Comprehensive Exercise Report

Victor Leclercq 240AEB041
Thomas VAUDELEAU 240AEB040
Augustin VIEL 240AEB014

Requirements/Analysis 2
Journal 2
Software Requirements 3
Black-Box Testing 3
Journal 4
Black-box Test Cases 5
Design 6
Journal 6
Software Design 8
Implementation 9
Journal 9
Implementation Details 10
Testing **Error! Bookmark not defined.**
Journal 11
Testing Details 12
Presentation 14
Preparation 14
Grading Rubric **Error! Bookmark not defined.**

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.**
    - Connect-Four
        - Using a simple enough programming language capable of doing calculus
        - The language must be compatible with UX/UI technologies.
        - It is not useful to use a particularly low-level language as execution speed should not be a key factor.
- **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.**
    - Should the game allow you to play against a bot?
    - Do we need to save game states in case we want to come back to our game later.
    - Do you have a specific GUI framework in mind for the game?
- **Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.**
    - UX/UI
- **Describe the users of this software (e.g., small child, high school teacher who is taking attendance).**
    - Anybody can use this software if they know the basic principles of a computer. Therefore, we can define the user-base as people from 5 to 99 years old.
- **Describe how each user would interact with the software.**
    - Using the arrow keys and the space bar or the mouse to place their token.
- **What features must the software have? What should the users be able to do?**
    - The software must allow you to count the scores for players 1 and 2. It must randomly choose who starts the first time then alternate. Naturally, it should allow players to play Connect-Four.
- **Other notes:**

Software Requirements

This project will be a computer version of the game of connect 4, in which 2 players can choose a column into they can drop a token [input a column number]. In each turn, a player can use 1 token. When a winning sequence is detected (4 tokens of the same color are aligned horizontally, vertically or in diagonal), the game ends and the player with the color of the winning sequence wins.

Requirements:

- Indication of player’s turn
  Game board representation (2D array), visual feedback after each turn.
- Check winning conditions and if the board is full after each turn.
- Scalability for different board sizes
- Input validation.

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 only input to play is a mouse click. Other inputs, such as board size are integers set at the start of the game.
- **What does output for the software look like (e.g., what type of data, how many pieces of data)?**
  - A string describing the status of the game: correct/incorrect placement, player turn, win, lose.
  - The GUI of the game relies on a matrix generated by the code.
- **What equivalence classes can the input be broken into?**
  - Valid column numbers
  - Invalid column numbers
  - Non integer column numbers
- **What boundary values exist for the input?**
  - Minimum: 1, Max: [board size]
- **Are there other cases that must be tested to test all requirements?**
  - Empty board
  - Full board
  - Win conditions.
- **Other notes:**
  - Check after each turn: win sequence present? & full board?

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**
0	Beginning of the game	Empty board	
1	Board size input	Validated only for a positive, non-null integer	
1	Correct placement	Accept a token only if placed in a column where there is still room	
2	Horizontal win	When the 4th token of a horizontal sequence of same color tokens is placed, this color's player wins.	
3	Vertical win	When the 4th token of a vertical sequence of same color tokens is placed, this color's player wins.	
4	Diagonal win	When the 4th token of a diagonal sequence of same color tokens is placed, this color's player wins.	
5	Draw	When the board is full but there is no winning token sequence, the game is a draw	

Design

Instructions: Week 6

Journal

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

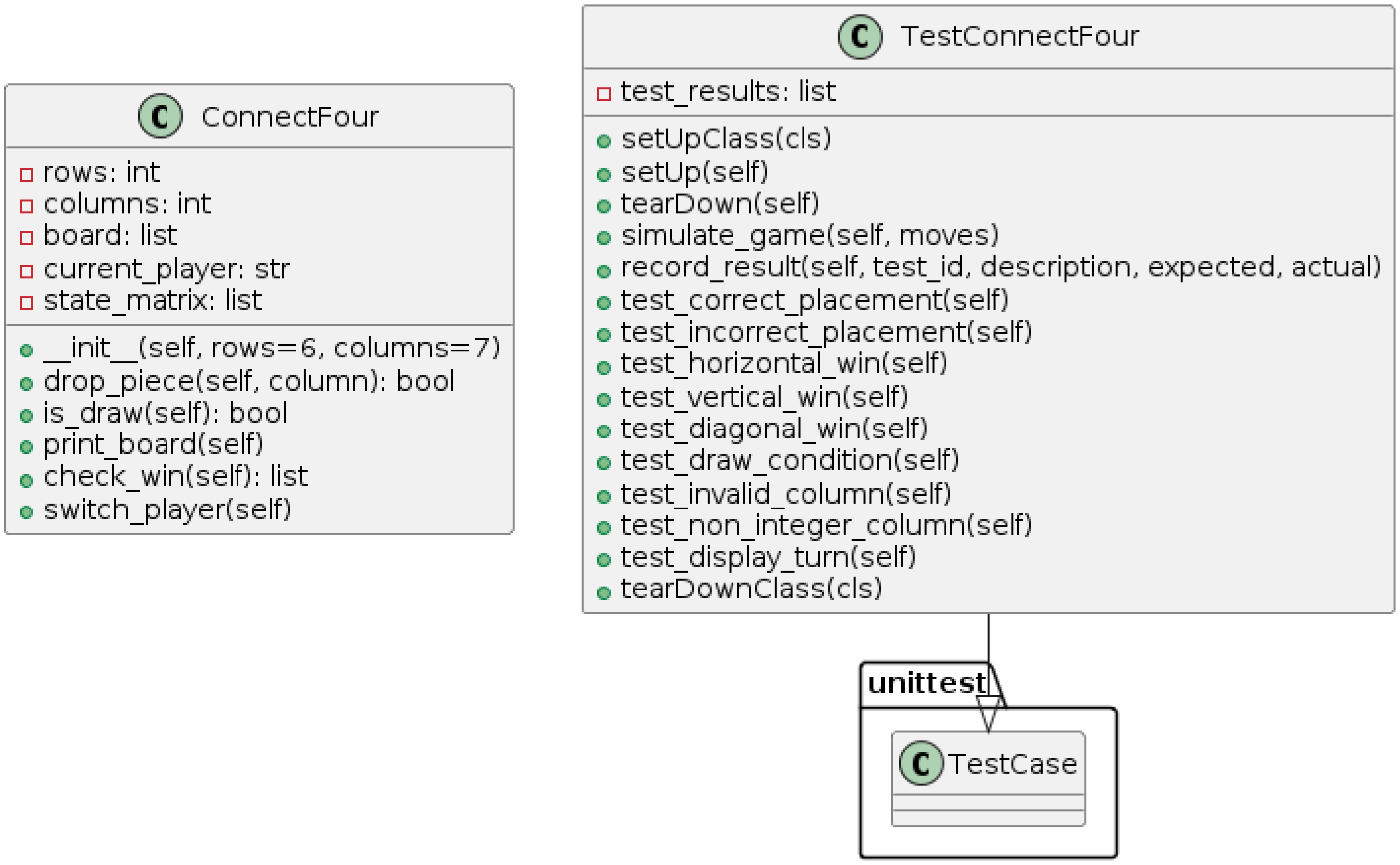
The following prompts are meant to aid your thought process as you complete the design portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- List the nouns from your requirements/analysis documentation.
    - Player
    - Token
    - Board
    - Column
    - Turn
    - Sequence
- Which nouns potentially may represent a class in your design?
    - Player
    - Board
    - Game
    - Token
- Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
    - Player Class:
        - ID
        - Color
    - Board class:
        - Size
        - Grid
    - Game class:
        - Current player
        - State
    - Token class:
        - Position
        - Color
- 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:
            - Player
            - Board
            - Game
            - Token
        - Pros:
            - Clear separation

                - Easy to manage.
            - Cons:
                - More classes could increase the complexity of the code.
    - Option 2
        - Classes:
                - ConnectFour (board, game, and player functions)
                - GUI (interface)
        - Pros:
            - Simplified structure with less classes
            -
        - Cons:
            - Could complicate things depending on code structure.
            - Less easy the modify certain aspects in the future without good code knowledge

- Which design do you plan to use? Explain why you have chosen this design.
    - Option 2 : We've chosen this design because the code probably won't be updated after the final version and made for relatively easy to read code.
- List the verbs from your requirements/analysis documentation.
    - Play
    - Drop
    - Check
    - Validate
    - Alternate
    - Display
- Which verbs potentially may represent a method in your design? Also list the class each method would be part of.
    - Game
        - start
        - end
    - Board
        - Drop
        - Check
        - Validate
    - Player
        - Take (turn) or play
- Other notes:

Software Design



ConnectFour
- rows: int
- columns: int
- board: list
- current_player: str
- state_matrix: list
+ __init__(self, rows=6, columns=7)
+ drop_piece(self, column): bool
+ is_draw(self): bool
+ print_board(self)
+ check_win(self): list
+ switch_player(self)
TestConnectFour
- test_results: list
+ setUpClass(cls)
+ setUp(self)
+ tearDown(self)
+ simulate_game(self, moves)
+ record_result(self, test_id, description, expected, actual)
+ test_correct_placement(self)
+ test_incorrect_placement(self)
+ test_horizontal_win(self)
+ test_vertical_win(self)
+ test_diagonal_win(self)
+ test_draw_condition(self)
+ test_invalid_column(self)
+ test_non_integer_column(self)
+ test_display_turn(self)
+ tearDownClass(cls)
unittest
TestCase

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.
    - SRS (Software requirements specifications)
    - URS (User requirements specifications)
    - Pair programming
    - Object-oriented analysis and design
- Other notes:

Implementation Details

README

Connect Four Game

This is a simple implementation of the classic Connect Four game in Python. The game is played in a terminal and uses a GUI for the game board.

Getting Started

To start the game, navigate to the project directory and run the GameLoop.py file from the src/connect_four directory:

Game Play

The game board is a grid of 6 rows and 7 columns. Players take turns dropping their tokens into any of the columns. The token will fall to the lowest empty row within the selected column. The goal is to be the first player to form a horizontal, vertical, or diagonal line of four of one's own tokens.

The game is played in a GUI window. When it's your turn, you can drop a token into a column by clicking on that column. The game will automatically switch turns between the two players.

If a player successfully forms a line of four tokens, the game will display a win message and highlight the winning tokens. If the board is filled without any player forming a line of four, the game will display a draw message.

```
python ./GUI/GUI.py
```

Testing

Unit tests for the game logic are in the test directory. You can run the tests with the following command:

```
python -m unittest test/test_connect_four.py
```

Dependencies

This project uses the pygame library for the GUI. You can install it with pip:

```
pip install pygame
```

It also uses unittest

```
Pip install unittest
```

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
- List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.
    - ConnectFour
        - Equivalence classes: game states (game start, mid-game, end of game, draw)
        - Boundary values (edge rows and columns)
            - Tests: drop_piece, check_win, is_draw, switch_player
    - GUI
    - There is no verification for the GUI
- Other notes:

Testing Details

Each test follows a similar pattern: it describes what is being tested, the expected results, and checks the actual results using assertions. The results are then recorded.

1. **`test_correct_placement`**
   - ⊄ **Description**: Verifies that a token is accepted only if placed in a column where there is still room.
   - ⊄ **Test**: Places a token in an empty column and verifies that the placement is correct.
2. **`test_incorrect_placement`**
   - ⊄ **Description**: Verifies that a token is rejected if the column is full.
   - ⊄ **Test**: Fills a column and tries to add an extra token, which should fail.
3. **`test_horizontal_win`**
   - ⊄ **Description**: Verifies the detection of a horizontal win.
   - ⊄ **Test**: Simulates a sequence of moves that leads to a horizontal win and verifies win detection.
4. **`test_vertical_win`**
   - ⊄ **Description**: Verifies the detection of a vertical win.
   - ⊄ **Test**: Simulates a sequence of moves that leads to a vertical win and verifies win detection.
5. **`test_diagonal_win`**
   - ⊄ **Description**: Verifies the detection of a diagonal win.
   - ⊄ **Test**: Simulates a sequence of moves that leads to a diagonal win and verifies win detection.
6. **`test_draw_condition`**
   - ⊄ **Description**: Verifies the detection of a draw condition when the board is full without a winning sequence.
   - ⊄ **Test**: Fills the board without creating a winning sequence and verifies that the game is declared a draw.
7. **`test_invalid_column`**
   - ⊄ **Description**: Verifies the rejection of tokens if the column index is out of range.
   - ⊄ **Test**: Attempts to place a token in an invalid column and verifies that it raises a `ValueError`.
8. **`test_non_integer_column`**
   - ⊄ **Description**: Verifies the rejection of tokens if the column index is not an integer.
   - ⊄ **Test**: Attempts to place a token with a non-integer column index and verifies that it raises a `ValueError`.
9. **`test_display_turn`**
   - ⊄ **Description**: Verifies the correct display of the current player's turn.
   - ⊄ **Test**: Checks that the GUI correctly displays the current player's turn.

Running the Tests

The code ends with the following statement that executes all the defined tests when the script is run directly:

```
python
if __name__ == '__main__':
    unittest.main()
```

Summary

This test code for the Connect Four game in Python uses `unittest` to verify several essential aspects of the game, including token placement, win condition detection (horizontal, vertical, diagonal), draw condition management, and input validation. Test results are recorded in a CSV file for later analysis.

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
    - Our project is a Connect Four game playable on the same machine. Two players can compete by taking turns placing tokens into a grid, with the objective of aligning four tokens of the same color horizontally, vertically, or diagonally.
- Describe your requirement assumptions/additions.
    - The game does not need to be played against a bot.
    - The game does not save game states, as it is played in a single session.
    - The user interface will be built using a simple framework like Tkinter for Python.
    - The game will be playable using the mouse to place the tokens.
- Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?
    - **Option 1: Separate classes for Player, Board, Game, and Token**
        - **Pros:** Clear separation of responsibilities, easy to manage and maintain.
        - **Cons:** Increases complexity by having multiple classes.
    - **Option 2: Nested classes (Game with internal classes)**
        - **Pros:** Simplified structure with fewer classes, which may ease initial development.
        - **Cons:** Could complicate long-term code management.
    - We chose Option 2 to ensure better code organization and facilitate unit testing and maintenance.
- How did the extension affect your design?
    - The extension of features such as the ability to resize the grid influenced our design by requiring us to make the system flexible and modular, with classes and methods that can easily adapt to different scenarios.
- Describe your tests (e.g., what you tested, equivalence classes).
    - We used black box testing to verify the main functionalities:
        - Correct and incorrect token placement
        - Win conditions (horizontal, vertical, diagonal)
        - Draw condition
        - Input validation (valid columns, invalid columns, non-integer values)
        - Correct display of game status and current player
- What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?
    - The importance of planning and design before starting to code.
    - The necessity of rigorous testing to ensure software reliability.
    - Effective use of classes and objects to structure the code.
    - Collaboration and communication within the team to ensure a common understanding of goals and requirements.
- What functionalities are you going to demo?
    - Token placement by the players
    - Indication of the current player's turn

    - Detection of winning conditions
- 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.)
    - Augustin: Intro
    - Thomas: Design and Code
    - Victor: Testing and Demo
- Other notes: