# Comprehensive Exercise Report

Team 03 of Section 001

Tess Colavecchio - tcolave
Jamea Speight - jaspeig2
Johnathan Rhyne - jprhyne
Jay Vandeveld - jtvandev

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

# Requirements/Analysis

Instructions: http://go.ncsu.edu/csc116-ce-reqt

## 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.
    - An application of the game ConnectFour that the user can play
        - Has to be 8x8 grid
        - Able to be played with two players
        - Alternating Turns
        - When the player places a piece in the column it lands in the bottom most available slot
        - The player needs to be updated with the pieces they have played
        - The player needs to know the max number of pieces they currently have
- 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.
    - Incomplete information
        - No indication as to if no connect 4 is accomplished when a board is full
            - Output that there was a draw/tie/etc.
        - Should there be a max number of pieces (relevant because of pieces counter)
            - Enough to fill board (32 pieces per person)
        - Is the game replayable after winning? Or must it be started over
            - Prompt user to play another
                - Keep track of how many times each player has won
        - Where the player is able to click to add pieces (anywhere in the column or top or…) in the context of a gui. Or if there should be a text input
            - Any is acceptable.
                - Choice is button at the top that disappears when a column is full.
        - How the player wants to be updated about pieces played and how many in a row. (Text box, separated for each player, etc.)
            - As long as it is visible, is ok
                - Probably separated text box will be best?
- Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.
    - Unfamiliar with GUI (not much work with). Relevant labs are Lab22 and the chapter 14 reading
- Describe the users of this software (e.g., small child, high school teacher who is taking attendance).
    - Kids
    - Adults who like Connect Four
    - Friends who do not have the physical game but still want to play
- Describe how each user would interact with the software
    - Competitive, but enjoyable game
    - Since users take turns, the game doesn't need to be completed in one sitting
        - Time taken between turns for strategic purposes

    - Users may be able to click which row their piece goes in (not a set design, but possible way they interact within the game).
- What features must the software have? What should the users be able to do?
    - Must be able to run the game to a conclusion
    - Each player knows what their specific piece is
    - Users should only be able to play after the other
    - Users can only play one piece at a time
    - Pieces goes to the bottom most empty slot
- Other notes:
    - TBD

# Software Requirements

This project will emulate a game of Connect4 with an 8x8 grid. This game consists of placing pieces alternating between 2 players, and its win condition is 4 pieces in a straight line (horizontal, vertical, or diagonal). The program will also include a visual display of how many pieces are in play by each player along with how many pieces they have in a row. User input will also be needed to coordinate where each player's piece is placed.
Requirements:

- The player will be able to tell the program where they will be placing a piece.
- The player will be able to see where their pieces are placed.
- The player will be able to see how many pieces they have in a row.
- The player will be able to see how many pieces they have placed total.
- The player will be informed when the win condition is met.
- The players will be able to differentiate their pieces.
- The player will be able to fill the board until a total of 64 pieces are placed (or victory is met).
- The player will not be able to fill a column more than 8 pieces tall.

<<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: http://go.ncsu.edu/csc116-ce-bbt

## 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)?
    - User input
    - Alternates between users
    - Keeps going until a player wins or the board is full
- What does output for the software look like (e.g., what type of data, how many pieces of data)?
    - The image of the connect four grid
        - That updates every time the user inserts a piece
    - How many pieces each player has available
    - A Notification of the winner
    - A re-prompting of playing again
    - A count of wins/losses
- What equivalence classes can the input be broken into?
    - 3 cases for each player
        - 1 pieces in a row
        - 2 pieces in a row
        - 3 pieces in a row
- What boundary values exist for the input?
    - Only eight columns/ rows
    - Only 32 pieces per user
    - Checking to make sure that at 4 in a row the game ends
    - Checking to make sure that counter displays 0 at beginning
    - Make sure that column input greys out once a column is full
    - Make sure that the program prompts for a new game and executes the yes/no input properly
- Are there other cases that must be tested to test all requirements?
    - Pieces display properly
- Other notes:
    - For below, expected results will be in text/descriptive form, and actual Results will take the form of screenshots
    - First 3 tests will be done in one instance of Connect4

## 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 |
|---|---|---|---|
| testFirstMove | Player 1 starts game by placing a piece in Column 1.<br><br>Column 1 | Print Grid:<br><br>Piece for player 1 @ R(ow)1 C(olumn)1<br><br>Print Display Box: | Print Grid:<br><br>Piece for player 1 @ R(ow)1 C(olumn)1<br><br>Print Display Box: |
| testSecondMove | Player 2 places their first piece in Column 2.<br><br>Column 2 | Print Grid:<br><br>Piece for player 2 @ R1 C2<br><br>Print Display Box: | Print Grid:<br><br>Piece for player 2 @ R1 C2<br><br>Print Display Box: |
| testDisplayCount | Player 1 and 2 alternate placing pieces.<br><br>Player 1 will place 2 pieces in Column 1 and then 3 pieces in Column 3<br><br>Player 2 will do the same for Columns 2 and 4 respectively | Print Grid:<br><br>Print Display Box:<br><br>Player 1 name has 0 Win(s), and has 3 piece(s) in a row<br><br>Player 2 name has 0 Win(s), and has 3 piece(s) in a row | Print Grid:<br><br>Print Display Box:<br><br>Player 1 name has 0 Win(s), and has 3 piece(s) in a row<br><br>Player 2 name has 0 Win(s), and has 3 piece(s) in a row |
| testTwoPlaysBySamePlayer | Player 1 tries to play twice in a row, not waiting until Player 2 has played their turn. | Print Grid:<br><br>Print Display Box:<br><br>Player 1 has piece in R1 C1 and Player 2 has piece in R2 C1 | Print Grid:<br><br>Print Display Box:<br><br>Player 1 has piece in R1 C1 and Player 2 has piece in R2 C1 |
| testVictory | Player 1 and 2 alternate placing | Print Grid: with Column 1 at 4 | Print Grid: with Column 1 at 4 |

| | | | |
|---|---|---|---|
| | pieces in Columns 1 and 2 respectively, until Game stops with player 1 as victor | pieces and Column 2 at 3.<br><br>Print Display Box: Player 1 name has 0 Win(s), and has 3 piece(s) in a row<br><br>Print Display Box: Player 2 name has 0 Win(s), and has 3 piece(s) in a row<br><br>Print Victory message for player 1. Prompt for another game | pieces and Column 2 at 3.<br><br>Print Display Box: Player 1 name has 0 Win(s), and has 3 piece(s) in a row<br><br>Print Display Box: Player 2 name has 0 Win(s), and has 3 piece(s) in a row<br><br>Print Victory message for player 1. Prompts for another game |
| testRepromptAfterVictory | Player 1 won the first game over Player 2. | Print Victory Message for player 1.<br><br>“Player 1 wins.”<br><br>Ask if they want to play again or if they want to end. | Print Victory Message for player 1.<br><br>“Player 1 wins.”<br><br>Ask if they want to play again or if they want to end. |
| testFullColumn | Player 1 & Player 2 have each placed 4 of their pieces in C(olumn) 1 on the grid. Filling up a column. Player 1 selects that column put their next piece. | Print Grid:<br><br>**Column 1 button goes away** | Print Grid:<br><br>**Column 1 button goes away** |

# Design

Instructions: [http://go.ncsu.edu/csc116-ce-design](http://go.ncsu.edu/csc116-ce-design)

## Journal

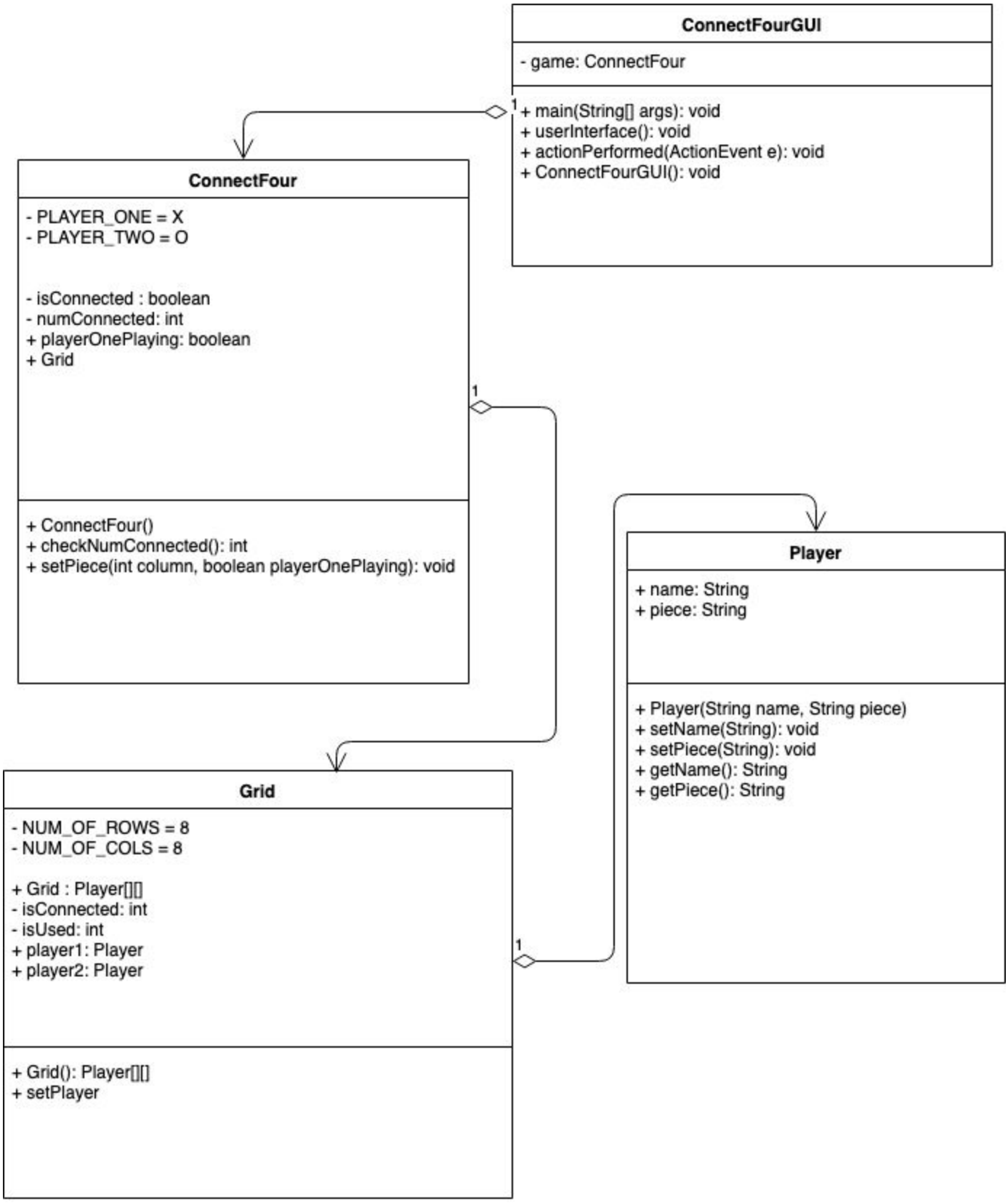
***Remember:*** You still will not be writing code at this point in the process.

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.

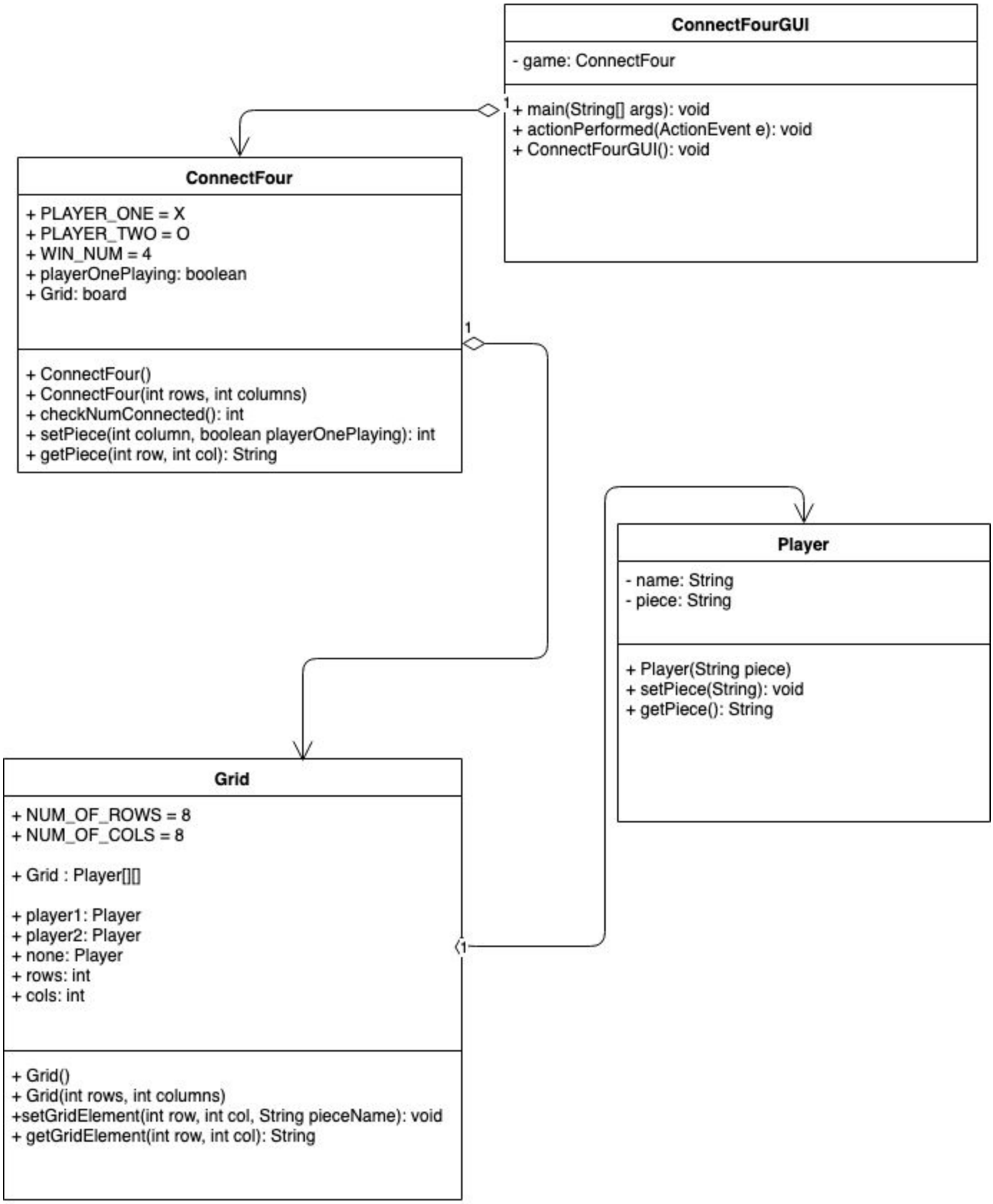
- List the nouns from your requirements/analysis documentation.
    - Player 1
    - Player 2
    - Grid
    - Used Pieces
    - Connected Pieces
- Which nouns potentially may represent a class in your design?
    - Grid
- Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
    - Color: Piece
    - Connected: Grid
    - Used: Grid
    - Filled/Empty/Occupied: Grid
- 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.
    - Overarching class for grid. The attributes are the black pieces and the red pieces separately.
    - Or another option being a Class for the grid, and a class for the pieces where the grid will be an array of piece objects whose color is determined by the piece class. Knowing if the piece is filled or not (null or contains a piece object), would change how it is displayed, which would be controlled by an overarching GUI file
- Which design do you plan to use? Explain why you have chosen this design.
    - We plan on using the second one because it allows us to break down the process of creating the pieces more, which in turn will allow us to debug easier along with making changes more intuitive.
- List the verbs from your requirements/analysis documentation.
    - Places
    - Updates
    - Output
    - Prompt
    - Keep Track
    - Play
- Which verbs potentially may represent a method in your design? Also list the class each method would be part of.
    - Places: Grid
    - Updates: Grid
    - Output (display): Grid
    - Prompt: ConnectFourGUI
    - Keep track (victories): ConnectFourGUI
- Other notes:
    -

# Software Design

UML Before Implementation



UML After Implementation



# Implementation

Instructions: http://go.ncsu.edu/csc116-ce-imp

## 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.
    - We will need to implement reference semantics with making 2 player objects and then referencing them within each element of the grid
    - We will use the idea of 2D arrays. This is applied during the creation of the grid, and allows us to display a message/indication of who is playing (player 1 and 2)
    - We will use loops to check for how many pieces in a row and also to check which element of a certain column is null.
- Other notes:
    - Make 2 player objects at the beginning and prompt the user for their names

# 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.>>

# Testing

Instructions: http://go.ncsu.edu/csc116-ce-test

## 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 longer displays how many pieces placed nor how many pieces are left
- List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.
    - White-Box
        - Player
            - Set up player objects
            - Tested get name for both player objects
            - Tested get pieces for both
        - Grid
            - Set up a grid object
            - Tested preconditions for invalid rows or columns
            - Tested get and set for piece placements of both players
        - ConnectFour
            - Set up connect four board
            - Tested preconditions for invalid spots
            - Tested the count
            - Tested get and set for set pieces
        - ConnectFourGUI (BlackBox)
            - Tested proper display
            - Tested correct piece representation
            - Tested win counter
            - Tested pieces in a row proper updates
            - Tested proper win messages
- Other notes:
    - Double victory happens when the player clicks really fast after starting a new game.
        - Issue fixed with hiding the display before prompting for another game as it saved the input otherwise if double clicked while waiting for the user to say if they want another game

# 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: http://go.ncsu.edu/csc116-ce-presentation

## 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 product is the game ConnectFour with a visual element. We display a Graphical User Interface with some command line interaction as our game
- Describe your requirement assumptions/additions.
    - Requirements
        - Players to be differentiated between each other
        - Players knowing how many are in a row
        - Players knowing how many wins they have and who won each game
        - Players being allowed to play again
        - Players being able to see the board
    - Assumptions
        - The end user wanted their pieces to be X's and O's as stated in the client file
- Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?
    - We could have made the entire project on the command line or in a GUI. We opted for a mix as the command line was easier but GUI more visually appealing.
- How did the extension affect your design?
    - It made us add how many pieces were in a row
    - We also had to prompt the players for their names as the customer wanted the names displayed
- Describe your tests (e.g., what you tested, equivalence classes).
    - We tested the 3 object class files via white box and the GUI via black box
        - We test to make sure that variables are stored properly and we can access all the methods and variables we need within all 4 java files
        - We test to make sure that the GUI displays everything that we need to display and done properly
            - Player names, wins, pieces in a row
            - Proper prompts following the game's end states (draw and a player winning)
- What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?
    - We learned that software development takes a lot longer and a lot more effort than previously thought
- What functionalities are you going to demo?
    - We will show that you can place a piece in every column, that once a column is full you can no longer place a piece, victory display works properly, reprompt works and has the intended effect, and wins are updated properly

- 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.)
    - Johnathan
        - Discussing the implementation along with requirements and basic assumptions. (leaving the GUI display to Jay as it is better to give visuals)
    - Jamea
        - Discussing the design approach we decided to take and why. Also what we learned from this project
    - Tess
    - Jay
        - Demo along with explaining the process behind it, and demonstrating that everything works (essentially demonstrating the blackbox test cases)
- Other notes:
    - To extend Jay's time, he could talk about some bugs that we ran into and how we worked with them

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

# Grading Rubric

http://go.ncsu.edu/csc116-ce-grading