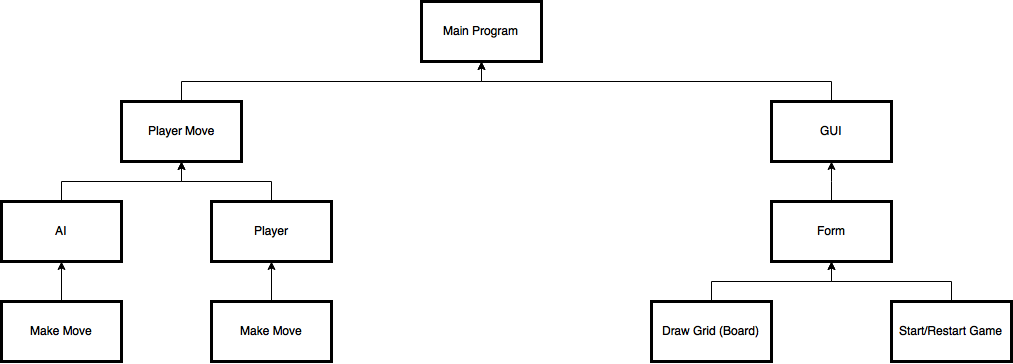
Documented Design

Structure

My project consists of five units:

* The User Interface unit (PCheckers)
* Board unit (UBoard)
* Move unit (UMove)
* Artificial Intelligence unit (UAI)
* Save/Load unit (USaveLoad)

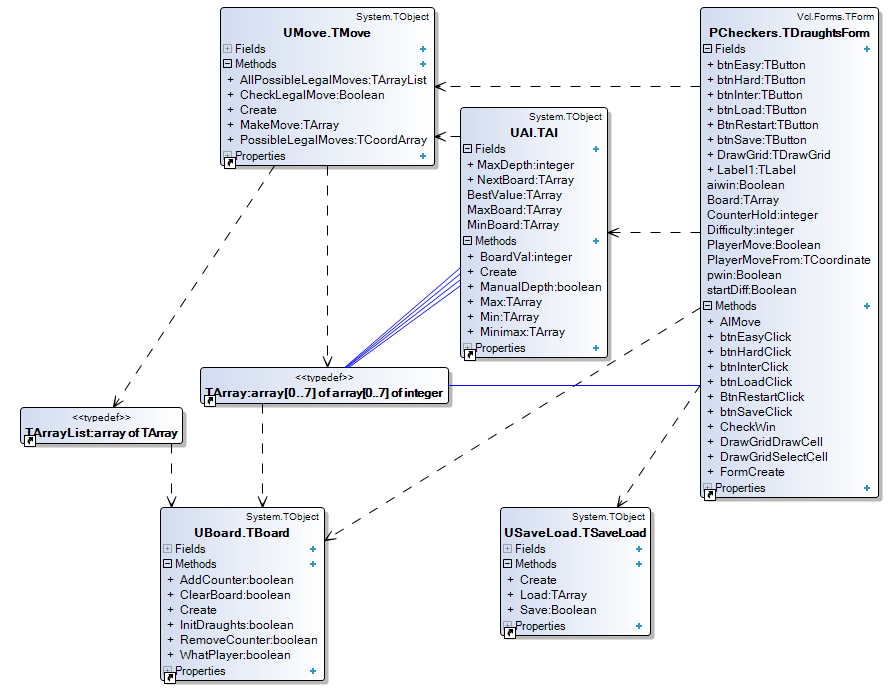
I began by planning my project, I started by creating a hierarchy chart that would help me plan how to implement each feature of the program defined by the objectives. As you can see in figure 2 below, I planned a general structure for the program:

  
Figure 1: Hierarchy chart[[1]](#footnote-1)

After getting the gist of the general structure I decided to go further into planning by creating a class diagram, shown in figure 2, which would help implement a robust structure that I could use in my technical solution. The class diagram would also help me avoid creating global variables, which are generally not preferred, as they remove locality and allow no access control.

Illustrated in figure 2, I planned for the User Interface Unit to use every unit, as it would be dealing with: AI turns which are handled by UAI; player turns which use UMove; generating the board which is handled by UBoard, and Saving/Loading a game which is handled by USaveLoad.

  
Figure 2: Class diagram at the start of the project[[2]](#footnote-2)

  
Figure 3: Class diagram at the end of the project[[3]](#footnote-3)

Components

The User Interface unit (PCheckers)  
  
The User Interface Unit is responsible for graphically displaying the program and handing the user interaction with the program. This unit interacts with all other units in the program, like an aggregator, which lets each unit communicate and transfer data with each other. The main functions within the unit are; CheckWin, DrawCell, DrawCellSelectCell and btnLoadClick/btnSaveClick.

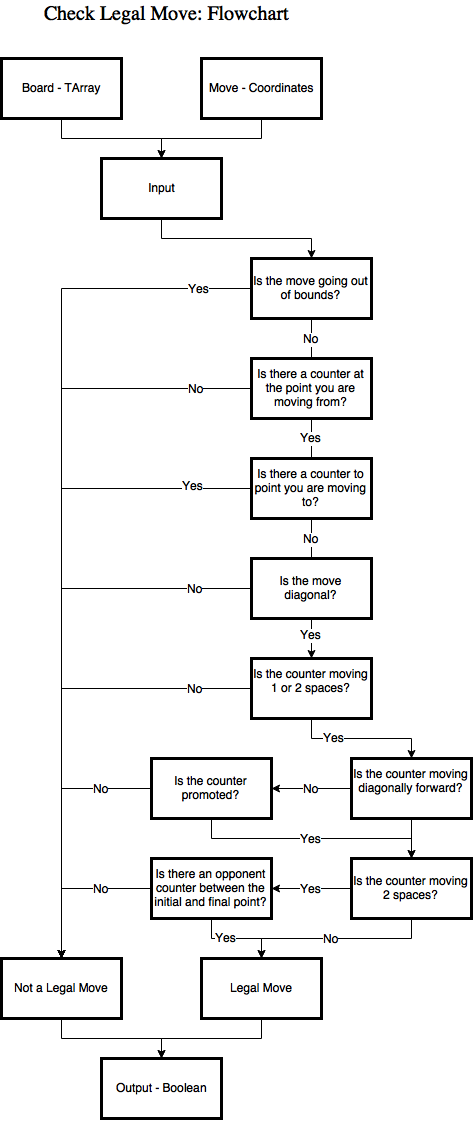
CheckWin alters the ‘pwin’ and ‘aiwin’ Boolean variables, that are used to communicate with other functions to determine the end of a game. DrawCell translates the values in the array to the board, which the player can interact with, by drawing and colour filling each cell with a counter. The user can interact with the graphical board by the function DrawCellSelectCell, which communicates with UAI and UMove, as the cell selection event indicates a player move and the start of the AI turn (at the end of the player move). BtnLoadClick and BtnSaveClick communicates with USaveLoad, to save and load the game.

Board unit (UBoard)  
  
The Board unit is responsible for initialising the draughts board and handling the counters in the board. The unit is independent of all the other units, it doesn’t use any of the other units, but it is used by UAI, UMove and the User Interface unit. The most important function in the board unit is InitDraughts.

InitDraughts initialises the board array, which represents the counters on the draughts board, by alternating counters and ‘snaking’, which is when the function populates the next row’s columns in reverse order. This results in a checker pattern, which ignores the middle two rows, that is ready for use by the User Interface unit and other units.

Move unit (UMove)  
  
The Move unit is responsible for changing the states of cells in the board array, to simulate a checker move; check if a counter move is legal and generate a list of legal moves a counter and a player can make. These features are implemented by these main functions: MakeMove, CheckLegalMove and PossibleLegalMoves.

MakeMove requires the input of two coordinates to remove the counter from the first coordinate and place it in the second coordinate, which correspond to cells in the board. However, you need to check if the move is legal before you can proceed with moving a counter, so CheckLegalMove returns if a move is legal by checking it against certain criteria. As shown in figure 4, the function has a multitude of conditions that a move must pass to be a legal move.

  
Figure 4: CheckLegalMove flowchart[[4]](#footnote-4)

Artificial Intelligence unit (UAI)  
  
The Artificial Intelligence unit is responsible for making the move for the artificial intelligence by using the Minimax algorithm and assigning values to each board. This unit is dependent on UBoard and UMove, as it needs UBoard for board representation and UMove to simulate counter moves. The most important function within UAI is Minimax, which is a recursive algorithm that allows the artificial intelligence to choose a move based on a ‘tree’ created by recursion that allows you to predict the value of the board, assuming the player is playing perfectly.

Minimax requires the input of the board array and the depth of the tree, because it needs the board to generate the next possible moves, by using UMove.AllPossibleMoves, and the depth allows you to indirectly change the difficulty of the artificial intelligence. A higher depth allows the AI to predict better outcomes as it can see if a branch, which looks like it can lead to a win or loss on a limited depth, leads to a better or worse outcome. The function returns an array for the next turn, which is used by the User Interface Unit to update the board.

Save/Load unit (USaveLoad)  
  
The Save/Load unit is responsible for saving and loading a board to a text file. The unit is dependent on UBoard, as it requires board representation to translate the board to a text file and vice versa. The functions that implement these features are USaveLoad.Save and USaveLoad.Load.

The save function requires the board array, AI difficulty and file name, which are retrieved from the User Interface Unit. The board array gets read by the nested FOR-loop in the function, to translate it to text, and the AI difficulty is appended to the end of the file, so that it can preserved the array as a save file. The file name is a string which is input by the user, through the User Interface unit, and the save function will either overwrite or create a file in that name.

The load function requires a file name and the ability to change the difficulty of the game, the file name is derived from the User Interface Unit by user input. The function will cycle through the file converting the text values to integers for use in the board array, and using the value of the difficulty appended at the end of the file to change the difficulty. The function returns the array, which the User Interface Unit assigns the board to.

Data  
  
In this section, I go through all the key variables I have used though my project, which are not global variable, as I have not used any global variables.

|  |  |  |
| --- | --- | --- |
| Key Variables used in the program | Variable Type | What it is used for |
| Board | TArray, an 8x8 array of integer declared in UBoard. | Used by the User Interface Unit and other global variables to store the positions of checkers on the board. |
| CBoard | TBoard, an object class declared in UBoard. | Used by the User Interface Unit to initialise the array, at the beginning of a game, also used to determine which players counter is at a position on the board. |
| CMove | TMove, an object class declared in UMove. | Used to move checkers on the board; check if a move is legal; and generate all legal moves for a player. |
| CAI | TAI, an object class declared in UAI. | Used to find the next move for the opponent. |
| CSaveLoad | TSaveLoad, an object class declared in USaveLoad. | Used by the User Interface Unit to save and load the draughts game. |
| NextBoard | TArray, an 8x8 array of integer declared in UBoard. | Used by the artificial intelligence and user interface units to communicate the next move of the AI. |
| PlayerMoveFrom | TCoordinate, an 1x2 array of integer declared in UMove. | Used by the User Interface Unit to store the position of the cell selected by the player, when it is the player’s move. |

TArray

0 . . . 7

0 . . . 7

0 . . . 7

TArray is an 8x8 grid which only allows integer values, it is one of the most important data structures in the project, as it is used for the board and the transfer of data between the major units.

0 . . . 7

TArrayList

TArrayList is an array of undefined length that stores TArray. It is mainly used by UMove and UAI to communicate possible moves.

0 . . . 7

0 . . . 7

0 . . . 7

0 . . . 7

0 . . . 7

0 . . . 7

0 . . . 7

0 . . . 7

0 . . . 7

0 . . . 7

0 . . . 7

0 . . . 7

0 . . . ∞

TCoordinate

TCoordinate is an array of length 2 that stores integers. It is mainly used by UMove and the User Interface Unit to for player and AI moves.

0 . . 1

TCoordArray

TCoordArray is an array of undefined length that stores TCoordinate. It is mainly used by UMove, to to store the multiple moves a counter can make.

0 . . . ∞

0 . . 1

0 . . 1

0 . . 1

TextFile

The pre-defined TextFile type is used by USaveLoad for saving the game to a text file and loading a text file to an array.

Algorithms

Minimax – UAI  
  
The minimax function takes in the board, finds all possible moves that are legal for the player, and then recursively finds all the possible legal moves from that board. This occurs until it hits maximum depth, then it compares all the boards at maximum depth to find the best board, which it bases its move on.  
  
Pseudocode:

function Minimax(board, depth, maxPlayer) {

if depth is not 0 then

listOfMoves ← allPossibleLegalMoves  
if maxPlayer then  
for I ← low(listOfMoves) to high(listOfMoves) do

BestValue ← -∞  
BestValue ← Max(BestValue, I)  
result ← BestValue

else

for I ← low(listOfMoves) to high(listOfMoves) do

BestValue ← ∞  
BestValue ← Min(BestValue, I)  
result ← BestValue

}

CheckLegalMove – UMove  
  
  
The CheckLegalMove function is a series of conditions a counter move has to abide by for it to be a legal move in draughts. Some of these conditions are: check if is it a diagonal move; and check if it doesn’t move out of bounds.  
  
Pseudocode:

function CheckLegalMove(board, aX, aY, bX, bY) {

result ← false

if (aX, aY) has counter then

if move from (aX, aY) to (bX, bY) is diagonal then

if distance((bX, bY), (aX, aY)) = 1 then

if counter at (aX, aY) moves in correct direction then

result ← true

else if counter at (aX, aY) is promoted

result ← true

else if distance((bX, bY), (aX, aY)) = 2 then

if midpoint((bX, bY), (aX, aY)) has opp. Counter then

if counter at (aX, aY) moves in correct direction then

result ← true

else if counter at (aX, aY) is promoted

result ← true

}

InitialiseArray – UBoard

The InitialiseArray function will set up the draughts board, by using nested for-loops and an alternating Boolean value that alternates the counters.

Pseudocode:

function InitialiseArray(board) {

tempBool ← false

for each row in board except rows 4 and 5 do

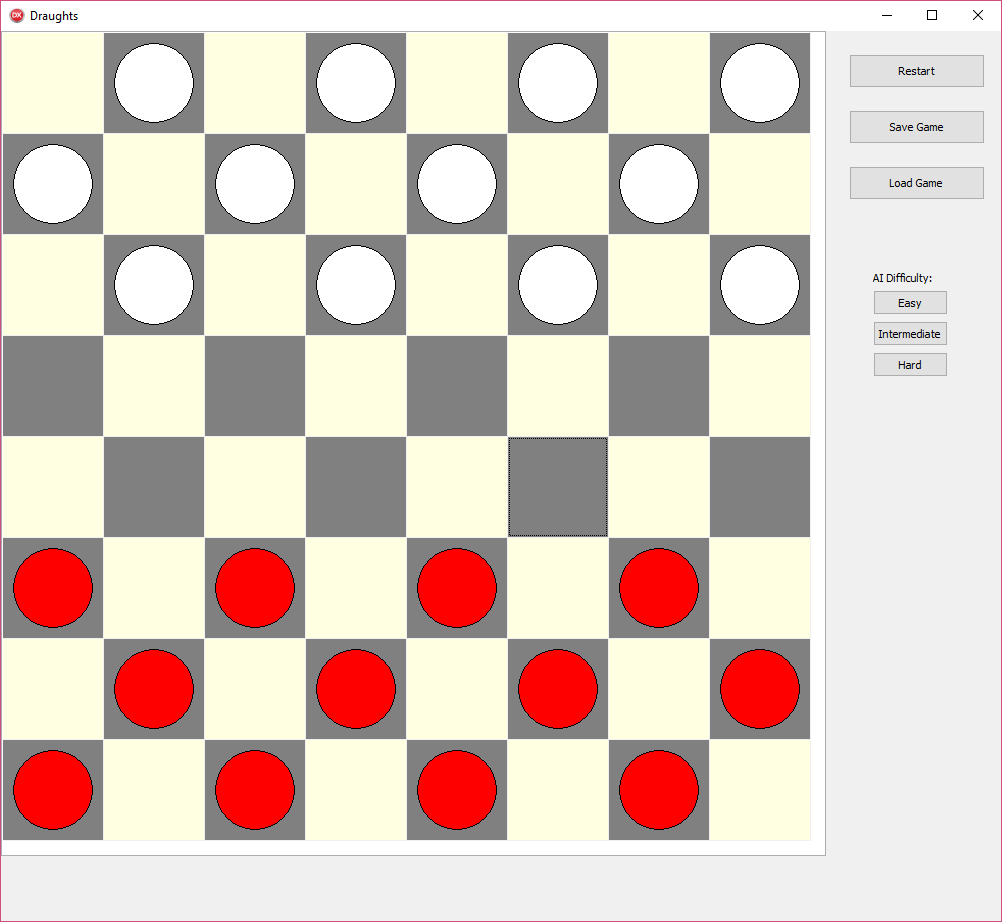
for every column in board do

if tempBool then

board cell ← counter  
 tempBool ← not tempBool

}

User Interface  
  
The user interface consists of a form (window) which encloses the draw grid, start and restart buttons. The draw grid refreshes, to update when a checker has been moved, and other operations have been carried out on the board. The player moves counters by selecting a counter and then selecting where they want to put it. While this occurs, the program is checking if the move is correct and updating the board. Below shows an annotation of the user interface:



AI checkers

P1 checkers

Restart game

Draw Grid

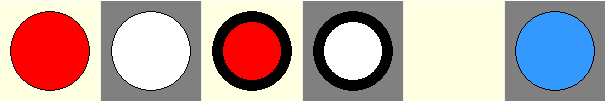
Form

Save game

Load game

Difficulty settings

The five states of the board are:



P1 Counter  
(C\_P1)

AI Counter  
(C\_AI)

Promoted AI Counter  
(C\_AI\_P)

Promoted P1 Counter  
(C\_P1\_P)

No Counter  
(NC)

Selected Counter  
(HIGHLIGHT)

1. Figure 1 – Rahul Yadav, 2017 [↑](#footnote-ref-1)
2. Figure 2 – Rahul Yadav, 2017 [↑](#footnote-ref-2)
3. Figure 3 – Rahul Yadav, 2017 [↑](#footnote-ref-3)
4. Figure 4 – Rahul Yadav, 2017 [↑](#footnote-ref-4)