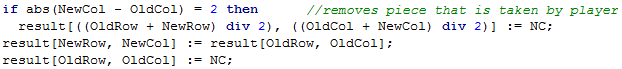
Testing

Informal Testing

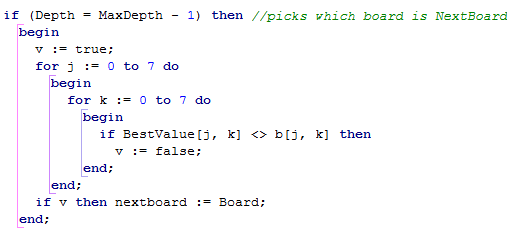
Error 1: UMove.MakeMove not removing counter when it is taken out of play

When a move that causes an opponent’s counter to be removed from play occurs, MakeMove will not remove it from play. This was fixed by checking if the counter moves 2 spaces, if it does move 2 spaces that means that it has removed a counter from play, which it proceeds to do. An excerpt of the code is shown below:



Error 2: UAI.Minimax not returning the correct board for the next turn

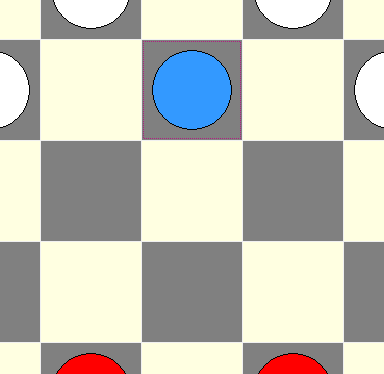
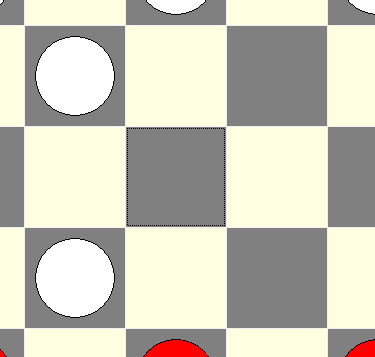
The move that Minimax returned wasn’t a legal move, after checking that the UMove functions didn’t have any errors I concluded that there must be an error with the Minimax function. I found that the function was assigning a board to NextBoard which was at a depth that didn’t correspond with the next move. An excerpt of the working code is shown below:



The code checks that the depth 1 less than the starting depth, and then it checks if it is the same as the BestValue board. If it satisfies the two conditions, then it is assigned to NextBoard which is used for its move.

Error 3: Ability to move the AI counter

The player should be able to only move their counters only, however I found that the player could select the AI counter and another cell, which would lead to an AI move. This resulted in the 2 consecutive AI turns. The pictures below show that the user can select an AI counter and the AI moves the piece it owns:



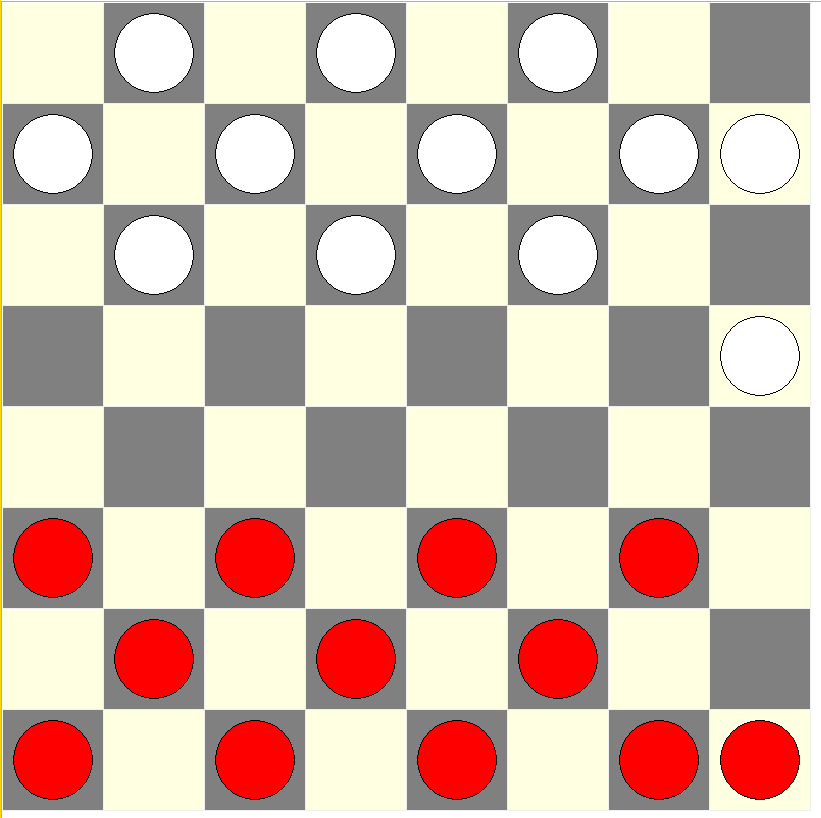
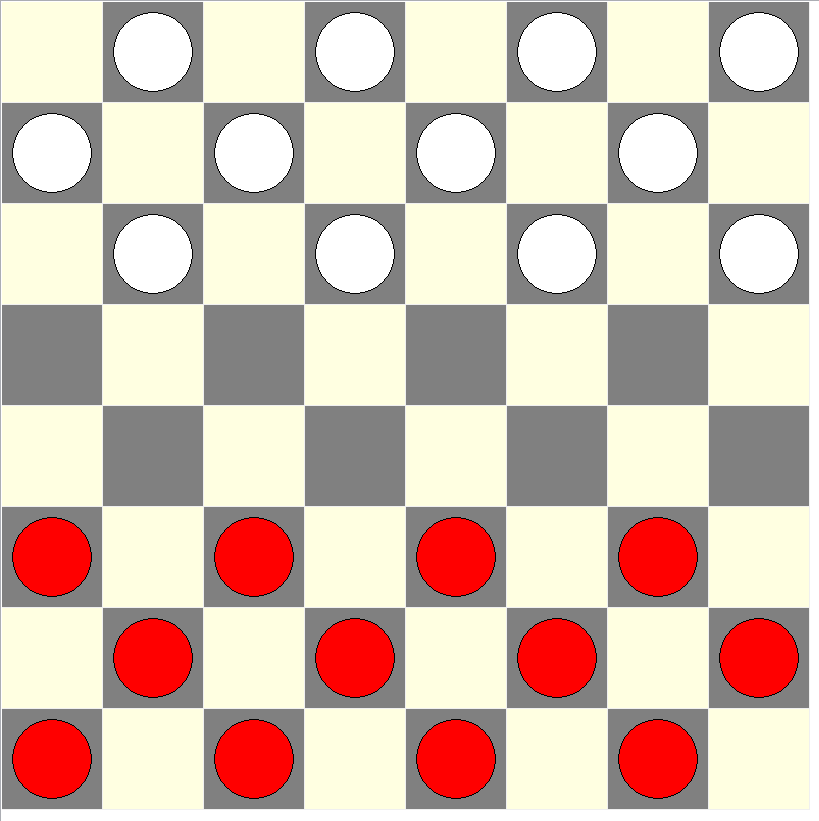
I fixed this by not allowing the player to select the AI counter, this was done by adding another condition to the IF statement in:  
DrawGridSelectCell(Sender: TObject; ACol, ARow: Integer; var CanSelect: Boolean)

The condition was: not ((Board[ARow, ACol] = C\_AI) xor (Board[ARow, ACol] = C\_AI\_P))

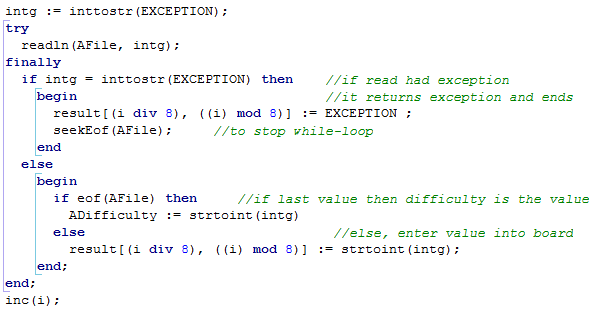
This condition would return FALSE if there it is an AI counter and TRUE if it were any other counter.

Error 4: USaveLoad.Load not loading the save file correctly

When loading the standard draughts setup, as a test file, the board would not show the correct pieces in the correct places, as shown below:

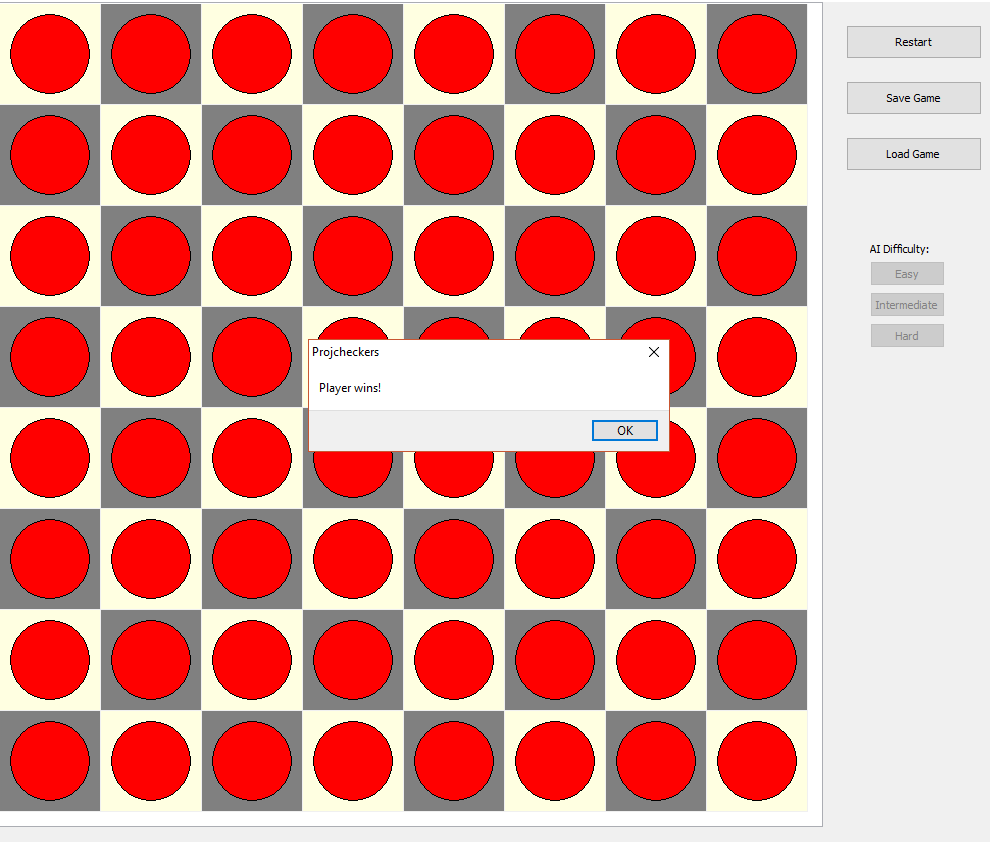
What it should be: What is loaded:  


Evidently, the loaded file must be saved or loaded incorrectly. After looking through the file, I found that there was no saving error, thus I concluded that there must have been a loading error. I found that the error must be due to a shift at the end of the board, because I am using DIV to identify what row the counter should go on. I fixed the error by making the counter variable increment after placing a counter, instead of placing it before. An excerpt of the working code is shown below:

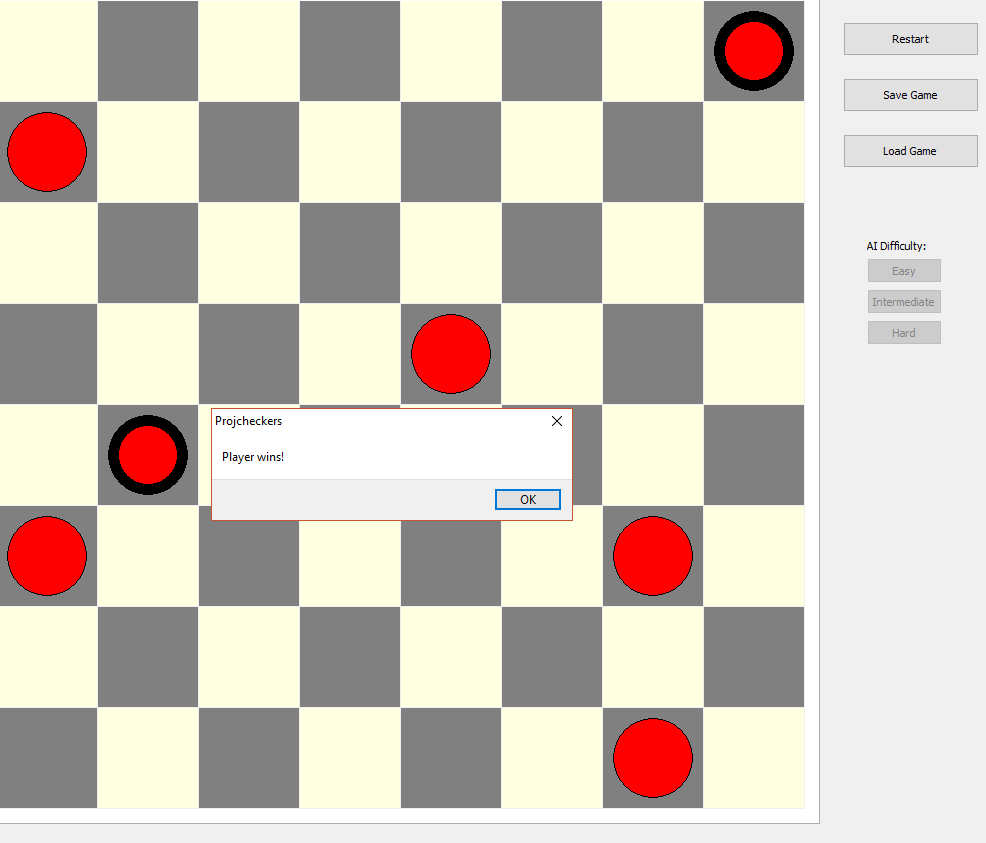


Error 5: Board populated with counters at the end of a game

When the game would end, a win/loss condition was met, the board would get populated with the winner’s counters, as shown below:



The only reason could find to explain this was UAI.Minimax returning the MaxBoard/MinBoard values, which consisted of a counter layout shown in the image above. This was fixed by not allowing any moves after a win condition was met.



The picture above shows the program working, with the error removed.

Formal Testing

Test plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Unit | What is getting tested | Input | Expected Result |
| 1 | UBoard. InitDraughts | This function should setup the draughts board at the start of each game. | Empty TArray | Functions returns an array with all the cells in the correct state, determined by the initial draughts board. |
| 2 | UBoard. WhatPlayer | This function should be able to identify if a counter belongs to the player or the AI. | TArray populated with all counter types | Function returning FALSE for player counters and TRUE for AI counters. |
| 3 | UMove. MakeMove | This function should to use the user input to move a counter from one cell to another. | 4 integers: old row, old column, new row and new column | A counter getting removed from one cell and added to another cell, while maintaining its original value. |
| 4 | UMove. CheckLegalMove | This function should check the move, against certain criteria, in order to check its legality. | TArray and 4 integers: old row, old column, new row and new column | Function outputs TRUE if it is a legal move and FALSE if it is an illegal move. |
| 5 | UMove. PossibleLegalMoves | This function should use UMove.CheckLegalMove on a specified counter to find what moves it can make. | TArray and 2 integers: row and column | The function returns a list of the coordinates for all the legal moves for each counter. This list is checked by UMove.CheckLegalMove to see if they are legal. |
| 6 | UAI. Min | This function should return the board with the lesser value, for use in USI.Minimax. | Two TArrays | The function returns the board with the lowest value, evaluated by UAI.BoardVal. |
| 7 | UAI. Max | This function should return the board with the greater value, for use in UAI.Minimax. | Two TArrays | The function returns the board with the highest value, evaluated by UAI.BoardVal. |
| 8 | UAI. Minmax | This function should output the next board, based on the move the AI has made, for use in the user interface. | TArray, Boolean and an integer (MaxDepth) | The function returns the array for the next turn, based on the parameters that it was given. |
| 9 | USaveLoad. Save | This function should be able to translate the board to text, so that it can save it to a text file. | TArray, difficulty (integer) and file name (string). | The function returns a save file with the value of each cell in the array and a value for the difficulty of the game. |
| 10 | USaveLoad. Load | This function should be able to translate a text file to variables for use in the board, then return that board for use in the user interface. | File name (string) and difficulty (integer). | The function loads each counter from the save file to their respective cell in the array. It must also correctly set the difficulty specified in the save file. |

Testing

|  |  |  |
| --- | --- | --- |
| Test # | Unit | Actual result, was it a success or failure? |
| 1 | UBoard. InitDraughts | There were no errors whenever I started and restarted the game, this is illustrated by the user interface which displayed a graphical representation of the board:  **Test Success** |
| 2 | UBoard. WhatPlayer | I used a quick test script:  if CBoard.WhatPlayer(0, 1, Board)  then ShowMessage('TRUE')  else ShowMessage('FALSE');  if CBoard.WhatPlayer(1, 0, Board)  then ShowMessage('TRUE')  else ShowMessage('FALSE');  if CBoard.WhatPlayer(7, 7, Board)  then ShowMessage('TRUE')  else ShowMessage('FALSE');  The test script output FALSE for a player counter and TRUE for an AI counter. The function returned the correct values (shown in the dialog boxes), corresponding to the draughts board in test #1.  **Test Success** |
| 3 | UMove. MakeMove | I called the function and tested it by moving certain counters from one location to another.  The first counter I moved was from (5, 4) to (3, 2) and the next counter I moved was from (5, 6) to (3, 6).  As shown by the screenshot, the test was a **Success**. |
| 4 | UMove. CheckLegalMove | I loaded a test board, with the 4 types of checkers, shown in the screenshot.  I tested each counter separately to see if they would per their respective properties:  C\_AI: **working**  C\_AI\_P: **working**  C\_P1: **working**  C\_P1\_P: **working**  **Test Success**  As a side note, I had previously encountered an error with this function where it wouldn’t check if there is a counter in selected cell, which I had previously fixed. |
| 5 | UMove. PossibleLegalMoves | I created a test script which used the function to generate a list of moves, then I cycled through these moves and checked if they were legal, by using the previously tested UMove. CheckLegalMove.    The result should have been 3 possible moves, that respond TRUE for the CheckLegalMove test. However, the result (shown in the screenshots) output 4 moves.  **Test Failure**  I found that the error was for normal counters only, UMove.PossibleLegalMove would also output a 2-space move. This was easily fixed as it was a simple logical error. |
| 6 | UAI. Min | I tested the function by inputting the standard draughts board and a board with less AI counters (compared to player counters). Board A was the standard draughts board and Board B was the board with less AI counters. The function should output Board B.  **Test Success** |
| 7 | UAI. Max | I tested the function by inputting the standard draughts board and a board with more AI counters (compared to player counters). Board A was the standard draughts board and Board B was the board with more AI counters. The function should output Board B.  **Test Success** |
| 8 | UAI. Minimax | I tested this function by checking what the function returned after the moves the player made. I made three consecutive moves to check that the minimax algorithm was basing its move based on the depth it is given. The screenshots show that the AI (white counters) is making legal moves that are in response to the players move.  **Test Success** |
| 9 | USaveLoad. Save | I tested this by saving a board with a few counters, then creating an array that is identical to the board was saved, so that I can loop through both the file and array to see if they are equal. The test was conducted by saving the board to a file, then an identical array would populate the board array. After this occurs, the test program checks if the save file and board are equal. The program returns an appropriate message based the equality.  **Test Success** |
| 10 | USaveLoad. Load | I tested this by saving a board with a few counters, then I restarted the program and loaded the save file. I compared this with a screenshot of the board before creating the save file, to check for discrepancies.  Before: After:    By looking at the board before saving and after loading, I can see that the the counters have been put in the correct cells of the board and the difficulty has been preserved.  **Test Success** |