Connect Four Report

Valeri Vladimirov
SET08122
BEng Software Engineering
Edinburgh Napier University
40399682@live.napier.ac.uk

1. Introduction

Connect Four is a two-player connection game where players take turns dropping a disc from the top into a seven-column, six-row vertically suspended grid. The winner is the first player to form a horizontal, diagonal or vertical line of four same color disks (*Wikipedia)*. This report will explain how I have implemented the game in C language, all of the functionality I have added and the data structures and algorithms I have used. Some of the features talked about in this report include: creating and displaying the game board, checking for a winner (diagonally, horizontally and vertically), placing a token on the board, removing one or more tokens from the board (multiple undo feature), redo feature if the user is not happy with the changes, saving replays of the games so they can be automatically replayed and also various game modes (normal, competitive, versus the computer, pop out, pop 10 and 5-in-a-row).

2. Design

2.1 Displaying and creating the game board

To create the game board, I have created a Board structure which has a row, column and a token (X, O or -), using this structure I create a 2D array (*2D Arrays)* '6x7' in size, which I initialize with dashes using my "board_create" method. The reason I have chosen to use a 2D array for this implementation is because it allows easy and fast access to all the positions on the game board, which would improve the time and functionality all of my other methods. An example of the benefits a 2D array provides would be my "board_display" method in which by only using 2 for loops I display the whole board (*Image 1*).

2.2 Placing tokens and winner checking

The "place_token" (*Image 2)* and vertical, horizontal and diagonal checkers (*Image 3)*, and also all of the variations of these methods, are again great examples of how easy the 2D array makes everything. As previously explained only with a few for loops I can access all of the positions on the board and add tokens or check for a win. I have implemented my "place_token" method to return the column on which the token has been placed. Combining this with a stack data structure (*Stack)* I push every column number of a token placed, which helps me with my undo functionality. The reason I have chosen a stack for this implementation is because I can easily peek and pop the top of the stack to see where the last token was placed and more importantly because it is a lot faster than iterating through an array. More on this in the next section.

2.3 Undo and redo functionality

After placing a token (in Normal Connect Four or Pop 10 game modes only) the player will be asked whether they want to undo their move (*Image 4*). If they want to undo, they will be further asked if only one move or multiple moves, which can be right back to the initial state of the game.

Implementing only one undo was fairly easy using a stack. After placing a token its column number gets pushed to the top of the stack and if the player decides to undo their move, I peek the value, find the last token that has been placed for the given column and swap it with a dash. Afterwards remove the value from the stack. When adding a redo feature (*Image 5)*, I added another temporary stack where after I undo a move, I push the column value to the temporary stack. If the player decides that he is not happy with the change, I peek the value from the temporary stack, push it to the original and place the token in the given column as if no changes have been made (*Image 6)*.

Implementing multiple undo's had almost the same logic as only one undo. I ask the user how many times he wants to undo. Afterwards, I again peek all the values and pop them from the stack using a for loop and swap the tokens with dashes. Adding all the values of columns to a temporary stack if the user wants to redo the changes. If they want to redo, again as stated above, I peek the values from the temporary stack, push to the original one and place the tokens back to their places. Stack came in very handy here, because when popping from the original and pushing to the temporary they get placed in the exact order I needed them to make the redo. Rather than spending time iterating or searching I just push and pop from stacks.

2.4 Replays functionality

For the implementation of replays (*Image 9)*, I have used both an array and a 2D array, as well as the stack from the undo functionality. After each game I pop all the values from the stack to a temporary array. Since the values were added from the last move to the first, I decided to also create a 2D array where I can store each game and all of the column values for it in the correct order after getting the values in the correct order. A simple for loop made all of that possible.

When replaying a game, a user is asked for a number of the game he wants to see and afterwards accessing the game and the given values for it was made easy by using the 2D array and iterating. In addition to that I have used the C language function Sleep to further improve the experience of the user when replaying a game.

2.5 Game modes

I have implemented various game modes (*Image 7*) to improve the experience of the player.

1. **Normal Connect Four** is the default play of the game where players try to connect 4 tokens and win the game. Undo and redo functionality are supported, and also all of the games can be replayed.

2. **Competitive Connect Four** is similar to normal however it is like a competition and players are not allowed to undo their moves. Replays are supported.

3. **Versus bot** is for players that want to practice against the computer. I have implemented easy and hard difficulty. The difference between them is that when on easy the bot will place a token anywhere on the board, whereas on hard it will place it near the players token, either on top of it or on the sides. Undo is not allowed, replays are supported.

4. **Pop Out** is a game mode where two players have the choice to either place a token or remove a token from the board. Undo is not allowed; replays are not supported.

5. **Pop 10** is a game mode where the players have to fill the whole lower row before moving on to the next one. Undo is allowed and replays are supported

6. **5-in-a-Row** is a game mode (*Image 8*) which is a bit different from the other ones. The board is different '6x9', having alternatingly placed tokens on 1 column on each side. This time players have to have 5 tokens connecting to win. In this implementation I have created new checking and placing methods similar to the original ones but for 5 tokens. Undo is not allowed; replays are not supported.

3. Critical Evaluation

While evaluating the game I have created I realized that there are both excellently working and not that great working features.

Some of the features that work well are my checking methods, placing and removing the tokens (undo functionality). In my opinion this is because of the choice of using 2D arrays and stacks. They made it very easy and fast to retrieve positions, which helped me a lot with the implementation of the game.

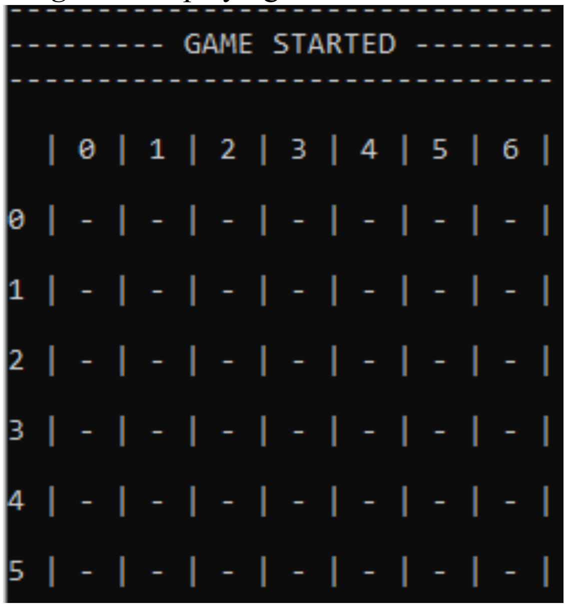
What I think could be improved and is not working as great is probably the replays functionality. Even though it does the job, and it is good enough it could probably be further improved by using a Queue (*Queue)* or even a simple Array. At the moment I add items to the stack and when I pop them and add to an array, they are not in the correct order, so I have to reverse it. If instead of doing that I added the column numbers to an Array and iterate through it or add them to a Queue which is a "first in first out" data structure I would have had faster access to the player moves from the start and easily added them to the 2D array of games.

4. References

Wikipedia: https://en.wikipedia.org/wiki/Connect_Four
Stack: https://www.geeksforgeeks.org/stack-data-structure-introduction-program/
2D Arrays: https://beginnersbook.com/2014/01/2d-arrays-in-c-example/
Queue: https://www.geeksforgeeks.org/queue-data-structure/

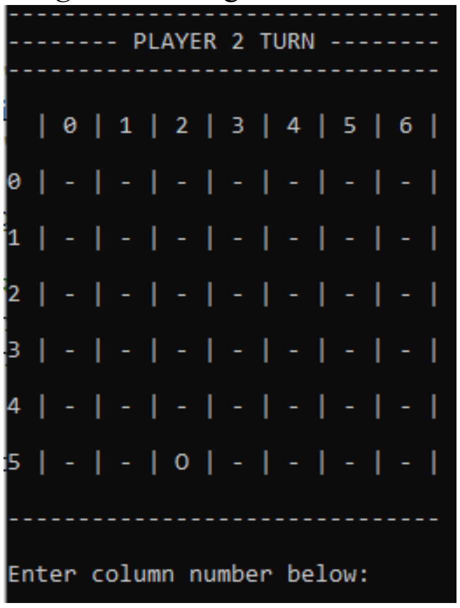
5. Appendices

Image 1 – Displaying the board



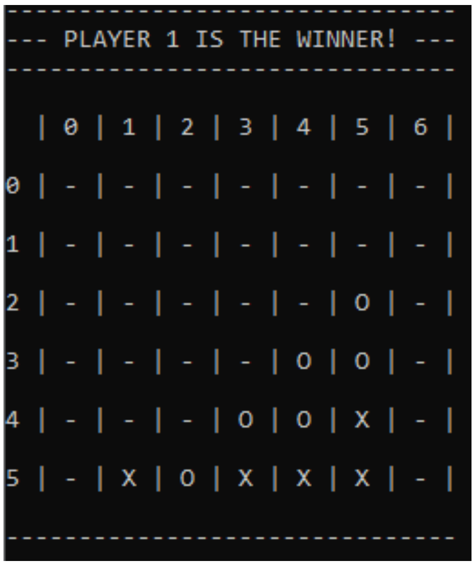
```
-------------------------------
--------- GAME STARTED --------
-------------------------------

  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
0 | - | - | - | - | - | - | - |
1 | - | - | - | - | - | - | - |
2 | - | - | - | - | - | - | - |
3 | - | - | - | - | - | - | - |
4 | - | - | - | - | - | - | - |
5 | - | - | - | - | - | - | - |
```

Image 2 – Placing a token



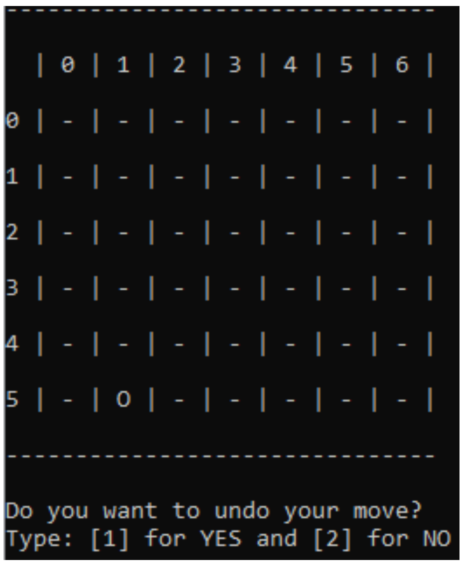
```
-------------------------------
-------- PLAYER 2 TURN --------
-------------------------------

  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
0 | - | - | - | - | - | - | - |
1 | - | - | - | - | - | - | - |
2 | - | - | - | - | - | - | - |
3 | - | - | - | - | - | - | - |
4 | - | - | - | - | - | - | - |
5 | - | - | O | - | - | - | - |

-------------------------------
Enter column number below:
```

Image 3 – Checking for a winner



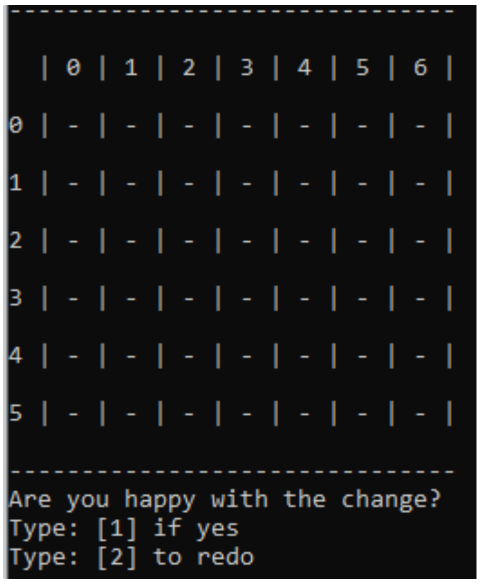
```
-------------------------------
--- PLAYER 1 IS THE WINNER! ---
-------------------------------

  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
0 | - | - | - | - | - | - | - |
1 | - | - | - | - | - | - | - |
2 | - | - | - | - | - | O | - |
3 | - | - | - | - | O | O | - |
4 | - | - | - | O | O | X | - |
5 | - | X | O | X | X | X | - |

-------------------------------
```

Image 4 – Undo Functionality



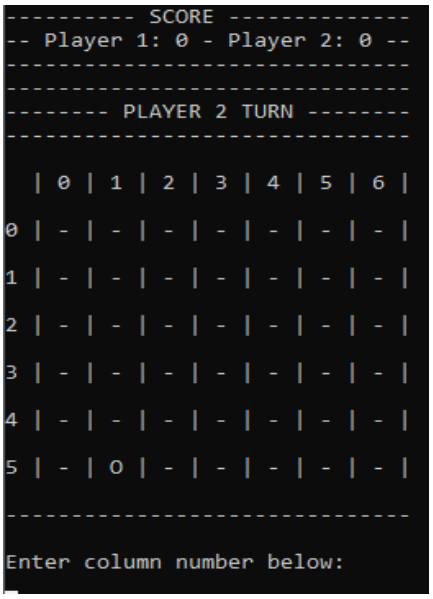
```
-------------------------------

  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
0 | - | - | - | - | - | - | - |
1 | - | - | - | - | - | - | - |
2 | - | - | - | - | - | - | - |
3 | - | - | - | - | - | - | - |
4 | - | - | - | - | - | - | - |
5 | - | O | - | - | - | - | - |

-------------------------------
Do you want to undo your move?
Type: [1] for YES and [2] for NO
```

Image 5 – Asking to redo the change



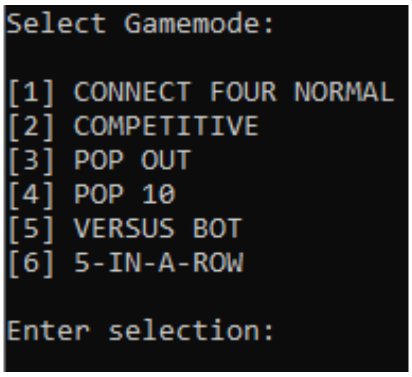
```
-------------------------------

  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

0 | - | - | - | - | - | - | - |

1 | - | - | - | - | - | - | - |

2 | - | - | - | - | - | - | - |

3 | - | - | - | - | - | - | - |

4 | - | - | - | - | - | - | - |

5 | - | - | - | - | - | - | - |

-------------------------------
Are you happy with the change?
Type: [1] if yes
Type: [2] to redo
```

Image 6 – After redo and undo



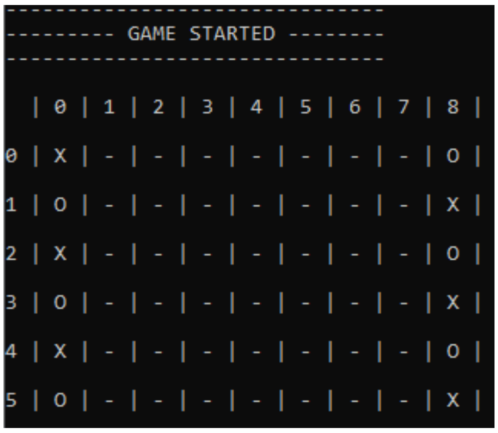
```
---------- SCORE --------------
-- Player 1: 0 - Player 2: 0 --
-------------------------------
-------------------------------
-------- PLAYER 2 TURN --------
-------------------------------

  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

0 | - | - | - | - | - | - | - |

1 | - | - | - | - | - | - | - |

2 | - | - | - | - | - | - | - |

3 | - | - | - | - | - | - | - |

4 | - | - | - | - | - | - | - |

5 | - | O | - | - | - | - | - |

-------------------------------

Enter column number below:
```

Image 7 – Game modes



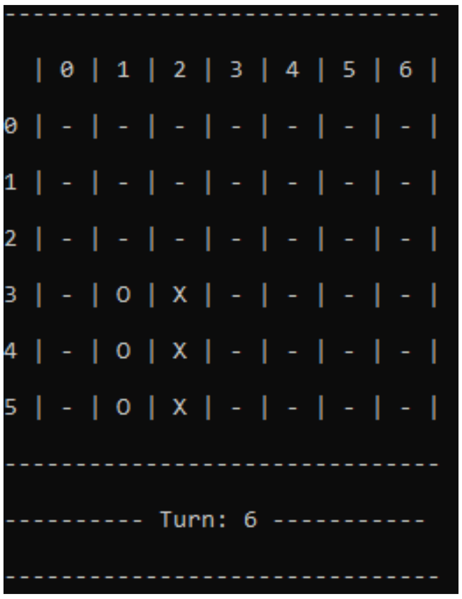
```
Select Gamemode:

[1] CONNECT FOUR NORMAL
[2] COMPETITIVE
[3] POP OUT
[4] POP 10
[5] VERSUS BOT
[6] 5-IN-A-ROW

Enter selection:
```

Image 8 – 5-in-a-Row game mode



```
-------------------------------
--------- GAME STARTED --------
-------------------------------

  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

0 | X | - | - | - | - | - | - | - | O |

1 | O | - | - | - | - | - | - | - | X |

2 | X | - | - | - | - | - | - | - | O |

3 | O | - | - | - | - | - | - | - | X |

4 | X | - | - | - | - | - | - | - | O |

5 | O | - | - | - | - | - | - | - | X |
```

Image 9 – Replay functionality



```
-------------------------------

  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

0 | - | - | - | - | - | - | - |

1 | - | - | - | - | - | - | - |

2 | - | - | - | - | - | - | - |

3 | - | O | X | - | - | - | - |

4 | - | O | X | - | - | - | - |

5 | - | O | X | - | - | - | - |

-------------------------------

---------- Turn: 6 -----------

-------------------------------
```