University of Central Florida

DEPARTMENT OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE

COMPUTER SCIENCE DIVISION

CIS 4004 Web Based Information Technology

Assignment 5
Due, Sunday, April 21, 2024 for 100% credit
Monday, April 22, 2024 for 90% credit
Tuesday, April 23, 2024 for 80% credit
Wednesday, April 24, 2024 for 70% credit

Deliverables

Compress the React app **connectfour public** and **src** folders ONLY (i.e., do **NOT** include the **node_modules** folder) and submit the compressed file (e.g., .zip)

Reference files from Assignment 3 JavaScript Connect Four

1. index.html
2. connectfour.js
3. connectfour.css

File provided

1. ConnectFour_template.js
2. ConnectFour.css

Project description

This assignment is focused on ReactJS and will require students to generate a Connect Four board and replicate the board game based on game components, game setup, object of the game, game play, valid moves and end of game.

Game components
The Connect Four game is a classic strategy game in which two players go head-to-head in a battle to own the grid!

- Players choose their disc colors.
- Empty board in a grid sized six row by seven columns.

Object of the game
Players stack their colored discs upwards, horizontally, or diagonally to get four in a row to win.

Game play

- "Yellow" goes first.
- Players take turns dropping the discs into the grid, starting in the middle or at the edge to stack their colored discs upwards, horizontally, or diagonally.
- Use strategy to block opponents while aiming to be the first player to get four in a row to win.

End of game
One player gets four discs in a row upwards, horizontally, or diagonally.

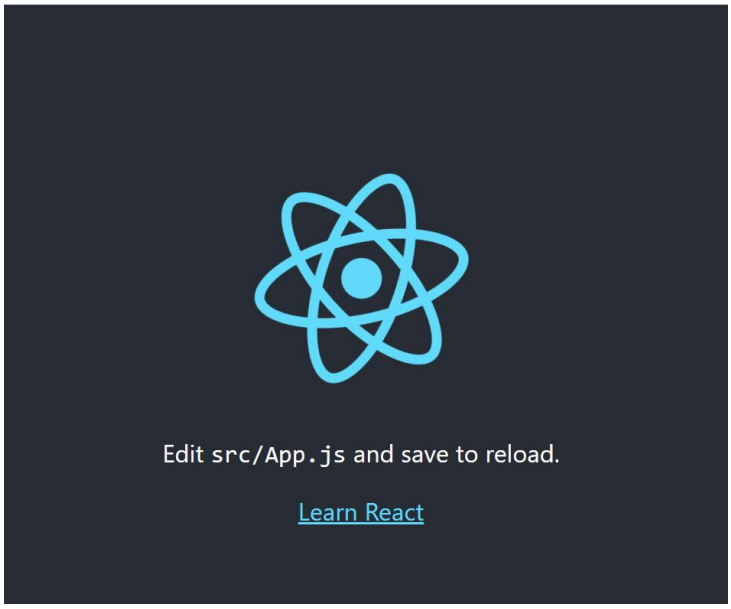
ReactJS installation resources and steps

Note: For any UNIX flavor operating system (e.g., Linux, MacOS) you may need to add '**sudo'** (do not include quotes) in front of the installation commands to be elevated to superuser permissions.

Node.js and ReactJS

1. Download and install **Node.js**
    a. **https://nodejs.org/en/download/**
2. Using the command prompt or terminal window, install the ***serve*** static server
    a. **npm install serve -g**
3. Using the command prompt or terminal window, install Babel and its React companion
    a. **npm install babel-cli@6 babel-preset-react-app@3 -g**
4. Using the command prompt or terminal window, install Create React App toolchain
    a. **npm install -g create-react-app**
5. Using the command prompt or terminal window, create a React application in desired workspace
    a. **create-react-app connectfour**
6. Using the command prompt, change directory to the application folder
    a. **cd connectfour**
7. Using the command prompt or terminal window, install the default React app

   a. **npm install**
8. Using the command prompt or terminal window, test the default React app
   a. **npm start**
   b. The web browser should display



9. Browse to the **src** folder of the **connectfour** app directory
   a. Create folder named **components**
   b. Change directory to the **components** folder
      i. Create empty file **ConnectFour.js**
      ii. Create empty file **ConnectFour.css**

Activity	
index.html	1. Update the **<title>** element to **Connect Four in React**
index.js	2. Comment out or delete unused import App to eliminate warnings 3. import **ConnectFour** from **'./components/ConnectFour'** 4. Change **root.render** a. from: **<App />** b. to: **<ConnectFour />**
ConnectFour.css	1. Reuse connectfour.css from Assignment 3 **OR** 2. Modify the following colors to any color of your choice a. body, background-color b. .container, background-color c. .player1:before, background d. .player2:before, background e. .grid-box, background-color
ConnectFour.js	1. import the **React** library from **'react'** 2. import the **ConnectFour.css** 3. Define class **ConnectFour** that **extends React.Component** a. Define the **constructor** method, parameter list includes **props**

i. Call the **super** constructor method, pass parameter **props** as an argument
ii. Initialize the **state** object
1. Add property **initialMatrix** initial state **[[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], [0, 0, 0, 0, 0, 0, 0]]**
2. Add property **currentPlayer** initial state to **1**

b. Define function **fillbox**, receives one parameter, **e**
i. Declare variable **colValue** set equal to function **parseInt**() of parameter **e**, object **target**, function **getAttribute**, passing as argument **"data-value"**
ii. Call function **this.setPiece**, passing arguments **5** (because we have 6 rows, 0 - 5) and variable **colValue**
iii. Call method **this.setState** to update the state of property **currentPlayer**, if currently 1 then 2, if currently 2, then 1

c. Define function **setPiece**, receives two parameters, **startCount** and **colValue**
i. Declare variable **initialMatrix** intialized to state property **initialMatrix**
ii. Declare variable **rows** initialized to object **document**, method **querySelectorAll**, passing argument class **".grid-row"**
iii. Write exception handling with **try**/**catch** to catch index out of bounds exception when array column is full
1. **try**
a. If the element in array **initialMatrix** at indexes parameters **startCount** and **colValue** is NOT identical to 0
i. Decrement parameter **startCount** by 1
ii. Call function **this.setPiece**, passing as arguments parameters **startCount** and **colValue**
b. Else
i. Declare variable **currentRow** initialized to array **rows**, index **startCount**, method **querySelectorAll**, passing as an argument class **".grid-box"**
ii. Modify **currentRow**, index **colValue**, object **classlist**, method **add**, passing as arguments **"filled"** and **player${`this.state.currentPlayer}`**
iii. Update array **initialMatrix**, indexes **startCount** and **colValue**, set equal to **this.state.currentPlayer**
iv. If function call **this.winCheck** is true
1. Display an **alert** dialog box with message **"Player " +**

this.state.currentPlayer + " wins!"

2. **return true**

2. **catch** with parameter **e**
    a. Display an **alert** dialog box with message **"Column full, select again"**
3. Call function t**his.gameOverCheck**

d. Define function **winCheck**, no parameters
    i. Write decision making logic, if function call **this.checkHorizontal**, **this.checkVertical**, **this.checkPositiveDiagonal**, or **this.checkNegativeDiagonal** is true
        1. **return true**
    ii. Else
        1. **return false**
e. Define function **checkHorizontal**, no parameters
    i. Write a nested for loop to iterate through the rows and columns
        a. If the currentPlayer has four discs in a row horizontally, **return true**
    ii. **return false**
f. Define function **checkVertical**, no parameters
    i. Write a nested for loop to iterate through the rows and columns
        a. If the currentPlayer has four discs in a row vertically, **return true**
    ii. **return false**
g. Define function **checkPositiveDiagonal**, no parameters
    i. Write a nested for loop to iterate through the rows and columns
        a. If the currentPlayer has four discs in a row diagonally, bottom right to top left, **return true**
    ii. **return false**
h. Define function **checkNegativeDiagonal**, no parameters
    i. Write a nested for loop to iterate through the rows and columns
        1. If the currentPlayer has four discs in a row diagonally, bottom left to top right, **return true**
    ii. **return false**
i. Define function **gameOverCheck**, no parameters
    i. Declare variable **count**, initialized to **0**
    ii. Declare variable **initialMatrix** initialized to **state** property **initialMatrix**
    iii. Write a **for/of** loop to iterate through the rows, loop control variable **innerArray**, in 2d array **initialMatrix**
        1. If object **innerArray**, function **every(val => (val) != 0))**
            a. increment variable **count** by 1
        2. Else
            a. return **false**
    iv. If variable count is identical to 6
        1. Display **alert** dialog box that the game is over
        2. return **true**

	j. Define function **render() {** i. **return (** the JSX element 1. Define an open **div** with class **wrapper** a. Define an open **div** with class **container** i. Define an open **div** with class **grid-row** 1. Define a **div** element with class **grid-box**, **data-value** equal to 0, and **onClick** equal to **{(e) => this.fillBox(e)}** 2. Define a **div** element with class **grid-box**, **data-value** equal to 1, and **onClick** equal to **{(e) => this.fillBox(e)}** 3. Define a **div** element with class **grid-box**, **data-value** equal to 2, and **onClick** equal to **{(e) => this.fillBox(e)}** 4. Define a **div** element with class **grid-box**, **data-value** equal to 3, and **onClick** equal to **{(e) => this.fillBox(e)}** 5. Define a **div** element with class **grid-box**, **data-value** equal to 4, and **onClick** equal to **{(e) => this.fillBox(e)}** 6. Define a **div** element with class **grid-box**, **data-value** equal to 5, and **onClick** equal to **{(e) => this.fillBox(e)}** 7. Define a **div** element with class **grid-box**, **data-value** equal to 6, and **onClick** equal to **{(e) => this.fillBox(e)}** ii. Repeat above **grid-row** five more times b. Close div with class container c. Define an open **div** with id **information** i. Defined open **div** with class **"player-wrappers"** 1. Display explicit text **Player 1** 2. Define **div** element with class **"player1"** ii. Close div iii. Defined open **div** with class **"player-wrappers"** 1. Display explicit text **Player 2** 2. Define **div** element with class

	"player2" iv. Close div d. Close div with id information 2. Close div with class wrapper k. Write the **export default** statement

Test Cases		
	Action	**Expected outcome**
Test Case 1	**Execute "npm start" in terminal window or command prompt**	When the application loads, the web browser should look similar to Figure 1
Test Case 2	**Player 1 clicks column**	The web browser updates the Connect Four game which should look similar to Figure 2
Test Case 3	**Player 2 clicks column**	When the application reloads, the web browser should look similar to Figure 1
Test Case 4	**Refresh the web browser**	When index.html loads, the web browser should look similar to Figure 1
Test Case 5	**A player has four discs in a row horizontally**	The web browser updates the Connect Four game to state the players wins, similar to Figure 4
Test Case 6	**A player has four discs in a row vertically**	The web browser updates the Connect Four game to state the players wins, similar to Figure 5
Test Case 7	**A player has four discs in a row diagonally, bottom left to top right**	The web browser updates the Connect Four game to state the players wins, similar to Figure 6
Test Case 8	**A player has four discs in a row diagonally, bottom right to top left**	The web browser updates the Connect Four game to state the players wins, similar to Figure 7
Test Case 9	**The board is full with no winner, game is over**	The web browser updates the Connect Four game to state the game is over, similar to Figure 8
Test Case 10	**A column is full and a player clicks on the column**	The web browser updates to display an alert dialog box, similar to Figure 9
Test Case 11	**Web browser console**	The web browser console should have no errors, Figure 10

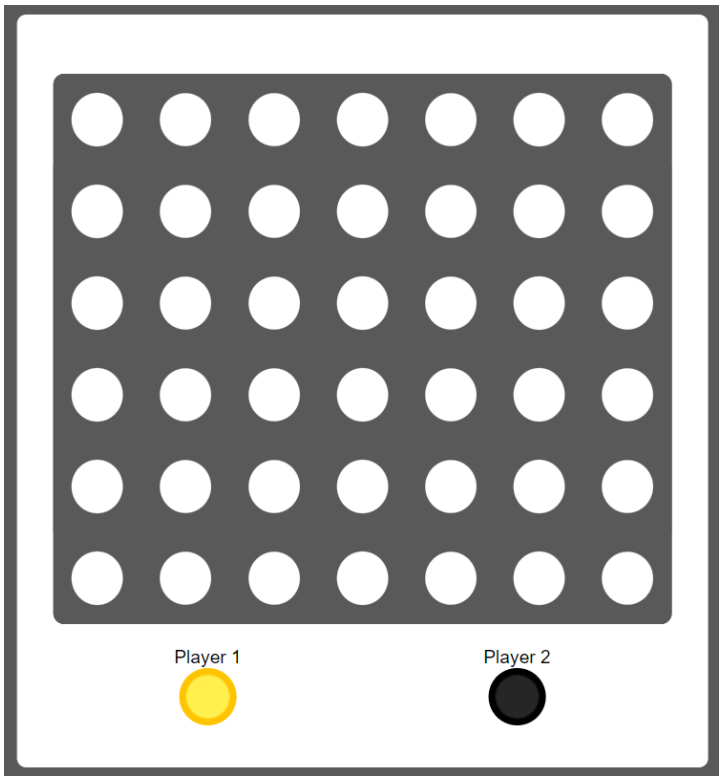


Figure 1 Connect Four

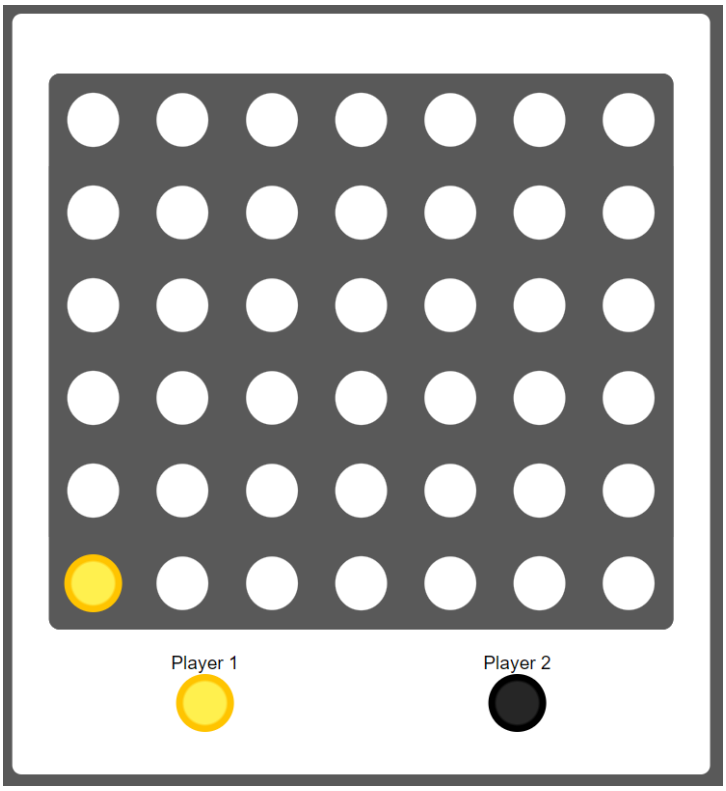


Figure 2 Player 1 turn

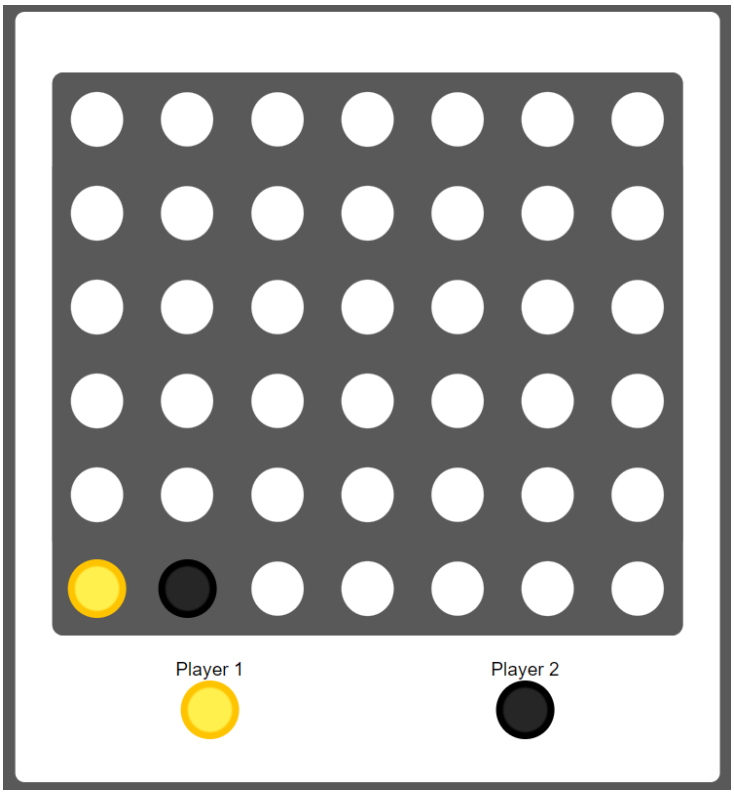


Figure 3 Player 2 turn

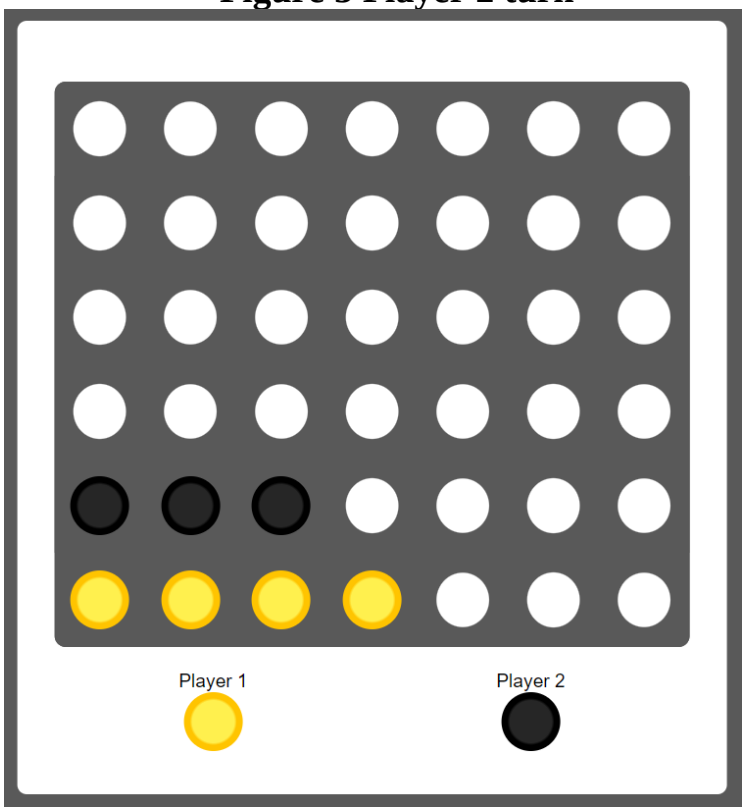


Figure 4 Horizontal win

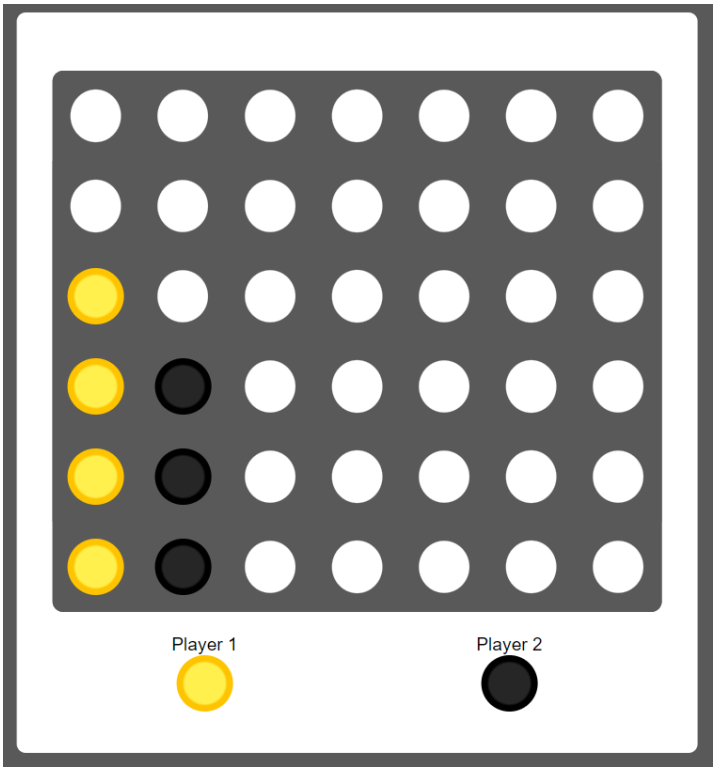


Figure 5 Vertical win

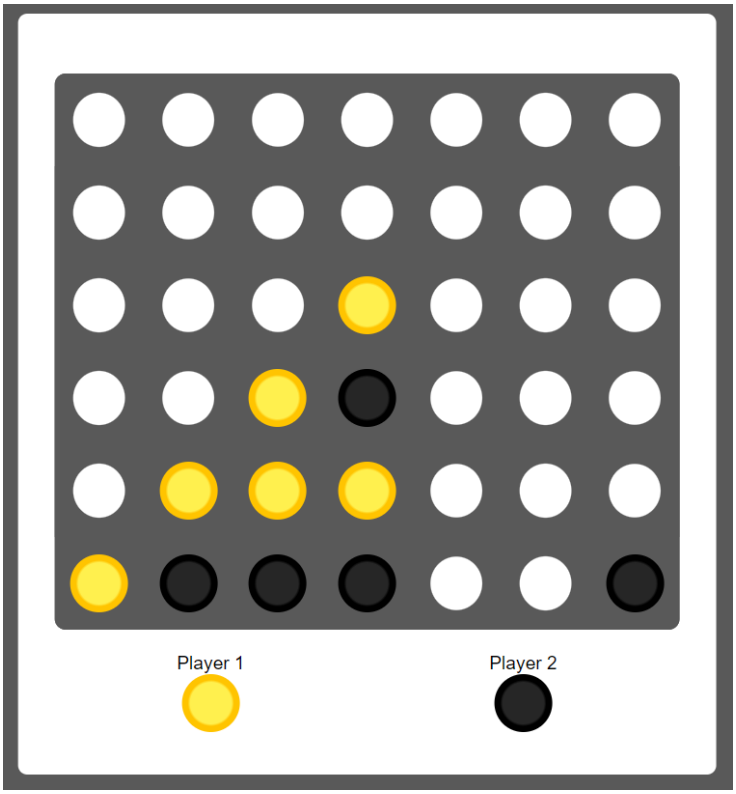


Figure 6 Diagonal win

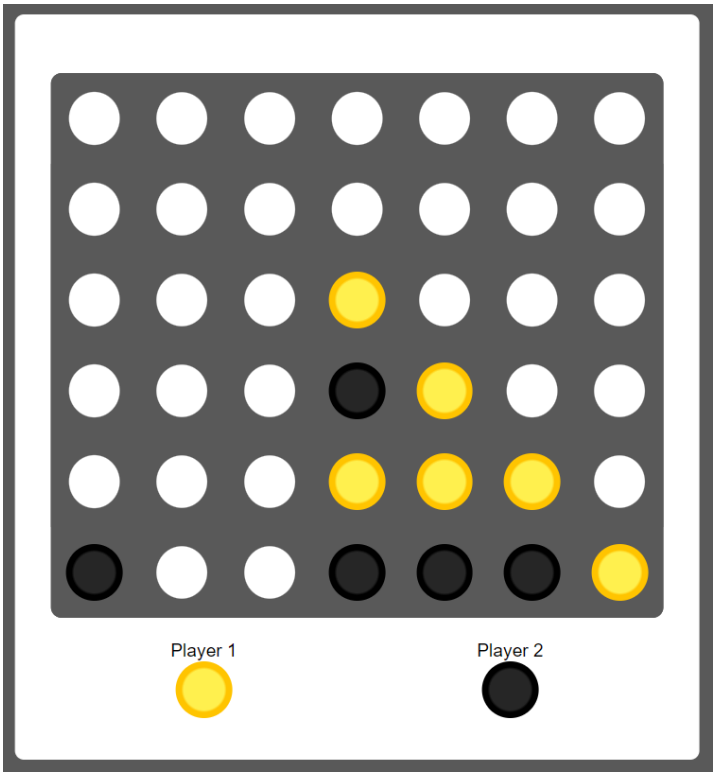


Figure 7 Diagonal win

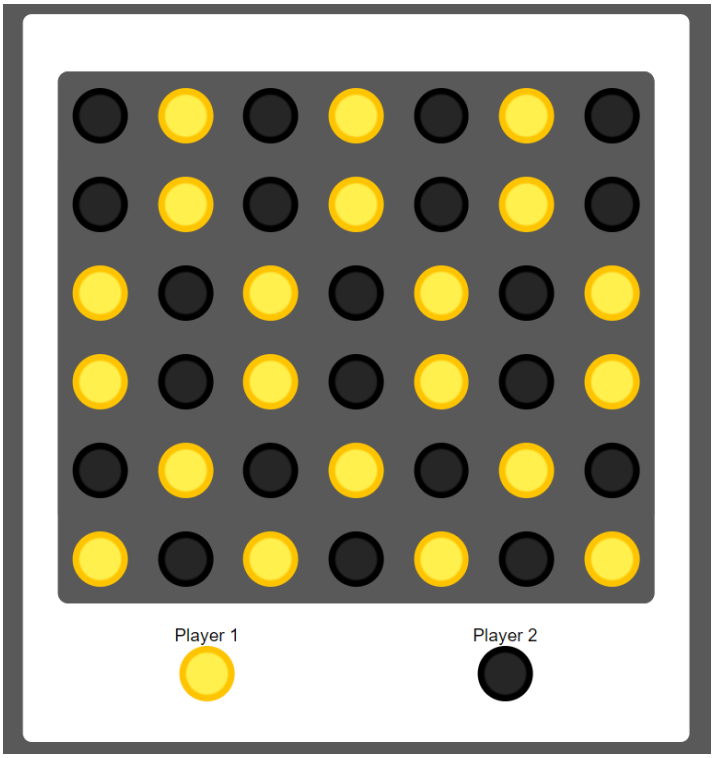


Figure 8 Game over, no winner

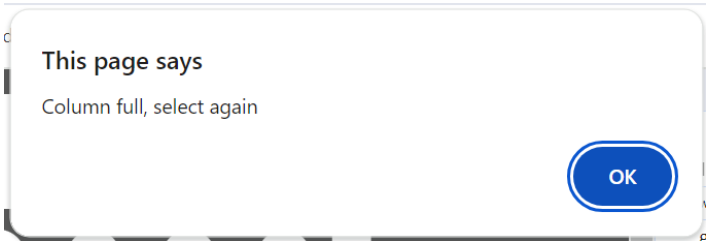


Figure 9 Column full

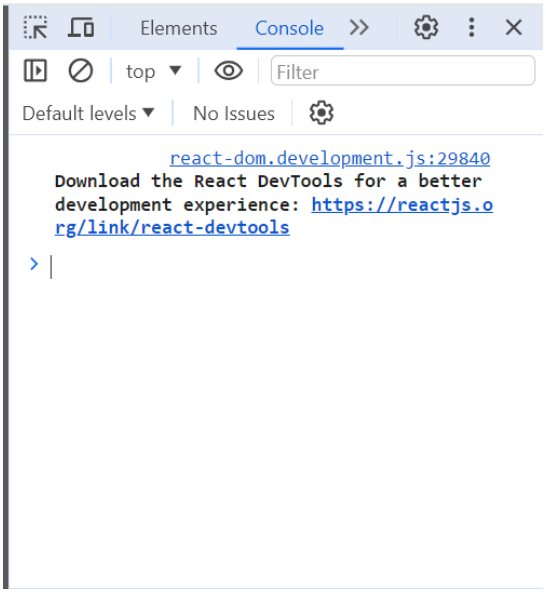


Figure 10 Web browser console