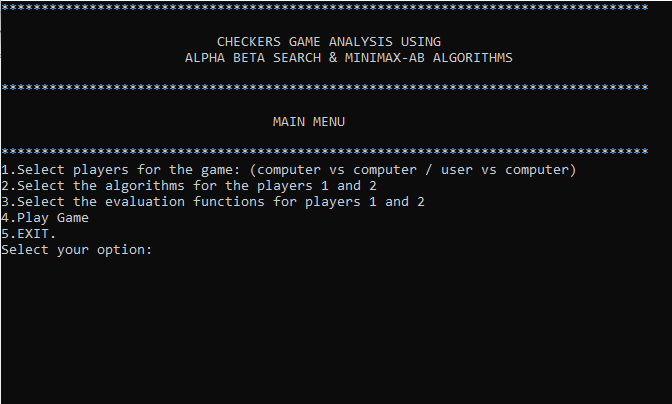
PROGRAM DESCRIPTION

The implemented program description is as follows:

* Main function:

Here the control of the program begins with a menu driven program as follows:



We will have to select the game type first i.e; Computer vs Computer or User vs Computer. For which the variable “play\_choice” will store the selection.

The variable “algo\_choice1” and “algo\_choice2” take the values for either “AlphaBetaSearch” or “MinimaxAB” for player1 and player 2 respectively.

Variables “eval\_choice1” and “eval\_choice2” accept the number selections from 1 to 4, for Eval1() through Eval4(), for player1 and player 2 respectively.

Player 1 is “R” pawn for Red and is in red color and its corresponding King is “S”.

Player 2 is “W” pawn for White and is in white color and its corresponding King is “Z”.

* Setcolor():

This function is used to set the colors of red and white respectively for players 1 and 2 respectively in the game interface. Uses the windows.h header file.

* Node class:

This class encapsulates all the functions related to the checkers board creation and move checks like jump\_left, jump\_right, etc, and evaluation functions etc.

Except the performance algorithms used.

* Utility():

This function calculates the vvalue and supplies it to the algorithms to gauge the value of heuristic of the winning move for any player.

* Terminal():

Checks whether any of the nodes are the leaf nodes or not.

* printboard():

Prints the checkers board after every move.

* MinMaxAB():

This function contains the algorithm for MinimaxAB as the name suggests.

It is implemented with the DeepEnough() function as the instructions and algorithm specifies. Tries to get the best path for player based on the evaluation function used.

* Alpha\_beta\_search():

This function contains the algorithm for alphabeta search as the name suggests again. And makes use of max\_value() and min\_value() functions.

Tries to get the best path for player based on the evaluation function used.

* MoveGen():

This function is used by both the algorithms to guess and perform the next plausible move of a pawn or king of players 1 and 2.

* Alpha\_BetacomVsAlpha\_Betacom():

The function is called to simulate and play a game of checkers between players 1 and 2, where both are computers, and both are using alphabeta search algorithms. But a combination of varying evaluation functions are selected to determine the winner.

* MinMaxABcomVsMinMaxABcom():

The function is called to simulate and play a game of checkers between players 1 and 2, where both are computers, and both are using minimaxAB algorithms. But a combination of varying evaluation functions are selected to determine the winner.

* MinMaxABVsAlpha\_Beta():

The function is called to simulate and play a game of checkers between players 1 and 2, where both are computers, and both are using a common evaluation function. But a combination of varying algorithms are selected to determine the winner, between alphabeta search and minimaxAB.

* Alpha\_BetacomVsAlpha\_Betauser():

The function is called to simulate and play a game of checkers between players 1 and 2, one is user and the other is computer, and both are using alphabeta search algorithms. But a combination of varying evaluation functions are selected to determine the winner.

* MinMaxABcomVsMinMaxABuser():

The function is called to simulate and play a game of checkers between players 1 and 2, one is user and the other is computer, and both are using minimaxAB algorithms. But a combination of varying evaluation functions are selected to determine the winner.

* Cutoff:

A cuttoff of game path length of 80 is placed to ensure the game doesn’t go into a long loop, and to enable a shorter approach to determining the winning player.

* Road to analysis:

Using the above functions and evaluations we achieve the length of game, number of nodes generated and expanded, execution time for each algorithm and use it to determine best algorithm combination.