ORBITAL DODGE

Manual and Game Developed by:
Derek Revilla
Design and Layout:
Derek Revilla
Translated by:
Fernando Lopez

USER MANUAL OF THE GAME “ORBITAL DODGE🌀

Introduction

The game “Orbital Dodge 🌀” is a mini game built compleately inHTML, CSS and JavaScript.
The goal is to control a sphere that orbits around a planet and dodge meteors.
The player can change the orbit’s direction pressing the keys ⬅️ (left) or ➡️ (right).
Each meteor that misses the planet increases the score by 1.
The project is composed by 1 single HTML file, inside of it the CSS styles and the JavaScript coding takes place
However, this document will explain it.

📄 1. HTML file’s structure

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>Orbital Dodge 🌀</title>
```

- <!DOCTYPE html>: Indicates that the file is HTML5..
- <html lang="es">: Sets the main language (Spanish).
- <head>: Contains the technical information and the file resources.
- <meta charset="UTF-8">: Allows to show correctly special characters (like accents or parenthesis).
- <meta name="viewport">: Adapts the game to small or mobile screens.
- <title>: Sets the name that will appear in the google browser tab (“Orbital Dodge 🌀”).

2. CSS styles

- The styles are written in the label <style> in the <head> of the file.

a) Import the font

```
@import url('https://fonts.googleapis.com/css2?family=Montserrat:wght@700&display=swap');
```

Import the font Montserrat, this is used in texts of games.

b) General body styles

```
body {
  margin: 0;
  background: radial-gradient(circle at center, #0a1128 0%, #001f54 80%);
  overflow: hidden;
  color: #e0e0e0;
  font-family: 'Montserrat', sans-serif;
  user-select: none;
  display: flex;
```

- Errases the browser margin (margin: 0).
- Creates a space background with a rounded dark blue gradient.
- Prevents that the player selects text with the mouse.
- Centers all the elements vertically.
- Applies the font Montserrat and the color’s text in light grey.

c) Title and marker

```
h1 {
  margin: 20px 0 5px 0;
  color: #00ffc8;
  text-shadow: 0 0 8px #00ffc8aa;
}

#score {
  font-size: 1.3rem;
  margin-bottom: 15px;
  font-weight: 700;
  color: #00ffc8cc;
}
```

- h1: Shows the title “Orbital Dodge 🌀”with a green-bluish glowing effect
- #score: Shows the player’s score in cyan

d) Canvas (<canvas>)

```
canvas {
  background: transparent;
  border-radius: 15px;
  box-shadow: 0 0 20px #00ffbdcc;
  display: block;
  max-width: 100vw;
}
```

- The canvas is where the game is built.
- It has rounded corners and it shines around.
- Uses all the browser’s available space.

e) Instructions and reboot messages

```
#instructions {
  font-size: 1rem;
  margin-bottom: 10px;
  color: #008066cc;
}

#restartMsg {
  margin-top: 15px;
  font-size: 1.1rem;
  color: #00ffbdcc;
  opacity: 0.85;
}
```

- #instructions: Shows how to play (Left and right keys).
- #restartMsg: Appears at the end of the game, indicating to press R to restart the game

3. Body’s structure (<body>)

```
<h1>Orbital Dodge 🌀</h1>
<div id="score">Puntos: 0</div>
<div id="instructions">Presiona ⬅️ o ➡️ para cambiar direccion</div>
<canvas id="gameCanvas" width="500" height="500"></canvas>
<div id="restartMsg"></div>
```

- Title: name of the game.
- Score: starts in 0.
- Instructions: Shows the control buttons.
- Canvas: place where the meteors, the planet and the player are drawn.
- Restart message: It will appear when the game ends.

4. JavaScript code – game logic

The <script> block contains all the coding that are separated in sections:

a) Main elements and default variables

```
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreEl = document.getElementById('score');
const restartMsg = document.getElementById('restartMsg');
```

You get the HTML elements and the 2D context of the canvas (where everything is drawn)

```
const centerX = canvas.width / 2;
const centerY = canvas.height / 2;
```

Calculates the center of the canvas, used as the planet reference point.

b) Player (the sphere that orbits)

```
const player = {
  radius: 120,
  angle: 0,
  size: 18,
  speed: 0.03,
  direction: 1,
  color: '#00ffc8',
};
```

- radius: distance of the player to the center (orbit).
- angle: angular position around the planet.
- speed: speed of rotation.
- direction: 1 = right, -1 = left.
- color: color of the player (glowing green).

c) Meteors and score

```
let meteors = [];
let meteorSpeed = 2.2;
let spawnTimer = 0;
let spawnInterval = 1500;

let score = 0;
let gameOver = false;
```

- It controls the creation, speed and frequency of the meteors
- score Stores the players score.
- gameOver Indicates if the game is over.

d) Drawing of the elements

Main planet:

```
function drawPlanet() {
  const gradient = ctx.createRadialGradient(centerX, centerY, 20, centerX, center
  gradient.addColorStop(0, '#008066ff');
  gradient.addColorStop(1, '#002b2200');
  ctx.fillStyle = gradient;
  ctx.beginPath();
  ctx.arc(centerX, centerY, 70, 0, Math.PI * 2);
  ctx.fill();
  ctx.strokeStyle = '#00ffbd';
  ctx.lineWidth = 3;
  ctx.stroke();
}
```

Draws the planet with gradient and glowing edges.

d) Drawing of the elements

Main planet:

```
function drawPlanet() {
  const gradient = ctx.createRadialGradient(centerX, centerY, 20, centerX, centerY,
  gradient.addColorStop(0, '#008066ff');
  gradient.addColorStop(1, '#002b2200');
  ctx.fillStyle = gradient;
  ctx.beginPath();
  ctx.arc(centerX, centerY, 70, 0, Math.PI * 2);
  ctx.fill();
  ctx.strokeStyle = '#00ffbd';
  ctx.lineWidth = 3;
  ctx.stroke();
}
```

Draws the planet with gradient and glowing edges.

Player orbiting:

```
function drawPlayer() { ... }
```

Calculates its position with cosine and sinus according to its angle.

Meteors:

```
function spawnMeteor() { ... }
function drawMeteors() { ... }
```

→ They are generated outside the canvas and move towards the center.

Player's orbit:

```
function drawOrbit() { ... }
```

→ It draws a circular dotted line that marks the player's path.

e) Movement and collisions

```
function updatePlayer() { player.angle += player.speed * player.direction; }
function updateMeteors() { ... }
function checkCollisions() { ... }
```

- The player rotates continuously.
- The meteors get closer to the planet.
- If a meteor touches the planet the game ends → gameOver = true.

f) Score system

```
function updateScore() {
  meteors.forEach(m => {
    const dist = Math.hypot(m.x - centerX, m.y - centerY);
    if (dist <= 75) score++;
  });
}
```

g) Main game loop

```
function gameLoop(timestamp) { ... requestAnimationFrame(gameLoop); }
```

- Draws and updates the entire game continuously.
- Uses requestAnimationFrame for a fluent animation.
- Gets harder with time (more and faster meteors).

h) keyboard controls

```
window.addEventListener('keydown', e => {
  if (e.code === 'ArrowLeft') player.direction = -1;
  if (e.code === 'ArrowRight') player.direction = 1;
  if (e.code === 'KeyR' && gameOver) resetGame();
});
```

Allows to change the player's direction
The key R starts the game.

i) Restart of the game

```
function resetGame() {
  meteors = [];
  score = 0;
  spawnInterval = 1500;
  meteorSpeed = 2.2;
  player.angle = 0;
  gameOver = false;
  restartMsg.textContent = '';
}
```

Conclusion

The code of the game "Orbital Dodge 🌀"combines simple physics , animation and collision detection in the sameHTML file.
Thanks to the canvas,the elements can be animated fluently, creating a visually attractive experience.
The player only needs to use the arrows to rotate the orbit and dodge the meteors while the score system and increasing difficulty makes the game more dynamic and entertained.