CIS 434 - Software Engineering

Group 4

Connect Four

April 24th, 2021

Sunil Khatri, Valerie Jarvis, Milica Jandric

Contribution Breakdown:

In the beginning of the project we were all coding together during our weekly Thursday group meetings, so the first half of the code that we created was largely spread out equally. The people who were coding specifically during each part are as follows: Val coded specifically the solution checking functions, Sunil did the player setup, and Milica did the initial setup of the program. We then switched our model after the progress checkup as it was made clear to us that that was not what was necessarily wanted for this project. Our contributions to the subsequent code are as follows:

Val: Setup of the board with the button grid system and aesthetic improvements to the board such as color touch ups and rounding the buttons.

Sunil: Connecting the previously made Connect Four code to the GUI created by using Action Events.

Milica: Designing the best of series and mocking up code to implement it within the Connect Four base code.

For the report, Val wrote out the objectives (modifying from previous proposal objectives) and the conclusion, Sunil wrote out the project description and abstract, and Milica wrote out the professional awareness section. Sunil, Milica, and Val created a progress update diagram at a meeting together on how they were going to distribute the rest of the work, as well as the gantt chart was created in a similar fashion at the beginning of the project. Milica created the user manual that was submitted with this report. Val created the system process model for this project. Sunil and Val wrote the contribution breakdown together.

The operating system required to run this program successfully is Microsoft Windows. This program will not run in any other operating system unless there are specific configuration changes in the files and compilers. In order to successfully run this game, one must have a Java Runtime Environment installed. To be able to modify the program, a Java Development Kit would need to be installed. All of the developers for this program had the above requirements met; each individual was able to contribute their code on this project.

Requirements:

- Microsoft Windows Operating System
- Java Runtime Environment (JRE) - to establish an environment for the Java program to be developed.
- Java Development Kit (JDK - to develop and modify the program)

Abstract:

In 1973, Howard Wexler and Ned Strongin had invented the game Connect Four. The game uses a 6 x 7 slotted rack, where there are 42 slots total. As this is a two player game, each player gets one designated color, blue or red. Each player takes turns inserting one of their respectively colored discs into one of the slots in hopes of matching 4 like-colored discs, horizontally, vertically, or diagonally in a row. This project's goal is to create that same game, but for a computer. The program will be written in Java along with its GUI. The objectives of the program include checking win conditions repeatedly, ensuring a stable GUI, as well as maintaining efficient time-complexity for the program to ensure minimal delays. The overall methodology involves writing the program first, creating the GUI after the code is fully functioning, and then touching up code to ensure efficiency.

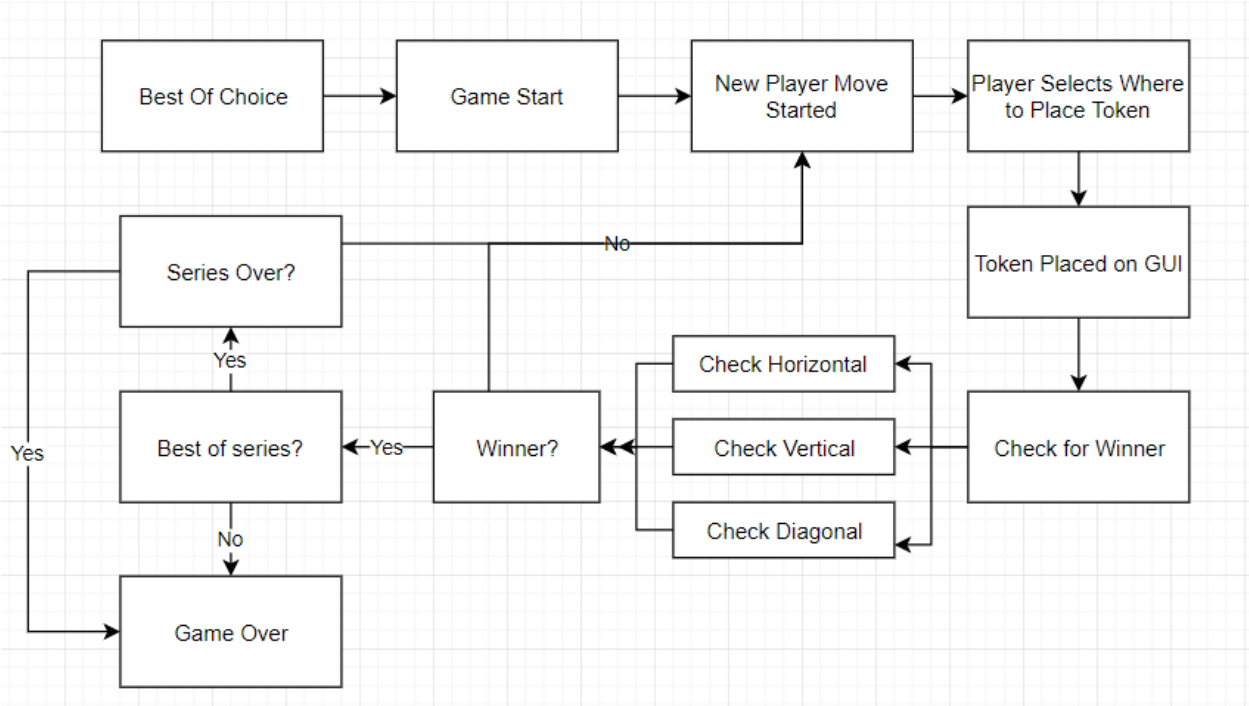
Objectives:

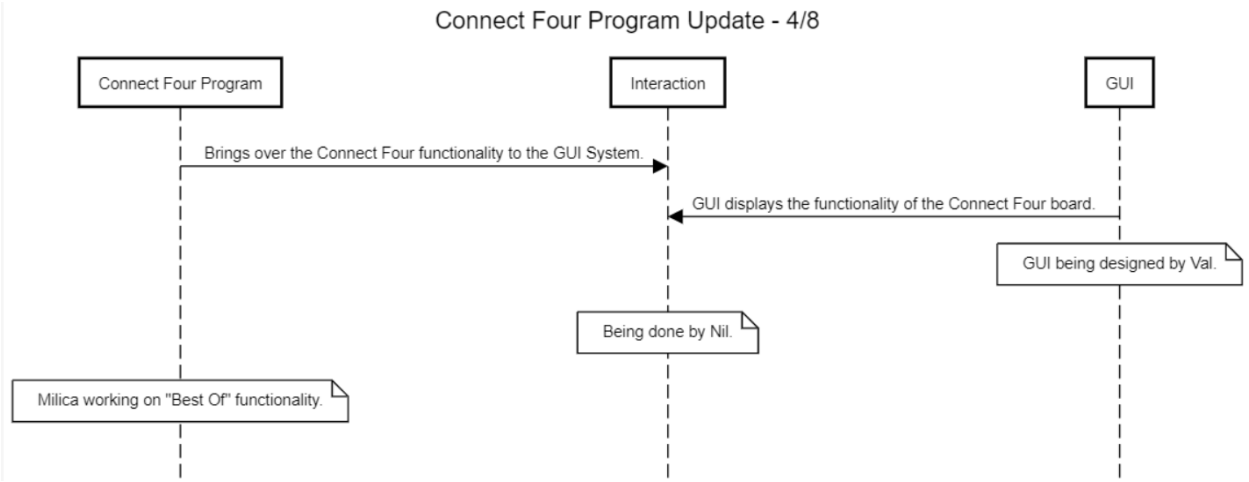
Our main objective in this project was to create a java-based application that will run a game of Connect Four between two players at the same terminal. This game of Connect Four will be able to tell if a player wins vertically, horizontally, or diagonally, as well as take input from each player one after the other. The game will be shown via a java-based GUI (or graphical user interface) so that each player can see in a streamlined manner where their play goes. The GUI includes a grid of JButtons that allows the user to click on where they want their piece to go. The GUI is also styled in the traditional Connect Four game style of yellow and blue with user tokens being red and blue.

Another goal of ours was to create a program that runs efficiently so that there were no noticeable delays after the user places the chip while the computer checks to see if the user has won. This was done by minimizing the check functions until they were actually useful. This was done by delaying the checking of diagonals until the ninth turn was over. The GUI itself displays seamlessly, and we wanted to make sure the time complexity was never more than O(n^2).

The main methods that were established are those that checked the solution of each type (vertical, horizontal, and diagonal), got each of the players moves, established a GUI representation of the board, and maybe one or two helper functions that helped us to reach the workability of each of those main functions. We also created a ConnectFour class itself to house the GUI aspects.

Project Description:





Very detailed explanation of what process you went through to create your project.

• All the features of the software.

• Difficulties and solutions.

• Charts/graphs on the software development process.

• How you would improve your project given more time.

Currently, our project includes several features to ensure a smooth game operation workflow. These features involve best of series to include multiple games in a set, automatic turn switching, as well as constant win condition checks for each game in a series. There are also some precautionary features implanted such as invalid entry checks for invalid user inputs. These features all work together in our project to ensure the users could operate this program efficiently and smoothly.

Our project's "best of series" implementation ensures users can set the number of games to be played in a set, based on users' preference. This is done by asking the user how many games they would like to play and then the program waits for the user's input. This input is then held in memory and will be checked after the completion of each game to see if any users' win count matches the number of games in the set count. Once they match, the game alerts who the winner of the set is.

After each player contributes their turn, win condition checks are completed by our game to check if the current game state contains any winners. If there are no winners, the game continues, and again performs a check after the next player. The win condition for this game is to have 4 like-colored tokens in a row, whether horizontal, vertical, or diagonal. These check methods check the current game state's instance to see if there are 4 like-colored tokens horizontally, vertically, as well as diagonally; if there are, then the game will notify the user that the game is over and indicates the winner. The above-mentioned "best of series" check will be

performed again to see if any users' win count matches the number of games to win in a set count. If it does match, the game is over and an alert pops up to indicate who won the set. Based on the number of games in the set, the game will start again with a new instance, after clearing the previous instance

To further ensure smooth operation of this game for users, invalid user input checks were also implemented. This is to ensure the program does not crash or run into any other errors based on users' input. A very important check implemented was to ensure that the row being entered is one of the 7 rows available. Any other input will alert the user that the input was incorrect and will suggest them to enter in a value from 1-7, again.

Some of the difficulties encountered was getting Java key listeners to function properly – mostly override function related issues. After doing further research and tests based on Oracle's Java documentation, it was advised to use Action objects for Java in conjunction with Java's Key Binding API. Based on the documentation and taking Oracle's advice, the Key Binding API as well as Action objects were used to help solve that problem.

This project would certainly be able to grow and have a lot more features if time permits. Some features could include using mouse-click functionality instead of key input listeners, providing users more customization options such as changing background colors and token colors, and including an audio interface in our program. Time complexity would also be investigated further to ensure we have the best algorithm to compute the calculations as efficiently as possible.

Professional Awareness:

Professionalism is when an expert in their field is dedicated and hardworking to produce quality results with no mistakes. Being professional in software engineering requires a high level of dedication, skills and knowledge related to computer systems which can have a large impact on real world applications. This is because software is in every aspect of our lives, from personal interactions all the way up through enterprise transactions. All this data and our personal private information is being stored in these systems. As software engineers, it is our responsibility to give our best at everything to ensure that the software we produced exemplifies who we are as professionals. As a software engineer, you are directly responsible for shaping the world around us, whether it is improving society in general or avoiding any harm that can come to it. This is the responsibility that comes with the profession and one should be aware of the issues and responsibilities. Therefore, it is important that Software Engineers should maintain being professional and never misuse other's personal information and should use proper encryption to protect each individual data.

Furthermore, some main qualities of software professionalism are demanding quality, stability, continually producing, estimating honestly and continuously learning. First, in terms of demand quality, we should always do everything in our power to ensure that every piece of code we write is thoroughly scrutinized and tested. Next is stability, where we should always be writing all of our code in an agile manner that provides the business with choices, while still providing a high level of value. Moreover, continually producing is where good practices are utilized by software engineers. Good practices such as always writing clean and easily readable

codes, proper documentation of changes made in codes, and continuously modifying code to add new functionality. Furthermore, in terms of estimating honestly, professional software engineers should provide an estimation range to the business of any point during the development process. Lastly, in terms of continuously learning, software engineers should always keep up to date so that they do not fall behind, like new and more stable implementations of a certain software, security and many more.

The software development process can be broken down in a couple of phases, such as communication, planning, modeling, construction and deployment. This is a very large process and requires multiple professionals that vary in skill. Therefore, one of the most important responsibilities is the ability to collaborate effectively, efficiently and overall in a productive manner. This also includes the ability to communicate with users, clients and stakeholders so that everyone is in terms of where the project or product is headed. Additionally, the product needs to be developed legally and ethically, but also fulfilling the desires of the contract.

Social responsibility is one of the most significant obligations that is defined in ethic codes for software engineering. Be that as it may, social responsibility is more ambiguous than other responsibilities. Once a software engineer obtains the right to practice his/her profession, he/she is required to obey by the proper responsibilities. However, the definition of responsibilities is not uniform but relied upon the environment and personal life experience of the software engineer. The social impact of software engineering is not always clear. This means that in many cases the impact of software engineering is not decided by the software itself but by the people that use it.

These responsibilities are drawn on the fact that software engineers have a better comprehension about their products and in this manner have the obligation to protect society from any damage or harm that may be caused by these products. In general, software engineers are required to develop products that improve society or at the very least do no harm to society. However, improvement and harm cannot be defined universally, which means they are left to the engineers and society to be defined. This is why social issues and responsibilities are an ambiguous area. Software engineers may have different views about what is "good" and "bad". This does not stem from the profession but from other factors, such as political views, environment, social position and religion.

In general, the main responsibilities are personal, professional and social responsibilities. Personal responsibilities are basic obligations which include honesty when dealing with others and the overall concern for their well-being. Professional responsibility are the obligations that go beyond personal responsibilities and are taken on because of the profession. And finally, as already stated, social obligations are the ones toward society as a whole. Software engineers, having received extensive knowledge and the right to practice, are required to share their knowledge with the goal of benefiting society. The world around us is running more and more on software each day. As a software engineer, you are directly responsible for shaping the world around us, whether its improving society in general or avoiding any harm that can come to it. This is the responsibility that comes with the profession and one should be aware of the issues and responsibilities.

An example that involves social issues and responsibility is a car software that might protect the interests of the car manufacturer but may be against the interest of the occupant of the car or the environment in general. Additionally, a poorly designed user interface could cause

people problems, while incorrect code can harm human life or damage property. Another example is a game that might be purposely designed to be highly addictive, which may be beneficial to the company but harm or damage the user.

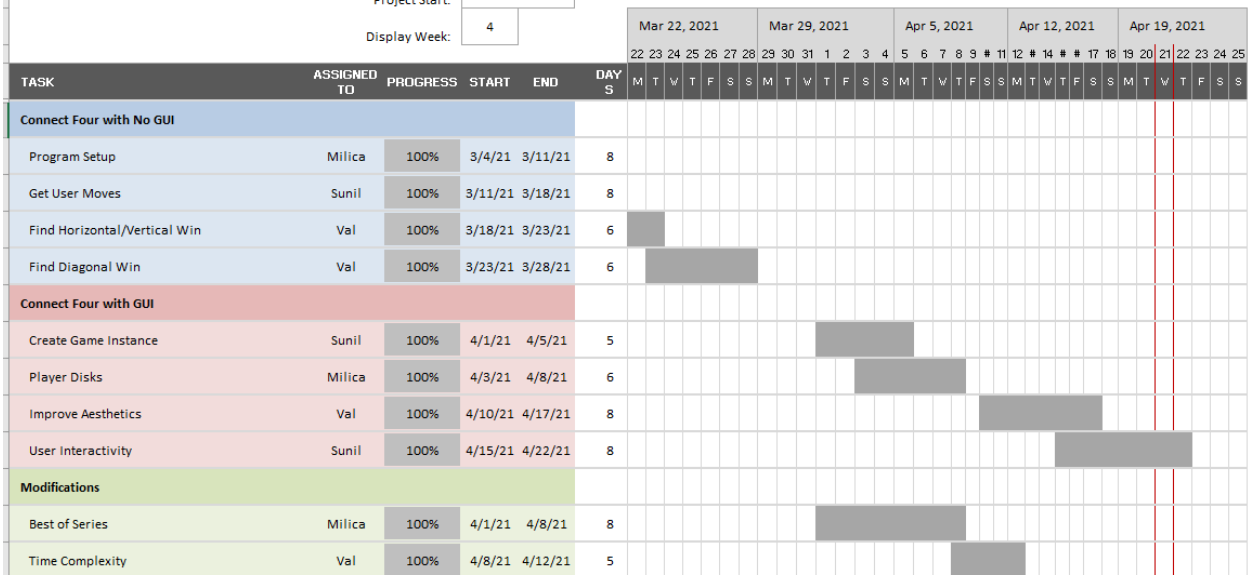
Project Timeline:

Sunil Khatri, Valerie Jarvis, Milica Jandric

Project Start: Thu, 3/4/2021

Display Week: 4

TASK	ASSIGNED TO	PROGRESS	START	END	DAYS
Connect Four with No GUI					
Program Setup	Milica	100%	3/4/21	3/11/21	8
Get User Moves	Sunil	100%	3/11/21	3/18/21	8
Find Horizontal/Vertical Win	Val	100%	3/18/21	3/23/21	6
Find Diagonal Win	Val	100%	3/23/21	3/28/21	6
Connect Four with GUI					
Create Game Instance	Sunil	100%	4/1/21	4/5/21	5
Player Disks	Milica	100%	4/3/21	4/8/21	6
Improve Aesthetics	Val	100%	4/10/21	4/17/21	8
User Interactivity	Sunil	100%	4/15/21	4/22/21	8
Modifications					
Best of Series	Milica	100%	4/1/21	4/8/21	8
Time Complexity	Val	100%	4/8/21	4/12/21	5



Gantt Chart showing our last weeks of production and completion statuses.

Conclusion:

We have used this semester to create a Connect Four program that can be played by two people at the same terminal. The main features include a “best of” series option, a GUI that the users can interact with to play their game, and time complexity updates to make the program function smoothly. Our course of doing this project saw us doing a gantt chart to keep track of where each of us was in the process of updating the code, a few diagrams to display exactly how we wanted each of the parts to work with each other, a few different versions of prototype designs until we landed on a proper look for the GUI, and weekly meetings to go over what needed to be done and who needed to do what. We also created a simple guide for our program so that people would know how to run it.

The main takeaways from this project would be the difficulties and benefits of working with a group of people on a long term project. For normal projects, we would simply meet over discord and code together, but as this one was assigned over such a long period of time, there was a lot of planning, preparations, delegating, and diagrams that needed to be done in order to organize all that had been done vs. what we still needed to do. Another important factor of this assignment was that we had to work with people we had likely never worked with before, this presented some unique challenges in working with people's different styles of production. However, by the end we were able to seamlessly understand what we needed out of each other and make sure that we ended up presenting a wonderful project.

References:

[1] Hill, G. (2018). Connect 4: A Novel Paradigm to Elicit Positive and Negative Insight and Search Problem Solving. Frontiers. https://www.frontiersin.org/articles/10.3389/fpsyg.2018.01755/full.

[2] Myers, C., Hall, T., & Pitt, D. (2007). The responsible software engineer: Selected readings in IT professionalism. London: Springer.

[3] Stellabotte, R. (2018, July 11). Toy Story: Catching Up with Howard Wexler, Inventor of the Classic Game Connect 4. Fordham Newsroom. https://news.fordham.edu/fordham-magazine/toy-story-catching-up-with-howard-wexler-inventor-of-the-classic-game-connect-4/.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.