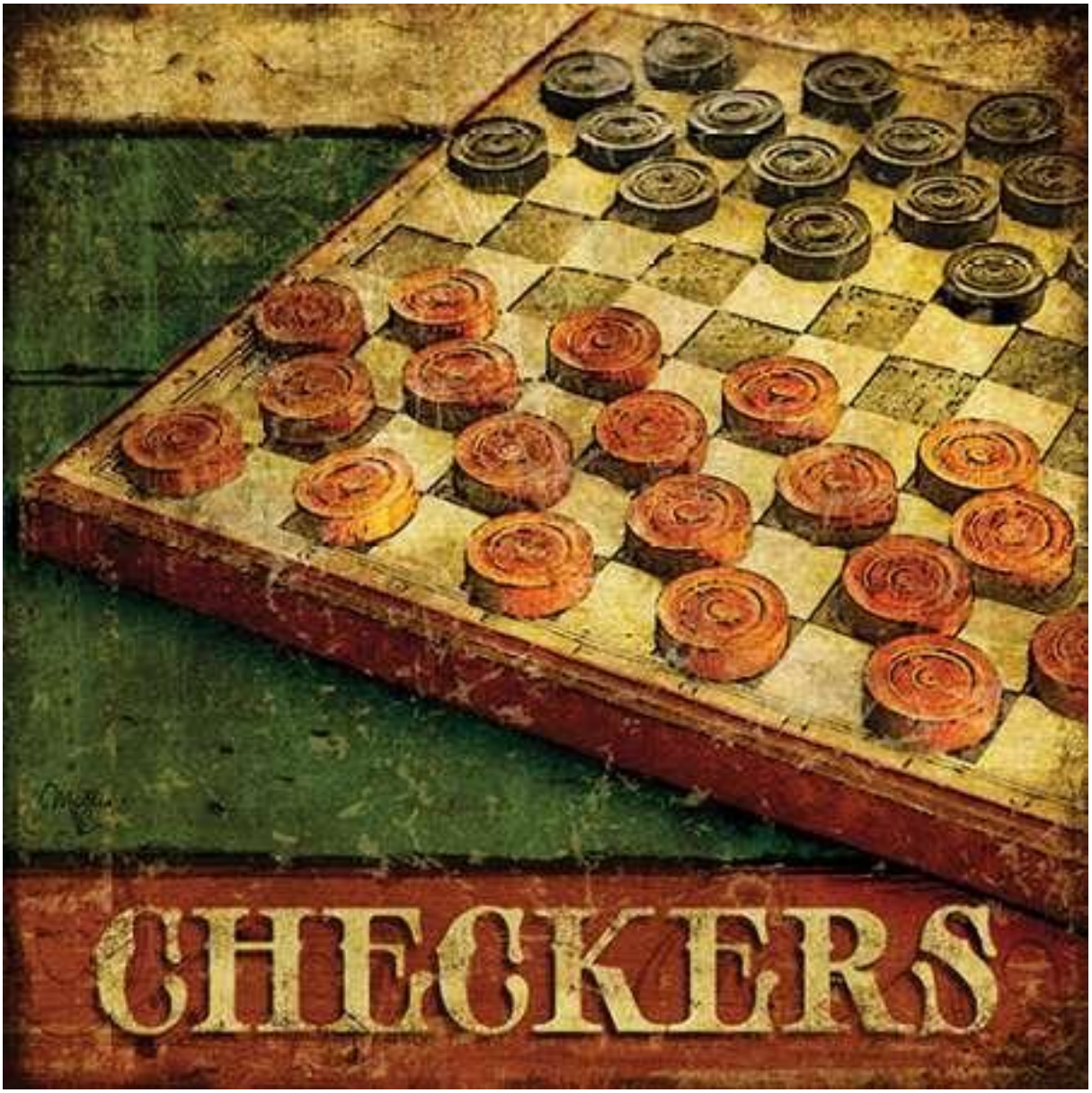

Final Project Report

Submitted to: Miss Noshaba Nasir
L15-4136 | L15-4256 | L15-4313
Artificial Intelligence - Section C

Checkers

Neha Akram, Alizeh Asim, Muhammad Ali

***Abstract*—this project is based upon the classical game of Checkers.**

Keywords—checkers; artificial intelligence;alpha-beta pruning; minimax

1. INTRODUCTION

The board game Checkers has been played by humans for decades now. In this project we have attempted to design a bot which will function in such a way as to excel human intelligence. In order to do so, we used the most common approach to solving any adversarial problem; the minimax algorithm with alpha-beta pruning. This strategy has proven successful in our attempts to outsmart the human mind.

2. LITERATURE REVIEW

The **University of Stanford**[1] published a paper based on creating an artificial intelligence for the game of checkers. The authors have used minimax algorithm with alpha-beta pruning. They've created an 8*8 board and have started off with a random move. As the program proceeds, it begins to use AI techniques. In the minimax algorithm, instead of exploring maximum depth which is quite large, they have limited their search to a particular level and have then predict the heuristics at that point. The simplest heuristic they have implemented is the difference between each player's numbers of pieces currently on the board. Other forms of heuristic functions discussed involve weight training.

The **University of Notre Dame**[2] presented a similar paper in which they too discussed techniques to solve checkers. They too used minimax algorithm with alpha-beta pruning, generating up to five moves in advance, but only to a pre-specified depth. They also discussed transposition tables and opening/endgame database increasing the efficiency

but didn’t implement either. They even talked about how in the past few decades, scientists have come up with search algorithms that use metalevel computations and interval bounds to prune the tree even better than the alpha-beta pruning method. The game they implemented was pretty straightforward, with the board, pieces, and the nodes of the search tree. The board was searched using the weight assigned to each piece and location along with the type of piece and positions of other pieces, and returning a positive or negative value.

3. GAME PROPERTIES

Our implementation relies on the rules of English draughts as they're described in Wikipedia [1], and on some additional rules that follow from them. Such as:

1. Regular capture is possible only in forward direction, except if the capture is a part of multiple stages capture on the same move, it's possible to capture also backward.
2. When a pawn reaches the last line it is crowned to King. Kings are able to move both backward and forward and they're also able to capture both backward and forward.

Our implementation is generic and can be easily expanded to even higher dimension boards that preserve the current game properties (The same rules, the first 3 lines of each side hold pieces).

4. PROBLEM STATEMENT

Our project’s focus is how to create an artificial intelligence that a user can play a game of checkers against. The main task is to gather the available moves, select the strongest action, and update the board accordingly.

5. METHODOLOGY

In this project we will use random and minimax algorithm with alpha-beta pruning to strategically play a game of checkers against a human opponent.

- **RANDOM:**

Our baseline approach was to build a randomized player that picks each move from a set of possible moves. As the moves are picked randomly, that causes an irrational player which doesn't play by any strategy. A property of checkers is that one mistake is enough to assure the other player a sure win (even if it takes few dozens of moves).

- **MINIMAX**

The entire minimax algorithm has been coded from scratch. We considered optimization known as **alpha-beta pruning** and the algorithm is as follows:

1. Have two values passed around the tree nodes:

   ➔ the alpha value which holds the best MAX value found

   ➔ the beta value which holds the best MIN value found.

2. At MAX level, before evaluating each child path, compare the returned value with of the previous path with the beta value. If the value is greater than it abort the search for the current node;.

3. At MIN level, before evaluating each child path, compare the returned value with of the previous path with the alpha value. If the value is lesser than it abort the search for the current node.

6. EVALUATION FUNCTIONS

The game works on two kinds of functions. The evaluations throughout the game and the other section that deals with ending of the game.

1. Opening-Middlegame evaluation functions

We will be using different heuristic functions to see how favorable a particular game state is for our bot. The heuristics are based on the following factors:

a. The number of pieces of both the system and the opponent.

b. The number of kings of both the players.

c. Whether or not the move will result in expulsion of opponent's piece.

d. Whether the move will result in an enemy jump.

2. Ending evaluation functions

The game can end with two possible cases:

1. **One player kills all checkers of other player**
   In this case, if all the checkers of one player die, the opponent player wins.
2. **Game is stuck with same moves and no attacks are being made**
   In case where there are no attacks by either of the players and moves exceed a certain limit, we count checkers for each player.
   a. WHITE WINS: Number of white checkers> Number of black checkers
   b. BLACK WINS: Number of white checkers< Number of black checkers
   c. DRAW: Number of white checkers = Number of black checkers

7. Experimental Setup

In order to make our gaming experience more interesting for the user, we created different modes for the user to play in. The following are the different modes:

- **Human vs Robot**

There are two possible cases when human vs bot game is played. They are as follows:

1. **HUMAN VS. MINIMAX ROBOT**

   In this scenario the user plays against an artificially intelligent which presents a good challenge to the user.

2. **HUMAN VS. RANDOM ROBOT**

   For those who don't like a challenge and would much rather win the game easily, we created a bot which does not apply any algorithm to improve its performance, rather, it selects a random move from the available moves and plays without any sound logic following no methodology. This makes it easier for the user to win. The statistics have been provided in the pages to come.

- **ROBOT VS ROBOT**

In order to further test the performance of the artificially intelligent bot we had created, we introduced the option to observe two bots competing against each other. The different variants of this are mentioned as follows:

1. **MINIMAX BOT VS. MINIMAX BOT**

   In this variant, the two artificially intelligent bots will go head to head against each other in attempts to outsmart the other one.

2. **RANDOM BOT VS. RANDOM BOT**

   As the name suggests, two bots functioning without any fixed guidelines or principles will compete against each other.

3. **RANDOM BOT VS. MINIMAX BOT**

   The randomly functioning bot will compete against one that is working based on a smart algorithm.

We had a total of 6 heuristics and to see if the game ended differently without any one of them, we conducted an experiment. We only applied one heuristic at a time and observed

the results as to who won. The results have been summarized in the tables presented further along the report.

Note: This experiment was conducted on ALL variants of the game.

8. 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.
- **The square** - or 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.
- **The checker** - Checker has its own class with its own attributes such as its coordinates and some boolean values.
- **The 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.
- **AI functions** - Rest involves various functions that help in the working of random and artificial bots.

9. Results

Mode	Heuristic	Winner	Number of black checkers	Number of white checkers	Avg running time of each move	Total number of moves	Total number of black moves
Human vs Minimax bot	Protect	Human	0	9	3.03	60	30
Human vs Minimax bot	Enemy Jump	Bot	8	5	2.21	80	40
Human vs Minimax bot	Escape	Human	0	6	0.77	62	31
Human vs Minimax bot	King	Human	0	7	0.16	58	29
Human vs Minimax bot	Move	Human	1	6	0.15	56	29
Human vs Minimax bot	Jump	Human	1	8	0.21	52	27
Human vs Random bot	N/A	Human	1	12	0.027	56	28
Human vs Minimax bot	All	Bot	7	4	5.4	80	40

Table1: Human vs Bot

Mode	Heuristic	Winner	Number of black checkers	Number of white checkers	Total number of moves	Avg time of Black move	Total number of black moves	Avg time of White move	Total number of white moves
Random vs Random	N/A	White	1	6	97	0.032	48	0.039	49
Minimax vs Minimax	Protect	Black	10	2	47	13.87	23	5.94	24

Minimax vs Minimax	Enemy Jump	Black	9	1	85	11.87	42	7.57	43
Minimax vs Minimax	Escape	Black	11	2	35				
Minimax vs Minimax	King	Black	7	1	55				
Minimax vs Minimax	Move	Black	10	2	39				
Minimax vs Minimax	Jump	Black	7	0	43	0.13	21	0.18	22
Random vs Minimax	Protect	Black	7	1	42	2.32	21	0.017	21
Random vs Minimax	Enemy Jump	Black	10	1	38	1.82	19	0.0074	19
Random vs Minimax	Escape	Black	8	3	94	0.75	47	0.0037	47
Random vs Minimax	King	Black	8	1	68	0.18	33	0.0066	34
Random vs Minimax	Move	Black	9	0	38	0.14	19	0.017	19
Random vs Minimax	Jump	Black	8	0	76	0.20	38	0.0059	38

Table2: Bot vs Bot

10. CONCLUSION

In this project minimax with alpha-beta pruning was used to solve the game of checkers. A variety of heuristics were used to test our bot against human players. While we were able to win the game when only one of the possible six heuristics was applied, we, as amateurs, could not match up to the bots intelligence and perception.

REFERENCES

[1] Hugo Kitano, Daniel Gallegos and Marco Monteiro, “An Artificial Intelligence For Checkers”, University of Stanford.

[2] Bill Grenzer and Steve Balensiefer, “Artificial Intelligence for Checkers”, University of Notre Dame.