLeft') {
    // Move blue car left
    blueCar.style.left = new position
  } else if (event.key === 'ArrowRight') {
    // Move blue car right
    blueCar.style.right =new position
  }
  if (event.key === A) {
    // Move red car left
    redCar.style.left = new position
  } else if (event.key === D) {
    // Move red car right
    redCar.style.right =new position
  }
});
```

We add an event listener to the keydown event, which listens for keyboard inputs. When the left or right arrow key is pressed, or a or d on the keyboard is pressed, the respecting car will move.
We then update the left or right CSS property to move the car smoothly to the desired position.

The obstacles will drop 15px per 0.3 seconds, and its speed will increase as the game goes until reaches a speed limit of 50px per 0.3 seconds.

We select the obstacle element using querySelector.
We set an initial obstacle-dropping speed of 15 pixels per 0.3 seconds.

```
obstacle.style.top = parseInt(topVal) + gameSpeed + "px";
```

The dropObstacle function is called at intervals using setInterval. It resets the obstacle's position to the top and increases its speed by reducing the transition duration.
This creates an animation where obstacles drop down at an increasing speed over time.

Check if the obstacle goes beyond boundary:

```
if (parseInt(topValUpdated) > 400) {
// Reset obstacle position and increase score
obstacle.style.top = "0px";
obstacle.style.left = Math.random() * 320 + 60 / 2 + "px";
}
```

If the obstacle goes beyond the boundary, it will go back to the top, visually people will feel that the old obstacle has passed, the new one has come.

2) Custom Interaction Mechanism

If your game involves moving and interacting elements, implement a custom collision detection mechanism.

This is how to avoid collision:

```
if (
parseInt(obstacle.style.top) > 340 &&
parseInt(car.style.left) + 35 > parseInt(obstacle.style.left) &&
parseInt(car.style.left) < parseInt(obstacle.style.left) + 90
) {
console.log("Game over.")
gameOver();
}
```

The collision detection logic checks if the bounding rectangles of the cars and the obstacle overlap in both the horizontal and vertical directions.
If a collision is detected, that user's car will stop moving, the score will stop updating, and the game ends on his/her side.

3) Custom Algorithms

1. An algorithm to show the users who is the winner by comparing the scores between these two players.
2. I will implement a simple scoring algorithm that increases the score when a player successfully avoids an obstacle.

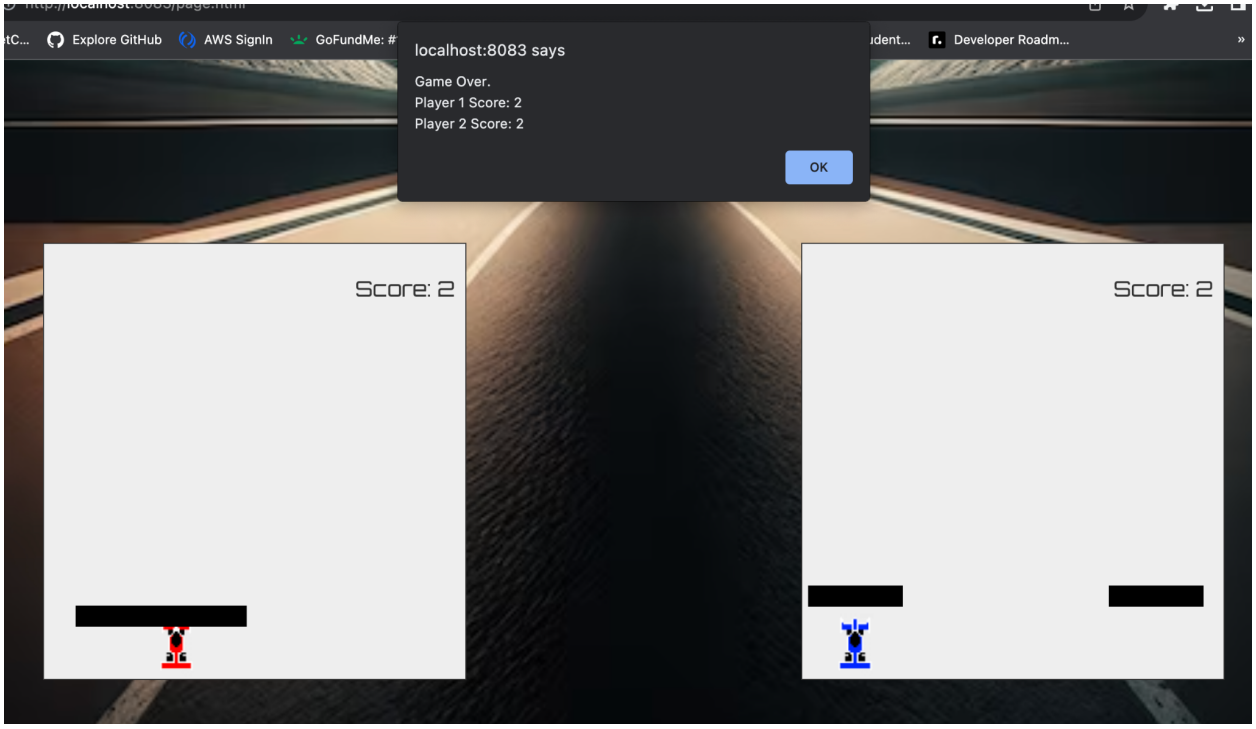
```
let score = 0;


function increaseScore() {
score += 10;
document.querySelector('.score').textContent = `Score:
${score}`;
}
```

We initialize a score variable to keep track of the player's score.
The increaseScore function increases the score by 1 point whenever it's called.
The game calls this function whenever a player successfully avoids an obstacle to increase their score, and after that, html element will also get updated.

3. Collision detection algorithm

Red Car avoiding the collision

Collision detected on Blue Car: Game over



Game Over Alert