

ChatGPT 4o

Share

ChatGPT

Test File Fixer

Doc

Three.js Mentor

Explore GPTs

Today

go-task2

Test

go-task1

ChatGPT 4o

Create a 6x6 space themed 2048 game using JS and HTML in a single file that uses a space background only using CSS. The player uses arrow keys to move the tiles. When two tiles with the same number touch, they merge into one. Player wins by forming 2048, loses when all cells are filled.

Here's a simple implementation of a 6x6 space-themed 2048 game using HTML, CSS, and JavaScript in a single file. The design includes a space-themed background and arrow key controls for the player to move tiles.

```
html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Space-Themed 2048</title>
    <style>
        body, html {
            margin: 0;
            padding: 0;
            height: 100%;
            font-family: Arial, sans-serif;
            display: flex;
            justify-content: center;
            align-items: center;
            background-color: black;
            background-image: url('https://www.publicdomainpictures.net/pictures/40000/velka/space-background-space-wallpaper.jpg');
            background-size: cover;
        }
        .game-container {
            display: grid;
            grid-template-columns: repeat(6, 100px);
            grid-template-rows: repeat(6, 100px);
            gap: 10px;
            background: rgba(255, 255, 255, 0.2);
            padding: 15px;
            border-radius: 10px;
        }
        .tile {
            width: 100px;
            height: 100px;
            background: rgba(255, 255, 255, 0.7);
            display: flex;
            justify-content: center;
            align-items: center;
            font-size: 2em;
            border-radius: 10px;
            color: #333;
            transition: all 0.2s ease-in-out;
        }
        .tile[data-value="2"] { background-color: #eee4da; }
        .tile[data-value="4"] { background-color: #eeca9c; }
        .tile[data-value="8"] { background-color: #f2b179; }
        .tile[data-value="16"] { background-color: #f99563; }
        .tile[data-value="32"] { background-color: #f67c5f; }
        .tile[data-value="64"] { background-color: #f65e2b; }
        .tile[data-value="128"] { background-color: #edcf72; }
        .tile[data-value="256"] { background-color: #edcc61; }
        .tile[data-value="512"] { background-color: #edc850; }
        .tile[data-value="1024"] { background-color: #edc53f; }
        .tile[data-value="2048"] { background-color: #edc22e; color: #fff; }
    </style>
</head>
<body>

    <div class="game-container" id="game-container"></div>

    <script>
        const gridSize = 6;
        const target = 2048;
        let gameBoard = [];
        const container = document.getElementById('game-container');

        // Initialize game board
        function initBoard() {
            gameBoard = Array.from({ length: gridSize }, () => Array(gridSize).fill(0));
            addTile();
            addTile();
            renderBoard();
        }

        // Render the game board
        function renderBoard() {
            container.innerHTML = '';
            gameBoard.forEach(row => {
                row.forEach(tile => {
                    const tileDiv = document.createElement('div');
                    tileDiv.classList.add('tile');
                    tileDiv.dataset.value = tile;
                    if (tile != 0) tileDiv.textContent = tile;
                    container.appendChild(tileDiv);
                });
            });
        }

        // Add a new tile to the board
        function addTile() {
            let emptyTiles = [];
            ...
        }
    </script>

```

```

        for (let i = 0; i < gridSize; i++) {
            for (let j = 0; j < gridSize; j++) {
                if (gameBoard[i][j] === 0) emptyTiles.push({ x: i, y: j });
            }
        }
        if (emptyTiles.length === 0) return;

        const { x, y } = emptyTiles[Math.floor(Math.random() * emptyTiles.length)];
        gameBoard[x][y] = Math.random() > 0.1 ? 2 : 4;
    }

    // Move tiles in the specified direction
    function move(direction) {
        let moved = false;
        for (let i = 0; i < gridSize; i++) {
            let row = gameBoard[i].filter(val => val);
            if (direction === 'right' || direction === 'down') row.reverse();

            for (let j = 0; j < row.length - 1; j++) {
                if (row[j] === row[j + 1]) {
                    row[j] *= 2;
                    row[j + 1] = 0;
                    moved = true;
                }
            }
        }

        row = row.filter(val => val);
        while (row.length < gridSize) row.push(0);
        if (direction === 'right' || direction === 'down') row.reverse();
        if (direction === 'down') {
            for (let x = 0; x < gridSize; x++) {
                gameBoard[x][i] = row[x];
            }
        } else {
            gameBoard[i] = row;
        }
        if (gameBoard[i].join() !== row.join()) moved = true;
    }

    if (moved) {
        addtile();
        renderBoard();
        if (checkWin()) alert('You win!');
        if (checkClose()) alert('Game over!');
    }
}

// Check if the player has won
function checkWin() {
    for (let i = 0; i < gridSize; i++) {
        for (let j = 0; j < gridSize; j++) {
            if (gameBoard[i][j] === target) return true;
        }
    }
    return false;
}

// Check if the player has lost
function checkClose() {
    for (let i = 0; i < gridSize; i++) {
        for (let j = 0; j < gridSize; j++) {
            if (gameBoard[i][j] === 0) return false;
            if (i > 0 && gameBoard[i][j] === gameBoard[i - 1][j]) return false;
            if (i < gridSize - 1 && gameBoard[i][j] === gameBoard[i + 1][j]) return false;
            if (j > 0 && gameBoard[i][j] === gameBoard[i][j - 1]) return false;
            if (j < gridSize - 1 && gameBoard[i][j] === gameBoard[i][j + 1]) return false;
        }
    }
    return true;
}

// Arrow key controls
document.addEventListener('keydown', (e) => {
    switch (e.key) {
        case 'ArrowUp':
            move('up');
            break;
        case 'ArrowDown':
            move('down');
            break;
        case 'ArrowLeft':
            move('left');
            break;
        case 'ArrowRight':
            move('right');
            break;
    }
});

initBoard();

```

#### Key Features:

- **Space Theme:** The game background uses a space image for a cosmic feel.
- **6x6 Grid:** The board is a 6x6 grid instead of the traditional 4x4.
- **Arrow Keys:** Use the arrow keys to move tiles.
- **Game Mechanics:** When two tiles of the same value collide, they merge. The game ends when you reach 2048 or if there are no more moves left.

Feel free to tweak the styling, add animations, or change the background to match your preferences!

For the following requirements and assumptions pairs, please create test cases in JEST with JSDom - provide a single JS file and consider completing all tests, start the test name with R1, R2...etc. Make sure all tests are written and not mocked:

Requirement 1:  
The game must be implemented using HTML, CSS, and JavaScript in a single file.

Assumption 1:  
The game logic, styling, and structure are contained within a single index.html file.

Requirement 2:  
The game must have a 6x6 grid layout for the game board.

Assumption 2:  
The game board is represented by a div element containing 16 cell elements.

Requirement 3:  
The game must have a space-themed background implemented using CSS.

Assumption 3:  
The body or a container element has a CSS background property having an image.

Requirement 4:  
The game must allow players to move tiles using arrow keys.

Assumption 4:  
The game listens for keydown events on the arrow keys (up, down, left, right) to trigger tile movements.

Requirement 5:  
The game must merge tiles with the same number when they collide.

Assumption 5:  
When two tiles with the same value move into each other, they combine into a single tile with double the value.

Requirement 6:  
The game must generate a new tile with a value of either 2 or 4 after each move.

Assumption 6:  
After each valid move, a new tile appears in an empty cell with a 90% chance of being a 2 and a 10% chance of being a 4.

Requirement 7:  
The game must end and display a "You Win!" message when a tile with the value 2048 is created.

Assumption 7:  
When a tile reaches the value of 2048, the game stops and a win message is displayed.

Requirement 8:  
The game must end and display a "Game Over" message when no more moves are possible.

Assumption 8:  
When all cells are filled and no adjacent tiles can be merged, the game stops and a game over message is displayed.

Requirement 9:  
The game must display the current score.

Assumption 9:  
There is a score element that updates with each tile merge, adding the value of the merged tile to the score.

Requirement 10:  
The game must have a "New Game" button to reset the game state.

Assumption 10:  
A button element with text "New Game" exists and, when clicked, resets the game board, score, and generates initial tiles.

Requirement 11:  
The game must start with two randomly placed tiles on the board.

Assumption 11:  
When the game initializes or is reset, two tiles (with values of either 2 or 4) are placed in dom empty cells.



Here is a single JavaScript file with Jest and JSDom test cases for the requirements and assumptions mentioned. These tests verify the functionality of the game without mocking, ensuring that the game logic is properly implemented.

```
javascript
/*
 * @jest-environment jsdom
 */

const { fireEvent } = require('@testing-library/dom');

// Setup a simple HTML structure that mimics the game's initial state.
document.body.innerHTML = `
<div class="game-container" id="game-container"></div>
<div id="score">0</div>
<button id="new-game">New Game</button>
`;

const gameScript = require('./game.js'); // Assume game logic is imported here

describe('Space-Themed 2048 Game Tests', () => {
  beforeEach(() => {
    // Reset the game board and score before each test
    // ...
  });
  // Test cases...
});
```

```

        gameScript.initBoard();
    });

    // R1: The game must be implemented using HTML, CSS, and JavaScript in a single file.
    test('R1: The game is implemented in a single index.html file with inline JS and CSS references.', () => {
        expect(document.querySelector('html')).not.toBeNull();
        expect(document.querySelector('script')).not.toBeNull(); // Checks that JS is inline
        expect(document.querySelector('style')).not.toBeNull(); // Checks that CSS is inline
    });

    // R2: The game must have a 6x6 grid layout for the game board.
    test('R2: The game board has a 6x6 grid.', () => {
        const container = document.getElementById('game-container');
        expect(container.children.length).toBe(36); // 6x6 grid = 36 tiles
    });

    // R3: The game must have a space-themed background implemented using CSS.
    test('R3: The game has a space-themed background using CSS.', () => {
        const bodyStyles = getComputedStyle(document.body);
        expect(bodyStyles.backgroundImage).toContain('url');
    });

    // R4: The game must allow players to move tiles using arrow keys.
    test('R4: The game responds to arrow key events for moving tiles.', () => {
        const moveSpy = jest.spyOn(gameScript, 'move'); // Spy on the move function
        fireEvent.keyDown(window, { key: 'ArrowUp' });
        fireEvent.keyDown(window, { key: 'ArrowDown' });
        fireEvent.keyDown(window, { key: 'ArrowLeft' });
        fireEvent.keyDown(window, { key: 'ArrowRight' });

        expect(moveSpy).toHaveBeenCalledTimes(4);
    });

    // R5: The game must merge tiles with the same number when they collide.
    test('R5: Merging tiles with the same value.', () => {
        gameScript.gameBoard = [
            [2, 2, 0, 0, 0, 0], ↓
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0]
        ];
        gameScript.move('left');
        expect(gameScript.gameBoard[0][0]).toBe(4); // Merged tile
        expect(gameScript.gameBoard[0][1]).toBe(0); // Empty cell after merge
    });

    // R6: The game must generate a new tile with a value of either 2 or 4 after each move.
    test('R6: A new tile is generated after each move.', () => {
        const emptyTilesBefore = gameScript.getEmptyTiles().length;
        gameScript.move('left');
        const emptyTilesAfter = gameScript.getEmptyTiles().length;
        expect(emptyTilesBefore - emptyTilesAfter).toBe(1); // One new tile is added
        const newValue = gameScript.getRandomTileValue();
        expect([2, 4]).toContain(newValue); // The new tile should be 2 or 4
    });

    // R7: The game must end and display a "You Win!" message when a tile with the value 2048 is created.
    test('R7: The game ends and shows "You Win!" when a 2048 tile is created.', () => {
        gameScript.gameBoard = [
            [2048, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0]
        ];
        gameScript.checkWin(); ↓
        expect(document.body.textContent).toContain('You Win!');
    });

    // R8: The game must end and display a "Game Over" message when no more moves are possible.
    test('R8: The game ends and shows "Game Over" when no more moves are possible.', () => {
        gameScript.gameBoard = [
            [2, 4, 2, 4, 2, 4],
            [4, 2, 4, 2, 4, 2],
            [2, 4, 2, 4, 2, 4],
            [4, 2, 4, 2, 4, 2],
            [2, 4, 2, 4, 2, 4],
            [4, 2, 4, 2, 4, 2]
        ]; // No possible moves
        gameScript.checkClose();
        expect(document.body.textContent).toContain('Game Over');
    });

    // R9: The game must display the current score.
    test('R9: The score updates correctly when tiles are merged.', () => {
        gameScript.gameBoard = [
            [2, 2, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0]
        ];
        gameScript.move('left');
        const scoreElement = document.getElementById('score');
        expect(scoreElement.textContent).toBe('4'); // The score should update after merge
    });

    // R10: The game must have a "New Game" button to reset the game state.
    test('R10: The "New Game" button resets the game board.', () => {
        gameScript.gameBoard = [
            [2, 2, 0, 0, 0, 0],
            [2, 2, 0, 0, 0, 0]
        ];
    });

```

```

    [0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0];
};

fireEvent.click(document.getElementById('new-game'));
expect(gameScript.gameBoard.flat().filter(v => v != 0).length).toBe(2); // After reset
});

// R11: The game must start with two randomly placed tiles on the board.
test('R11: The game initializes with two random tiles.', () => {
  const nonEmptyTiles = gameScript.gameBoard.flat().filter(val => val !== 0);
  expect(nonEmptyTiles.length).toBe(2); // Two initial tiles at game start
  nonEmptyTiles.forEach(tile => {
    expect([2, 4]).toContain(tile); // Initial tiles should be either 2 or 4
  });
});
});
});
});

```

#### Explanation:

- Setup: A basic HTML structure is created, mimicking the game's initial state. The game script (assumed to be `game.js`) handles the game logic.
- Tests (R1 - R11): Each requirement is turned into a test case. These tests cover various aspects such as the grid layout, background theme, keyboard events, merging logic, and game-over conditions.

Make sure to adjust the paths in the `require` statements if necessary, and update any additional functions or game logic that are used within the script.

no external js is present, file to be tested should be loaded from  
`'./index_o.html'`. Append Chat GPT 4o before test name.

Memory updated

Here is the Jest test file that loads the HTML file from `'./index_o.html'`, with all test names prefixed with "Chat GPT 4o".

javascript

```

/* @jest-environment jsdom */

const fs = require('fs');
const path = require('path');
const { fireEvent } = require('@testing-library/dom');

// Load the HTML file to be tested
const html = fs.readFileSync(path.resolve(__dirname, './index_o.html'), 'utf8');

describe('Space-Themed 2048 Game Tests', () => {
  let gameScript;

  beforeEach(() => {
    document.documentElement.innerHTML = html.toString();
    gameScript = require('./index_o.html'); // Assume all game logic is in this file
    gameScript.initBoard(); // Initialize the board before each test
  });

  // Chat GPT 4o R1: The game must be implemented using HTML, CSS, and JavaScript in a single index_o.html file with inline JS/CSS
  test('Chat GPT 4o R1: The game is implemented in a single index_o.html file with inline JS/CSS', () => {
    expect(document.querySelector('html')).not.toBeNull();
    expect(document.querySelector('script')).not.toBeNull(); // Checks that JS is inline
    expect(document.querySelector('style')).not.toBeNull(); // Checks that CSS is inline
  });

  // Chat GPT 4o R2: The game must have a 6x6 grid Layout for the game board.
  test('Chat GPT 4o R2: The game board has a 6x6 grid.', () => {
    const container = document.getElementById('game-container');
    expect(container.children.length).toBe(36); // 6x6 grid = 36 tiles
  });

  // Chat GPT 4o R3: The game must have a space-themed background implemented using CSS.
  test('Chat GPT 4o R3: The game has a space-themed background using CSS.', () => {
    const bodyStyles = getComputedStyle(document.body);
    expect(bodyStyles.backgroundImage).toContain('url'); // Checks if background image is set
  });

  // Chat GPT 4o R4: The game must allow players to move tiles using arrow keys.
  test('Chat GPT 4o R4: The game responds to arrow key events for moving tiles.', () => {
    const moveSpy = jest.spyOn(gameScript, 'move'); // Spy on the move function
    fireEvent.keyDown(window, { key: 'ArrowUp' });
    fireEvent.keyDown(window, { key: 'ArrowDown' });
    fireEvent.keyDown(window, { key: 'ArrowLeft' });
    fireEvent.keyDown(window, { key: 'ArrowRight' });

    expect(moveSpy).toHaveBeenCalledTimes(4);
  });

  // Chat GPT 4o R5: The game must merge tiles with the same number when they collide.
  test('Chat GPT 4o R5: Merging tiles with the same value.', () => {
    gameScript.gameBoard = [
      [2, 2, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0]
    ];
  });
});

```

```

gameScript.move('left');
expect(gameScript.gameBoard[0][0]).toBe(4); // Merged tile
expect(gameScript.gameBoard[0][1]).toBe(0); // Empty cell after merge
});

// Chat GPT 4o R6: The game must generate a new tile with a value of either 2 or 4 after
test('Chat GPT 4o R6: A new tile is generated after each move.', () => {
  const emptyTilesBefore = gameScript.getEmptyTiles().length;
  gameScript.move('left');
  const emptyTilesAfter = gameScript.getEmptyTiles().length;
  expect(emptyTilesBefore - emptyTilesAfter).toBe(1); // One new tile is added
  const newTile = gameScript.getRandomTileValue();
  expect([2, 4]).toContain(newTile); // The new tile should be 2 or 4
});

// Chat GPT 4o R7: The game must end and display a "You Win!" message when a tile with t
test('Chat GPT 4o R7: The game ends and shows "You Win!" when a 2048 tile is created.', 
gameScript.gameBoard = [
  [2048, 0, 0, 0, 0],
  [0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0],
];
gameScript.checkWin();
expect(document.body.textContent).toContain('You Win!');
});

// Chat GPT 4o R8: The game must end and display a "Game Over" message when no more moves
test('Chat GPT 4o R8: The game ends and shows "Game Over" when no more moves are possible',
gameScript.gameBoard = [
  [2, 4, 2, 4, 2, 4],
  [4, 2, 4, 2, 4, 2],
  [2, 4, 2, 4, 2, 4],
  [4, 2, 4, 2, 4, 2],
  [2, 4, 2, 4, 2, 4],
  [4, 2, 4, 2, 4, 2],
];
}); // No possible moves
gameScript.checkLose();
expect(document.body.textContent).toContain('Game Over');
});

// Chat GPT 4o R9: The game must display the current score.
test('Chat GPT 4o R9: The score updates correctly when tiles are merged.', () => {
gameScript.gameBoard = [
  [2, 2, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0],
];
gameScript.move('left');
const scoreElement = document.getElementById('score');
expect(scoreElement.textContent).toBe('4'); // The score should update after merge
});

// Chat GPT 4o R10: The game must have a "New Game" button to reset the game state.
test('Chat GPT 4o R10: The "New Game" button resets the game board.', () => {
gameScript.gameBoard = [
  [2, 2, 0, 0, 0, 0],
  [4, 4, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0],
];
fireEvent.click(document.getElementById('new-game'));
expect(gameScript.gameBoard.flat().filter(v => v !== 0).length).toBe(2); // After rese
});

// Chat GPT 4o R11: The game must start with two randomly placed tiles on the board.
test('Chat GPT 4o R11: The game initializes with two random tiles.', () => {
  const nonEmptyTiles = gameScript.gameBoard.flat().filter(val => val !== 0);
  expect(nonEmptyTiles.length).toBe(2); // Two initial tiles at game start
  nonEmptyTiles.forEach(tile => {
    expect([2, 4]).toContain(tile); // Initial tiles should be either 2 or 4
  });
});
});
});
});

```

#### Explanation:

- **Test file loading:** The `../index_o.html` is loaded using `fs.readFileSync` to simulate a scenario where everything (HTML, CSS, and JavaScript) is bundled into one file.
- **Tests:** Each test is prefixed with "Chat GPT 4o" as requested. These tests cover all the requirements and assumptions.
- **JSDOM environment:** Since we are using Jest with JSDOM, the test environment replicates the DOM within the HTML file, allowing us to directly manipulate it and simulate events like key presses.

You can run these tests using Jest after placing the HTML file and the test script in the proper directory structure.



