Comprehensive Exercise Report

Team <<X>> of Section <<000>>

<<Group Member names and unity IDs>>

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

After reading the client’s brief (possibly incomplete description), write one sentence that describes the project (expected software) and list the already known requirements.

* Project Description: The project aims to develop a graphical implementation of the game Connect 4 using Python and Pygame.
* Known Requirements:
  + Implement the game logic for Connect 4.
  + Utilize Pygame for graphical user interface.
  + Allow two players to take turns placing their pieces on the board.
  + Detect and announce the winner when four pieces of the same color are connected.

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.

* How should the game handle scenarios such as a tie (i.e., the board fills up without a winner)?
* Are there any specific visual preferences or design elements the client has in mind for the game interface?
* Should the game include any additional features beyond basic gameplay, such as a replay option or a menu screen?

Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.

* No, the project covers familiar topics such as Python programming and game development with Pygame.

Describe the users of this software (e.g., small child, high school teacher who is taking attendance).

* The users of this software could be anyone interested in playing the game Connect 4, including children, teenagers, or adults.

Describe how each user would interact with the software

* Each user would interact with the software by taking turns placing their pieces on the game board using the mouse, aiming to connect four pieces of their color either horizontally, vertically, or diagonally.

What features must the software have? What should the users be able to do?

* The software must allow two players to play the game by taking turns placing their pieces on the board.
* It should include visual indicators for the current player's turn.
* The users should be able to see the game board and the pieces placed by each player.
* The software should announce the winner when a player successfully connects four pieces.
* Optionally, the software could include features such as a replay option or a menu screen for starting or resetting the game.

Other notes:

* It might be beneficial to implement error handling for scenarios such as invalid user input or unexpected program behavior.
* Consider adding sound effects or animations to enhance the gaming experience.

## Software Requirements

**Project Overview:** The project aims to develop a graphical implementation of the classic game Connect 4 using Python and Pygame. Connect 4 is a two-player game where the objective is to connect four of one's own discs of the same color vertically, horizontally, or diagonally before the opponent. The software will provide a user-friendly interface allowing players to take turns placing their colored discs on a grid. The game will feature visual indicators for the current player's turn, detect winning combinations, and announce the winner accordingly. Optional features such as a replay option or a menu screen for starting or resetting the game may be included to enhance the user experience.

**Software Requirements:**

1. **Game Logic Implementation:**
   * The software shall implement the game logic for Connect 4.
   * It shall provide functions to drop a piece into a specified column and detect winning combinations.
   * The game logic shall handle scenarios such as a tie when the board fills up without a winner.
2. **Graphical User Interface:**
   * The software shall utilize Pygame library to create the graphical user interface.
   * It shall display a game board grid where players can place their pieces.
   * The interface shall include visual indicators for the current player's turn.
3. **Player Interaction:**
   * Players shall take turns placing their colored discs on the board by clicking on the desired column.
   * The software shall validate user input to ensure it corresponds to a valid column.
4. **Win Detection and Announcement:**
   * The software shall detect winning combinations when four pieces of the same color are connected either horizontally, vertically, or diagonally.
   * It shall announce the winner and end the game when a winning combination is detected.
5. **Optional Features:**
   * The software may include additional features such as a replay option to start a new game or a menu screen for game setup.
   * It may incorporate sound effects or animations to enhance the gaming experience.

**User Stories:**

* As a player, I want to be able to click on the game board to place my colored disc in a desired column.
* As a player, I want to see visual indicators indicating whose turn it is during the game.
* As a player, I want the game to detect and announce when I have won by connecting four of my discs.
* As a player, I want the option to replay the game or start a new game from a menu screen.

# Black-Box Testing

Instructions: Week 4

**Black-Box Testing:**

**What does input for the software look like?**

* Input for the software includes:
  + Column selections made by players during their turns, represented as integers corresponding to the column number (e.g., 0 for the first column, 1 for the second column, etc.).

**What does output for the software look like?**

* Output for the software includes:
  + Visual updates to the game board based on player actions.
  + Messages announcing the winner of the game or indicating a tie.

**What equivalence classes can the input be broken into?**

* Equivalence classes for input can be categorized based on valid and invalid column selections:
  + Valid input: Integers within the range of 0 to (COLUMN\_COUNT - 1).
  + Invalid input: Integers outside the valid range or non-integer inputs.

**What boundary values exist for the input?**

* Boundary values for the input include:
  + Lower boundary: 0 (minimum column index).
  + Upper boundary: COLUMN\_COUNT - 1 (maximum column index).
  + Invalid boundary: Negative integers or integers greater than COLUMN\_COUNT - 1.

**Are there other cases that must be tested to test all requirements?**

* Other cases that must be tested include:
  + Testing for win detection: Ensuring that the software correctly identifies winning combinations horizontally, vertically, and diagonally.
  + Testing for tie scenarios: Ensuring that the software correctly detects when the board is full without a winner.
  + Testing for graphical updates: Ensuring that the game board is visually updated after each player action.

**Other notes:**

* Error handling: Test cases should include scenarios where invalid input is provided, such as non-integer inputs or selections outside the valid range.
* Game flow: Test cases should cover the entire game flow, including player turns, win detection, and game termination.

**Black-Box Test Cases:**

| **Test ID** | **Description** | **Input** | **Expected Results** |
| --- | --- | --- | --- |
| T1 | Valid column selection within range | 3 | Update game board with player's piece placed in column 3. |
| T2 | Invalid column selection (negative integer) | -1 | Display error message and prompt player to make a valid column selection. |
| T3 | Invalid column selection (greater than maximum column index) | 7 | Display error message and prompt player to make a valid column selection. |
| T4 | Valid column selection at lower boundary | 0 | Update game board with player's piece placed in the first column. |
| T5 | Valid column selection at upper boundary | 6 | Update game board with player's piece placed in the last column. |
| T6 | Win detection: Horizontal win | (sequence of valid column selections leading to four pieces in a row) | Announce the player who achieved the win and end the game. |
| T7 | Win detection: Vertical win | (sequence of valid column selections leading to four pieces in a column) | Announce the player who achieved the win and end the game. |
| T8 | Win detection: Diagonal win | (sequence of valid column selections leading to four pieces diagonally connected) | Announce the player who achieved the win and end the game. |
| T9 | Tie scenario | (sequence of valid column selections resulting in a filled board without a winner) | Announce a tie game and end the game. |
| T10 | Graphical update after player's turn | (valid column selection made by a player) | Display updated game board with player's piece placed in the selected column. |

Note: Actual results will vary depending on the implementation of the software. These expected results are based on the requirements and assumptions made during the black-box testing process.

Haut du formulaire

Bas du formulaire

# Design

**List of Nouns from Requirements/Analysis Documentation:**

* Software, input, output, equivalence classes, boundary values, test cases, game, player, column, piece, board, turn, winner, message, error, game flow, graphical updates.

**Which Nouns Potentially May Represent a Class in Your Design?**

* Classes such as Player, Game, Board, Piece can potentially represent objects in the design.

**Which Nouns Potentially May Represent Attributes/Fields in Your Design? Also List the Class Each Attribute/Field Would Be a Part Of:**

* Player:
  + Attributes: name
* Game:
  + Attributes: board
* Board:
  + Attributes: grid
* Piece:
  + Attributes: color, position

**Design Options:**

1. **Design Option 1: Separate Classes for Player, Game, Board, Piece:**
   * Pros:
     + Clear separation of concerns, each class responsible for its specific functionality.
     + Modularity and ease of maintenance.
   * Cons:
     + Potential for increased complexity in managing interactions between classes.
     + More overhead in coordinating communication between classes.
2. **Design Option 2: Combined Class for Game Management:**
   * Pros:
     + Simplified design with a single class responsible for managing game logic and player interactions.
     + Reduced overhead in managing game state.
   * Cons:
     + Less modular compared to option 1, potentially leading to less flexibility in future modifications.
     + May become more complex to maintain as the game logic grows.

**Which Design Do You Plan to Use? Explain Why You Have Chosen This Design:**

* I plan to use Design Option 1 with separate classes for Player, Game, Board, and Piece. This design offers clear separation of concerns, making it easier to understand and maintain the code in the long run. Additionally, it allows for greater modularity, facilitating future enhancements or modifications to individual components of the game.

**List of Verbs from Requirements/Analysis Documentation:**

* Place, update, detect, announce, end, display, prompt.

**Which Verbs Potentially May Represent a Method in Your Design? Also List the Class Each Method Would Be Part Of:**

* Game:
  + Methods: placePiece(), updateBoard(), detectWinner(), announceWinner(), endGame().
* Board:
  + Methods: displayBoard().
* Player:
  + Methods: promptColumnSelection().

**Other Notes:**

* Consideration should be given to error handling and validation in the design, especially for user input and game state transitions.
* Design should allow for scalability and flexibility to accommodate potential future requirements or enhancements.

## 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.****>>*

# Implementation

Instructions: Week 8

**Programming Concepts Needed:**

1. **Classes and Objects:** I'll need to implement classes such as Player, Game, Board, and Piece based on the design. Each class will represent an object type in the game, and objects of these classes will interact to implement the game logic.
2. **Methods:** Methods defined within each class will encapsulate specific behaviors associated with the corresponding object type. For example, methods like placePiece(), detectWinner(), and displayBoard() will be implemented to perform actions related to the game.
3. **Data Structures:** I'll utilize data structures such as lists and tuples to represent the game board and pieces. Lists will be used to store the grid of the board, and tuples will represent the position of each piece.
4. **Control Structures:** Control structures like loops and conditional statements will be used to manage the flow of the game. Loops will handle iterations, such as checking for a winning condition or updating the game board after each turn. Conditional statements will determine the outcome of actions, such as detecting a winner or validating player input.
5. **Error Handling:** I'll implement error handling techniques to handle unexpected situations, such as invalid user input or game state inconsistencies. This will ensure smooth execution of the program and provide informative error messages to the user when necessary.

**Other Notes:**

* Modular Approach: I'll break down the implementation into smaller, manageable modules corresponding to the classes defined in the design. This modular approach will facilitate easier testing, debugging, and maintenance of the code.
* Testing: I'll conduct thorough testing of each module to ensure that individual components function correctly and interact as expected. This will involve both unit testing and integration testing to verify the behavior of the entire system.
* Documentation: I'll maintain clear and concise documentation throughout the implementation process to aid in understanding and future modifications of the code. This documentation will include comments within the code and possibly external documentation files describing the functionality of each module and class.

Haut du formulaire

Bas du formulaire

## Implementation Details

**Implementation Details:**

Below are the implementation details for the Connect 4 game based on the design provided earlier.

**Player Class:**

class Player:

def \_\_init\_\_(self, name: str):

self.name = name

def getName(self) -> str:

return self.name

**Game Class:**

class Game:

def \_\_init\_\_(self):

self.board = Board()

def startGame(self):

# Start the game by displaying the empty board

self.board.displayBoard()

def placePiece(self, column: int, player: Player) -> bool:

# Place a piece on the board

# Implement logic to place the piece and update the board

pass

def detectWinner(self) -> bool:

# Detect if there is a winner in the game

# Implement logic to check for winning combinations

pass

def announceWinner(self, winner: str):

# Announce the winner of the game

# Display a message indicating the winner

pass

def endGame(self):

# End the game

# Display a message indicating the end of the game

pass

**Board Class:**

class Board:

def \_\_init\_\_(self):

self.grid = [[0 for \_ in range(7)] for \_ in range(6)]

def displayBoard(self):

# Display the current state of the game board

# Implement code to visualize the board using Pygame or another graphical library

pass

**Piece Class:**

class Piece:

def \_\_init\_\_(self, color: int, position: tuple[int, int]):

self.color = color

self.position = position

# 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.
  + <<Insert answer>>
* List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.
  + <<Insert class>>
    - <<Insert needed tests>>
  + <<Insert class and tests for each class>>
* Other notes:
  + <<Insert notes>>

## 

## 

## 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.>>

# 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
  + <<Insert answer>>
* Describe your requirement assumptions/additions.
  + <<Insert answer>>
* Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?
  + <<Insert answer>>
* How did the extension affect your design?
  + <<Insert answer>>
* Describe your tests (e.g., what you tested, equivalence classes).
  + <<Insert answer>>
* What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?
  + <<Insert answer>>
* What functionalities are you going to demo?
  + <<Insert answer>>
* 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.)
  + <<Insert answer>>
* Other notes:
  + <<Insert notes>>

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