Connect 4

Final Report

Team Nein

Jake Snitily, Cody Kesselring, Tu Do, Alex Sautereau

Course: CPSC 224 - Software Development

I. Introduction and Project Description

Welcome to our Connect 4 Game Project, where we're transforming the classic Connect 4 by Hasbro into a digital gaming experience. In this 2-player strategy game, participants drop colored tokens into a grid, fighting to align four tokens in a row either vertically, horizontally, or diagonally. This document serves as a progress report, delving into the technical aspects of our development efforts. It outlines the evolution of the user interface, the implementation of key features like multiplayer capabilities, token placement, winning conditions, and game state management.

The purpose of this document is to highlight the broad strokes we took in the development process of Connect 4.

II. Team Members - Bios and Project Roles

Jake Snitily is a computer science student concentrating in cybersecurity and statistics, with a passion for biological systems. His prior projects include but are not limited to, Farkle, DataScience algorithm implementations, and a pumpkin cannon. For this project, Jake's contribution to the Connect 4 project came in the form of win/draw checking, GUI construction, and report creations.

Alex Sautereau is a computer science student concentrating in software security. His past projects include a Farkle game, Poker, a recent movie release recommendation app, as well as the recreating of Flappy bird using Swift. For this project, Alex's contributions consisted of creating the base UI for the game and adding the dark mode feature and bug fixing.

Cody Kesselring is a computer science student with interests in cyber security and artificial intelligence. His prior projects include a spotify data analysis, farkle, Connect 4, and assignments from his Freshman and Sophomore classes. For this project, Cody's contributions to the Connect 4 project were laying the foundation by creating the main and board class which initialized the board within the console, additionally making tweaks to the GUI to make it better.

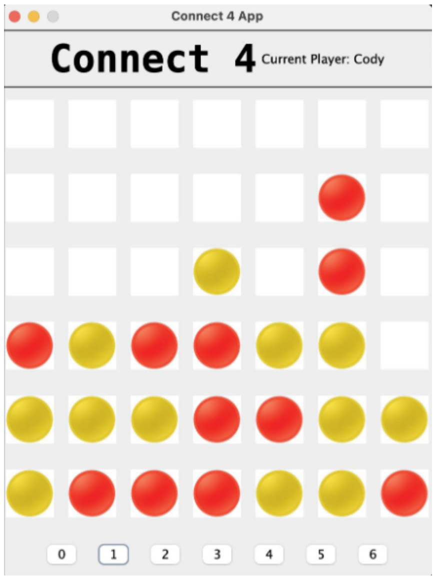
Tu Do is a computer science student with an interest in software development. Tu has a diverse background in various projects including contributions to projects such as Farkle, data science implementations, and class assignments that demonstrate a strong foundation in programming. His job in the Connect 4 implementation is GUI, including the logic behind placing pieces on the board and displaying the correct pieces.

III. Project Requirements

The success of a game lies not only in its engaging gameplay but also in the seamless integration of user interface features, dynamic token interactions, and effective game state management. In our project, focused on developing a Connect 4 game, we have identified key requirements that form the foundation of an enjoyable multiplayer experience. In order for a virtual Connect 4 game to work properly, it should roughly mimic the physical game in order to make the experience intuitive and recognizable. With that in mind, communication is a big deal when working with others, so we frequently kept in touch through discord and git commit messages; as far as the technical framework, we laid out the design before writing our program in the table below:

Multiplayer UI features	The game must be entirely setup & played through it's UI, allowing for multiple player rotation through the turns. Players must be assigned pieces easily differentiable from each other. This feature is necessary in connect 4
Adding Tokens to Grid	A player needs to be able to click on a button at the top of a column to insert their token, which will fall down to the lowest unoccupied position in that column. The color of the spot (aka image) in which the token is placed should turn into the token's color to indicate that the spot is not occupied. Without this feature, the game would not be able to progress towards its ending state.
Winning screen	A player must be able to win once they place a token that will create a 4-token line, whether horizontally, vertically or diagonally. This is a key feature of the program as the game would always end in a draw without it.
Game ending in a draw	If every spot on the grid is being occupied and no 4-token line has been made, then an ending screen must be shown to indicate that the game ended in a draw. This is also an important feature as it provides an ending to our game.
Game State Management	The game needs to know who's turn it is, detecting a win or tie, and transitioning states of the game (new game, win screen, scoring)

IV. Solution Approach



For this project, our team decided to split up our class implementations between our three classes MainGame, connect4GUI and Board (see **Appendix B**). MainGame class initiates the GUI & originally launched a terminal based prototype. The class connect4GUI builds a playable GUI & manages GUI events with interactions with our game. Lastly, the board class manages the player movement, keeps track of pieces as well as win and draw checking.

We wanted to handle our player interaction through a dialog box at the beginning of the game, consisting of a text field asking for the players' names. Players simply pick the column to drop a token in every turn. Some of the GUI features that needed to be implemented were the GUII dynamic reflection of every move made within the game, as well as current player update after every valid move.

As far as Game logic, we decided to handle Win & Draw detecting by checking after every move made by the player whether four tokens were placed in a row vertically, horizontally or diagonally. Once the game has ended, the player would be presented with an option to either quit the app or replay. Replayability was added by resetting the board and GUI, making the loser go first for the next game

V. Test Plan

Our Connect 4 project used tests to ensure the correctness of its implementation. The testing process encompassed various aspects, including unit, functional, and user tests. The BoardTest class was created to validate critical functionalities. The tests covered initialization, making moves, checking for wins (horizontal, vertical, diagonal), determining tie games, and handling invalid moves. The test suite ensured that the Board class correctly initialized the game board, allowed valid moves, and accurately detected winning or draw scenarios. Additionally, the Connect4GUI class was tested for creating a functional graphical user interface, handling player interactions, and updating the display after each move. These tests, executed by the main method in BoardTest, provided confidence in the reliability of the Connect 4 implementation. The simple nature of the test suite addressed cases such as win/draw checking, GUI construction, and foundational class creation, ensuring the overall success of the project.

VI. Project Implementation Description

Once the application has booted up, the players can enter their names in a textfield, updating the appropriate variables to keep track. Once the first button to continue is pressed, the game changes into its “playing mode”, now presenting the main board with a grid and 7 buttons. Most of the Game logic is handled using a 2d array of characters called board in the Board.java class. When a column is clicked, our program uses that array to add a character in the appropriate place within the board, and updates the GUI. If a player tries to place a token in an already full column, a message error will pop up (see **Appendix A**) Our checkWin() function is called after every move,using the 2d array to check whether four tokens in a row have been placed. Once that condition has been met, another window pops up, now indicating that we have entered the last stage of our game: the ending (see **Appendix A)**. Our game will now use the Connect4GUI.java class file to collect data from the player’s score, asking them if they would like to replay or quit and showing which player has won. If the replay button is pressed, then our 2d array will be reset to no longer contain any token, as well as updating the GUI to display that. Our game also implements a “Dark mode” feature that switched the color of various labels, images and backgrounds to a darker color for visibility purpose.

Git repo: https://github.com/GU-2023-Fall-CPSC224/final-game-project-team-nein.git

VII. Future Work

In the future we could make the GUI more professional, as it looks pretty bare bones with the default java swing aesthetic, with the ability to change the color of the user's pieces. This would include a more thorough dark mode and making the board look like an actual Connect 4 board instead of a digital grid. Additionally we could add the ability to make the board larger to give players more options to form more articulated strategies that couldn’t be brought to fruition in the default 6x7 grid. Although the 6x7 grid allows for a quick and easy game, it could feel cramped and results in many ties.

VIII. References

Cite your references here if you’ve got any. At the very least you can cite this:

For the papers you cite, include the author(s), the title of the article, the journal name, journal volume number, date of publication and inclusive page numbers. Giving only the URL for the journal is not appropriate.

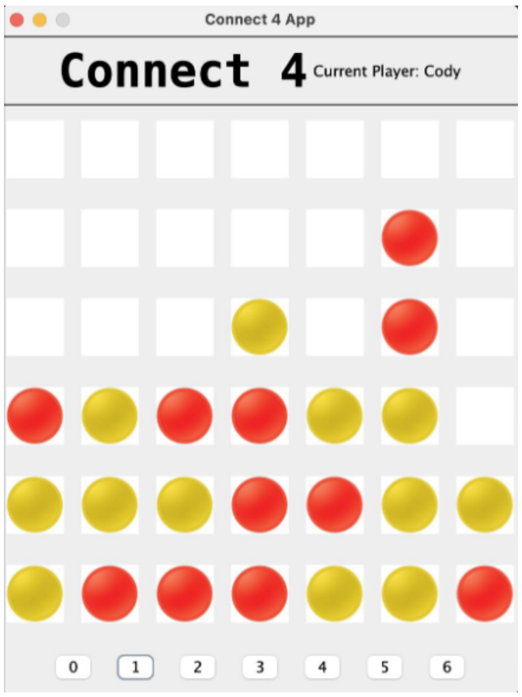
For the websites, give the title, author (if applicable) and the website URL.

Please use either Chicago or IEEE format for your citations

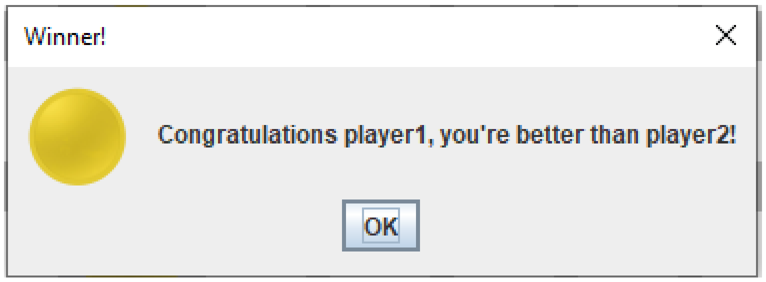
IX. Appendices

Appendix A: User Interface Mock-ups

UI Sample 1: Main Game Board

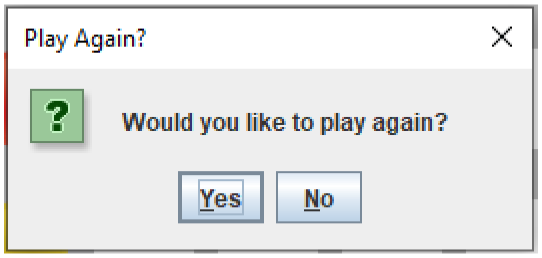


UI Sample 2: The Winner's Screen



Congratulations player1, you're better than player2!

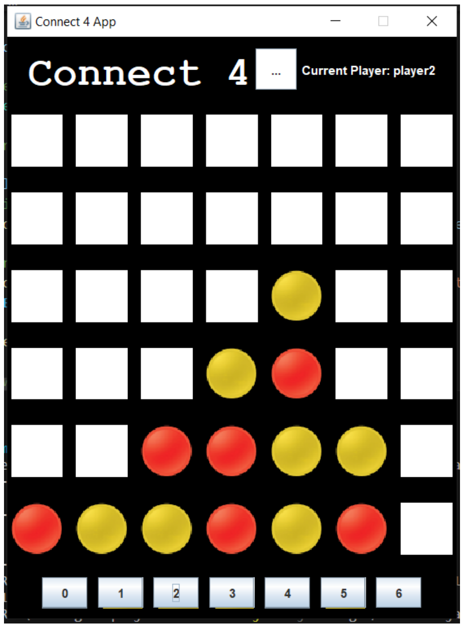
UI Sample 3: The replay screen



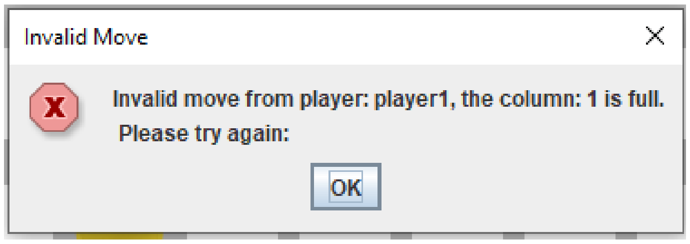
Play Again?

Would you like to play again?

UI Sample 4: Dark mode feature



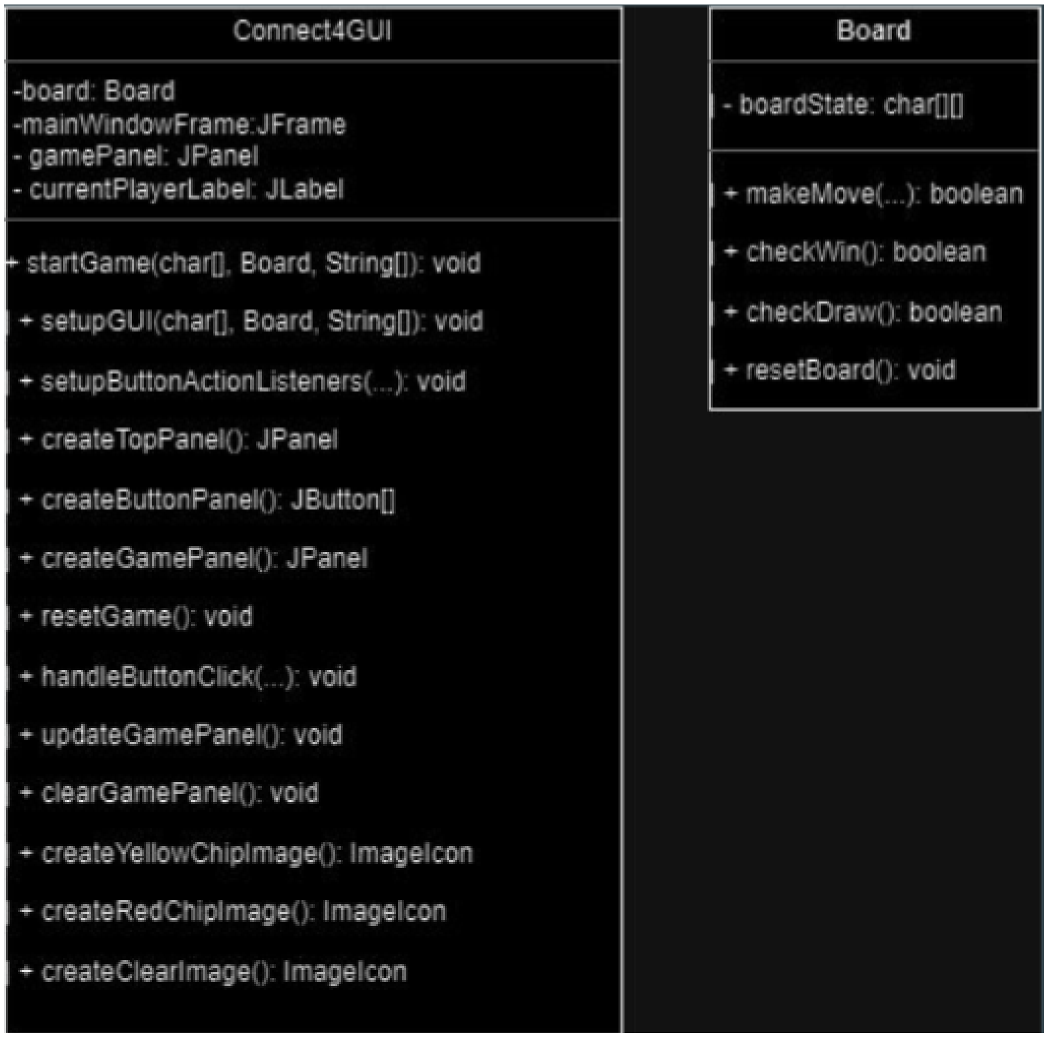
UI: Error message for invalid column move



Invalid Move

Invalid move from player: player1, the column: 1 is full.
Please try again:

Appendix B: Class Diagram for UML


Connect4GUI
-board: Board
-mainWindowFrame:JFrame
- gamePanel: JPanel
- currentPlayerLabel: JLabel
+ startGame(char[], Board, String[]): void
+ setupGUI(char[], Board, String[]): void
+ setupButtonActionListeners(...): void
+ createTopPanel(): JPanel
+ createButtonPanel(): JButton[]
+ createGamePanel(): JPanel
+ resetGame(): void
+ handleButtonClick(...): void
+ updateGamePanel(): void
+ clearGamePanel(): void
+ createYellowChipImage(): ImageIcon
+ createRedChipImage(): ImageIcon
+ createClearImage(): ImageIcon
Board
- boardState: char[][]
+ makeMove(...): boolean
+ checkWin(): boolean
+ checkDraw(): boolean
+ resetBoard(): void