Comprehensive Exercise Report

Team <<1>> of Section <<000>>

Pooja Odedara , 221ADB033 ,GITHUB ID:- <https://github.com/Monster2307>

Kushagra Singh, 221ADB063, GITHUB ID:-<https://github.com/Kushagrasingh23/DIP392-KP>

NOTE: You will replace all placeholders that are given in <<>>

[**Requirements/Analysis**](#_uwgqwd5ezv2w) **2**

[Journal](#_lsityg2iq9m6) 2

[Software Requirements](#_2h0vru1u2mla) 3

[**Black-Box Testing**](#_prhaxdxmf8n8) **4**

[Journal](#_18f11w613jft) 4

[Black-box Test Cases](#_2xn4jzot820y) 5

[**Design**](#_24fdizefyocn) **6**

[Journal](#_esp2ocs9j6bk) 6

[Software Design](#_aifbl1x6rddt) 7

[**Implementation**](#_hya8f3jqkba6) **8**

[Journal](#_acupzfhai7gz) 8

[Implementation Details](#_ojhtwkms2z3b) 9

[**Testing**](#_3qvya3vi836q) **10**

[Journal](#_ckfs4xbl5pyr) 10

[Testing Details](#_bzt1547yxzxi) 11

[**Presentation**](#_hdjvrbf45b1p) **12**

[Preparation](#_xbiquwtmf36n) 12

[**Grading Rubric**](#_u0hfnmdgusmf) **13**

# Requirements/Analysis

Week 2

## Journal

The following prompts are meant to aid your thought process as you complete the requirements/analysis portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* After reading the client’s brief (possibly incomplete description), write one sentence that describes the project (expected software) and list the already known requirements.
  + - Description: Create a software implementation of the Connect Four game.
    - Known requirements from client description
      * The game should allow two players to play against each other.
      * The game board consists of a 7-column and 6-row grid.
      * Players take turns to drop discs into columns.
      * The game ends when a player connects four discs in a row, column, or diagonal.
* After reading the client’s brief (possibly incomplete description), what questions do you have for the client? Are there any pieces that are unclear? After you have a list of questions, raise your hand and ask the client (your instructor) the questions; make sure to document his/her answers.

Questions for the Client

Questions:

* + - Is there a need for a computer AI player or just human players?
    - Should the game include a graphical user interface (GUI) or just a command-line interface (CLI)?
    - Are there specific platforms or operating systems the game needs to support?
    - Should the game have a feature to save and load game states?

Instructor's Answers:

* + Yes, a computer AI player would be a valuable addition.
  + A GUI is preferred to enhance user experience.
  + The game should be compatible with Windows, macOS, and Linux.
  + Implementing save and load functionality would be beneficial.
* Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.
  + Topics:
    - Implementation of AI in games.
    - GUI development in Python (using libraries like Tkinter or Pygame).
    - References:
    - "Artificial Intelligence: A Modern Approach" by Stuart Russell and Peter Norvig.
    - Tkinter documentation (<https://docs.python.org/3/library/tkinter.html>).
    - Pygame documentation (<https://www.pygame.org/docs/>).
* Describe the users of this software (e.g., small child, high school teacher who is taking attendance).
  + The primary users are casual gamers who enjoy playing classic board games.
* Describe how each user would interact with the software.
  + Players: Start a new game. Choose to play against another human. Select columns to drop their discs. Save and load game states.
* What features must the software have? What should the users be able to do?
  + Must have:
    - Two-player mode.
    - Option to play against computer AI.
    - GUI for intuitive interaction.
    - Game state saving and loading.
  + Should allow:
    - Viewing the game rules and instructions.
    - Customizing the look of the game (e.g., disc colors).
* Other notes:
  + Consider implementing sound effects for disc drops and win notifications.
  + Ensure the game is responsive and performs well across different devices.

## Software Requirements

<<Use your notes from above to complete this section of the formal documentation by writing a detailed description of the project, including a paragraph overview of the project followed by a list of requirements (see lecture for format of requirements). You may also choose to include user stories.>>

Overview of the Project:

The Connect Four game project involves creating a software implementation of the classic two-player connection game. In this game, players take turns dropping colored discs into a vertical grid with the objective of being the first to form a horizontal, vertical, or diagonal line of four of their own discs. The game requires strategic thinking and planning, making it both challenging and entertaining. The project will focus on developing a user-friendly and visually appealing interface, providing an engaging experience for players. The game will also feature a computer AI opponent to allow for single-player gameplay.

**Project Requirements:**

1. **Functional Requirements:**
   * **Gameplay Mechanics:**
     + Two-player mode where players can take turns.
     + One-player mode with an AI opponent.
     + A 7-column by 6-row grid where players drop discs.
     + Detection of win conditions (four discs in a row, column, or diagonal).
     + Handling of draw conditions when the grid is full with no winner.
   * **User Interface:**
     + Graphical User Interface (GUI) for intuitive interaction.
     + Visual representation of the grid and discs.
     + Highlighting the winning line when a player wins.
     + Buttons for starting a new game and saving/loading game states.
   * **Platform Compatibility:**
     + The game should run on Windows, macOS, and Linux.
   * **Additional Features:**
     + Sound effects for disc drops and win notifications.
     + Customizable settings for disc colors and other visual elements.
     + Instructions and game rules accessible from the main menu.
2. **Non-Functional Requirements:**
   * **Performance:**
     + The game should be responsive and perform well without lag.
   * **Usability:**
     + The interface should be easy to navigate, even for users unfamiliar with the game.
     + Clear visual cues and feedback for player actions.
   * **Reliability:**
     + The game should handle edge cases gracefully, such as invalid moves or full columns.
   * **Maintainability:**
     + The codebase should be well-documented and structured for ease of maintenance and future enhancements.

**Examples of User Stories for Connect Four Game Project**

* **Title**: Two-Player Mode
  + **Story**: As a casual gamer, I want to play Connect Four with a friend on the same computer so that we can enjoy a classic game together.
* **Title**: Single Player Mode with AI
  + **Story**: As a solo player, I want to play against a computer AI so that I can enjoy the game even when I don't have a friend available.
* **Title**: Turn Indicator
  + **Story**: As a player, I want the game to clearly indicate whose turn it is so that we don't get confused about the gameplay.
* **Title**: Highlight Winning Line
  + **Story**: As a player, I want the game to highlight the winning line when someone wins so that it is easy to see how the game was won.
* **Title**: Save and Load Game
  + **Story**: As a player, I want to save my game and load it later so that I can continue playing from where I left off.
* **Title**: Visual and Sound Effects
  + **Story**: As a user, I want the game to be visually appealing and have sound effects to enhance the overall experience

# Black-Box Testing

Instructions: Week 4

## Journal

***Remember:*** Black box tests should only be based on your requirements and should work independent of design.

The following prompts are meant to aid your thought process as you complete the black box testing portion of this exercise. Please review your list of requirements and respond to each of the prompts below. Feel free to add additional notes.

* What does input for the software look like (e.g., what type of data, how many pieces of data)?
  + The input for the software includes player actions such as selecting a column to drop a disc, color selections for the discs via color input fields, and mode selection between single-player and two-player. The data types include integers for column selection, color codes for disc customization, and boolean values for mode selection.
* What does output for the software look like (e.g., what type of data, how many pieces of data)?
  + The output includes visual updates to the game board, status messages indicating the current player’s turn, win or draw notifications via modal dialogs, and highlighted winning lines. The output is primarily visual, supported by text messages.
* What equivalence classes can the input be broken into?
  + Equivalence classes for input can be categorized as:
  + Valid column selections (0 to 6).
  + Invalid column selections (less than 0 or greater than 6).
  + Full column selections (when a column is already filled with discs).
  + Color inputs (valid hex color codes).
  + Mode selections (single-player or two-player).
* What boundary values exist for the input?
  + Boundary values include:
  + Column indices: -1, 0, 6, 7 (to test edge cases for valid and invalid column selections).
  + Grid boundaries: Testing for the top-most and bottom-most rows when dropping discs.
  + Full grid: Testing when all cells are filled and checking for draw conditions.
* Are there other cases that must be tested to test all requirements?
  + Yes, other cases include:
  + Switching between game modes (single-player to two-player and vice versa).
  + Testing AI move logic in single-player mode.
  + Checking the persistence and loading of game states if implemented.
  + Customizing disc colors and ensuring visual updates occur correctly.
* Other notes:
  + Consider stress testing by simulating rapid consecutive moves to ensure the game handles quick interactions smoothly. Additionally, validate the responsiveness of the UI on different screen sizes.

## Black-box Test Cases

Use your notes from above to complete the black-box test plan section of the formal documentation by writing black box test cases (other than actual results since no program currently exists). Remember to test each equivalence class, boundary value, and requirement.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| 1 | Select a valid column to drop a disc | Disc is placed in the selected column | Disc is placed in the selected column |
| 2 | Select an invalid column (e.g., -1 or 7) | No disc is placed, and error is handled | No disc is placed, and error is handled |
| 3 | Select a full column | No disc is placed, and error is handled | No disc is placed, and error is handled |
| 4 | Switch to single-player mode | AI makes a move after the player’s turn | AI makes a move after the player’s turn |
| 5 | Switch to two-player mode | Players take turns without AI interference | Players take turns without AI interference |
| 6 | Customize player disc colors | Discs reflect the selected colors | Discs reflect the selected colors |
| 7 | Fill the board without any player winning | Game declares a draw | Game declares a draw |
| 8 | Connect four discs horizontally | Game declares a win and highlights the line | Game declares a win and highlights the line |
| 9 | Connect four discs vertically | Game declares a win and highlights the line | Game declares a win and highlights the line |
| 10 | Connect four discs diagonally | Game declares a win and highlights the line | Game declares a win and highlights the line |
| 11 | Open and close instructions | Instructions modal opens and closes correctly | Instructions modal opens and closes correctly |
| 12 | Restart the game | Game resets to initial state | Game resets to initial state |

# Design

Instructions: Week 6

## Journal

***Remember:*** You still will not be writing code at this point in the process.

The following prompts are meant to aid your thought process as you complete the design portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* List the nouns from your requirements/analysis documentation.
  + - Game
    - Player
    - AI
    - Grid
    - Disc
    - Turn
    - Win Condition
    - Draw Condition
    - GUI (Graphical User Interface)
    - Settings
    - Color
    - Instructions
    - Game State
    - Column
    - Row
* Which nouns potentially may represent a class in your design?

#### Potential Classes

* Game
* Player
* AI
* Grid
* Disc
* Settings
* Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.

 **Game:**

* players (list of Player)
* grid (Grid)
* currentPlayer (Player)
* gameState (GameState)
* settings (Settings)

 **Player:**

* name (string)
* discColor (string)
* isAI (boolean)

 **AI (inherits from Player):**

* difficultyLevel (string)

 **Grid:**

* columns (int)
* rows (int)
* cells (2D list of Disc)

 **Disc:**

* color (string)
* position (tuple)

 **Settings:**

* player1Color (string)
* player2Color (string)
* Now that you have a list of possible classes, consider different design options (***lists of classes and attributes***) along with the pros and cons of each. We often do not come up with the best design on our first attempt. Also consider whether any needed classes are missing. These two design options should not be GUI vs. non-GUI; instead, you need to include the classes and attributes for each design. Reminder: Each design must include at least two classes that define object types.

**Option 1:**

* **Classes:** Game, Player, Grid, Disc, Settings
* **Attributes:** Defined as above

**Pros:**

* Simple and straightforward design.
* Easy to understand and maintain.
* Clear separation of responsibilities.

**Cons:**

* Might require more inter-class communication, leading to increased coupling.

**Option 2:**

* **Classes:** Game, Player, AI (inherits from Player), Grid, Disc, Settings
* **Attributes:** Defined as above, with AI inheriting from Player

**Pros:**

* Better handling of AI-specific logic by separating AI from Player.
* More extensible for future features like different AI difficulty levels.

**Cons:**

* Slightly more complex due to inheritance.
* Might require additional logic to handle AI behavior separately.
* Which design do you plan to use? Explain why you have chosen this design.

**Chosen Design:** Option 2

* **Reason:** This design separates the AI logic from the Player class, making it more modular and easier to extend in the future. It also keeps the design clean by leveraging inheritance, which aligns with object-oriented principles.
* List the verbs from your requirements/analysis documentation.

 Play

 Drop

 Take Turn

 Detect Win

 Detect Draw

 Highlight

 Start New Game

 Save Game

 Load Game

 Change Settings

 Display Instructions

* Which verbs potentially may represent a method in your design? Also list the class each method would be part of.

 **Game:**

* startNewGame()
* saveGame()
* loadGame()
* switchTurn()
* checkWinCondition()
* checkDrawCondition()

 **Player:**

* takeTurn()

 **AI:**

* makeMove()

 **Grid:**

* dropDisc(column)
* highlightWinningLine(startPosition, endPosition)

 **Settings:**

* updateSettings()
* Other notes:
  + **Encapsulation:** Ensure each class has private fields with public getters and setters.
  + **Modularity:** Each class should handle its specific responsibility to adhere to the single responsibility principle.
  + **Extensibility:** The design should be easy to extend with additional features like different AI difficulty levels or additional game modes.

## Software Design

<<Use your notes from above to complete this section of the formal documentation by planning the classes, methods, and fields that will used in the software. Your design should include UML class diagrams along with method headers. ***Prior to starting the formal documentation, you should show your answers to the above prompts to your instructor.****>>*

Class Design Overview

Based on the analysis and implementation notes, we have identified the following key classes for the Connect Four game:

1. **Game**
2. **Player**
3. **AI (inherits from Player)**
4. **Grid**
5. **Disc**
6. **Settings**

#### Method Headers

Here are the method headers for the key methods in each class:

**Game Class:**

**Game Logic (in script.js):**

javascript

// Function to start a new game

function startNewGame() {

// Initialize a new game

}

// Function to switch to the next player's turn

function switchTurn() {

// Switch to the next player's turn

}

// Function to check if the current player has won

function checkWinCondition(row, col) {

// Check if the current player has won

return false;

}

// Function to check if the game is a draw

function checkDrawCondition() {

// Check if the game is a draw

return false;

}

**Player Class (Conceptual):**

javascript

class Player {

constructor(name, discColor) {

this.name = name;

this.discColor = discColor;

}

takeTurn(col) {

// Player takes a turn

}

}

**AI Class (Conceptual):**

javascript

class AI extends Player {

constructor(name, discColor, difficultyLevel) {

super(name, discColor);

this.difficultyLevel = difficultyLevel;

}

makeMove() {

// AI makes a move

}

}

**Grid Logic (in script.js):**

javascript

// Function to create the game board

function createBoard(gameBoard) {

gameBoard.innerHTML = '';

for (let i = 0; i < ROWS \* COLS; i++) {

const cell = document.createElement('div');

cell.classList.add('cell');

gameBoard.appendChild(cell);

}

}

// Function to drop a disc into the specified column

function dropDisc(colIndex) {

for (let rowIndex = ROWS - 1; rowIndex >= 0; rowIndex--) {

if (board[rowIndex][colIndex] === null) {

board[rowIndex][colIndex] = currentPlayer;

return rowIndex;

}

}

return -1;

}

// Function to highlight the winning line on the grid

function highlightWinningLine(row, col) {

const directions = [

{ rowDir: 0, colDir: 1 },

{ rowDir: 1, colDir: 0 },

{ rowDir: 1, colDir: 1 },

{ rowDir: 1, colDir: -1 }

];

for (const { rowDir, colDir } of directions) {

let count = 0;

let cells = [];

for (let i = -3; i <= 3; i++) {

const r = row + i \* rowDir;

const c = col + i \* colDir;

if (r >= 0 && r < ROWS && c >= 0 && c < COLS && board[r][c] === currentPlayer) {

count++;

cells.push(document.querySelector(`#game-board`).children[r \* COLS + c]);

if (count === 4) {

cells.forEach(cell => cell.classList.add('winning-cell'));

return;

}

} else {

count = 0;

cells = [];

}

}

}

}

**Settings Logic (in script.js):**

javascript

// Function to update game settings

function updateSettings() {

const player1Color = document.getElementById('player1-color').value;

const player2Color = document.getElementById('player2-color').value;

const style = document.createElement('style');

style.innerHTML = `

.cell.player1 { background-color: ${player1Color}; }

.cell.player2 { background-color: ${player2Color}; }

`;

document.head.appendChild(style);

}

#### Notes on Implementation

* **Encapsulation:** Each part of the game logic is encapsulated within functions and object properties.
* **Modularity:** The design ensures that each function handles a specific part of the game’s functionality, adhering to the single responsibility principle.
* **Extensibility:** The design is easy to extend with additional features, such as different AI difficulty levels or additional game modes.

# Implementation

Instructions: Week 8

## Journal

The following prompts are meant to aid your thought process as you complete the implementation portion of this exercise. Please respond to each of the prompt below and feel free to add additional notes.

* What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation.

To outline the programming concepts we will need to implement our design for the Connect Four game project. These concepts come from our course and will be applied as follows:

1. **Object-Oriented Programming (OOP):**
   * **Classes and Objects:** We used a class-like structure in JavaScript to encapsulate game logic and state. Functions and variables are scoped to manage game flow, players, and the grid state.
     + **Game:** Manages the overall game flow, players, and game state.
     + **Player:** Represents a player, either human or AI.
     + **Grid:** Manages the grid where discs are placed.
     + **Disc:** Represented by HTML elements managed within JavaScript.
   * **Encapsulation:** Variables like currentPlayer, gameOver, and board are managed within the script, ensuring controlled access to the game’s internal state.
2. **Event-Driven Programming:**
   * **Event Listeners:** JavaScript event listeners handle user interactions, such as clicking on a column to drop a disc, starting a new game, or changing settings. This updates the game state and UI in response to user actions. For example, listeners are set up for the game board, restart button, and color input changes.
3. **Data Structures:**
   * **Arrays and Lists:** A 2D array represents the game grid, storing the state of each cell (either null or the current player's disc).
   * **Grid:** The grid is dynamically created and updated using HTML and CSS, managed by JavaScript functions.
4. **Control Structures:**
   * **Loops:** We use loops to iterate over the grid to check for win conditions, fill empty columns, and handle the AI’s moves. For example, for loops are used to create the game board and to check win conditions.
   * **Conditional Statements:** We implement game logic using if statements to validate moves, detect wins or draws, and toggle turns between players.
5. **Algorithm Design:**
   * **Win Detection:** We implemented algorithms to check for four consecutive discs in rows, columns, or diagonals. This is done within the checkDirection function.
   * **AI Move Calculation:** We implemented a basic algorithm for the AI to choose a column randomly, as seen in the aiMove function.
6. **GUI Programming:**
   * **HTML/CSS/JavaScript:** We created a user interface that allows players to interact with the game.
     + **HTML:** Defines the game board, buttons, and other UI elements. Elements like #game-board, #status, and buttons for game control are defined here.
     + **CSS:** Styles the game board and discs to provide a visually appealing interface. CSS classes like. cell, .player1, and .player2 define the appearance of game elements.
     + **JavaScript:** Manages game logic, updates the UI in response to game events, and handles user input. Functions such as createBoard, dropDisc, and event listeners manage game interactions.

#### Other notes:

* **User Experience:** We ensured the game is user-friendly by providing clear instructions, intuitive controls, and responsive feedback. Features like highlighting the winning line, playing sound effects, and allowing customization of disc colors are included to enhance the experience.
* **Testing:** While not explicitly shown in the code provided, testing was conducted manually to ensure functions like win detection and AI move calculation work correctly.
* **Documentation:** We commented the code to explain the purpose of each significant function and block of code. This helps maintain readability and ease of understanding for others who may work on the code.

#### Code Implementation Highlights:

* **Two-Player and Single-Player Mode:** Implemented using buttons to toggle between modes.
* **Color Customization:** Color input elements allow players to choose their disc colors.
* **Win and Draw Detection:** Functions checkWin, checkDirection, and isBoardFull handle game outcomes.
* **AI Moves:** Simple AI implemented to make random moves in single-player mode.
* **User Interface:** Dynamic creation and updating of the game board, with visual and modal feedback for game status and instructions.

## Implementation Details

<<Use your notes from above to write code and complete this section of the formal documentation with a README for the user that explains how he/she will interact with the system.>>

**README for Connect Four Game**

Welcome to the Connect Four Game! This README file will guide you through the features and instructions on how to interact with the game. Follow the steps below to get started.

**Features**

1. **Two-Player and Single-Player Modes:**
   * Play against another human or an AI opponent.
2. **Color Customization:**
   * Customize the disc colors for both players.
3. **Win and Draw Detection:**
   * The game detects and announces wins and draws automatically.
4. **AI Moves:**
   * The AI makes random moves in single-player mode.
5. **User Interface:**
   * Dynamic and intuitive UI with visual feedback for game status and instructions.

**How to Play**

1. **Start the Game:**
   * Open the index.html file in your web browser.
   * You will see the main game interface with buttons for Two Player Mode and Single Player Mode.
2. **Select Game Mode:**
   * Click the "Two Player Mode" button to play against another human.
   * Click the "Single Player Mode" button to play against the AI.
3. **Customize Disc Colors:**
   * Use the color input fields to select colors for Player 1 and Player 2/AI discs.
4. **Playing the Game:**
   * Player 1 starts the game. Drop a disc by clicking on the desired column in the grid.
   * Players take turns to drop their discs into one of the seven columns.
   * The game will automatically detect four consecutive discs horizontally, vertically, or diagonally to declare a win.
   * If the grid is full and no player has won, the game will declare a draw.
5. **Game Status:**
   * The current player's turn is displayed at the top of the game board.
   * When a player wins, the winning line is highlighted, and a modal displays the winner.
   * In case of a draw, a modal displays the draw status.
6. **Restarting the Game:**
   * Click the "Restart Game" button in the modal to start a new game with the same settings.
7. **Instructions:**
   * Click the "Instructions" button to view the game rules and instructions.
   * Close the instructions by clicking the "Close" button.

**Code Structure**

The code is structured into HTML, CSS, and JavaScript files:

1. **index.html:**
   * Defines the layout and elements of the game, including the game board, settings, and instructions modal.
2. **style.css:**
   * Styles the game elements, including the game board, discs, and modals for a visually appealing interface.
3. **script.js:**
   * Contains the game logic, including event listeners for user interactions, game state management, win and draw detection, AI move logic, and updating the UI based on the game state.

### Key Functions

Below is a brief explanation of some key functions in the script.js file:

* **createBoard(gameBoard):** Dynamically creates the game board with grid cells.
* **dropDisc(colIndex):** Handles the logic for dropping a disc into a column and returns the row index where the disc lands.
* **checkWin(row, col):** Checks for a winning condition starting from the specified row and column.
* **checkDirection(row, col, rowDir, colDir):** Checks a specific direction for four consecutive discs.
* **isBoardFull():** Checks if the board is full, indicating a draw.
* **showModal(message):** Displays a modal with the specified message.
* **restartGame():** Resets the game state for a new game.
* **highlightWinningLine(row, col):** Highlights the winning line when a player wins.
* **aiMove():** AI logic for making a move in single-player mode.
* **updateColors():** Updates the disc colors based on user selection.

### Example Usage

1. Open index.html in a web browser.
2. Select either "Two Player Mode" or "Single Player Mode."
3. Customize the disc colors if desired.
4. Click on the columns to drop discs and play the game.
5. The game will indicate a win or draw automatically.
6. Restart the game or view instructions as needed.

Enjoy playing Connect Four!!!

# Testing

Instructions: Week 10

## Journal

The following prompts are meant to aid your thought process as you complete the testing portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* Have you changed any requirements since you completed the black box test plan? If so, list changes below and update your black-box test plan appropriately.
  + No requirements have been changed since the completion of the black-box test plan.
* List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.
  + **Classes**:
  + **Game**:
  + **Equivalence Classes**: Valid game states, invalid game states, saved game states.
  + **Boundary Values**: Start of the game, end of the game, full board.
  + **Paths to Test**: Starting a game, switching turns, checking win conditions, restarting a game.
  + **Player**:
  + **Equivalence Classes**: Human player, AI player.
  + **Boundary Values**: Player 1, Player 2.
  + **Paths to Test**: Taking turns, disc color updates.
  + **Grid**:
  + **Equivalence Classes**: Empty grid, partially filled grid, fully filled grid.
  + **Boundary Values**: Top row, bottom row, full column.
  + **Paths to Test**: Dropping discs, checking grid updates, highlighting winning lines.
* Other notes:
  + Ensure all edge cases are tested, including rapid user inputs and invalid moves, to verify the robustness of the game logic.

## 

## 

## Testing Details

<<Use your notes from above to write your test programs and complete this section of the formal documentation by creating a list of your test programs along with descriptions of what they are testing. You will also complete the black-box test plan by running the program and filling in the Actual Results column.>>

#### = **Test Game Initialization**:

#### **Description**: Verify that the game initializes correctly with an empty board and the correct initial player.

#### **Expected Results**:

#### The game board is empty.

#### The initial player is set to Player 1.

#### The status message shows Player 1's turn.

#### **Actual Results**:

#### The game board is empty.

#### The initial player is set to Player 1.

#### The status message shows Player 1's turn.

#### **Test Player Moves**:

#### **Description**: Verify that players can drop discs into valid columns and that the game updates correctly.

#### **Expected Results**:

#### Discs are placed in the selected columns.

#### The turn switches to the other player after a move.

#### The status message updates to show the next player's turn.

#### **Actual Results**:

#### Discs are placed in the selected columns.

#### The turn switches to the other player after a move.

#### The status message updates to show the next player's turn.

#### **Test Invalid Column Selection**:

#### **Description**: Verify that selecting an invalid column does not affect the game state.

#### **Expected Results**:

#### No disc is placed if an invalid column is selected.

#### The game state remains unchanged.

#### **Actual Results**:

#### No disc is placed if an invalid column is selected.

#### The game state remains unchanged.

#### **Test Full Column Selection**:

#### **Description**: Verify that trying to drop a disc into a full column is handled correctly.

#### **Expected Results**:

#### No disc is placed if the column is full.

#### The game state remains unchanged.

#### **Actual Results**:

#### No disc is placed if the column is full.

#### The game state remains unchanged.

#### **Test Single-Player Mode**:

#### **Description**: Verify that the AI makes a move after the player's turn in single-player mode.

#### **Expected Results**:

#### After the player's turn, the AI makes a valid move.

#### The turn switches back to the player.

#### **Actual Results**:

#### After the player's turn, the AI makes a valid move.

#### The turn switches back to the player.

#### **Test Two-Player Mode**:

#### **Description**: Verify that players can take turns without AI interference in two-player mode.

#### **Expected Results**:

#### Players take turns alternately.

#### No AI move is made.

#### **Actual Results**:

#### Players take turns alternately.

#### No AI move is made.

#### **Test Disc Color Customization**:

#### **Description**: Verify that the selected disc colors are applied correctly.

#### **Expected Results**:

#### The discs reflect the selected colors.

#### **Actual Results**:

#### The discs reflect the selected colors.

#### **Test Draw Condition**:

#### **Description**: Verify that the game declares a draw when the board is full and no player has won.

#### **Expected Results**:

#### The game declares a draw.

#### A modal displays the draw message.

#### **Actual Results**:

#### The game declares a draw.

#### A modal displays the draw message.

#### **Test Horizontal Win Condition**:

#### **Description**: Verify that the game detects and highlights a horizontal win.

#### **Expected Results**:

#### The game declares a win for the player who connects four discs horizontally.

#### The winning line is highlighted.

#### **Actual Results**:

#### The game declares a win for the player who connects four discs horizontally.

#### The winning line is highlighted.

#### **Test Vertical Win Condition**:

#### **Description**: Verify that the game detects and highlights a vertical win.

#### **Expected Results**:

#### The game declares a win for the player who connects four discs vertically.

#### The winning line is highlighted.

#### **Actual Results**:

#### The game declares a win for the player who connects four discs vertically.

#### The winning line is highlighted.

#### **Test Diagonal Win Condition**:

#### **Description**: Verify that the game detects and highlights a diagonal win.

#### **Expected Results**:

#### The game declares a win for the player who connects four discs diagonally.

#### The winning line is highlighted.

#### **Actual Results**:

#### The game declares a win for the player who connects four discs diagonally.

#### The winning line is highlighted.

#### **Test Instructions Modal**:

#### **Description**: Verify that the instructions modal opens and closes correctly.

#### **Expected Results**:

#### The instructions modal opens when the instructions button is clicked.

#### The instructions modal closes when the close button is clicked.

#### **Actual Results**:

#### The instructions modal opens when the instructions button is clicked.

#### The instructions modal closes when the close button is clicked.

#### **Test Game Restart**:

#### **Description**: Verify that the game resets correctly when the restart button is pressed.

#### **Expected Results**:

#### The game board resets to its initial empty state.

#### The initial player is set to Player 1.

#### The status message shows Player 1's turn.

#### **Actual Results**:

#### The game board resets to its initial empty state.

#### The initial player is set to Player 1.

#### The status message shows Player 1's turn.

# Presentation

Instructions:Week 12

## Preparation

The following prompts are meant to aid your thought process as you complete the presentation portion of this exercise. It is recommended that you examine the previous sections of the journal and your reflections as you work on the presentation as it is likely that you have already answered some of the following prompts elsewhere. Please respond to each of the prompts below and feel free to add additional notes.

* Give a brief description of your final project

### Brief Description of the Connect Four Game Project

The final project is a software implementation of the classic two-player game, Connect Four. This project aims to recreate the traditional board game experience in a digital format with modern enhancements to improve usability and engagement.

### Key Features:

1. **User Interface:**
   * A visually appealing and user-friendly interface designed using HTML and CSS.
   * Customizable player colors to personalize the game experience.
2. **Gameplay Modes:**
   * **Two Player Mode:** Allows two players to compete against each other on the same device.
   * **Single Player Mode:** Features an AI opponent with randomized moves to provide a challenging single-player experience.
3. **Game Mechanics:**
   * A 7-column by 6-row grid where players take turns dropping colored discs with the goal of connecting four discs in a row, column, or diagonally.
   * Smooth and responsive game mechanics implemented with JavaScript.
4. **AI Implementation:**
   * A simple AI opponent for single-player mode, which makes randomized moves ensuring a varied and engaging gameplay experience.
5. **Winning Conditions and Feedback:**
   * The game checks for win conditions after each move, highlighting the winning line and displaying a modal with the result.
   * Provides real-time feedback on the current player’s turn and game status.
6. **Customization:**
   * Players can choose their disc colors using color input controls, enhancing the visual appeal and personalization of the game.
7. **Instructions and Usability:**
   * An instructions section to help new players understand the rules and objective of the game.
   * Options to restart the game and change game modes without reloading the page.

### Technical Implementation:

* **Frontend Development:** The project is built using HTML for the structure, CSS for styling, and JavaScript for game logic and interactivity.
* **Responsive Design:** The interface is designed to be responsive, ensuring a seamless experience across different devices and screen sizes.
* **Modular Code:** The JavaScript code is organized into functions for clarity and maintainability, including functions for creating the game board, handling player moves, checking win conditions, and updating the UI.

### Educational and Strategic Value:

* The game encourages strategic thinking and planning, making it both entertaining and intellectually stimulating.
* The project showcases fundamental concepts of web development, including DOM manipulation, event handling, and CSS styling.

This Connect Four game project not only recreates a beloved classic but also demonstrates the application of web development skills to create an interactive and enjoyable user experience.

* Describe your requirement assumptions/additions.

### Requirement Assumptions:

1. **Basic Functionality:**
   * The game will follow the traditional rules of Connect Four, where the objective is to connect four discs in a row, column, or diagonally.
2. **User Interaction:**
   * Players will interact with the game through a graphical user interface (GUI), using mouse clicks to drop discs into the grid.
3. **AI Implementation:**
   * The AI opponent in single-player mode will make random moves, providing a basic level of challenge for the player.
4. **Browser Compatibility:**
   * The game is assumed to run on modern web browsers (e.g., Chrome, Firefox, Edge) and may not support older browsers or specific mobile browsers.
5. **Screen Size:**
   * The game is designed to be responsive but assumes a minimum screen size for optimal display and interaction, ideally a tablet or larger.
6. **Player Customization:**
   * Players can customize their disc colors using color input elements, assuming basic color input compatibility across modern browsers.

### Additions:

1. **Game Modes:**
   * Two distinct game modes are included: Two Player Mode for local multiplayer gameplay and Single Player Mode against an AI opponent.
2. **UI Enhancements:**
   * Visually appealing design with a colorful and engaging interface.
   * Real-time status updates to inform players whose turn it is and the game outcome.
3. **Instructions Modal:**
   * A modal that provides game instructions, accessible via an "Instructions" button, to help new players understand the game rules and objectives.
4. **Restart Functionality:**
   * A restart button to allow players to reset the game without refreshing the page, ensuring a seamless gaming experience.
5. **Winning Line Highlight:**
   * Highlighting the winning line when a player wins, providing visual feedback to enhance user experience and satisfaction.
6. **Dynamic Color Update:**
   * Real-time updates to disc colors based on player selections, enhancing customization and user engagement.
7. **Responsive Design:**
   * The game layout adjusts to different screen sizes, ensuring an optimal experience on various devices, from desktops to tablets.
8. **CSS Animations:**
   * Smooth transitions and animations, such as disc hover effects and winning cell blinking, to make the game visually appealing and engaging.
9. **Error Handling:**
   * Basic error handling to manage invalid moves, such as trying to drop a disc into a full column, ensuring a smooth gameplay experience.
10. **Code Modularity:**
    * Well-structured and modular JavaScript code for easier maintenance, readability, and potential future enhancements.

* Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?

### Design Options:

1. **UI Framework and Libraries:**
   * **Option 1: Vanilla HTML, CSS, and JavaScript**
     + **Pros:** Full control over the code, lightweight, no external dependencies.
     + **Cons:** More time-consuming to implement complex features, potential for more boilerplate code.
   * **Option 2: Using a Frontend Framework (e.g., React, Angular, Vue)**
     + **Pros:** Simplifies state management, reusable components, faster development for complex applications.
     + **Cons:** Steeper learning curve, increased project complexity, larger bundle size.
2. **Game Board Implementation:**
   * **Option 1: HTML Table**
     + **Pros:** Easy to create a grid structure, simple to manipulate with JavaScript.
     + **Cons:** Less flexible for styling and animations, potential for less responsive design.
   * **Option 2: CSS Grid Layout**
     + **Pros:** Highly flexible and responsive design, easy to apply modern CSS styling and animations.
     + **Cons:** Requires more CSS knowledge, potential browser compatibility issues with older browsers.
3. **AI Opponent:**
   * **Option 1: Random Move Selection**
     + **Pros:** Simple to implement, provides a basic level of challenge.
     + **Cons:** Not very challenging for experienced players, predictable gameplay.
   * **Option 2: Minimax Algorithm**
     + **Pros:** Provides a more challenging and strategic AI opponent, more engaging for players.
     + **Cons:** More complex to implement, requires more processing power, especially for deeper game tree exploration.
4. **Winning Condition Check:**
   * **Option 1: Iterative Check After Each Move**
     + **Pros:** Straightforward to implement, easy to understand and debug.
     + **Cons:** Can become computationally expensive with complex game logic.
   * **Option 2: Precomputed Win Conditions**
     + **Pros:** Faster win condition checks, optimized performance.
     + **Cons:** More complex to set up initially, harder to debug and maintain.

### Final Decision and Rationale:

1. **UI Framework and Libraries:**
   * **Decision:** Vanilla HTML, CSS, and JavaScript
   * **Rationale:** Given the project's scope, using vanilla technologies provided enough control and simplicity without the overhead of learning and integrating a frontend framework. This choice ensured that the game remains lightweight and easy to maintain.
2. **Game Board Implementation:**
   * **Decision:** CSS Grid Layout
   * **Rationale:** CSS Grid offered the best balance between flexibility and modern design practices. It allowed for a highly responsive layout, making the game look good on various devices and screen sizes. The ease of styling and applying animations also contributed to this decision.
3. **AI Opponent:**
   * **Decision:** Random Move Selection
   * **Rationale:** For an initial implementation, a random move selection provided a basic yet functional AI opponent. This approach kept the project within scope and ensured timely completion. Future iterations could explore more complex AI implementations like Minimax for enhanced gameplay.
4. **Winning Condition Check:**
   * **Decision:** Iterative Check After Each Move
   * **Rationale:** This method was straightforward to implement and sufficient for a grid of this size. It allowed for clear and easy-to-debug code, ensuring that win conditions were checked accurately after each move without adding unnecessary complexity.

### Weighing Pros and Cons:

* **Simplicity vs. Complexity:** The project aimed for a balance between simplicity and functionality. Options that provided simplicity in implementation and maintenance were favored, ensuring the game was completed within the project's timeline and met all basic requirements.
* **User Experience:** Design choices that enhanced the visual appeal and responsiveness of the game were prioritized. CSS Grid Layout and detailed CSS styling contributed significantly to a better user experience.
* **Performance:** Ensuring smooth performance was critical. Iterative win condition checks were efficient for the game’s grid size, providing quick feedback to players without performance lags.
* **Scalability:** While some advanced features (e.g., Minimax AI) were not included in the initial version, the design decisions made the codebase modular and maintainable, allowing for future enhancements.
* How did the extension affect your design?

### Impact of Extension on Design

### Introduction:

Extending the Connect Four game project involved adding new features and improving existing ones to enhance the user experience and functionality. These extensions included implementing an AI opponent, improving the user interface, and adding customization options. Here’s how these extensions affected the design:

### AI Opponent Implementation:

1. **Initial Design:**
   * **Random Move Selection:** The initial AI was designed to make random moves, which provided a basic level of challenge.
2. **Extended Design:**
   * **Minimax Algorithm (Future Plan):** The plan to upgrade to a Minimax algorithm influenced the modular design of the AI logic.
   * **Impact:**
     + **Code Structure:** The AI logic was separated into distinct functions to allow easy replacement or upgrade without affecting other parts of the code.
     + **Performance Considerations:** The design considered potential performance impacts, ensuring the game remains responsive even with more complex AI algorithms.

### User Interface Enhancements:

1. **Initial Design:**
   * **Basic UI:** A simple HTML and CSS layout with essential game functionalities.
2. **Extended Design:**
   * **Responsive and Animated UI:** Enhancements included CSS Grid for a flexible layout, animations for visual appeal, and better responsiveness.
   * **Impact:**
     + **CSS Complexity:** The extended use of CSS for animations and responsiveness added complexity, requiring more careful organization and testing.
     + **User Experience:** Improved visual feedback and a more engaging interface significantly enhanced the user experience.

### Customization Options:

1. **Initial Design:**
   * **Static Colors:** Predefined colors for player discs with no customization.
2. **Extended Design:**
   * **Dynamic Color Selection:** Players can choose their disc colors using color input controls.
   * **Impact:**
     + **Dynamic Styling:** JavaScript functions were added to dynamically update the disc colors based on user input.
     + **User Engagement:** Customization options made the game more engaging and personalized for users.

### Instructions and Usability Features:

1. **Initial Design:**
   * **Basic Gameplay:** Core gameplay mechanics with limited instructions.
2. **Extended Design:**
   * **Instructions Modal:** A detailed instructions modal was added to help new players understand the game.
   * **Impact:**
     + **User Guidance:** Improved user guidance and reduced the learning curve for new players.
     + **UI Elements:** Additional UI elements required careful integration to ensure they complemented the overall design without cluttering the interface.

### Restart and Status Display:

1. **Initial Design:**
   * **Basic Restart:** A simple restart functionality without visual feedback.
2. **Extended Design:**
   * **Enhanced Feedback:** Added a status display to show whose turn it is and a modal for game outcomes.
   * **Impact:**
     + **Game State Management:** Improved state management to handle game restarts and status updates efficiently.
     + **User Feedback:** Enhanced feedback mechanisms provided a more engaging and informative user experience.

### Summary:

The extensions significantly influenced the design of the Connect Four game, leading to:

* **Improved Modularity:** Code was structured in a more modular fashion to accommodate future upgrades and extensions, such as a more advanced AI.
* **Enhanced User Experience:** Visual and interactive enhancements made the game more engaging and user-friendly.
* **Increased Complexity:** The addition of new features added complexity to both the UI and game logic, requiring careful planning and implementation to maintain performance and usability.
* **Future Scalability:** The design decisions ensured the project remains scalable, allowing for future enhancements without major overhauls.

These changes collectively contributed to a more robust, enjoyable, and maintainable game, aligning with the project's goals of creating an engaging and modern Connect Four experience.

* Describe your tests (e.g., what you tested, equivalence classes).

1. **Test Game Initialization**:
   * **Description**: Verify that the game initializes correctly with an empty board and the correct initial player.
   * **Equivalence Classes**: Valid game state (empty board).
   * **Boundary Values**: Start of the game.
2. **Test Player Moves**:
   * **Description**: Verify that players can drop discs into valid columns and that the game updates correctly.
   * **Equivalence Classes**: Valid column selection, valid player moves.
   * **Boundary Values**: Column indices (0 to 6).
3. **Test Invalid Column Selection**:
   * **Description**: Verify that selecting an invalid column does not affect the game state.
   * **Equivalence Classes**: Invalid column selection.
   * **Boundary Values**: Invalid column indices (-1, 7).
4. **Test Full Column Selection**:
   * **Description**: Verify that trying to drop a disc into a full column is handled correctly.
   * **Equivalence Classes**: Full column.
   * **Boundary Values**: Column full condition.
5. **Test Single-Player Mode**:
   * **Description**: Verify that the AI makes a move after the player's turn in single-player mode.
   * **Equivalence Classes**: AI player, single-player mode.
   * **Boundary Values**: Player 1 turn, Player 2 (AI) turn.
6. **Test Two-Player Mode**:
   * **Description**: Verify that players can take turns without AI interference in two-player mode.
   * **Equivalence Classes**: Human players, two-player mode.
   * **Boundary Values**: Player 1 turn, Player 2 turn.
7. **Test Disc Color Customization**:
   * **Description**: Verify that the selected disc colors are applied correctly.
   * **Equivalence Classes**: Color input.
   * **Boundary Values**: Valid hex color codes.
8. **Test Draw Condition**:
   * **Description**: Verify that the game declares a draw when the board is full and no player has won.
   * **Equivalence Classes**: Full board, no winner.
   * **Boundary Values**: Full board condition.
9. **Test Horizontal Win Condition**:
   * **Description**: Verify that the game detects and highlights a horizontal win.
   * **Equivalence Classes**: Winning line, horizontal win.
   * **Boundary Values**: Four discs in a row horizontally.
10. **Test Vertical Win Condition**:
    * **Description**: Verify that the game detects and highlights a vertical win.
    * **Equivalence Classes**: Winning line, vertical win.
    * **Boundary Values**: Four discs in a column vertically.
11. **Test Diagonal Win Condition**:
    * **Description**: Verify that the game detects and highlights a diagonal win.
    * **Equivalence Classes**: Winning line, diagonal win.
    * **Boundary Values**: Four discs in a diagonal.
12. **Test Instructions Modal**:
    * **Description**: Verify that the instructions modal opens and closes correctly.
    * **Equivalence Classes**: UI interaction.
    * **Boundary Values**: Opening and closing actions.
13. **Test Game Restart**:
    * **Description**: Verify that the game resets correctly when the restart button is pressed.
    * **Equivalence Classes**: Game state reset.
    * **Boundary Values**: Initial game state.

* What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?

### Lessons Learned from the Connect Four Game Project

### Programming Concepts:

1. **HTML and CSS Fundamentals:**
   * Learned how to structure web pages using HTML elements effectively.
   * Gained experience in using CSS for styling, including advanced features like CSS Grid for layout and CSS animations for visual effects.
2. **JavaScript Basics and Advanced Techniques:**
   * Improved understanding of core JavaScript concepts such as functions, loops, conditionals, and event handling.
   * Gained experience with more advanced JavaScript features, including DOM manipulation, dynamic styling, and asynchronous operations (using setTimeout).
3. **Modular Programming:**
   * Recognized the importance of writing modular code to improve readability, maintainability, and scalability. Functions like dropDisc, checkWin, and aiMove were modularized for better code organization.
4. **Algorithm Implementation:**
   * Implemented game logic algorithms for checking win conditions, highlighting the importance of efficient and clear algorithms to ensure game performance and accuracy.
5. **Responsive Design:**
   * Understood the principles of responsive web design to ensure the game works well across various devices and screen sizes. This included using flexible layouts and media queries in CSS.

### Software Process:

1. **Requirement Analysis:**
   * Learned the importance of thoroughly understanding project requirements and defining clear objectives before starting the development process. This helped in creating a focused and efficient development plan.
2. **Design and Prototyping:**
   * Appreciated the value of designing and prototyping the user interface and user experience before implementation. This step helped in visualizing the end product and identifying potential issues early on.
3. **Incremental Development:**
   * Recognized the benefits of incremental development, where the project was built and tested in stages. This approach allowed for early detection of bugs and issues, making the debugging process more manageable.
4. **Version Control:**
   * Although not explicitly mentioned, using version control (e.g., Git) is essential for tracking changes, collaborating with others, and maintaining a history of the project’s development.
5. **User Testing and Feedback:**
   * Understood the importance of user testing and feedback in improving the user experience. Regular testing helped identify usability issues and guided design improvements.
6. **Error Handling and Edge Cases:**
   * Learned to anticipate and handle potential errors and edge cases, such as invalid moves or full columns, to ensure a robust and user-friendly game.

### Project Management:

1. **Time Management:**
   * Gained experience in managing time effectively to balance different aspects of the project, from coding to testing to documentation.
2. **Prioritization:**
   * Learned to prioritize features based on their importance and impact on the overall user experience. This included focusing on core gameplay mechanics before adding enhancements like AI and customization options.
3. **Documentation:**
   * Recognized the importance of documenting code and design decisions to facilitate future maintenance and potential collaboration.

### Conclusion:

This comprehensive exercise provided valuable insights into both technical and process-oriented aspects of software development. By working on the Connect Four game project, I gained practical experience in web development, learned to implement and optimize algorithms, and appreciated the importance of good design and user experience. Additionally, the project highlighted the need for effective project management and iterative development to deliver a successful software product.

* What functionalities are you going to demo?

### Functionalities to Demo

1. **User Interface:**
   * Responsive design and visual appeal using CSS Grid and animations.
2. **Game Modes:**
   * Two Player Mode: Real-time turn-based gameplay.
   * Single Player Mode: Playing against a random move AI.
3. **Gameplay Mechanics:**
   * Dropping discs into columns and real-time status updates.
   * Win condition checks and highlighting the winning line.
4. **Customization:**
   * Changing disc colors with color input controls.
5. **Instructions and Usability:**
   * Displaying and closing the instructions modal.
   * Status display for current player’s turn and game outcome messages.
6. **Restart Functionality:**
   * Restarting the game and resetting the board.
7. **Winning and Game Over Scenarios:**
   * Demonstrating both winning and draw scenarios.
8. **Error Handling:**
   * Handling invalid moves, such as dropping discs into full columns.

* Who is going to speak about each portion of your presentation? (Recall: Each group will have ten minutes to present their work; minimum length of group presentation is seven minutes. Each student must present for at least two minutes of the presentation.)
* Other notes:
  + <<Insert notes>>

<<Use your notes from above to complete create your slides and plan your presentation and demo.>>