Documented Design  
  
Data

|  |  |  |
| --- | --- | --- |
| Key Variables | Variable Type | What it is used for |
| Board | TArray – 8x8 array of integer | Used by main methods and other global variables to store the positions of checkers on the board. |
| CBoard | TBoard – object class | Used by main methods 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 – object class | Used to move checkers on the board, check if it is a legal move, and generate all legal moves for a player. |
| CAI | TAI – object class | Used to find the next move for the opponent. |

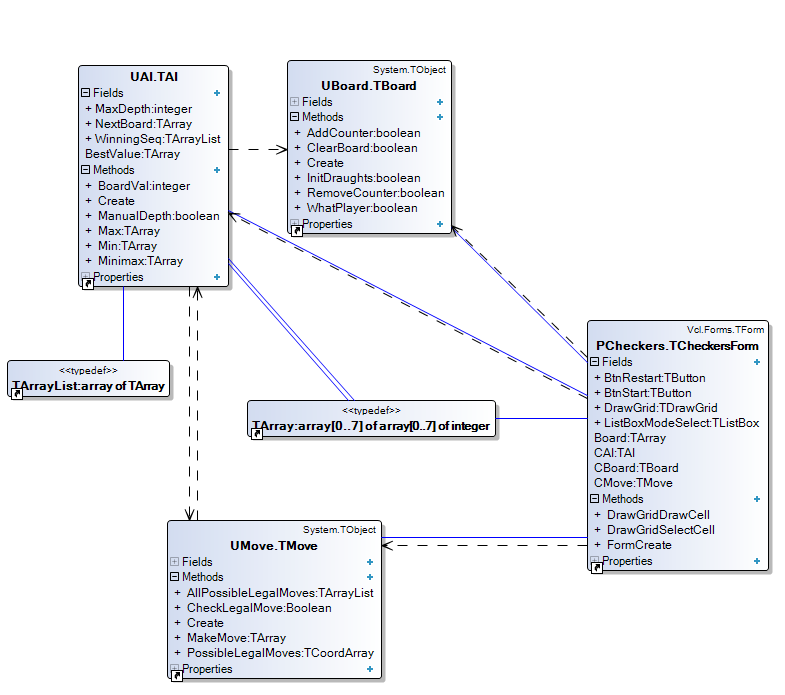
Other data structures:

TArrayList – an array of undefined length that stores TArray.

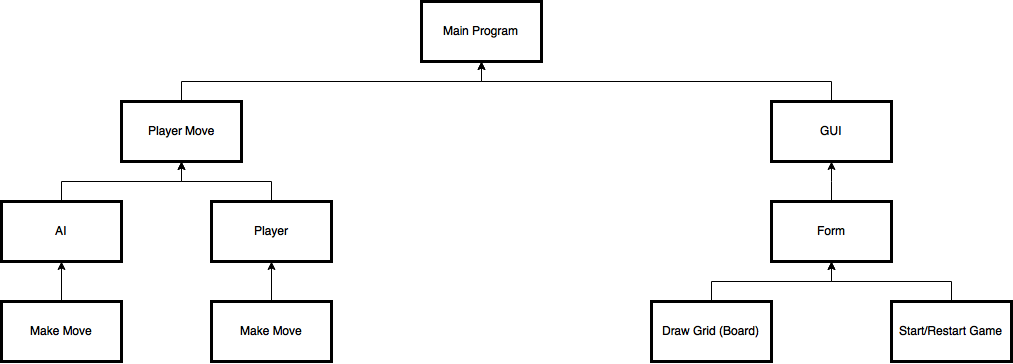
TArray – an 8x8 array of integers.

TCoordinate – an array of length 2 that stores integers.

TCoordArray – an array of undefined length that stores TCoordinate.  
  
Class-Relationship diagram



Hierarchy chart



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

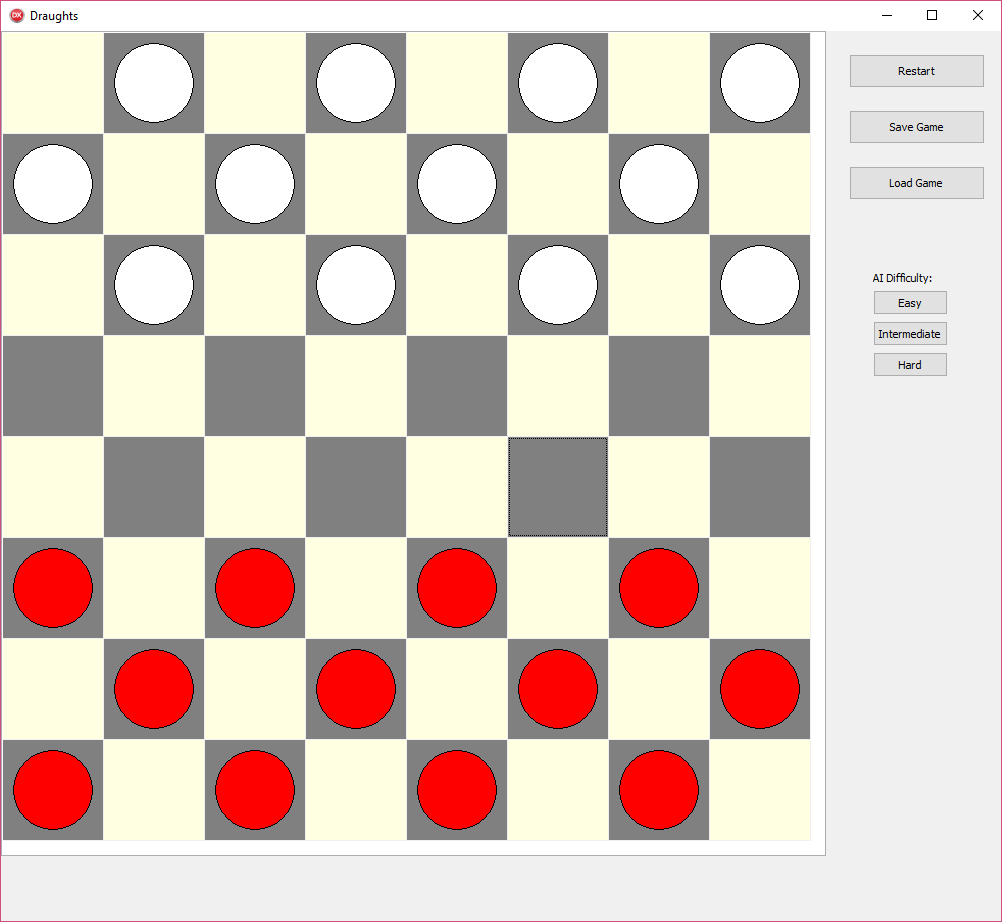
for every column in board do

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:

Selected Counter  
(HIGHLIGHT)

No Counter  
(NC)

Promoted P1 Counter  
(C\_P1\_P)

Promoted AI Counter  
(C\_AI\_P)

AI Counter  
(C\_AI)

P1 Counter  
(C\_P1)

