## Artificial intelligence

### TOPIC:

Checkers playing bot using alpha beta pruning and min max

## Team members:

16BCE0223- SR NAVYA SREE

### **15MIS0271**- **AKSHAYA DHANANJAYAN**

17BCE0685-UJAYINI DAS

## Faculty:

RAJESHKANNAN R

**ABSTRACT:**

Checkers is a traditional board game played on an 8x8 board with 12 pieces each of black and white. The rules of the game are complex enough to take a checkers playing bot a few seconds to come up with each move during the initial phase of the game when there are a lot of possible moves to consider. In this project we have tried to make the bot learn a new rule added to the regular rules of the game that depends on the time that the algorithm starts looking for new moves. Given enough board states to play the game and enough winning states, the bot should be able to attach more priority to the moves that lead it closer to learning the additional rule and thus improve its learning.

Minimax is a recursive algorithm which is used to choose an optimal move for a player assuming that the other player is also playing optimally. It is used in games such as tic-tac-toe, go, chess, checkers, and many other two-player games. Such games are called games of perfect information because it is possible to see all the possible moves of a particular game.

The method that we are going to look in this article is called alpha-beta pruning. If we apply alpha-beta pruning to a standard minimax algorithm, it returns the same move as the standard one, but it removes (prunes) all the nodes that are possibly not affecting the final decision.

The min max algorithm is the most commonly used algorithm for this game however for optimization purpose we are using alpha beta pruning with min max algorithm

**INTRODUCTION:**

From the very early days of their existence, humans have played games for relaxation and for measuring their skills against each other. Many games test physical skill and stamina, for example tennis or football. Other games, such as checkers, chess, or go test the mental prowess and ingenuity of the players. Some games contain elements of both categories.

This project concentrates on mental or thinking games, in which players must use careful

Thought to weigh different options. These thinking games are often in the category of board

Games, i.e, games that are played with tokens or pieces on a two-dimensional board.

Checkers is a two-player game played on a 8x8 grid with black and white pieces denoting each player.

Existing rules of the game –

1. Black makes the first move.
2. Regarding placement of pieces: Each player begins the game with 12 pieces, or checkers, placed in the three rows closest to him or her.
3. Only the 32 dark coloured squares are used in play.
4. Basic movement is to move a checker one space diagonally forward. You cannot move a checker backwards until it becomes a King.
5. When one of your checkers reaches the opposite side of the board, it is crowned and becomes a King. Your turn ends there.
6. If a jump is available, you must take the jump.

The method that we are going to use is called alpha-beta pruning along with min max algorithm. Minimax search is a simple, yet relatively effective, tree-based planning algorithm. Minimax is based on a tree building process for games that are played by two opponents, called Min and Max.

The algorithm assumes that the current state, or root node of the tree, is the position

where it is the Max player’s turn to move. To generate the next layer, all the legal moves for Max are computed. Thereafter, the next layer contains all the possible

moves for the Min player. The tree building process alternately creates layers for each player until one of the termination criteria is met.

A big computational drawback of the minimax search is that it has to evaluate a large number of nodes that do not lead to promising states.

If we apply alpha-beta pruning to a standard minimax algorithm, it returns the same move as the standard one, but it removes (prunes) all the nodes that are possibly not affecting the final decision.

**Alpha-beta pruning Algorithm :**

function ALPHA-BETA-SEARCH(state) returns an action

v ← MAX-VALUE(state, −∞, +∞)

return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value

if TERMINAL-TEST(state) the return UTILITY(state)

v ← −∞

for each a in ACTIONS(state) do

v ← MAX(v, MIN-VALUE(RESULT(state, a), α, β))

if v ≥ β then return v

α ← MAX(α, v)

return v

function MIN-VALUE(state, α, β) returns a utility value

if TERMINAL-TEST(state) the return UTILITY(state)

v ← +∞

for each a in ACTIONS(state) do

v ← MIN(v, MAX-VALUE(RESULT(state, a), α, β))

if v ≤ α then return v

β ← MIN(β, v)

return v

**Project working model:**

In this project, we will program a checkers game. The player would have the option of playing against another player or AI. The AI would be programmed to 3 diﬃculties (easy, medium and hard).

• For the hard mode Alpha beta pruning algorithm will be implemented to design the AI.

• For the medium mode the AI would follow the general rules of the game that could lead to winning.

• For the easy mode the AI would make random moves and save the progress to improve.

Description of User interface:

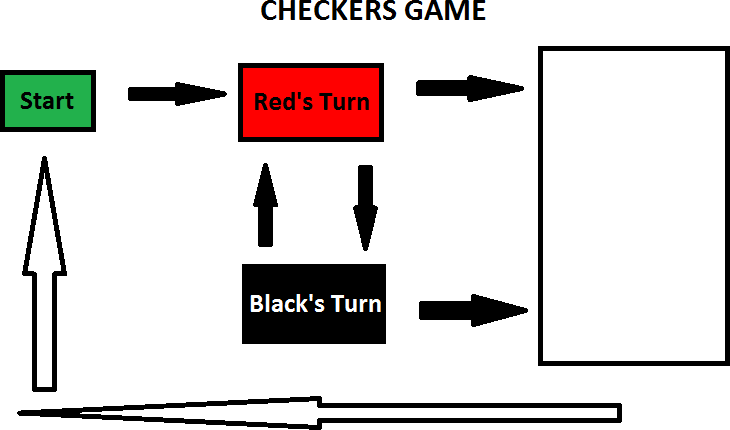
• Main menu scene: this would allow the player to choose either to play against another player or choose one diﬃculty mode of AI. (Other features might be added such as instructions).

• Game Scene: This would be a normal checkers board set up.

• Result Scene: This would show which player won.

Design:

We will be using python and its GUI to design the checkers game . The game will start, blue will move first, then black. This will continue until one of the players has lost all of their pieces. At that point, the game will declare the winning player then winner, and give the choice to start a new game or to quit. The functionality is illustrated in the state diagram in Figure 1



Red Wins

Winning Screen

Red Moves

Black Moves

Black Wins

Implementation:

In this part of the project we will explain our baselines in the implementation and the thoughts behind them.

**The board** – we chose to implement the board as a matrix (dimension X dimension). We think that this is the most logical way to do it and it will also make the GUI making process a bit simpler.

**Possible moves** – for each game stage (or position if you will) there is an array of moves that the player can make. This is also quite useful for the GUI.

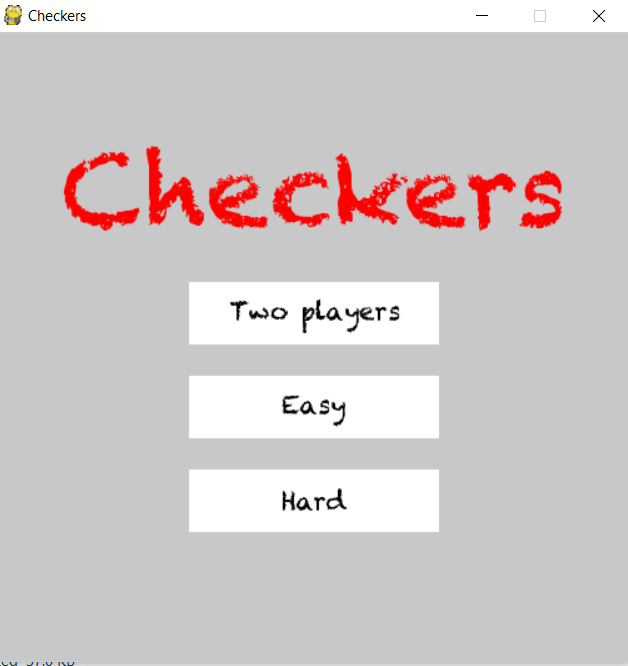
**Attack moves** – a Boolean to tell us if some of the possible moves are attack moves. This is needed for the game to play properly.

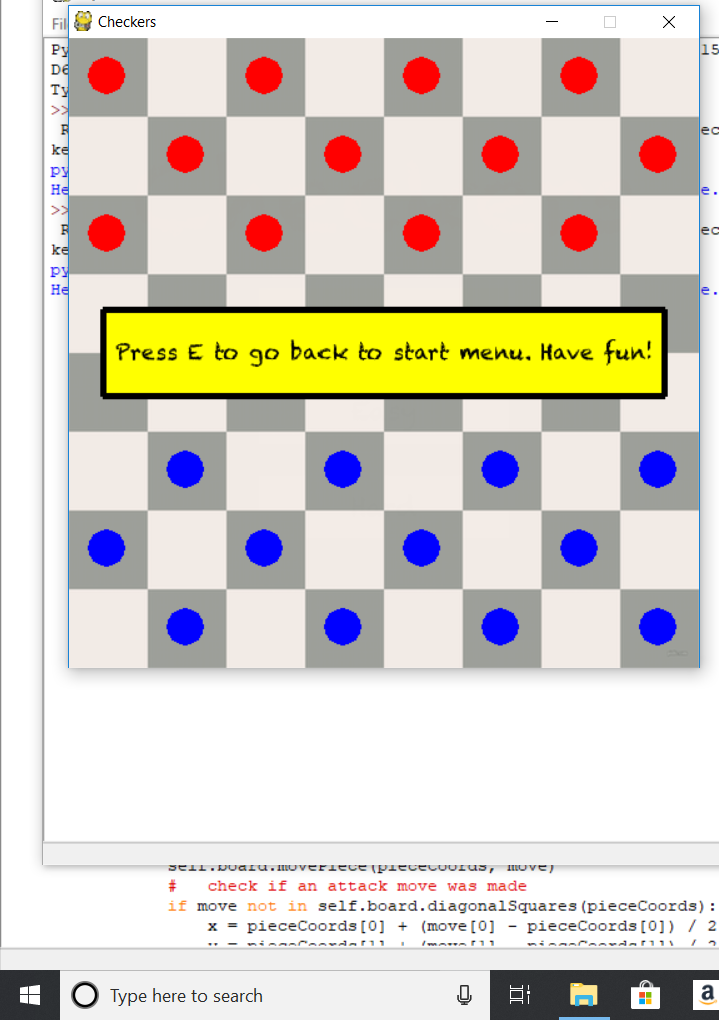
**Move** – a class representing a move in the game. Contains more than just the Src and the Dest, but also all the data we need.

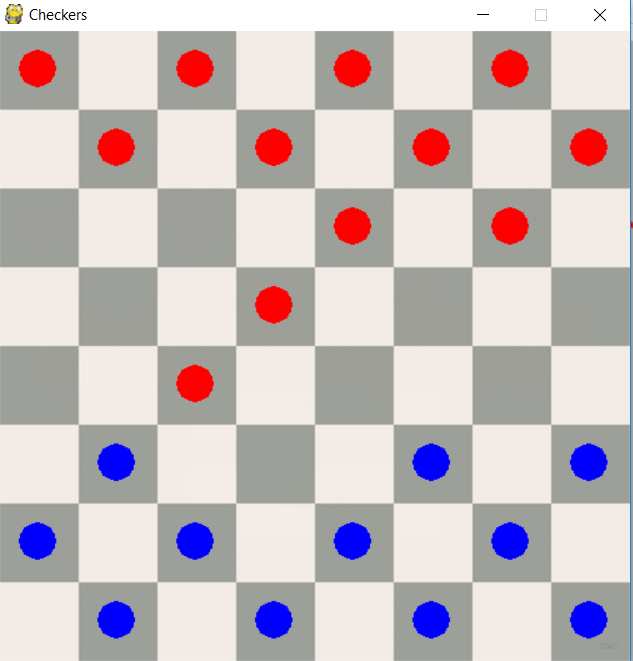
**Pieces** – for each game stage there is an array of pieces for both players. This is for knowing when the game ends, if the move if legal and other various issues.

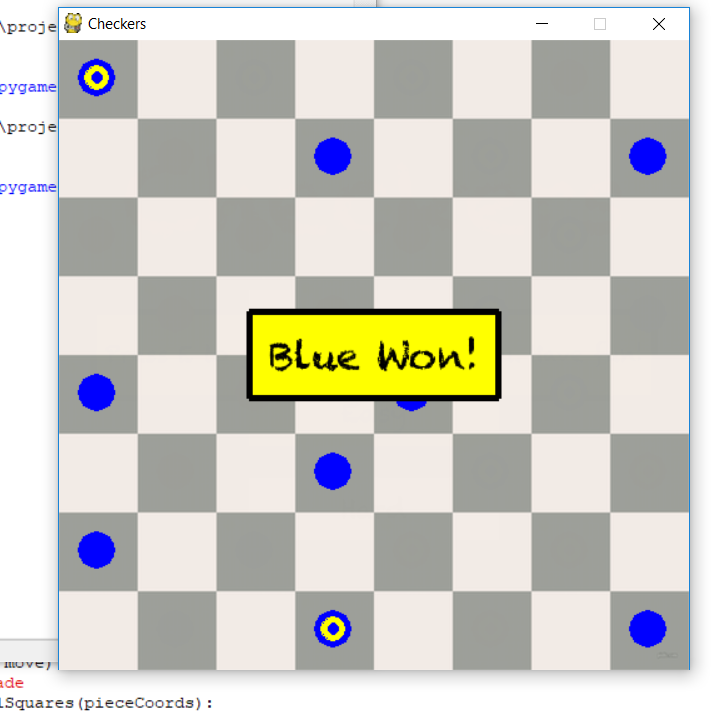
**Draw counter** – in the ending of each game (see the end-heuristics section) if there was no attack move for 80 (can be changed) moves we declare a draw.

**RESULTS:**









Conclusion:

* + - There are different approaches for heuristics.
    - Heuristics can be drastically improved by adding specific features.
    - The depth of the game tree has significant influence on the quality of the computer player.
    - is exponentially improving in comparison to Minimax as the depth grows.
    - Certain heuristics are clearly better than others but some of the “bad” ones still work well in some cases.
    - Simple algorithms as the random player don’t stand a chance against Alpha-Beta at depth greater than 1 so we use it for easy game
    - There are many other ways to approach zero-sum games but Minimax seems like a good on

## **REFERENCES:**

1)https://kartikkukreja.wordpress.com/2015/07/12/creating-a-bot-for-checkers/

2) http://www.cs.columbia.edu/~devans/TIC/AB.html

3) https://medium.freecodecamp.org/simple-chess-ai-step-by-step-1d55a9266977