Connect Four

Algorithms and Data Structures coursework report

Author: Klaudia Jaros 40439268

Contents

Introduction

Connect Four is a two-player game where two players take turns dropping distinctive discs into a seven-column, six-row grid. The objective of the game is to be the first to form a horizontal, vertical, or diagonal connection of four of one's own discs. (Wikipedia, 2021).

In this coursework, an attempt has been made to implement Connect Four game as a command line application using the C programming language, C standard libraries and self-written libraries.
The result of the design and implementation of the above idea is a working protype of the game Connect Four, featuring two play modes: single player – a user against a simple randomly generated computer moves and a multiplayer – two users playing together. Both modes can be played using the assisted mode that allows players to undo and redo moves or in a competitive mode without allowing the users to undo their moves. Additionally, any game played during the application runtime is being saved to allow for game replays. Past games can be found the the “Game History” in the main menu and can be re-watched.

Design

User Interface

As mentioned above, Connect Four is a command line application therefore a simple Text User Interface has been implemented to provide the basic functionalities (Appendix A).

Data Structures

To allow the game to function correctly, several data structures were needed to store information about the game, players and moves.

1. **Game board**

Game board was implemented using a simple **char array** of size 42 since the board grid is 6 by 7 in size. The array stores either 'X' or 'O' as the players' moves or ' ' an empty space if not occupied by any move. The reason for using an array is that a fixed size was provided, it is easy to mark where the move was made by direct access using index and offset, and it is easy to print the entire board for the user to see. It also does not require a lot of memory.

2. **Players**

To store information about the players such as their names, token ('X' or 'O') and flags if the player is a "computer" for the single player mode or if the player is the winner a **Player struct** was used. The reason for that is that a struct groups all the information together and it is easily accessible throughout the game play. A pointer to a struct can also be easily saved to Game History and then used to replay the game using player's names.

3. **Saving game moves**

To easily save players' moves and to be able to undo them and redo them easily a **stack structure** was needed. It was implemented as a struct with an array and a integer to keep track of the top of the stack. In the assisted mode where the players can undo and redo their moves, two stacks were used: one for players' moves and one for storing undone moves to support the redo moves option. Stacks allow for an easy pop and push operations on the last element of the list which is very useful when dealing with undoing moves.

Because a stack is also an array, it was fairly easy to turn it into a **queue** for the purposes of re-watching a past game. Instead of using pop and push, a simple loop was set up that starts at the beginning of the array and replays the players' moves following a first in first out order, rather than stack's way of last in first out.

4. **Saving games history**

To display unspecified number of past games and then replay them, another list was needed. A **LinkedList** structure was used to create a GameHistory struct that stores a list of moves taken in a game(a pointer to a Stack struct), two players (pointers to Player structs) and a pointer to the next GameHistory struct. Because different data structures were needed to be stored within that list, a simple array would not work in this case. Linked List seemed like the best option - it is easy to add a new element, and when displaying all the saved games, the starting point is at the beginning of the list and then it follows to the next link.

Algorithm to check for winners

After every turn, the program needs to check for possible winners. To achieve that, **a linear search** algorithm with a counter was deployed to check the char array (board) for a horizontal, vertical, or diagonal connection of four of a kind.

To check for winners horizontally, a simple loop was used to move from one cell to another, resetting the counter at the beginning of a new row.

Table 1: Check horizontally: moving from the start point to the right

Start point	0	1	2	3	4	5	6
	7	8	9	10	11	12	13
	14	15	16	17	18	19	20
	21	22	23	24	25	26	27
	28	29	30	31	32	33	34
	35	36	37	38	39	40	41

To check it vertically, two loops were needed: one to start at the bottom of each column and a second one to move up. An offset was needed to move from a cell at the bottom of one column to a cell up in the same column – minus 7 for the 6x7 board.

Table 2: Check vertically: move from the start point up

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
Start point						

To check for winners diagonally down, two nested loops were needed: one to check the first half of the board starting from the beginning of each row and moving diagonally down by adding an offset of 8 for the 6x7 board, and a second one to check the second half of the board starting at the top of each column and moving diagonally down again by adding an offset of 8.

Table 3: Check diagonally down: moving from the start point and going diagonally down.

	Start point						
	0	1	2	3	4	5	6
Start point	7	8	9	10	11	12	13
	14	15	16	17	18	19	20
	21	22	23	24	25	26	27
	28	29	30	31	32	33	34
	35	36	37	38	39	40	41

To check diagonally up, the same approach was used as above, but instead of moving down at each row, the loop moves up by subtracting 6 for 6x7 board and for the second half of the board, it starts at the bottom each column moving up by subtracting 6.

Table 4: Check diagonally up: moving from the start point and going diagonally up

Start point	0	1	2	3	4	5	6
	7	8	9	10	11	12	13
	14	15	16	17	18	19	20
	21	22	23	24	25	26	27
	28	29	30	31	32	33	34
	35	36	37	38	39	40	41
	Start point						

Gameplay algorithm

ConnectFour application consists of several loops and decisions based on the user's input. The entire program logic can be seen on Connect Four Flowchart in the appendix part of this report (Appendix B).

Single player mode

A "computer" opponent was required to allow for a single player game. To achieve that, the Player struct was equipped with a flag to indicate whether the next player is a "computer". If yes, instead of allowing a human player to take turn, a random number generator was used to generate a number between 1 and 7 which is the number of columns. If the generated move is invalid, the random number generator will try again and so on until a valid move is played (Appendix B).

Assisted mode

Every move is being saved in the moves stack. The undo/redo move option in the assisted mode utilises stack properties: if the player undoes their move, the move stack pops an element and the undone moves stack pushes one. If the user redoes their move, the undone moves stack pops an element, and the moves stack pushes one.

In the multiplayer mode, each user can only undo/redo their own moves and the moves have to stay balanced during the gameplay.

In the single player mode, this is slightly different. The player can undo both their own moves as well as the computer's. The reason for that is that computer cannot undo its own moves therefore it is impossible to go back to the beginning of the game. Also, if the player is still learning how to play, it is more convenient to be able to undo computer's moves.

Libraries and methods

Stack struct, Game History struct and their methods have been places into separate source files and compiled into a library to be then linked with the main application to keep the code readable and maintainable.

The main application has also been divided into small functions to make it more concise and easier to modify. Depending on the user's choices throughout the gameplay, the correct method will be called.

Critical Evaluation

Overall Connect Four is a fully working game. Features that work well are single player and multiplayer mode, saving user's moves, displaying the game history, and replaying past games. Undo and redo options work well too.

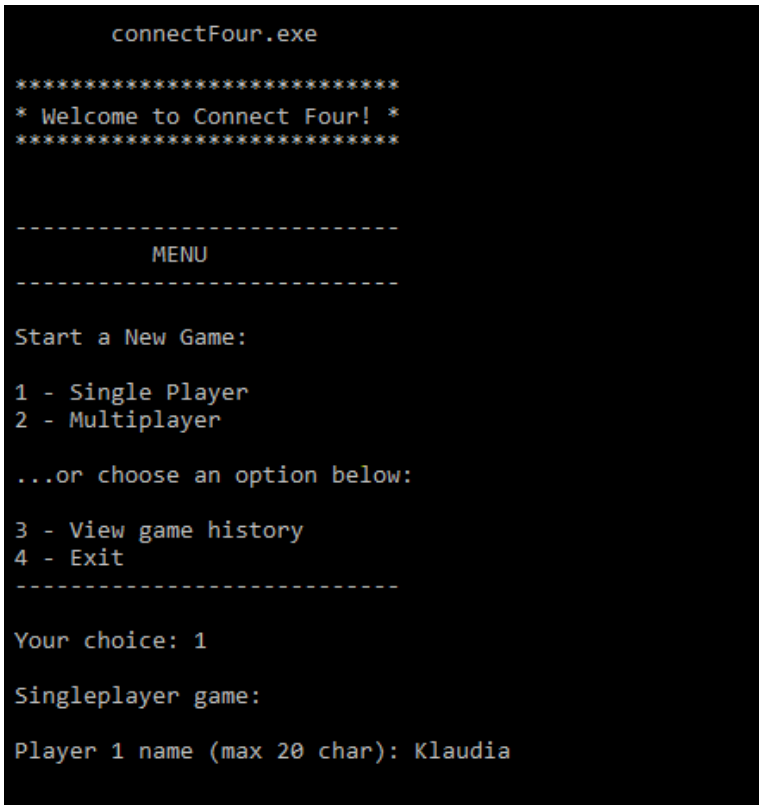
Things that could be improved include single player mode could have difficulty settings. For now, the "computer" only uses randomly generated moves which can rarely be challenging. Another thing is that the board size is fixed at 6x7 – Connect Four original size. It would be good to make it adjustable for the user.

References

Wikipedia 2021, *Connect Four*, viewed on 22 March 2021, https://en.wikipedia.org/wiki/Connect_Four

Appendices

Appendix A - Screenshot of the Text User Interface designed for Connect Four



Appendix B – Connect Four application Flowchart