Mobile Applications Development Coursework Assignment Using Knowledge of Algorithms and Data Structures to Develop a Tic-Tac-Toe Game

Aaron Campbell

Contents

1. Introduction 3
2. Design 4
2. Enhancements 5
2. Critical Evaluation 6
2. Personal Evaluation 7
2. References 8

1. Introduction

The specific objective of this coursework is to implement a Tic-Tac-Toe game with an emphasis on the particular algorithms and data structures used within the program.

Tic-tac-toe is a paper-and-pencil for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. [1]

There are several elements that will have to be represented to have a base tic-tac-toe game, namely: the game board, players, the pieces i.e. 'X' and 'O', moves and positions of the pieces when a valid move is made.

The way that a base tic-tac-toe game would work is fairly simple – the program would print the initial board showing the empty board squares. Player 1 would then make their first move, followed by Player 2 and so on until the game is resolved. After every move the program would check that the move is is valid and check the board against win conditions.

Checking for a valid move would also be fairly simple, the program would simply check the player's choice against the current board. If the board square the player has chosen is currently empty, then the move can be made. If the board square is taken, a message should be printed to the user and the player counter should be decremented so that the current player can make a choice again.

Checking for win conditions, as mentioned earlier, would be performed after either player makes a move. This function would check for all valid win conditions i.e if Square 1, Square 2 and Square 3 match this is a win, and so on. A value should be returned on a win so that the game can be stopped inside the function that gets the player moves.

While this would be sufficient to have the functionality of a basic tic-tac-toe game, there are some extra features that could be added to improve upon it. A replay feature could be added to allow the user to see the sequence of moves of a previous game. This could be dependent on the games played since the program was run or the results of each game could be made persistent and written to a file.

Another useful feature to add would be an undo/redo feature. This would give the user the chance to take away their move and return the game to the immediate previous state. This would be a very useful functionality to implement as the user is liable to make a mistake, and as tic-tac-toe is such a short game, any mistake could prove to be costly to the end result. Having an undo feature would also be much more efficient than making the user confirm their every move.

Another feature that could be implemented is to have an automated player so that the program implements both Player vs. Player and Player vs. AI. The AI could have varying degrees of complexity; in its most basic form it would simply select a random legal move. Through more robust algorithms, such as Minimax, it is possible to create an unbeatable tic-tac-toe AI.

2. Design

Taking into account all the elements that had to be represented in the program, I decided to implement my game board first. The obvious way to implement the board is to break the board down into its squares and represent these squares as an array of chars. The reason I opted on using an array over chars over an array on integers is that you have the ability to hold both the squares the the user's move in one array. I initially planned on using a two-dimensional array to represent the board, as a board square is simply a 3x3 grid. After some consideration I found some problems with this approach – firstly, having the user enter an (X, Y) co-ordinate rather than a single number could cause some confusion when it comes to entering a move. Secondly, it is arguably more simple to run loops through a 1D array. Thirdly, 1D arrays are faster and smaller. [2]

I have a function called checkWinConditions which returns an int. This function simply checks whether the current values of the board squares would represent a win for either player. If any combination has resulted in a win, the function will return the value of 1. If the game has resulted in a draw, the function will return the value of 0. Else, if neither is true i.e. the game is still to continue, the function returns the value of -1.

I also have a function called getMove which returns an int. Inside of this I have represented the player as an int. The player is initialised as 1, i, choice. 1 represents the fact that Player 1 moves first, choice is taken in from the user's input using scanf. The modulo operator is used to set player to iterate between 1 and 2 i.e. Player 1 and Player 2.

I decided to represent my pieces as a char. This is assigned using a conditional statement - piece = (player == 1) ? 'X' : 'O'. This means that if the current player is Player 1, set the value of piece to X. Otherwise, set the value of piece to O. This means that piece will iterate between X and O, as player iterates between 1 and 2.

These are contained within a do…while loop which prints the board and assigns the variables as such. It then runs a series of if statements to check that the user has made a valid move i..e check that the square the user is trying to occupy is empty. If the square is empty, the value of the chosen board square is set to the value of piece. If the square is full, the user is told to move another move and the value of player is decremented so that the current player gets to move again. After each valid move, i is set to the checkWinConditions function. While i is -1, the board is printed and the loop continues. If i is 1, the Player who won is congratulated on their win. This is performed by printing a statement and decrementing the player variable by one, as it is incremented by one after every move is made. Else, the game is a draw.

In order to implement the undo feature I used a Stack. Perhaps the most familiar example of a stack in everyday life is the pile of trays in a cafeteria, where newly washed trays are placed on the top of the pile and customers remove trays also from the top of the pile. [3] The stack is adapted from Lab 3 [4] and utilises both the push and pop functions. Any time the user makes a valid move, the position is pushed to the stack. Whenever they wish to undo their move this item is popped from the stack and the player is decremented so that the player can make another move.

3. Enhancements

With respect to the base tic-tac-toe game I have implemented, there are several enhancements that could be made to make it a better game.

The most obvious enhancement that could be made is that a redo functionality could be added to the program, in addition to the undo feature that currently exists. This would require the item being undone to be held somewhere so that it can be pushed back onto the top of the stack when the user decided to redo. This would be useful feature as it would give the user more optionality when it comes to undoing their mistakes.

Another enhancement that could be made is to add a replay feature, as mentioned introduction. This feature would likely require a queue or a linked list and would store the sequence of moves.

A major enhancement that could be made is to have some form of AI working. As mentioned earlier, the AI could have varying degrees of complexity. The simplest form of AI would simply choose a random valid move. While this would qualify as a working AI, it means the computer is liable to choose an ‘incorrect’ move i.e. opting not to block the player’s winning move, or not recognising that it is one move away from a winning condition.

Using the Minimax algorithm, it possible to create an unbeatable AI in tic-tac-toe. Minimax looks steps ahead of the current game conditions and assigns values to to “end game conditions” i..e 10 for a win, -10 for a loss, 0 for a draw. The key to the Minimax algorithm is a back and forth between the two players, where the player whose “turn it is” desires to pick the move with the maximum score. In turn, the scores for each of the available moves are determined by the opposing player deciding which of its available moves has the minimum score. [5]

One way to improve on the code that already exists is to make it more efficient. As it stands the program currently runs through a long list of if statements when checking win conditions, getting the player moves and when popping the item off the top of the stack when undoing. This could likely be turned into some kind of loop for efficiency. The printBoard function could also likely be made more efficient.

4. Critical Evaluation

Overall I feel as though the core features of tic-tac-toe work well in the program. The user makes a move and the program checks that this move is valid i.e. the move has not already been made. Once this has been checked, the program checks the win conditions to see if it was a winning move. The move is then pushed onto the top of the stack so that the user can undo it if they wish to do so. The game then iterates between the players until win conditions are met or the match has resulted in a draw. The way the game works is an accurate reflection of how tic-tac-toe should work and so is a fairly good part of the program.

I also feel as though undo works well. The user has the ability to enter and undo their moves, and undo can be used multiple times and the win/draw conditions still work as expected. It implements a stack which allows you to easily pop the top item off of the stack in an efficient manner. Overall I feel as though this is the best feature of the program as it allows the user more optionality when playing the game.

An area where I feel the program lacks is that it has no additional features outside of undo. While having a replay feature and a working AI would be good additions, not having a redo feature in particular is a weak part of the program as there is a working undo feature. This means that the game itself is rather simple.

Another part of the program that I feel is weak is that there is a lot of inefficient code. For example, the program runs through a long list of if/else if statements when getting the player's input in order for the stack to work properly. While I tried to create a loop when popping an item off the stack, I ran into errors until I created if statements to cover every possible move the user would wish to undo. This is very inefficient and could be improved upon to require a much smaller number of lines.

5. Personal Evaluation

Overall, I am fairly disappointed with the final result. While I feel my tic-tac-toe was acceptable and meets the base requirements in the specification, the lack of additional features and originality brings it down.

The main challenge I faced was trying to get my undo working. While I had the architecture set up for the stack early, I had a number of problems getting it to work correctly. I was able to check that the items were being added to the stack early on by implementing a size() function. This allowed me to see the size of the stack when the game was completed - if the game was won in 7 moves, the stack had 7 items. This showed that the stack was being filled, however when printing the item that was being pushed to to the stack I found that I was pushing the wrong item. I had a similar problem with popping off the top of the stack, I found that it was printing arbitrary values rather than the value of the position that should be popped.

I feel that the biggest thing I learned was how to problem solve better. When trying to fix my undo feature I broke the code down into chunks and individually tested each feature to see where exactly it was going wrong. I managed to work out what I needed to set the value of the boardSquare to when the top item of the stack is popped and wrote some working, but inefficient as mentioned earlier, code.

On a practical level I learned how to implement the stack from the labs. I feel as though I have a much greater understanding of stacks and how they can be implemented and I enjoyed working through the errors before getting it to work as intended.

In conclusion, I'm mostly disappointed that I didn't manage to include any additional features other than the undo. I spent a lot of time making sure that my undo was working and I feel that this held me back when it came to doing work on the additional features. If I had more time I feel like I would try to implement the redo feature and, although possibly unrealistic, try to get a working AI with minimax.

6. References

[1] Wikipedia, “Tic-tac-toe,” [Online]. Available: https://en.wikipedia.org/wiki/Tic-tac-toe.

[2] Pixelchemist. [Online]. Available: https://stackoverflow.com/questions/17259877/1d-or-2d-array-whats-faster.

[3] I. T. Adamson, Data Structures and Algorithms: A First Course, 1996.

[4] S. Wells, “Lab 3 - Data Structures #1,” [Online].

[5] n. s. building, “Tic Tac Toe: Understanding the Minimax Algorithm,” [Online]. Available: https://www.neverstopbuilding.com/blog/minimax.