Coursework Report

Namo Najem
40313888@napier.ac.uk
23rd March 2019
Algorithms and data structures -
Edinburgh Napier University

1 Introduction

1.1 Tick Tack Toe in C

Tick Tack Toe is a simple game with a set number of variables at play, however many things change while a game is being played certain things will always be the same for example the max number of turns that can be taken or the size of the board. Because of this and C's regulation of memory it is a great language of choice to implement this game in an efficient and meaningful way.

1.2 Goals

C is a lower level language than the ones more commonly used these days and this level of control can be a blessing as much as it can be a curse. The increased control over the memory locations of data allows for greater understanding of exactly what is happening with the code. This level of control is what allows the game, written properly, to require less memory than possible with other languages. However with this increased there are more things to go wrong and issues can be harder to track down and elminate.

2 Design

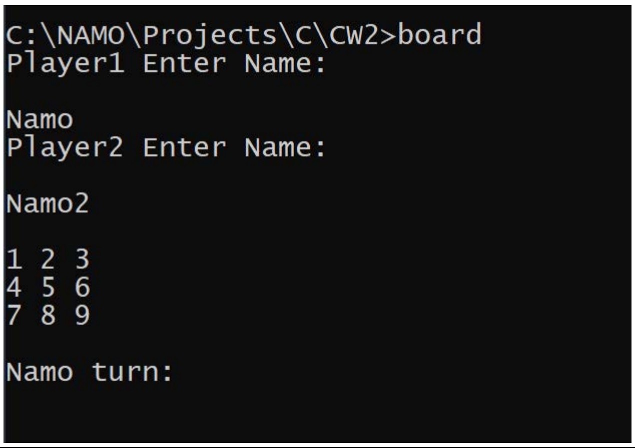
2.1 Data Structures

The data structure of choice for this assignment includes a CHAR array to keep track of the board in it's current state and a Doubly Linked List with that hold an Integer as it's data. The Doubly Linked list isn't the most efficient structure that can be implemented but it's very close so it was chosen. An integer variable was chosen to store a game move as a single integer matches the max number of choices exactly. Pointing to the next and previous nodes is how the game keeps track of the order of turns to allow for undo capabilities.

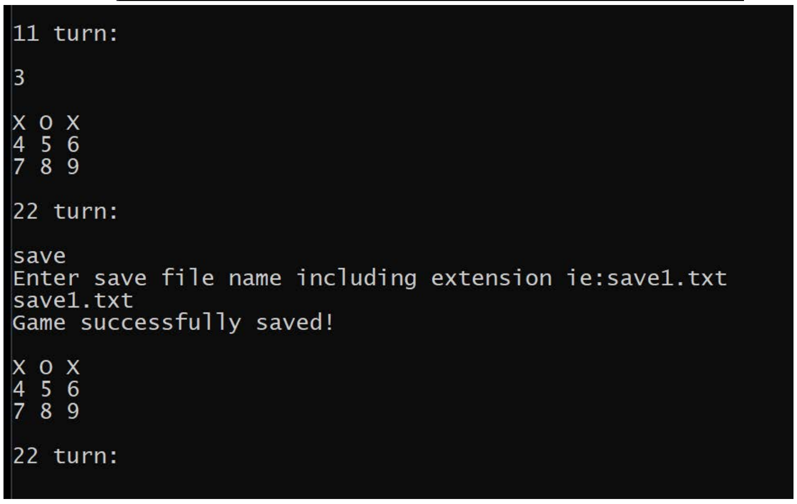
2.2 Game Design

The first things to happen when the game is run are prompts for the users to input their names, once the names have been entered a new game will automatically commence. A while loop starts running until the game is won by one of the players. Within this while loop we have the necesarry iterations to switch back and forth between turns. This is acheived by switching a flag every time a valid move and turn is made successfully.

Figure 1.1 Starting point for game



```
C:\NAMO\Projects\C\CW2>board
Player1 Enter Name:

Namo
Player2 Enter Name:

Namo2

1 2 3
4 5 6
7 8 9

Namo turn:
```



```
11 turn:

3

X O X
4 5 6
7 8 9

22 turn:

save
Enter save file name including extension ie:save1.txt
save1.txt
Game successfully saved!

X O X
4 5 6
7 8 9

22 turn:
```

Figure 1.2 Demonstration of a save

2.3 Input Flags

In order to be able to load and save the game at any moment the same input used to take turns are also used to check whether or not the user has typed on of: undo, save, load. When one of these flags are hit the while loop doesn't register that input as a turn and it remains the turn of that player. The player is then prompted for further input to save and load. If undo is requested checks are made to see if there is a move to be undone.

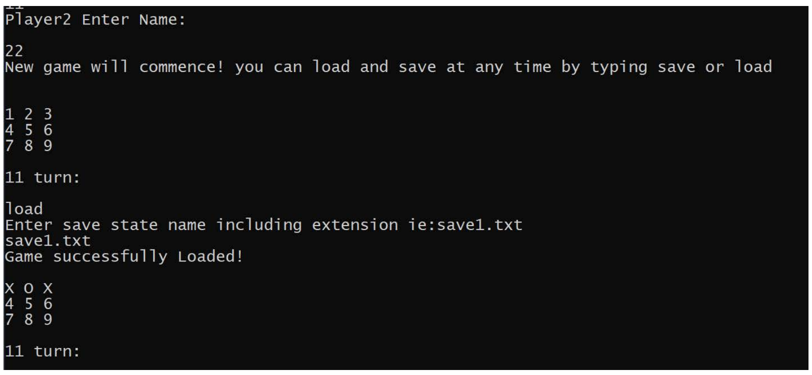
2.4 Save and Load

Save and load functionality has been implemented in a very basic fashion. If a player chooses to save they are prompted to enter a name for their save file. The application then itterates over the linked list and saves what turns are made in order asa single string consisting of integers marking the correct place on the board. This string is then saved to the first line of a text file in the directory of the application. When a player decides to load a saved game, this save file is taken and the load function appends the moves to the linked list in the order it was recorded as a string. The linked list is emptied before we can append the moves from the save file otherwise moves made before a load will break the game.

2.5 Undo

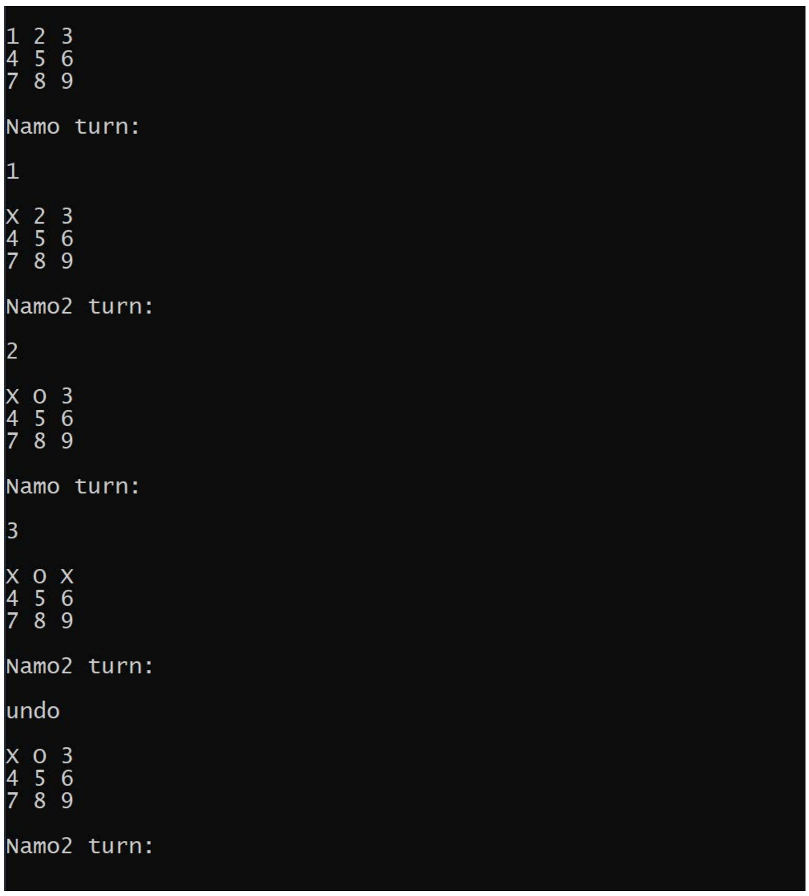
The undo function is responsible for undoing a move. When a move wants to be undone first some temporary nodes are created that point to the same memory locations as our list. This needs to be done so that our list can always point at the head of our linked list, this is neccessary because most other functions in our application depend on list being head when it is passed into parameters. A while loop then itterates until it reaches the second to last node. Once here the next nodes are freed and list remains the head of our linked list.

Figure 2.1 Loading a game



```
Player2 Enter Name:

22
New game will commence! you can load and save at any time by typing save or load


1 2 3
4 5 6
7 8 9

11 turn:

load
Enter save state name including extension ie:save1.txt
save1.txt
Game successfully Loaded!

X O X
4 5 6
7 8 9

11 turn:
```

Figure 2.2 Undo being done



```
1 2 3
4 5 6
7 8 9

Namo turn:

1

X 2 3
4 5 6
7 8 9

Namo2 turn:

2

X O 3
4 5 6
7 8 9

Namo turn:

3

X O X
4 5 6
7 8 9

Namo2 turn:

undo

X O 3
4 5 6
7 8 9

Namo2 turn:
```

2.6 Append

The append used for the linked list is identical to that from the practicals. If there is no first node it is created and populated with data, if a node exists a new node is initialized and the next node for the previous node is made to point at the new one.

2.7 Empty List

Part of what makes loading possible is emptying the list and freeing the memory of whatever turns have been made prior to a load, if this isn't done then all the moves from both instances of the game will remain in the linked list and effectively break the game. The function itterates and frees every node as it goes along, using a temporary node to once again leave the list as the head of the linked list.

2.8 Print board

The print_board function has two functions. One is to take the board string that represents the current state of the boad and use a for loop and special indexing to print it in a 3x3 grid. The second function is to update the board string every time a change is made to the linked list.

3 Evaluation

3.1 Critical Evaluation

In todays age hardware has evolved to where there is enough overhead for developers to spread their wings and higher level and more object orientated languages will be chosen over these for certain projects but with languages like C higher levels of efficiency can be achived for certain usecases and therefore C remain the programming language of choice for lots of developers especially when it comes to embedded systems as it strikes a great balance between control and ease of use.

3.2 Personal Evaluation

I've not had this much fun with a coursework in ages. I beleive having a simple goal in mind for projects like this allows for strengths and weaknesses of a student to be uncovered easier than more difficult assignments as there's more time to reflect on doing things better than to just get it working. Most of my time bug fixing for this coursework was mistakes I made coming back to C after spending long with higher level languages. The complier would fail to show certain syntax errors so next time I'll probably invest time getting more than the basic visual studios compiler as it'll save me alot of time. I like figuring stuff out as i go along and bugs with memory allocation or simply C's tendancy to do certain things even if they are not perfect syntax can make for some interesting bugs.

3.3 Enhancements

Perhaps the biggest and most obvious enhancement to make is to have less happen in main method and more set methods to pass parameters into. This is a consequece of the bolt on as you go mindset that was used to build the application. functionally not much would change but itd do wonders for readability and