Algorithms and Data Structures Coursework Report:

Introduction:

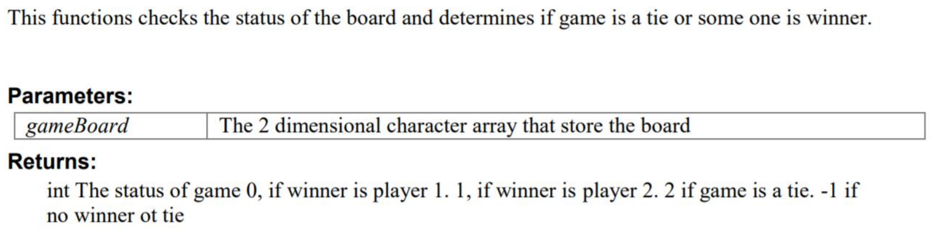
The objective of this coursework is to demonstrate my understanding of both theory and practice in relation to the content of the Algorithms and Data Structures module. My specific task is to implement a text-based Tic-Tac-Toe (Noughts & Crosses) game using the C programming language. The core goal of my implementation should focus on my choice of data structure and algorithms needed to implement the game. I must think carefully about which data structures and algorithms are appropriate to my solution.

Design:

My game of tic tac toe works via a game menu with a list of options listing from 1-6 that allows both players to perform several actions and makes it easier to implement the several features this game requires; e.g. redo, undo, record and make a move in general. I never really used a data structure other than arrays, but I did use different kinds of arrays (2d and 3d). Although I would have loved to flex my overall knowledge of the course and the multiple kinds of structs like stacks, records, queues, binary etc. I never thought it necessary to do so as the task was simplistic enough to rely on my knowledge of said structures. I mainly used 2d arrays to store the state of the board (see parameter gameBoard) which allowed me to implement loops and conditions to see if a player had won the game or if a specific action could be done. For more advanced actions like undo and redo I used 3d arrays (see parameter allMoves) which stores all the current game's moves and allows easy access to removing and re-adding moves that the players have inflicted upon the board and keeps a track of what number each move is so it can easily revert to the game's original state. The game also has an import and export function which uses both of the previously mentioned arrays as it requires them to recreate a specific game state and to play on from that point in the game.

List of features:

- "checkWinner"



This functions checks the status of the board and determines if game is a tie or some one is winner.

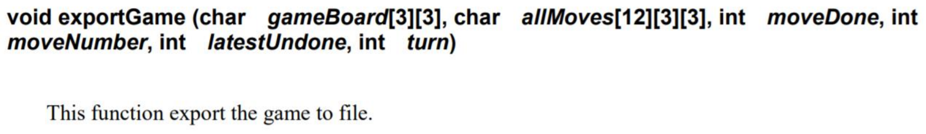
Parameters:

gameBoard	The 2 dimensional character array that store the board

Returns:

int The status of game 0, if winner is player 1. 1, if winner is player 2. 2 if game is a tie. -1 if no winner ot tie

- "exportGame"



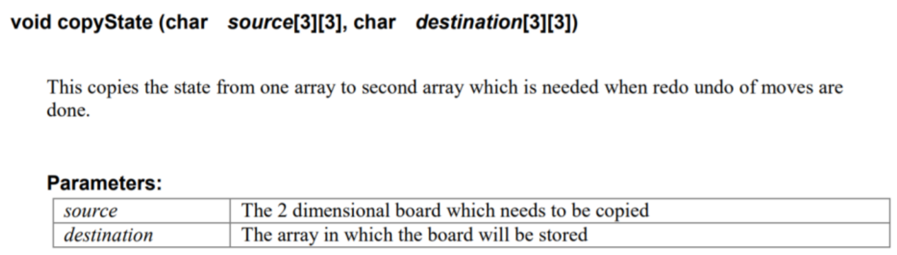
void exportGame (char *gameBoard*[3][3], char *allMoves*[12][3][3], int *moveDone*, int *moveNumber*, int *latestUndone*, int *turn*)

This function export the game to file.

Parameters:

gameBoard	The 2 dimensional character array that store the board values
allMoves	The 3 dimensional array that stores all of the boards state when a move is placed
moveNumber	The overall move number in the game
undoNumber	The number of the move when undone
Turn	The variable to detect the player's turn
moveDone	The variable which stores the total move
latestUndone	The variable which stores the latest undone move number

- “copyState”



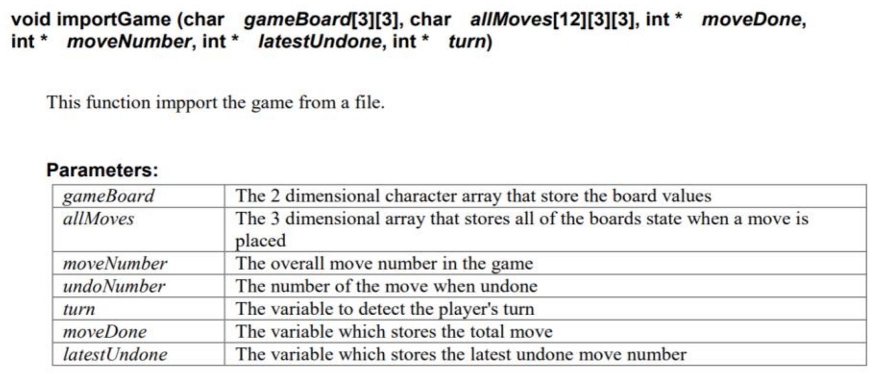
void copyState (char *source*[3][3], char *destination*[3][3])

This copies the state from one array to second array which is needed when redo undo of moves are done.

Parameters:

source	The 2 dimensional board which needs to be copied
destination	The array in which the board will be stored

- “importGame”



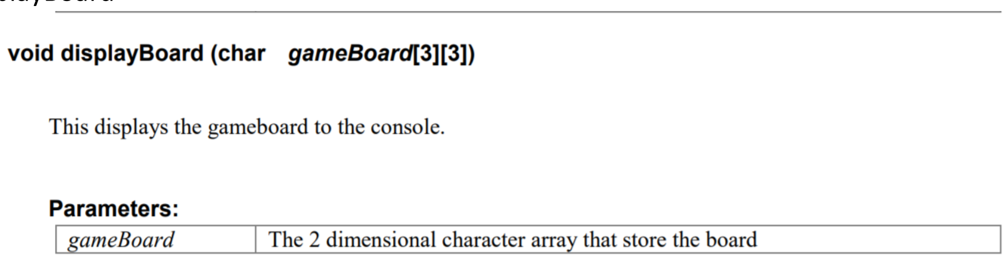
void importGame (char *gameBoard*[3][3], char *allMoves*[12][3][3], int * *moveDone*, int * *moveNumber*, int * *latestUndone*, int * *turn*)

This function impport the game from a file.

Parameters:

gameBoard	The 2 dimensional character array that store the board values
allMoves	The 3 dimensional array that stores all of the boards state when a move is placed
moveNumber	The overall move number in the game
undoNumber	The number of the move when undone
turn	The variable to detect the player's turn
moveDone	The variable which stores the total move
latestUndone	The variable which stores the latest undone move number

- “displayBoard”



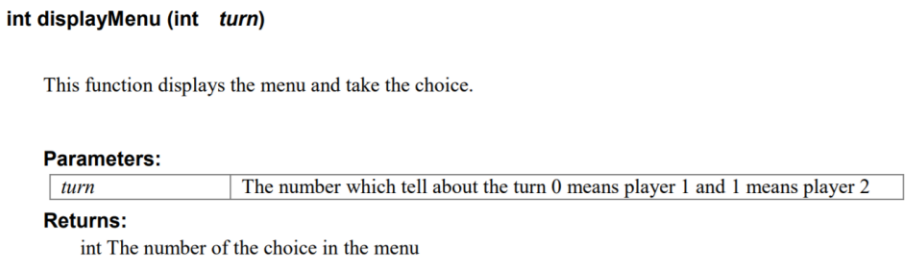
void displayBoard (char *gameBoard*[3][3])

This displays the gameboard to the console.

Parameters:

gameBoard	The 2 dimensional character array that store the board

- “displayMenu”



int displayMenu (int *turn*)

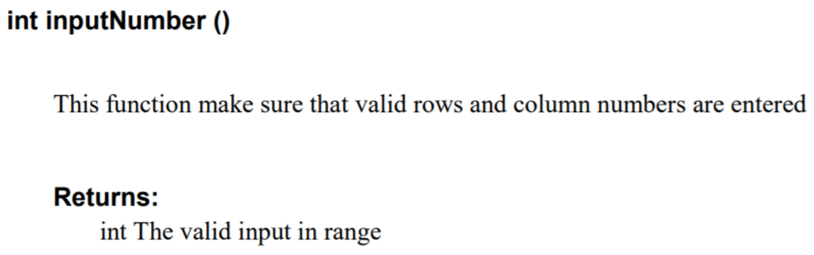
This function displays the menu and take the choice.

Parameters:

turn	The number which tell about the turn 0 means player 1 and 1 means player 2

Returns:

int The number of the choice in the menu



int inputNumber ()

This function make sure that valid rows and column numbers are entered

Returns:

int The valid input in range

- “inputNumber”

- “makeTurn”

  void makeTurn (char *gameBoard*[3][3], char *allMoves*[12][3][3], int *turn*, int * *moveNumber*)

  This function takes the block choice and update the gameBoard.

  Parameters:

gameBoard	The 2 dimensional character array that store the board
allMoves	The 3 dimensional array that stores all of the boards state when a move is done
turn	The variable that will tell about player's turn
moveNumber	The overall move number in the game passed as reference so changes reflect back in main

- “redoMove”

  int redoMove (char *gameBoard*[3][3], char *allMoves*[12][3][3], int *turn*, int * *moveNumber*, int * *undoNumber*)

  This function redo the moves that are undone.

  Parameters:

gameBoard	The 2 dimensional character array that store the board values
allMoves	The 3 dimensional array that stores all of the boards state when a move is placed
turn	The variable to detect the player's turn
moveNumber	The overall move number in the game
undoNumber	The number of the move when undone

  Returns:

  int Return 1 if successful and 0 if not successful

- “resetTheGame”

  void resetTheGame (char *gameBoard*[3][3])

  This functions resets the board and the game

  Parameters:

gameBoard	The 2 dimensional character array that store the board

- “undoMove”

  int undoMove (char *gameBoard*[3][3], char *allMoves*[12][3][3], int *turn*, int * *moveNumber*, int * *undoNumber*)

  This function undoes the game’s moves

  Parameters:

gameboard	The 2 dimensional character array that store the board values
allMoves	The 3 dimensional array that stores all of the boards state when a move is placed
turn	The variable to detect the player's turn
moveNumber	The overall move number in the game
undoNumber	The number of the move when undone

Returns: int Return 1 if successful and 0 if not successful

Enhancements:

If I had more time with project I would have probably attempted more features within the game. Including;

- The ability to have different sized game boards leading to different and more complex types of games.
- The option to play against an AI/the computer with varying levels of difficulty (easy, medium, hard)

Critical Evaluation:

- Feel the choosing of moves is perhaps a bit clunky where you have to choose a row and then a column when it could just be labeled from 1-9. Had a problem when first implementing the moves feature where the game would crash if you chose a specific row and column
- Like the menu approach as it clearly labels what the game can and can't do – easily modified and implemented. Originally just asked you to make a move without even mentioning you could redo or undo a possible move.
- Syntax and possible spelling errors are rife within the code making it perhaps look a bit unprofessional – perhaps a bit more time testing and playing would avoid this.

Personal Evaluation:

- C isn't as scary as originally thought, allows for a lot of wiggle room and a good understanding of the complexities and basics of the language allow for a lot implementation.
- Implementing actual structures from knowing how they work in theory into actual code was a bit daunting at first but luckily there's a lot of useful and in-depth information online which helps in a lot of specific circumstances.

- Feel like I performed under the circumstances of personal problems, the course and C language in general allows for a lot of personal learning and understanding which is easy to pick up.

Christopher Maxwell

40314579