Edinburgh Napier UNIVERSITY

Algorithms and Data Structures Coursework

Marcos Ramirez Aceves
40284094@live.napier.ac.uk
Edinburgh Napier University - Algorithms and Data Structures (SET08122)

1 Introduction

The aim of this project is to create an excellent functional Tic Tac Toe game, that will run and compile from the commandline, using the C programming language. This will be achieved through the implementation of different data structures and algorithms which will be based on the knowledge gained from the lectures and from some self-directed learning. The features implemented in the project are the following:

- Functional Tic Tac Toe game.
- Functional Connect 4 game.
- Recording and replaying games.
- Undo functionality.
- Redo functionality.
- Simple AI for Tic Tac Toe.

This report will cover in more detail the features mentioned, the algorithms and data structures used for those features as well as the decisions behind the implementation of the ones selected instead of others, the description of further features and improvement of the ones already implemented, and finally, a critical and a personal evaluation of the project.

2 Design

2.1 Functional Tic Tac Toe and Connect 4

For the first two features, the data structures and algorithms used were designed taking into account that their implementation had to work for both game types. In the case of the data structures, two were implemented, one that represents a player and one that represents the game board.

The player struct is form by a simple char to represent the player's piece, a boolean to know if the player is being controlled by an AI and a dynamic array for the name instead of a static one, with the purpose of avoid wasting memory.

About the game board struct, its most important part is the actual grid, and for that reason, several data structures where taking into account before selecting one to represent it. The first one considered was a static array, the main problem with this approach was that having a static array would have meant either having two almost identical structs, with the only difference being the size of the array, or having a struct in which the size of the array would have been the size of the Connect 4 game board and in that case, memory would have been wasted for Tic Tac Toe. For those reasons, this approach was discarded.

The next data structures considered were a one dimensional and two dimensional dynamic arrays. Based on the performance, one dimensional arrays are faster and more efficient than two dimensional arrays. In addition, one dimensional arrays are simpler to implement and the accessing of its elements is also easier than with a two dimensional array. However, when considering the algorithms that needed to be implemented using the grid, having a two dimensional array would lead to a more natural organization and interaction, Tic Tac Toe and Connect 4 are basically grids form by rows and columns, and to an easier design of the algorithms. For that reason, after consideration, the data structure selected was a two dimensional dynamic array.

When it comes to the algorithms, one algorithm was used to check if there were any available positions left and three algorithms were used to check whether a player has won the game or not. The first algorithm was quite simple as the only thing to be done was to traverse the array looking for a position that was empty. In the case of the winner check algorithm, its implementation was more complex and in consequence, it was divided in two main parts. The first part is also divided in two smaller parts that consist in checking firstly vertically and secondly horizontally if there were three or four pieces of the same type one after the other. The second part consisted in a diagonal check and for optimization, two different diagonal approaches were made, one for Tic Tac Toe and one for Connect 4. The Tic Tac Toe one is quite simple as only the main diagonal from right to left and from left to right needs to be checked. In the case of Connect 4, there are more diagonals and the checking is broken down into four parts, these parts being the left half and the right half of the diagonals that go from left to right and, the left half and the right half of the diagonals that go from right to left.

2.2 Recording and replaying games

In the case of the recording and replaying games implementation, the data structures used were a circular deque and a struct called Record.

The first approach for this part was to use a queue instead of a circular deque, the reason for that was that queues provide a natural way of interaction in which its elements are processed in order of arrival. This means that, the positions in which the players placed their pieces can be added (enqueue) to the tail of the queue and later on, those moves will be displayed in the same order they were done by removing (dequeue) them from the head of the queue. Furthermore, the game and the algorithms are designed in a way in which the player with the crosses is always first, and therefore, the queue only needed to hold the movements as no more information was needed to replay them later.

However, after implementing the Undo functionality, one flaw was spotted. This flaw consisted in that, after undoing a movement, the element at the tail of the queue had to be removed.

To solve the issue mentioned above, the queue was changed to a deque, allowing, in this way, to remove elements from both ends but with the restriction of only allowing elements to be added at the tail. Apart from that, it was also considered the possibility of having to shift elements in the deque to avoid having gaps when implementing further functionality, therefore, instead of using a normal deque, a circular one was implemented as in this way the shifting of the elements to fill the empty gaps is avoided.

About the Record struct used, its main purpose was to store the data about a specific game (the player names and corresponding pieces, a description about the match, the game type and the deque with the moves) when reading the file where all the recorded games were stored and retrieve it later for the displaying of the game.

Finally, in relation with some aspects of the algorithms used for this section, the recordings are created by reading the data from a file taking into account the order in which the elements are placed in the file and, as mentioned at the beginning of this section, the displaying of the game movements takes into account that the crosses player goes always first, making things a bit easier.

2.3 Undo and Redo functionality

For the undo and redo functionality, the data structure implemented was a stack, more concretely, a Undo stack and a Redo stack.

The selection of the stack as the data structure for these features was almost straightforward, and the reason behind this decision was that stacks follow a LIFO (Last In First Out) format which fits perfectly with the purpose of the features. Besides that, the only other data structure considered was a deque, however, for simplicity, it was discarded.

That fitness, mentioned above, refers to the simplicity achieved in the implementation by the usage of stacks, and it consists in the following: each time a player does a movement, the position is pushed into the Undo stack, later on, when a player wants to undo a previous movement, the last position added is popped out of this stack, the element at the tail of the deque that keeps track of all the movements is removed, the game board is updated with an empty element at the popped position and finally, the undone position is pushed into the Redo stack. In the case of the player performing a redo action, the process will be the same but the other way around. Apart from that, the Redo stack is emptied each time a player does a new movement.

2.4 AI in Tic Tac Toe

The AI implemented for Tic Tac Toe is very simple, it only consists in an algorithm that is going to select a random number from one to nine until it selects one that can be mapped to an available position in the grid. Besides that, the reason why this random AI is only used in Tic Tac Toe and not in Connect 4 is that, in Connect 4, the game board is too big. Because of that, making an AI that selects a random position will make things too easy for the player and, in addition, if the game board gets populated almost completely, genereting a random position until getting a correct one would be slow and inefficient

3 Enhancements

The first feature to be added if more time was available would be a more suitable and complex AI for the Connect 4 game type, this would be achieved by using a MiniMax algorithm or an Alpha-Beta pruning algorithm. Both algorithms make use of a search tree.

The way in which Minimax works is the following: taking into account that MAX and MIN refers to the players, starting with MAX, it is going to look at the children of the nodes that it has available and is going to try to select the node that maximizes the score of the children, while MIN is going to do the same but trying to minimize instead of maximize. In this way, both players basically try to maximize the worst case scenario for the other. Alpha-Beta pruning is mostly the same as Minimax but searches through less elements by performing pruning when it finds a worse children in the current node than in a previous one. In this project, one of the players would be a person and the other the AI.

Another two features to be added would be: display the records in a list form allowing the player to select one using a number, and allow the player to remove recordings from the file.

The fourth feature to be added if more time was available would be to include a new game mode with a third player. This game mode would use the Connect 4 game board, as it provides a big game space, and will have the winning condition of a Tic Tac Toe game. The combination of these two elements with the addition of a third player would create an interesting gameplay.

The final feature to be added would be to include an option for the user to select an AI vs AI match.

Apart from that, what could be improved in the current project would be the flexibility that the players have when selecting the action to do and to allow the players to move between the "menus" in a easier way, however, as the game works in the commandline, these are difficult tasks.

4 Critical Evaluation

Reflecting on my own work, all the features implemented work well and as expected, nonetheless, there are some considerations to have in mind.

As stated in a previous section, the approach for the game board grid was to use a two dimensional array. In terms of performance, a more optimal solution would have been to use a one dimensional array instead, but taking into account the design of the algorithms and that the game boards are basically grids, the two dimensional option seemed more sensible.

In the case of storing and replaying records feature, as stated in section 2.2, an issue was spotted in the first implementation after the undo feature was added to the game. This caused a delayed in the project as the implementation had to be changed from a queue to a circular deque, which could have been avoided if the Undo functionality would have been taken into account in this implementation.

About the AI implemented for Tic Tac Toe, in average, the time needed by the algorithm to select a position in the grid increases as the number of positions available decreases. As can be seen in figure 1, after each turn, the algorithm takes more time to select a position. It is quite certain to say that, if this algorithm was applied for a larger game board, like the Connect 4 one, there would be issues in the performance, but taking into account the game board size in Tic Tac Toe and the simplicity of the implementation, this approach is good enough for the Tic Tac Toe game mode. In the testing, ten games were played and the AI played as the second player.

Finally, the time needed by the winner check algorithm in Tic Tac Toe and Connect 4 is practically the same as can be seen in figure 2 and 3. This means that the approach chosen of using two different diagonal checks solves the problem in an appropiate way. In the testing, the amount of moves done by the players was maximized.

5 Personal Evaluation

Reflecting on my learning, the basics of the different data structures used in this project were already clear to me, however, when it came to actually implement them from scratch, I had to look for some documentation about them to make sure that I was making no mistakes. This helped me to gain a deeper understanding about how those data structures internally work and how they can be implemented in different ways.

During the development of the project I faced some challenges. The first one was already mentioned in section 2.1, the winner check. The reason why this was a challenge was that I did not want to hard code the positions to do the checking, which would have been the easiest thing to do, as this is bad practice. Therefore, I had to come up with an algorithm to check all the possible cases in which a player could win the match. For the horizontal and vertical cases, this was relatively easy, as the same algorithm could be applied for both game types. However, when it came to the diagonal checking, the number of diagonals in Tic Tac Toe and Connect 4 was very different, therefore, it would have been difficult to design a single algorithm that would have worked for both and, in the case of Tic Tac Toe, a single algorithm involved an unnecessary amount of checks. For those reasons, I decided to design two different algorithms for each game mode. The diagonal check for Tic Tac Toe was very easy, however, in Connect 4 it was a bit more complex. After considering the best way to proceed, I decided to divide the diagonal check of Connect 4 in four parts, which made the implementation of the algorithm much easier.

The second challenge I faced was the implementation of the Recording and Replaying feature. As mentioned in section 2.2, after adding the Undo feature, this one started to work unexpectedly, the reason being the element at the tail not being removed. After consideration, the decision I made was to change the queue for a deque.

Finally, the third challenge I faced was the implementation of the circular deque itself. After performing some research and finding code examples about the deque, I still had some doubts and trouble about understanding how it internally worked. The approach I took was to look at a simpler implementation with just a circular queue and use it in comparison with the circular deque to achieve a better understanding.

Generally, I feel that my performance in this project was quite good. I would have specially liked to have added a more complex AI for the Connect 4 game mode, however, due to some issues during the development that took longer to solve than expected and due to other projects taking place parallelly with this one, I did not have enough time.

References

[1] Circular Deque:
https://www.geeksforgeeks.org/implementation-deque-using-circular-array/

[2] Circular Queue:
https://www.geeksforgeeks.org/circular-queue-set-1-introduction-array-implementation/

[3] Stack:
https://www.geeksforgeeks.org/stack-data-structure-introduction-program/

[4] Queue:
https://www.tutorialspoint.com/data_structures_algorithms/queue_program_in_c.htm

[5] 1D vs 2D arrays:
http://pointlessdiversions.blogspot.com/2012/05/1d-vs-2d-arrays-performance-reality.html

Appendices

A Test Data

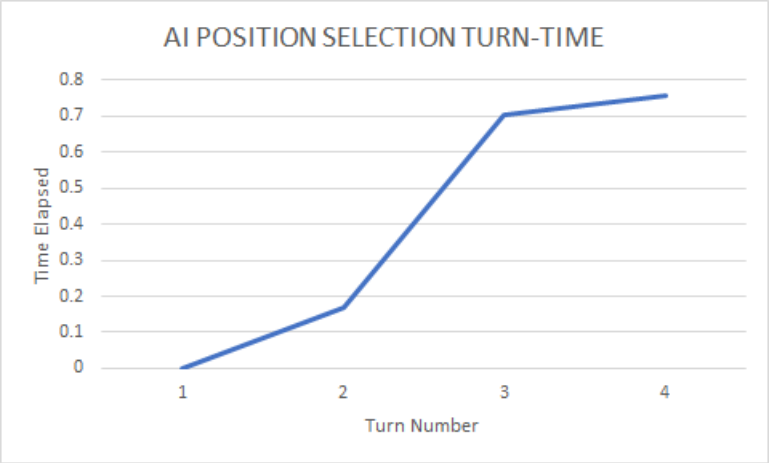

Figure 1: AI position selection algorithm test data

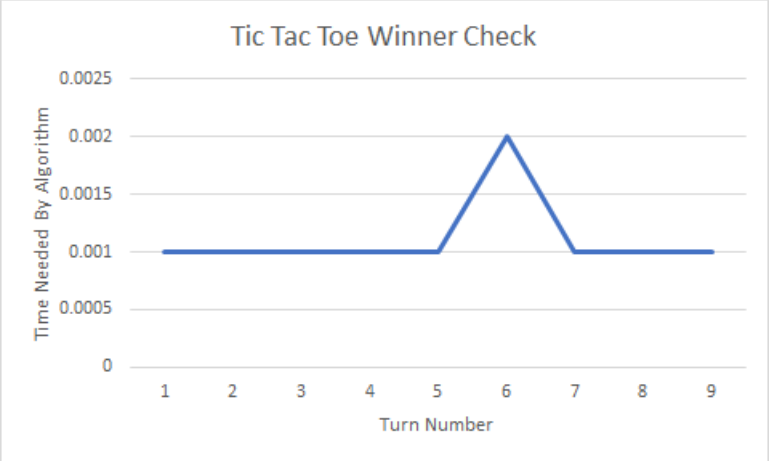

Figure 2: Winner check algorithm in Tic Tac Toe

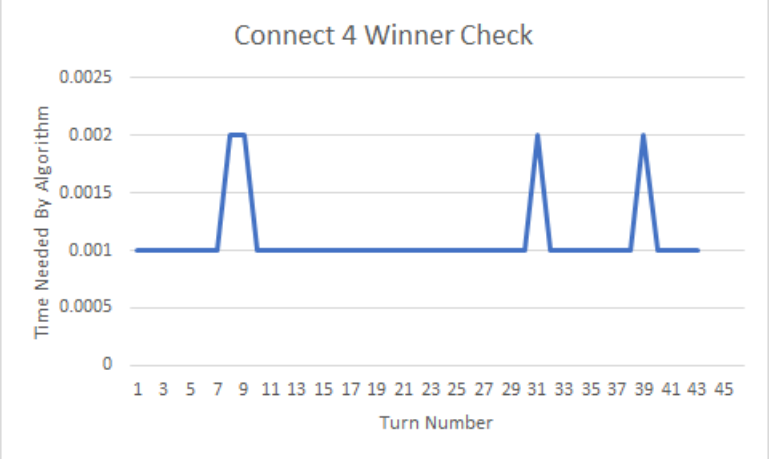

Figure 3: Winner check algorithm in Connect 4