

Assignment Report

Checkers game using Minimax Algo.

Devansh Goel

191010132

Devyansh Sehgal

1910110134

Som Pande

1910110398



Introduction

1. Explanation of the algorithm used in the game
2. References
3. Conclusion




The algorithm used in the game for generating the AI engine that plays against the user:

We decided to create the whole project using javascript and therefore the implementation of the algorithms has been wiped out in the checkers.js file itself. The User can launch the webpage on their local machine and play against the pc.

Minimax may be a general algorithm that's utilized in the game-supported discrete alternating moves. Minimax searches for the move which will minimize the choices of the opponent. This is often done by exploring a tree of moves with a predefined depth. The basis of the search tree is the current position of the sport board. Each branch off the basis represents a possible move by the opposite player. The minimax algorithm assumes each player will choose the move that maximizes its position.

The tree continues branching out, alternating between players at each level of the tree until there are no more possible moves within the game, or until the pre-determined depth has been reached. At the leaf nodes, the sport state is evaluated by returning a true value, indicating the strength of the board relative to the opponent.

Among a group of child nodes, the utmost score among them is passed up to their parent node. This reflects the very fact that the player would choose the move that maximizes its position. Among all the parent nodes that are siblings, and who themselves share a parent, the minimum score is then passed up to the parent.



this is often done keeping in mind that the pc will choose a move that minimizes the opponent's options and increases its own score.

This process continues all the way up to the basis(root) of the tree. At the basis(root), we will now see which of the possible moves should end in a rock bottom score for the opponent.

The front end:

Built using HTML and CSS the front end part consists of different buttons to make the game interactive. The user has an option to choose whether he wants to play against the AI randomly or compete against the algorithm which increases the difficulty of the game. We even print out various stages of the game like when a king is formed or when an illegal move is made on the console of the web browser.

Logic and various functions implemented:

All the functions have been implemented in the Checkers.js file. We created 2 Board-like 2D arrays, a static board that stored the initial positions of the pieces before starting the game and another dynamic board that kept on changing its content based on the current position of the pieces. We have considered 2 types of pieces (1 and 2) on the board and perform various operations on them based on their type.



Functions Implemented:

Legal(): this function checks whether the move initiated by the user is a legal move or not. For a move to be legal we had set constraints and conditions as described in the assignment description.

cut(): This function checks whether the next move being made can be an attacking move and eliminate an opponent's piece keeping in mind the rules of the game.

avoidReverse(): checks and prevents reverse motion of the pieces until a king is made.

box_fill(): function to check if an element is present at a reqd index of the board.

onchance(): Function to set the counter for only one chance at a time

boundaryconditionsOnBoundary(): to check for boundary conditions.

tamperlist(): list tampering to avoid repetition of code.

addtolist(): to avoid repetition of code

addimages() : adding the image classes and ids.

placement_of_pieces_initial(): to set the initial placement of pieces.


placement_change(): to set placement after game starts.

random() : function to make AI pieces move randomly.

minimax(): the function that implements the minimax algorithm and runs the AI engine.

onchance(): to allow only 1 chance per player at a time.

createboard(): function to create board.



Drag and drop functions for moving pieces

dragleave():

dragover():

dragenter()

dragStart():

drag(): to allow to drag a piece from its initial position

drop(): to allow to drop a piece on its final position before the next turn.




References

- Class Lectures and reference links for articles given in the assignment itself.
- Minimax Algorithm that was implemented in the game

<https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/>

- Reference for functions implemented in the code:

<https://frontendmasters.com/books/front-end-handbook/2018/learning/javascript.html>



https://www.researchgate.net/publication/342849109_Study_of_Artificial_Intelligence_into_Checkers_Game_using_HTML_and_JavaScript

https://www.researchgate.net/publication/339337169_checkers_research_paper_based_on_AI_2

Conclusion

■ ■ ■

We were able to complete the given assignment as per the instructions provided by Prof. Saroj Kaushik. We Implemented the Minimax Algorithm to construct the AI engine that plays against the user. **All the 3 members of the group contributed equally towards all parts of the assignment namely: ideating the approach, developing the code and testing the developed algorithm in the context of the board game, and creating the end report.** We thank Ma'am and all the TA's for their support in helping us understand the problem statement.