Comprehensive Exercise Report

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

Edouard TESSIER 230ADB022 Yanis BOUAOUN 230ADB019 Gaspard MOULIN 230ADB021 Hubert LEROY 230ADB020

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.
  + We want to create a connect 4 game using python
    - Graphical user interface with a 6 x 7 grid
    - Two players should be able to take turns dropping their pieces in the grid.
    - Detect when a player has won or when there is a tie.
    - Option to start a new game.
* 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.
  + Which language do we have to use?
  + Is there a specific design or theme that the client would like for the GUI?(colors)
  + What programming language should be used to develop the game?
* Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.
  + Graphical user interface on python
* Describe the users of this software (e.g., small child, high school teacher who is taking attendance).
  + The users of this software are expected to be individuals who enjoy playing games or seeking to pass the time. The age of the user is not specified.
* Describe how each user would interact with the software
  + They will open an executable file and play in a console.
* What features must the software have? What should the users be able to do?
  + Take turns dropping their pieces on the game board.
  + Detect when a player has won or when there is a tie.
  + Visualization of the game in real time.
  + Score counter.
  + Sentences to show which player must play.
  + Possibility to play against the computer.
* Other notes:
  + None

## 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.>>

# 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)?
  + <<Insert answer>>
* What does output for the software look like (e.g., what type of data, how many pieces of data)?
  + <<Insert answer>>
* What equivalence classes can the input be broken into?
  + <<Insert answer>>
* What boundary values exist for the input?
  + <<Insert answer>>
* Are there other cases that must be tested to test all requirements?
  + <<Insert answer>>
* Other notes:
  + <<Insert notes>>

## 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 Case ID** | **Test Scenario** | **Test Steps** | **Test Data** | **Expected Results** | **Actual Results** |
| 1 | Test GUI display | 1. Launch the application | N/A | A 6x7 grid is displayed with an empty board and appropriate colors/theme |  |
| 2 | Test player turn | 1. Start a new game | N/A | Player 1 starts and the game displays a message indicating it is Player 1's turn |  |
| 3 | Drop a piece | 1. Player 1 clicks on a column to drop a piece | Column number (1-7) | Piece is dropped in the selected column and the game switches to Player 2's turn |  |
| 4 | Test winning condition (horizontal) | 1. Simulate a game where Player 1 has 4 pieces horizontally connected | Appropriate game state | Game detects Player 1 has won and displays a winning message |  |
| 5 | Test winning condition (vertical) | 1. Simulate a game where Player 1 has 4 pieces vertically connected | Appropriate game state | Game detects Player 1 has won and displays a winning message |  |
| 6 | Test winning condition (diagonal) | 1. Simulate a game where Player 1 has 4 pieces diagonally connected | Appropriate game state | Game detects Player 1 has won and displays a winning message |  |
| 7 | Test tie condition | 1. Simulate a game where the board is full and no player has 4 connected pieces | Appropriate game state | Game detects a tie and displays a tie message |  |
| 8 | Test score counter | 1. Play multiple games and keep track of each player's wins | Number of games played and wins | Score counter accurately reflects each player's wins |  |
| 9 | Test playing against the computer | 1. Start a new game against the computer | N/A | Game starts with the computer as the second player, taking turns with Player 1 |  |
| 10 | Test new game option | 1. Start a new game after a completed game | N/A | Game resets the board and starts a new game with Player 1 |  |

# 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.

* Connect 4 game
* Graphical user interface
* 6 x 7 grid
* Players
* Pieces
* Game board
* Winner
* Tie
* New game
* Software
* Turns
* Real time visualization
* Score counter
* Sentences
* Computer

● Which nouns potentially may represent a class in your design?

* Connect4
* Player
* Piece
* Board
* Game
* Score

● Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.

* grid (Board)
* current\_player (Game)
* game\_over (Game)
* tie (Game)
* score (Score)
* play\_against\_computer (Game)

● 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.

Design option 1:

* Connect4 (board, player1, player2, current\_player, game\_over, tie)
* Player (name, color)
* Piece (color)
* Board (grid)
* Game (current\_player, game\_over, tie, score, play\_against\_computer)
* Score (red\_score, blue\_score)

Pros:

* Simple and straightforward design.
* Easy to understand and implement.

Cons:

* Board and Game classes have overlapping responsibilities.
* Limited functionality.

Design option 2:

* Connect4 (board, player1, player2, current\_player, game\_over, tie)
* Player (name, color)
* Piece (color)
* Board (grid)
* Game (current\_player, game\_over, tie, score, play\_against\_computer)
* Score (red\_score, blue\_score)
* GUI (canvas, reset\_button)

Pros:

* Separation of concerns.
* Scalable and extensible.

Cons:

* More complex design.
* More time-consuming to implement.

● Which design do you plan to use? Explain why you have chosen this design. I plan to use Design option 2 because it separates concerns and allows for scalability and extensibility.

● List the verbs from your requirements/analysis documentation.

* Create
* Drop
* Detect
* Start
* Take turns
* Win
* Tie
* Visualize
* Count
* Show
* Play
* Place
* Switch

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

* **init** (Connect4)
* play (Connect4)
* draw\_board (Connect4)
* draw\_piece (Connect4)
* reset\_game (Connect4)
* get\_available\_row (Board)
* place\_piece (Board)
* switch\_player (Game)
* check\_winner (Game)
* check\_tie (Game)
* computer\_play (Game)

● Other notes:

* It may be helpful to create a separate **Player** class to handle player attributes and behavior.
* The **Game** class can be further broken down into more specialized classes, such as a **Turn** class to handle turn-taking behavior.

## 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.****>>*

Une image contenant diagramme

Description générée automatiquement

Class Definitions :

Connect4 Class

* \_\_init\_\_(): Initializes the game board, current player, game over, tie, score, and whether the game is played against a computer.
* play(): Method to handle the game play when a player drops a piece on the board.
* draw\_board(): Method to draw the empty board on the GUI.
* draw\_piece(): Method to draw a piece on the GUI when a player drops it on the board.
* reset\_game(): Method to reset the game when a new game is started.

Board Class

* \_\_init\_\_(): Initializes the game board.
* get\_available\_row(): Method to get the next available row in a column where a player drops a piece.
* place\_piece(): Method to place a piece on the board.

Piece Class

* Represents a piece with a color.

Player Class

* Represents a player with a name and a color.

Game Class

* \_\_init\_\_(): Initializes the game with the current player, game over, tie, score, and whether the game is played against a computer.
* switch\_player(): Method to switch to the next player when a player finishes their turn.
* check\_winner(): Method to check if a player has won the game.
* check\_tie(): Method to check if the game is tied.
* computer\_play(): Method to generate a random move for the computer player.

Score Class

* \_\_init\_\_(): Initializes the score with the red score and blue score.

Fields :

Connect4 Class

* grid: A 2D list to represent the game board.
* current\_player: A string to represent the current player.
* game\_over: A boolean to represent if the game is over.
* tie: A boolean to represent if the game is tied.
* score: A Score object to keep track of the score.
* play\_against\_computer: A boolean to represent if the game is played against a computer.

Board Class

* grid: A 2D list to represent the game board.

Piece Class

* color: A string to represent the color of the piece.

Player Class

* name: A string to represent the name of the player.
* color: A string to represent the color of the player's pieces.

Game Class

* current\_player: A Player object to represent the current player.
* game\_over: A boolean to represent if the game is over.
* tie: A boolean to represent if the game is tied.
* score: A Score object to keep track of the score.
* play\_against\_computer: A boolean to represent if the game is played against a computer.

Score Class

* red\_score: An integer to represent the red player's score.
* blue\_score: An integer to represent the blue player's score.

Design Option :

Based on the analysis and design considerations, we have opted for a more complex but scalable and extensible design (Design option 2). The design consists of the following classes:

* Connect4
* Board
* Piece
* Player
* Game
* Score

The Connect4 class handles the overall game flow and initializes the Board, Player, Game, and Score objects. The Board class manages the game board and allows players to place their pieces. The Piece class represents the pieces in the game. The Player class represents the players with their names and piece colors. The Game class manages the game flow and checks for winners and ties, and also handles the computer player if playing against the computer. Finally, the Score class keeps track of the score for each player.

# 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.
  + Object-Oriented Programming: The Connect 4 game will be implemented using a class in Python, which is an example of OOP. This class encapsulates the attributes and methods that operate on this attributes.
  + Event-Driven Programming: User interaction with the game will be handled through events such as clicking on the canvas to place a piece.
  + Graphical User Interface (GUI): The game will use the tkinter library to create a GUI that allows players to interact with the game visually.
  + Conditional and Looping Constructs: These will be used in the game logic such as determining the next player or checking for a win or a tie.
  + Random Numbers: The random library will be used to select a random column for the computer's move when playing against the computer.
* Other notes:

## Implementation Details

OOP: The Connect4 class models the game board and the game logic, it contains methods for drawing the board, placing pieces, switching players and checking for a winner or a tie.

Event-Driven Programming: The play method is bound to the <Button-1> event so it's invoked whenever the player clicks on the canvas.

GUI: The game board is a tkinter canvas widget and a button widget is used for restarting the game.

Conditional and Looping Constructs: They are used throughout the code : for example, the check\_winner method uses nested loops to check the pieces in all directions from the last piece placed.

Random Numbers: The computer\_play method generates a random integer between 0 and 6 (included) to select a column for the computer's move.

## README:

**How to Play Connect 4**

* Start the game by running the script in a Python environment. The game opens in a new window.
* The red player goes first. Click on a column to drop a piece into the lowest empty place in that column.
* Players drops pieces into the board.
* The goal of the game is to be the first to connect four of your pieces in a row, either horizontally, vertically, or diagonally.
* If the board fills up before one of the players has connected four pieces, the game is a tie.

**Interacting with the System**

* Placing a piece: Click on the column where you want to drop your piece.
* Restarting the game: Click the "Restart" button below the game board to start a new game.
* Playing against the computer: To play against the computer, change play\_against\_computer to True in the Connect4 class initialization. The computer makes random moves.
* Closing the game: Close the game window to close the game.

**Known Issues**

The computer player simply makes a random move and doesn't have any strategy. Future improvements could include implementing a smarter AI for the computer player.

**Contact**

For any issues or suggestions, please reach out to leroyhubert@yahoo.fr

# 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.>>