Tic Tac Toe

Niall Wishart
40273691@napier.ac.uk
Edinburgh Napier University - set08122

1 Introduction

The purpose of the program was to demonstrate the the understanding of practice and theory of Algorithms and data structures. To do this, I created a text-based, command line Tic-Tac-Toe using the C language. The important features of this program include a easy to understand method of playing, while also having more in depth features such as undoing and redoing moves made, as well as allowing users to view and reload past games played.

2 Design

I began by designing the whole program, looking at what needed to be done and how it would be done, as well as splitting the plan into requirements and optional to improve my optimisation of what to prioritise within the given time. I decided that it would be best to focus on the bear minimal, that being a functioning Tic-Tac-Toe game, then to put work into all of the other features at the end. In doing this, I created two initial plans for the design, one that was bare bones and comprised of only the information for the board, as well as the board set up and win conditions, while the other had the general design for the linked lists needed to continually store the games information.

The initial part to look at, was the game board as that is the a firm base in which the rest of the program would run through. In order to do this, I created and array and added the numbers 1 through 9 into it in numerical order, which would serve as each square for Tic-Tac-Toe. From there I created a Display function, in which I formatted a general board, where each square was filled with one of the contents of the array. During the game itself, for example, when the X uses enters the number 1, the 0 element of the array (number 1) would be replaced with X in the array. After each input, the board re-creates itself, meaning that the board will update with the new contents of the array. Whenever a new game is started, or a replay is selected, the array will refresh itself, putting the original numbers back into their spots.

When the general creation for the board was completed, the next task was to implement the actual game logic that would allow the game to function as intended. When the user enters a number, it will initially be validated, first to make sure it is in fact a number and between 1 and 9. Once the input is validated, the number input will be deducted by 1, in order for it to be placed into the right place, and if the player is set as player 1, then an X will be put in its place in the array, otherwise an O will replace it. The draw function works from a turn count by adding 1 to a counter after every placement. After 9 turns, the game will return as a draw as every space will be taken. A check win function was then put in place, which checks the contents in the array match the same contents 3 times in a row, which will then return a true or false depending on the outcome. From there, if the game is still not over, the player number will add one if it is player 1, or remove 1 if its player 2 in order to move onto the next player, then the cycle will continue until a player wins, where a the player will be congratulated, and an ID for the game will be given, which I added in preparation for the planned replay function.

From there, I began to work on the undo and redo functions. In order to create the functions, I decided to use linked lists. In order to to this effectively, I created a linked list which will dictate the current state of the game, which would actively update as the game goes on, where each node consists of the current set of data used for the squares array which dictate what is on the board, by also storing it in a 9 slot char array, while also having a state which is used to look between the nodes. When the user requests to revert a move, it will take in the current game list, from there it will check if there is a current node, or previous node, and if not it then an error will be displayed. When a valid undo has been selected, it will set the squares array to the data stored in the previous node, putting the board back to the previous round. The redo function is basically the inverse of this, where in it will check if there is a node in front of it in the stack, then fill the squares array for the board with with the data array of the next node. For the undo and redo functions to be accessible, I added a menu feature when selecting the next placement, wherein if the user enters 0, they will be asked if they wish to undo, redo or quit the current match. Again, the input is validated to ensure it meets the requirements, then will run either the undo or redo function depending on the input.

After that the implementation of the replay function began. To do this, I created another linked list, which would store the games ID, as well as a node which would take in the other linked list in order to see every state of the game. For this to work, I created a main menu at the beginning of the program opening, which will ask the user if they wish to play a game, or replay a previous game. When replay is selected, it will prompt the user for the ID of the game requested, where it will then search through the linked list to find the ID, then get the related game information. It will then point back to the starting node of the list, where it will then change the display on screen one by one, showing when each move, redo and undo was done, until the end node is reached. A design that I intended that unfortunately didn't make it into the final build was persistence for the replay function, where the games

would be exported to a file, then read in on start up meaning that a user would be able to replay a game that was played before this instance of the application was opened.

3 Enhancements

I feel as though the current program works as intended and is rather well implemented in terms of stability and features, but there are some enhancements that could be made. One of which, is instead of players just replaying their games, allowing the users to fully load a game, where their game state will be loaded and they will be able to finish their game if uncompleted. In order to do this, all I would really need to do is to put take the program out of just the replay feature and push it into the actual game functions. The problem with this, would be importing the whole past state list into a new current game list, in order for the redo and undo functions to still work. As well as this, I also could have added, as mentioned before, persistence for the replays, where users would be able to close the game, and when they re-open the app they could use the same IDs and replay those games. A big problem with this would be remembering all of the IDs as a player, for example, its un-likely a person will remember the ID of a game they played last week.

Another idea I had was to implement a simple AI functionality into the game. I toyed with the idea in the beginning, to at the very least add a randomised AI, that would look at the available positions on the board and enter one of them at random. Unfortunately when I came to implement this, I realised that realistically there wouldn't be much fun in this for players, as potentially the AI would play stupid and not really pose a challenge for the player. In terms of adding proper AI that looks at the board properly and decides where the best place to go, in the end I decided that due to time constraints I wouldn't be able to complete this feature to a high enough degree.

As well as these enhancements, I could also have added the option to change the board size, allowing the player to select the size of the board, for example 3x3 or 4x4. This would allow the game to be more open, giving the players more choice in how they wish to play. It would also remove repetition for concurrent players.

There were also some more simple, but quality of life changes I could have made. For example, allowing users to enter their own names as apposed to player 1 and 2, which would allow the users to have some uniqueness and customisation, while also making it easier for users to remember which player they are if, for example, they use the replay function, but don't remember who was what player. Another feature would be a leader boards, where you could, instead of enter an ID, look through all the possible games, see the names of the players and who won. this would make the replay feature much neater and easier to use, while also making it so people can count their wins against each other, without having to watch the full game replay to see who won. Along with the IDs and player names, it could also have a date and time paired with it, which would help with the issue stated earlier, as users would be more likely to remember that the game they wanted to replay was from a specific date, rather than remembering its ID.

4 Critical Evaluation

I feel as though the program as a whole generally works as intended, to a fairly high degree. The game comes with several features to allow the players to play at ease, with enough control over their games to have fun playing. There is validation through, to ensure that players wont be able to break the game in any way. Below I have added an image that shows the user attempting to select a square which has been taken already, with the error message displayed. Along with this validation, I added validation for the undo/ redo functions, so that the user knows where there are no more possible moves to go forward or back on, as well as telling the user when an id for a game replay is invalid. I also added an error message if the game could not be found but the ID does match, but I am yet to encounter this as an issue.

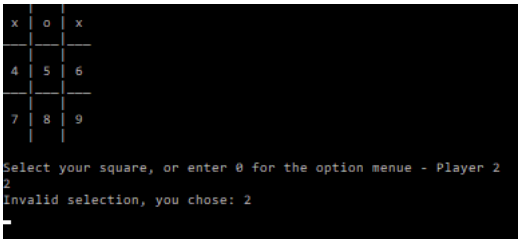

Figure 1: Validation for user selecting box already chosen

Originally I had the board display as dashes instead of the number to signify that the piece was empty. I then realised that it was not very user friendly, making the user count out which square they want to use, or them accidentally using the wrong number. Giving each square its own number made it alot more user friendly, being one of those simple things that can make a big difference. Although when doing this, it made resetting the board an issue, as the array wouldn't let me either reset the array or allow me to change the numbers one by one due to the numbers actually being chars. Eventually I managed to create a reset array, which I would then set the main array to that reset array at the beginning of each game before the board is created.

The re-do and undo features both work as intended, where in they will allow the user to continuously re-do moves until they are at the very beginning of the game, and much similarly, the redo will allow the users to forward again back to before any undos were done. There is also error messages that are displayed when a user makes an invalid undo/ redo that will tell them there is no more options. Undoing and redoing will put the played who was in play that turn to take their turn again, reverting the current player along with the board. Another neat piece I should have added was allowing the user to select how far they would like to go back, so instead of going into the options then typing redo 5 times, they could simply redo 5 then the program would automatically take the game back to 5 turns ago.

The whole visual design of the game is rather basic, there isn't a whole lot of things you can do while keeping it to command line, but I do feel as though the general aesthetic could be cleaned up. I avoided clutter on screen, making

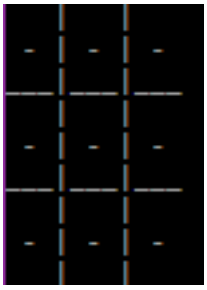

Figure 2: Old layout

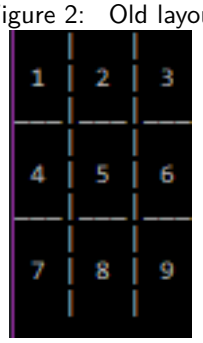

Figure 3: New Layout

sure that the screen is whip ed on every option chosen, and before any game or replay ends, I allow the user to stay on the screen until an input is pressed, which will then whip then move them onto the next screen.

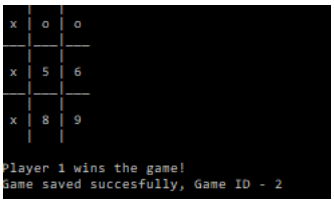

Figure 4: Winner screen

The lack of persistence for the replay feature could be argued that it sort of reduces the whole point in a replay feature, as realistically if people want quit their current game and save it to play again later, they would close the application down, which would just negate their save. Despite this, the replay functionality is still functioning in its basis, and giving I had more time, persistence would be very doable, as the hard part is done.

5 Personal Evaluation

I feel as though I learned more about linked lists from my past experiences with them in C, with this coursework giving a more hands on and practical use for them, unlike what I have done in the past. In the beginning I had some issues deciding how to connect the link lists together, where I store the current game information in one list, and would need to use that data again for the replay. I didn't want to just put one list into another, as that would be a great waste of storage and would be rather inefficient, which is why I looked into a better way to relate these two lists.

One of the biggest challenges I faced was when I changed the empty box dashes to their respective number, as it forced me to re-do my whole win check, as the original way I done it was if there were 3 boxes in a row that weren't dashes, meaning that the game ended as a win as soon as it started. I had to work around this by changed the validation from not being dashes, to the elements being either a X or an O.

Personally I feel as though I performed quite well. I managed to create a fully functioning game that provides plenty of features including redo/ undo, as well as allowing users to view previous games played. There are many additional features that could be added, but most of them are more optional additional to make the game more exciting, as apposed to being standards expected from a game like this.