**Introduction to Connect Four:**

Welcome to CONNECT 4!

To begin with, the code will welcome you to the game and list all the rules that should be abided by throughout playing. The rules that will be given are the following:

1. The game board has seven columns and six rows.

2. There are 21 red and 21 yellow tokens.

3. One player plays with red tokens, and the other with yellow tokens.

4. The tokens are inserted at the top of a column, and they will fall and land on the ground (if the column was empty) or on top of a previously inserted token.

5. Red starts.

6. Red and yellow take turns.

7. One can only insert tokens in one of the seven columns.

8. One cannot insert a token into a full column.

9. A line consists of several tokens, either in vertical, horizontal, or diagonal form, which contains only tokens of the same color.

10. A player wins if they manage to form a line of four tokens of their color.

11. The game ends if one of the players wins.

12. There will be no ties. In case of a tie on the board, the player that took less overall time wins.

**Specifications for the main program:**

A big part of our project is specifications which consist of what a function requires along with its effects. We developed numerous specifications for all functions that make it easier for whoever reads the code and facilitate their understanding of it. The following are some of the most important specifications that we have included in our main program:

**requires:** a column number greater than 7 or less than 1 should result in an error since the columns are numbered from 1 to 7. The program requires a winner; therefore, if there was a tie (all tokens are filled), then we should define a way for the player to win (calculating the time of each player, and the player that took less time in making decisions will be the winner). Requires us to enter the names of the player and randomly select who will start the game. If the move is not allowed the program should inform the player along with the corresponding reason the move is not allowed, then ask the same player for another move (until the player enters a legal move). Display the board game.

**effects:** a function that draws a board game. A play function that puts the tokens based on the column selected. check win function determines which player has formed 4 tokens (horizontally, vertically, or diagonally) to define as a winner else we will pick the winner based on less time taken by either player. no player has an unfair advantage over another player where there is a function implemented with a flipped coin that determines which player will start.

**Testing for the main program:**

For the test cases, we started by testing the conditions:

- If User1 and User2 inserted their names and if they appear on each player's turn

- If the Users inserted in the column range between 1 – 7

- If the Users inserted letters/words for the column number

- If the Users inserted negative numbers

- If the Users inserted outside the column range between 1-7

- If the Users are trying to insert in a full column

- If the board is filled with 0's initially

- If the program specifies the reason for an illegal move

- If the program announces that the winner has won by speed

When facing these conditions, the program will tell the user to insert another value until it is accepted by the conditions set by the algorithm. Furthermore, we tried implementing a coin system, that has a 50% probability of knowing which player starts. (if it lands on 1 or head then User1 starts, if landed on 2 or Tails then User2 Starts). We started by plotting on a piece of paper a 6x7 table, to test the cases and see if the code works properly. To begin with, we made sure that if one of the users has 4 in a row vertically, in the same column, starting from either row 1, row 2, or row 3, then the user wins. When we tested these cases, it worked for all the columns in the range of 1-7, then the user wins. In addition to that we tested for the cases where the user inserts 4 in a row horizontally, starting from any column, provided they belong to the same row, then the user wins. We also tested the condition for having 4 in a row diagonally, such that the row increments (upwards) by 1 on every input by the user, and the column is either deviated to the left or the right on every attempt of the user to get 4 in a row, then the user wins. We finally tested the condition if no user won (no 4 same values are consecutive), then it’s a draw, and the winner is based on the time taken for each user to insert a value. The user that took the most time loses, and the user with the least time wins.

**Problems and Experience:**

As for the specifications part, we tried our best for phase one but it turned out to be where we had several flaws. After attending the TA’s session and discussing phase one together, it turned out that we had lost like 10 grades for specifications and that this was the reason why we focused and worked more on the specifications for phase 2 and fixed them as much as we can. What we did is that we made our specifications more detailed in a way anyone who reads them would understand what each function can do along with what it requires and what it returns. Another problem that we faced was during the second phase since we implemented our functions in phase 1 using char 2d array. At first, we didn’t know how to change from char 2d array which was originally the parameter being used in every function mostly to int double pointer as wanted by the professor. The function needed should use an int double pointer called board where it accepts the current board and returns the index of the column at which it wishes to make the move. Therefore, at first, we started searching google to find out how are we going to do that as code. The logical idea at first that we came up with was to be able to access all the elements in the double int pointer and then typecast this int into a char or a char pointer in some way, and then put these chars in a 2d char array where we now can call the helper functions that we implemented in chars. Unfortunately, all the ideas we found initially whilst searching made us more confused since there was no similar thing to the idea that we came up with. However, there were other strategies, which we tried connecting to our current code, but faced multiple problems when trying to do that. At this moment, we realized 2 important things that we haven't thought about or noticed previously, which where that not all ideas or solutions can be found ion google directly where sometimes you need to use deep searching through specific websites using the right terminology to find the answers you have been looking for and even after that you might not always find what you want. Therefore, we went to certain senior computer science students that took the course previously and have more knowledge than us which helped a lot along with visiting the TA and asking them for help. We started thinking of another way to deal with this problem where we intended to change our version of all the helper functions to the needed one which is int \*\* board. This way of thinking leads us to that in the end, the code has run with the argument the professor asked for. Another realization is that working as a team is really important since one can help another one in figuring out how to do a certain code and explains it to him so that he can learn later. Others can open up ideas and ways that the rest of the group didn't come up with and start implementing the code. They can enlighten someone into a more interesting idea or shed the light on it for them.

**Specifications and description of the bot move along with the algorithm used:**

For the bot move, the helper functions that we implemented and used for the bot move all include an argument as int \*\* board[6][7], but for the competition and the project the argument should be as int \*\* board, so the first thing required to do is accessing the int \*\* board elements and then allocate them in an int\*\* board[6][7] in order to be able to call the helper functions we designed. This function will return a column number representing a bot move. This function is based on the Monte Carlo algorithm and probabilities where the algorithm is mainly designed in a way to suggest a move among the (at max) 7 possibilities. For each move, we play random games and store the ratio between won and lost games. Finally, the move with the best ratio is returned. If there are no valid moves, -1 is returned. This is one of the simplest implementations of a connect four AI we can write, yet it is strong enough to play with humans showing to be capable opponents. The implementation uses the Monte Carlo methods class of algorithms: for every potential move the bot can do, it does it and then plays random games, sampling the number of games won and lost. Then the decision on what move to make is made based on the move that appears to have the best won/lost ratio. The code is very simple but there is space for optimizations. we will be working on the optimizations later to make our function more accurate.

**Testing for the bot move:**

At first, we will check if the row at the bottom is empty. If it is, then my bot is the first player to play in the game. Therefore, for testing for this, the elements in the bottom row in the 2d array that we passed to the function will all be zeroes and we will check if our bot will be the first player to play or not. If the bottom row is not empty and contains a token in one of the columns, then our bot is the second player to play. To test this, we put a token (either the value 1 or the value 2) at any column in the bottom row and see the effect. The copy function will help us to get a temporary 2d array which is used to perform the random games separately from the main 2d array and have the calculations. For testing it, I passed with for loops on every element of the new copied array and checked if it is equal to the original one (note: here the size of the rows and the size of the columns are fixed, so the testing for these sizes only is enough for my purpose). The drop function will help us to drop and set a value at a specific row/column position. To test this, we tried to set a value of an empty position twice (the first one with value 1 and the second with the value 2), I then tried to set a value on a non-empty position and see what happens. getWinner function has the same testing as checkWin function (the testing of checkWin is in another document). randomGame function will generate different scenarios of playing a game. To test it, we tested it by a large number of games and saw if there were random results like our bot winning, losing, or if we have a tie. If the results were all the same, then there is something wrong. If the results contained only 2 result possibilities (like winning and losing or winning and tying or losing and tying), then there is something wrong. For testing the bot move in all, we played with the bot several times and started putting values that will aim to help us connect 4 tokens to win, and we observed how will our bot will respond to our moves (will it let us win or will it block it?). Also, we tried to put for the bot 3 tokens (either horizontally, vertically, or diagonally) to see if the bot will put the winning token or not. We searched how can we optimize the Monte Carlo algorithm more with a lower running time and we found out that by increasing the number of random games, it will be more likely for the bot to win; however, this is not always true since we are dealing with probabilities and the likelihood for the bot to lose may increase also. In the pdf we saw, there was an implementation of optimization of the Monte Carlo algorithm without the use of increasing the number of random games. It was really interesting to research this algorithm!

**Overall:**

It was a great project experience especially that it is our first time taking a course that includes projects. Although there were some hardships, but we tried to best to work as a team to hurdle all obstacles we face. Most of the times we tried to separate our work between the group members which improved our teamwork skills. It was also a beneficial experience since for most of us like I have mentioned before, this is our first real technical project in a team where we learned several skills like communicating with each other, proposing ideas, searching google, looking up resources, and researching interesting algorithms that we needed to use in while developing our program revolving around Connect Four such as Monte Carlo algorithm, minimax algorithm, alpha-beta pruning. In conclusion, I think more courses should be given the way 270 is given because it doesn’t only depend on memorization or understanding the lecture but actually enhances one’s self learning skills along with setting him one step closer to being an independent computer scientist.