Artificial Intelligence in Checkers

IIT2018028, Tarun Asudani

V Semester, B.Tech, Department of Studies in Information Technology,

IIIT Allahabad , Prayagraj, India

Abstract--In this report, we have devised the game of checkers in which a player can play optimally by determining the possible moves of its opponent using the Min-Max algorithm with alpha-beta pruning.

I. INTRODUCTION

Checkers is a strategy based game, played by two players such that both try to capture and block the pieces of the other. The pieces can only move diagonally, except in special circumstances, and must capture the opponent's pieces by jumping if such a condition is present, that is, capturing pieces is mandatory.

It is different in different countries, but in this report, we will implement the most famous version of the game: American Checkers. It consists of an 8X8 checkers boards and each player starts with 12 pieces.

There are two most generic rules:

1. Pieces that are not kings must travel diagonally in forward direction.
2. Kings may travel in all four diagonal directions.

The other rules that are implemented in this model will be discussed in the design and implementation section.

II. DEFINITIONS AND CONCEPTS USED

(A) Min Max Algorithm

This algorithm is used in two player games wherein one of the players is playing to make a result value maximum and the other is trying to decrease it to the minimum.

It is used in decision making which implements the combination of recursion and backtracking. It tries to get the best possible move for a player by using depth first search to determine the possible game tree and the path through which the chances of the player winning will be higher.

It is implemented with the notion that the second player is also playing optimally.

(B) Alpha Beta Pruning

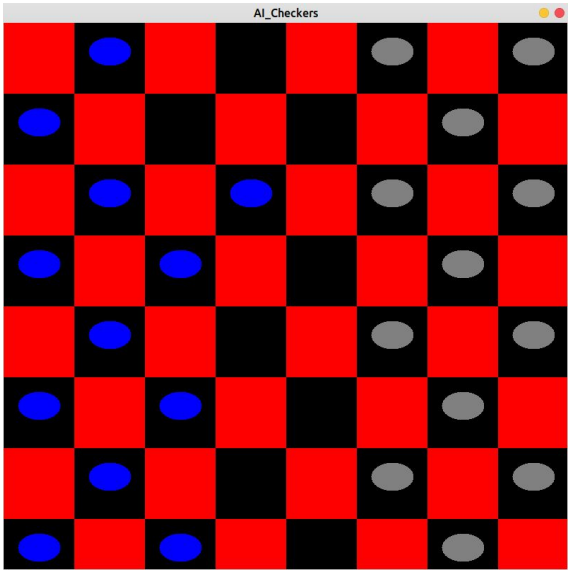
Pruning is used to optimise the min-max algorithm such that the player can go to deeper cut offs in a shorter period of time in such a way that without checking every node of the tree, it can get rid of the ones that do not hamper the decision of its parent node. Using this, it can even eliminate complete subtrees that have no role in affecting the decision of that particular move.

III. DESIGN AND IMPLEMENTATION

We have divided our problem in 3 different classes i.e Simulator, Board and Utility.
The following are the steps of our solution.

(A) For the Board class we have a grid of 800*800 pixels where each cell is 100*100 pixels. We colored each cell with black and red as checkers board. If the sum of row_number and the column_number of a cell is even then that cell is red otherwise that cell is black. Each cell has a value associated with it from the set : {0,1,2}. If the value is 1 then that cell contains a piece of the player, if the value is 2 then that cell has the piece of player otherwise 0 represents an empty cell. The player's

pieces are in the left side and the cpu pieces are on the right side.



(B) Next we have our Simulator class which has the main function. In this class we create the object of all the classes . We have submitted 2 projects one is a player vs cpu and the other is cpu vs cpu. In the player vs cpu we are taking the input from the user in an input dialog box. We first take x1 and y1 the coordinates of the initial piece to move and then the final position where we are moving our piece. If it is an invalid move the user will be prompted to enter again until he enters the move correctly. The coordinates for the pieces are as shown below.

```
   0  1   2  3  4 ……..
0_|__|___|___|__|__
1
2
3
4
.
.
```

After taking users input we apply the users move on the board and save that configuration. Then for the cpu's move we apply the minimax algorithm for predicting the most optimal outcome for the cpu. To get all the adjacent states and the heuristic function we use the utility class. We get the outcome of all the adjacent states by passing them in our minimax algorithm and select the most optimal move for our AI.

(C) The utility class has all the helper functions which are : get adjacent states, calc heuristic value for a state, clone a state, check if a move is valid. These functions are used in the main class to make the code clean.

IV. RESULT AND CONCLUSION

We can see that each player plays optimally and carefully before making any move. Moreover, the cutoff can be increased after applying alpha beta pruning to our min-max function. Thus, pruning has helped us in going deeper into the game tree and that too faster.

Using the min-max algorithm for this game of checkers is fitting since we not only have two players but also because we can assign positive to black and negative to red, and so while the former tries to maximise the score, the latter does the exact opposite.

We have used JAVA swing to implement this game and for getting its GUI.

V. REFERENCES

[1] Stuart Russel, Peter Norvig, "Artificial Intelligence, A Modern Approach"
[2]https://www.javatpoint.com/ai-alpha-beta-pruning
[3]https://www.arsdcollege.ac.in/wp-content/uploads/2020/03/Artificial_Intelligence-week3.pdf