Part 1: Module coupling and cohesion

Module coupling example 1:

1) Data Coupling:

```
1  //Initial references
2  const container = document.querySelector(".container");
3  const playerTurn = document.getElementById("playerTurn");
4  const startScreen = document.querySelector(".startScreen");
5  const startButton = document.getElementById("start");
6  const message = document.getElementById("message");
7  let initialMatrix = [
8    [0, 0, 0, 0, 0, 0, 0],
9    [0, 0, 0, 0, 0, 0, 0],
10   [0, 0, 0, 0, 0, 0, 0],
11   [0, 0, 0, 0, 0, 0, 0],
12   [0, 0, 0, 0, 0, 0, 0],
13   [0, 0, 0, 0, 0, 0, 0],
14 ];
15 let currentPlayer;
```

Explanation:

The integration of global variables, including but not limited to 'container,' 'playerturn,' 'startscreen,' 'startbutton,' 'message,' 'initialmatrix,' and 'currentplayer,' introduces a notably heightened degree of coupling within the codebase. These variables, serving as common threads woven across various functions and modules, establish intricate dependencies among them. The interwoven nature of these shared variables fosters a nuanced network of relationships, wherein any modification introduced to these global entities within a specific segment of the codebase has the potential to reverberate and influence the behavior of other interconnected parts, thereby cultivating a state of tightly-knit coupling. This inherent interdependence underscores the imperative of judiciously considering and managing these shared variables to strike a balance between cohesion and flexibility in the broader software architecture.

Module coupling example 2: code extract and explanation 2

2) Control Coupling:

```
//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`;
};



//Initialise game
window.onload = startGame = async () => {
  //Between 1 and 2
  currentPlayer = generateRandomNumber(1, 3);
  container.innerHTML = "";
  await matrixCreator();
  playerTurn.innerHTML = `Player <span>${currentPlayer}'s</span> turn`;
};
```

Explanation:

The event handler 'fillbox' intricately intertwines itself with both the 'setPiece' function and the 'update' mechanism pertaining to the current player, creating a perceptible tight coupling. This coexistence during the same timeframe introduces a delicate interdependence, wherein alterations to the order or structure of these functions might potentially trigger unforeseen consequences. It is in this delicate nexus that the significance of decoupling becomes apparent, as reducing this coupling could usher in a more resilient and modularized game logic. This intentional disentanglement not only mitigates the risk of unintended repercussions but also advocates for a refined and structured approach to the interplay between event handling and the broader game logic.

Module cohesion example 1: code extract and explanation

1) Logical Cohesion in winCheck Function:

```
//Win check logic
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;
};
```

Explanation:

The 'winCheck' function expertly showcases logical cohesion, centering its efforts on a singular, well-defined task — the nuanced exploration of a player's potential victory in the game. This strategic approach unfolds through a meticulous orchestration, where the win-checking logic gracefully delegates its responsibilities to a suite of specialized functions, including the discerning 'checkAdjacentRowValues,' 'checkAdjacentColumnValues,' and 'checkAdjacentDiagonalValues.' This deliberate design philosophy not only underscores the function's dedication to a streamlined purpose but also contributes significantly to the overall ecosystem's maintainability and readability. In essence, each of these intricately interwoven functions assumes a distinctive role, ensuring a harmonious balance between complexity and clarity within the broader scope of the game's logical framework.

Module cohesion example 2: code extract and explanation

2. Functional Cohesion in setPiece Function:

```
66  //Sets the circle to exact points
67  const setPiece = (startCount, colValue) => {
68    let rows = document.querySelectorAll(".grid-row");
69    //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
70    if (initialMatrix[startCount][colValue] != 0) {
71      startCount -= 1;
72      setPiece(startCount, colValue);
73    } else {
74      //place circle
75      let currentRow = rows[startCount].querySelectorAll(".grid-box");
76      currentRow[colValue].classList.add("filled", `player${currentPlayer}`);
77      //Update Matrix
78      initialMatrix[startCount][colValue] = currentPlayer;
79      //Check for wins
80      if (winCheck(startCount, colValue)) {
81        message.innerHTML = `Player<span> ${currentPlayer}</span> wins`;
82        startScreen.classList.remove("hide");
83        return false;
84      }
85    }
86    //Check if all are full
87    gameOverCheck();
88  };
89
```

Explanation:

The 'setPiece' function exemplifies functional cohesion through its adept management of both the strategic placement of a game piece and the subsequent meticulous examination for win or game-over conditions. This comprehensive encapsulation of interconnected operations within the confines of a singular function not only heightens the overall understandability and maintainability but also underscores a deliberate design philosophy that advocates for a discernible separation of concerns. Each meticulously crafted function within this design architecture assumes responsibility for a specific and nuanced facet of the game's dynamic state, contributing to a holistic and intricate yet comprehensible structure.