Part 1: Module coupling and cohesion

Module coupling: code extract and explanation

Example 1: Common Environment Coupling

When two software modules access the same data area, this is known as **common environment coupling**. The game state is managed by the globally defined intialMatrix in the setPiece and gameOverCheck functions.

```
const setPiece = (startCount, colValue) => {
  let rows = document.querySelectorAll(".grid-row");
  //Initially it will place the circles in the last row else if no place availabke we will decrement the count until we find empty slot
  if (initialMatrix[startCount][colValue] != 0) {
    startCount -= 1;
    setPiece(startCount, colValue);
  } else {
    //place circle
    let currentRow = rows[startCount].querySelectorAll(".grid-box");
    currentRow[colValue].classList.add("filled", `player${currentPlayer}`);
    //Update Matrix
    initialMatrix[startCount][colValue] = currentPlayer;
    //Check for wins
    if (winCheck(startCount, colValue)) {
      message.innerHTML = `Player<span> ${currentPlayer}</span> wins`;
      startScreen.classList.remove("hide");
      return false;
    }
  }
  //Check if all are full
  gameOverCheck();
};
```

In the setPiece function, it uses the intialMatrix to check if the current slot is occupied. The function moves the disc to the next available slot by continuously calling itself and adjusting the row if the slot is already occupied. In order to decide where to put the disc, the setPiece function accesses the global variable initialMatrix.

```
const gameOverCheck = () => {
  let truthCounnt = 0;
  for (let innerArray of initialMatrix) {
    if (innerArray.every((val) => val != 0)) {
      truthCounnt += 1;
    } else {
      return false;
    }
  }
  if (truthCounnt == 6) {
    message.innerText = "Game Over";
    startScreen.classList.remove("hide");
  }
};
```

To determine whether the game is over if every row is filled, the gameOverCheck function uses initialMatrix. InitialMatrix is used by both functions to control the game state. Both functions use and change the same data, the initialMatrix, which has an impact on how the game starts and finishes.

Example 2: Control Coupling

Control coupling is shown here as fillBox function communicated information to the setPiece function for explicit purpose of influencing the latter module's execution.

```
//When user clicks on a box
const fillBox = (e) => {
  //get column value
  let colValue = parseInt(e.target.getAttribute("data-value"));
  //5 because we have 6 rows (0-5)
  setPiece(5, colValue);
  currentPlayer = currentPlayer == 1 ? 2 : 1;

  playerTurn.innerHTML = `Player <span>${currentPlayer}'s</span> turn`;
};
```

In the fillBox function, it passes parameters startCount and colValue to the setPiece function.

```
const setPiece = (startCount, colValue) => {
  let rows = document.querySelectorAll(".grid-row");
  //Initially it will place the circles in the last row else if no place avail
  if (initialMatrix[startCount][colValue] != 0) {
    startCount -= 1;
    setPiece(startCount, colValue);
  } else {
    //place circle
    let currentRow = rows[startCount].querySelectorAll(".grid-box");
    currentRow[colValue].classList.add("filled", `player${currentPlayer}`);
    //Update Matrix
    initialMatrix[startCount][colValue] = currentPlayer;
    //Check for wins
    if (winCheck(startCount, colValue)) {
      message.innerHTML = `Player<span> ${currentPlayer}</span> wins`;
      startScreen.classList.remove("hide");
      return false;
    }
  }
  //Check if all are full
  gameOverCheck();
};
```

The startCount and colValue parameters are then passed to the setPiece function, which utilizes them to determine the disc's placement. By indicating which row to start from and which column the disc should be placed in, the data passed by the fillBox function affects the setPiece function. By sending information, the fillBox function controls what the setPiece function does, demonstrating control coupling.

Module cohesion: code extract and explanation

Example 1: Logical Cohesion

By checking the win condition in various directions but distinct operations, the winCheck function demonstrates **logical cohesion** in its tasks. Instead of concentrating on a single task, it combines related tasks to determine whether the game is won. With a focus on checking the win condition in multiple directions and

logically grouping these operations together, the winCheck function is very cohesive because all its tasks are directly related to determining whether the game is won.

```
const winCheck = (row, column) => {
  //if any of the functions return true we return true
  return checkAdjacentRowValues(row)
    ? true
    : checkAdjacentColumnValues(column)
    ? true
    : checkAdjacentDiagonalValues(row, column)
    ? true
    : false;
};
```

This function verifies the win condition in the row, column, and diagonal directions. Since they all work toward determining whether a player has won, these operations are related. They carry out different tasks, though. To handle each direction, the function invokes three distinct helper functions.

Example 2: Functional Cohesion

Functional cohesion is shown by the verifyArray function. When all of the tasks carried out by a software module work together to perform a single function, this is known as function cohesion. Because every operation in the function is connected to a single task checking for a win condition in a sequence it is extremely cohesive.

```
const verifyArray = (arrayElement) => {
  let bool = false;
  let elementCount = 0;
  arrayElement.forEach((element, index) => {
    if (element == currentPlayer) {
      elementCount += 1;
      if (elementCount == 4) {
        bool = true;
      }
    } else {
      elementCount = 0;
    }
  });
  return bool;
};
```

This function decides if each array contains the current player's four consecutive discs. Its primary goal is to confirm that the requirements for winning are fulfilled. The verifyArray function only checks to see if the array contains four consecutive values; it doesn't worry about other game-related tasks.