Game Manual

BREAKOUT

MANUAL AND GAME CREATED BY: DEREK REVILLA
DESIGN AND LAYOUT: MAURICIO CASTILLO
TRANSLATED BY: ALEXANDRA DIAZ

DETAILED MANUAL FOR THE GAME “BREAKOUT”

Description, Required Files, and Code Explanation

1. GAME DESCRIPTION

Breakout is a classic arcade video game originally created by Atari in the 1970s. Its objective is simple yet addictive: the player controls a bar (also called a paddle) located at the bottom of the screen, which must be used to hit a ball and destroy a series of bricks located at the top. Each time the ball hits a brick, the brick disappears, and the player earns points.

If the ball falls to the bottom of the screen without being intercepted by the paddle, the player loses a life. The game ends when all lives are lost or all bricks are destroyed, achieving victory.

This project recreates the classic Breakout using only HTML, CSS, and JavaScript, without the need for external libraries or game engines. Its purpose is didactic: it allows understanding fundamental concepts of 2D video game programming, such as canvas handling, collision detection, animations with requestAnimationFrame, and movement control via the keyboard.

2. REQUIRED DOCUMENTS FOR OPERATION

The most important document is a file with a .html extension, for example, index.html.

Within this file is all the necessary game code: the HTML structure, the CSS styles, and the JavaScript programming.

3. BASE HTML STRUCTURE

```
<!DOCTYPE html>
```

- Indicates that the document is in the HTML5 format, the most current and standard version. This guarantees compatibility with all modern browsers.

```
<html lang="es">
```

- Defines the main content language as Spanish, which aids search engines and screen readers.

4. DOCUMENT HEADER

```
<head>
  <meta charset="UTF-8" />
```

Allows the use of special characters like "ñ," "á," "ó," etc..

```
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
```

Without this, some texts might appear with strange symbols

```
<title>Breakout Game</title>
```

Makes the site look good on mobile devices, adapting the content size to the screen.
Game title. This is what appears on the browser tab.

3. STYLES (CSS)

```
<style>
```

This is where the visual aspect of the game is defined. It does not affect the logic, only the presentation.

Main Body

```css
body {
  display: flex;
  flex-direction: column;
  align-items: center;
  background: #222;
  color: #fff;
  font-family: sans-serif;
  margin: 0;
  padding: 20px;
}
```

- display: flex; → positions elements flexibly.
- flex-direction: column; → stacks them one below the other.
- align-items: center; → centers the elements horizontally.
- background: #222; → dark gray background color.
- color: #fff; → white text for contrast.
- font-family: sans-serif; → simple and legible typography.
- padding: 20px; → inner space around the content.
- margin: 0; → removes default browser margins.

```css
h1 {
  margin-bottom: 10px;
}
```

Adds space between the title and the following elements for visual separation.

Information (score and lives)

```css
.info {
  margin-bottom: 10px;
}
```

Creates separation between the information and the game area (canvas).

Game Canvas

```css
canvas {
  border: 2px solid #fff;
  background-color: #000;
}
```

● border draws a white frame around the game.
● background-color: #000 provides a black background that simulates a retro screen.

Button

```css
button {
  margin-top: 10px;
  padding: 8px 16px;
  cursor: pointer;
  background-color: #4caf50;
  color: white;
  border: none;
  border-radius: 4px;
}
```

Gives the button a modern shape:
● Inner (padding) and top (margin-top) space.
● "Hand" type cursor (cursor: pointer).
● Green background and white text.
● No visible border and with rounded corners.

```css
button:hover {
  background-color: #45a049;
}
```

Changes the color slightly on mouse hover, providing an interactive effect.

4. DOCUMENT BODY (BODY)

```
<body>
```

Everything visible and interactive in the game is here.
Game Title

```
<h1>Breakout</h1>
```

Displays the main game name at the top.
Information Panel

```
<div class="info">
  <span id="score">Puntuación: 0</span>
  <span id="lives">Vidas: 3</span>
</div>
```

Shows the score and remaining lives. The content is updated from JavaScript with textContent.
Game Area

```
<canvas id="canvas" width="800" height="400"></canvas>
```

- canvas creates the space where the entire game is drawn (ball, blocks, paddle).
- width and height define its size.
- It is an interactive 2D canvas controlled by JavaScript.

Restart Button

```
<button id="playBtn">Reiniciar</button>
```

Allows starting the game over after losing or wanting to restart.

All the game logic is here: movement, collisions, reset, and scoring.
Get HTML Elements

```
canvas {
  border: 2px solid #fff;
  background-color: #000;
}
```

● canvas gets the canvas from the HTML.
● ctx is the 2D graphic context necessary for drawing.

```
const scoreEl = document.getElementById('score');
const livesEl = document.getElementById('lives');
const playBtn = document.getElementById('playBtn');
```

🧠 Connects the code with the score text, lives text, and the restart button.
Ball

```
const ball = {
  x: canvas.width / 2,
  y: canvas.height - 30,
  size: 10,
  speed: 4,
  dx: 4,
  dy: -4,
};
```

● x, y: initial position.
● size: ball radius.
● speed: base speed.
● dx and dy: direction of movement (in X and Y).
● It moves diagonally from the start.
Paddle

```
const paddle = {
  width: 100,
  height: 10,
  x: (canvas.width - 100) / 2,
  y: canvas.height - 20,
  speed: 7,
  dx: 0
};
```

● Defines the bar that the player moves.
● Starts centered horizontally.
● dx changes with the left/right keys.
● speed defines the movement speed.
Bricks

```
const brick = {
  rowCount: 5,
  columnCount: 8,
  width: 75,
  height: 20,
  padding: 10,
  offsetX: 35,
  offsetY: 30
};
```

● 5 rows and 8 columns of bricks are created.
● Each brick measures 75×20 px.
● padding provides space between them.
● offsetX and offsetY set their initial position.

6. BRICK CREATION

```
function createBricks() {
  bricks = [];
  for (let r = 0; r < brick.rowCount; r++) {
    bricks[r] = [];
    for (let c = 0; c < brick.columnCount; c++) {
      const x = brick.offsetX + c * (brick.width + brick.padding);
      const y = brick.offsetY + r * (brick.height + brick.padding);
      bricks[r][c] = { x, y, status: 1 };
    }
  }
}
```

● Creates a two-dimensional array (rows and columns).
● Each brick has coordinates and a status ($1 =$ active, $0 =$ destroyed).

7. DRAWING OBJECTS

Bricks

```
function drawBricks() {
  for (let r = 0; r < brick.rowCount; r++) {
    for (let c = 0; c < brick.columnCount; c++) {
      let b = bricks[r][c];
      if (b.status === 1) {
        ctx.fillStyle = `rgb(${50 + r * 40}, ${100 + c * 15}, 200)`;
        ctx.fillRect(b.x, b.y, brick.width, brick.height);
      }
    }
  }
}
```

● Iterates through all the bricks.
● If the brick is active (status === 1), it

draws it with a variable color.

Ball

```
function drawBall() {
  ctx.beginPath();
  ctx.arc(ball.x, ball.y, ball.size, 0, Math.PI * 2);
  ctx.fillStyle = '#ffcc00';
  ctx.fill();
  ctx.closePath();
}
```

● Creates a circle (the ball) in yellow.

Paddle

```
function drawPaddle() {
  ctx.fillStyle = '#00aaff';
  ctx.fillRect(paddle.x, paddle.y, paddle.width, paddle.height);
}
```

Draws the blue paddle controlled by the player.

Information

```
function drawInfo() {
  scoreEl.textContent = `Puntuación: ${score}`;
  livesEl.textContent = `Vidas: ${lives}`;
}
```

Updates the visible texts according to game progress.

MOVEMENT AND COLLISIONS

Paddle Movement

```
function movePaddle() {
  paddle.x += paddle.dx;
  if (paddle.x < 0) paddle.x = 0;
  if (paddle.x + paddle.width > canvas.width) {
    paddle.x = canvas.width - paddle.width;
  }
}
```

- Moves the paddle according to dx.
- Prevents it from exiting the canvas sideways.

Ball Movement

```
function moveBall() {
  ball.x += ball.dx;
  ball.y += ball.dy;
```

Changes the ball's position every frame.

Wall Bounce

```
if (ball.x + ball.size > canvas.width || ball.x - ball.size < 0) {
  ball.dx *= -1;
}
if (ball.y - ball.size < 0) {
  ball.dy *= -1;
}
```

The ball bounces if it touches the side edges or the ceiling.

Paddle Bounce

```
if (ball.x > paddle.x && ball.x < paddle.x + paddle.width &&
    ball.y + ball.size > paddle.y) {
  ball.dy = -ball.speed;
}
```

Detects if the ball touches the paddle and sends it back up.

Brick Collision

```
for (let r = 0; r < brick.rowCount; r++) {
  for (let c = 0; c < brick.columnCount; c++) {
    let b = bricks[r][c];
    if (b.status === 1 &&
        ball.x > b.x && ball.x < b.x + brick.width &&
        ball.y - ball.size < b.y + brick.height &&
        ball.y + ball.size > b.y) {
      ball.dy *= -1;
      b.status = 0;
      score++;
    }
  }
}
```

- If the ball hits a brick, it changes the direction (dy) and destroys it (status = 0).
- Increments the score.

Loss of Lives

```
  if (ball.y + ball.size > canvas.height) {
    lives--;
    if (lives === 0) {
      alert('¡Juego terminado! Puntuación: ' + score);
      document.location.reload();
    } else {
      ball.x = canvas.width / 2;
      ball.y = canvas.height - 30;
      ball.dx = 4;
      ball.dy = -4;
      paddle.x = (canvas.width - paddle.width) / 2;
    }
  }
}
```

● If the ball touches the bottom, one life is subtracted.
● If lives reach 0, a message is displayed, and the game is reload

9. MAIN FUNCTION (update)

```
function update() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  drawBricks();
  drawBall();
  drawPaddle();
  drawInfo();
  movePaddle();
  moveBall();

  if (score === brick.rowCount * brick.columnCount) {
    alert('¡Ganaste! 🎉');
    document.location.reload();
  }

  animationId = requestAnimationFrame(update);
}
```

● Clears the screen.
● Draws all updated objects.
● Detects if all bricks have been destroyed (victory condition).
● requestAnimationFrame creates the continuous animation loop.

```
function keyDown(e) {
  if (e.key === 'ArrowRight' || e.key === 'Right') {
    paddle.dx = paddle.speed;
  } else if (e.key === 'ArrowLeft' || e.key === 'Left') {
    paddle.dx = -paddle.speed;
  }
}
```

10. KEYBOARD CONTROL

```
function keyUp(e) {
  if (e.key === 'ArrowRight' || e.key === 'Right' ||
      e.key === 'ArrowLeft' || e.key === 'Left') {
    paddle.dx = 0;
  }
}
```

Detects when an arrow key is pressed and moves the paddle.
When the key is released, the paddle stops.

11. RESETTING THE GAME

```
function resetGame() {
  score = 0;
  lives = 3;
  createBricks();
  ball.x = canvas.width / 2;
  ball.y = canvas.height - 30;
  ball.dx = 4;
  ball.dy = -4;
  paddle.x = (canvas.width - paddle.width) / 2;
  cancelAnimationFrame(animationId);
  update();
}
```

- Resets score, lives, bricks, and positions.
- Stops the previous animation and restarts the game.

12. MAIN EVENTS

```
document.addEventListener('keydown', keyDown);
document.addEventListener('keyup', keyUp);
playBtn.addEventListener('click', resetGame);

createBricks();
update();
```

- keydown and keyup: detect player movement.

- playBtn: resets the game on click.
- createBricks() creates the bricks upon startup.
- update() starts the overall animation.