

Implement an AI agent for a board game (Chess, Connect 4, or Checkers).

2. You can implement it using Prolog, or Python.

3. In case of prolog, play against human, in case of python, it is computer vs agent

4. Implement Minimax algorithm. (2 Marks)

5. Attached is an example for a connect-4 project setup in python, that take input from the game,

and perform actions based on your selected column, <http://kevinshannon.com/connect4/>, feel free

to use it, and continue based on it, or you got the idea if you want to choose another board game

6. Refine the AI agent using the Alpha-Beta pruning algorithm. (4 Marks)

7. Feel free to use any AI tool (ChatGPT, Copilot) to help with the project.

8. Create a GitHub repository for the project with a minimum of 5 commits, with each team member

contributing at least 1 commits that contain significant changes. (1 Mark)

Bonus (3 Marks):

1. Implement a GUI to select the algorithm type and difficulty level of the game. (1 Mark)

2. (1 Mark)

a. Write a post on LinkedIn to share your experience and discuss the benefits and risks of using AI tools to help you in the project.

b. Measure the performance of the two algorithms and create a graph to compare them.

c. Screenshots of how the AI tool helped you, for example, chat conversations in ChatGPT, for significant help.

3. Implement different game than connect-4 (1 Mark)

Policy:

1. Students must work in groups of 4-5 for their project. Students have to be from the same lab or

from another lab taught by the same TA.

2. Cheating Policy: Negative the project grade, if during the discussion, any of the team member do

not understand any part of the code

3. You cannot implement the games that is provided in the lab

Cairo University

Faculty of Computers & Artificial Intelligence

Department of Computer Science

Artificial Intelligence Course

4. Submission folder to include:

a. Source code

b. Link to github repo

c. Link to linkedin post (if exist)

d. Screenshots of conversation (if exist)

e. Graph (if exist)

5. Reference: <https://youtube.com/playlist?list=PLyLtvBVWUxsF7c4JvI-z6MwTWZTT3rQN>

this an AI Course Project summarize it and tell me the steps to do this project with python

Sure! Here's a summary of the AI course project and the steps to implement it using Python:

🔗 ↗ ↘

Project Summary:

The project involves implementing an AI agent for a board game (Chess, Connect 4, or

Checkers) using the Minimax algorithm and refining it with Alpha-Beta pruning. The AI agent

will play against a human player or the computer, depending on the game. The project also

includes creating a GitHub repository, implementing a GUI for algorithm selection and

difficulty level, and bonus tasks related to measuring algorithm performance and writing a

LinkedIn post.

Steps to Implement the Project:

1. Choose a Board Game: Select a board game to implement the AI agent. Chess, Connect

4, or Checkers are popular choices.

2. Set Up the Project: Create a new Python project and set up the necessary project

structure. You can use the provided Connect 4 project setup as a starting point if you

choose Connect 4 as the game.

3. Implement the Game Logic: Write the game logic for the selected board game. Define the

board, game rules, and valid moves. Implement the functions to check for a win or a draw.

4. Implement the Minimax Algorithm: Implement the Minimax algorithm to determine the

best move for the AI agent. The algorithm should recursively explore the game tree,

evaluating the utility of each board state and selecting the move that maximizes the AI's

chances of winning.

5. Test the AI Agent: Test the AI agent against a human player or the computer. Allow the

players to take turns making moves and update the game state accordingly. Use the

Minimax algorithm to determine the AI agent's moves.

6. Refine with Alpha-Beta Pruning: Enhance the AI agent's performance by implementing

Alpha-Beta pruning. This optimization technique reduces the number of nodes explored

during the Minimax search by pruning branches that are guaranteed to be worse than

previously explored branches.

7. Create a GitHub Repository: Set up a GitHub repository for the project. Initialize the

repository and commit the initial project files. Each team member should contribute at

least one commit with significant changes throughout the project.

8. Implement GUI and Difficulty Levels (Bonus): Develop a graphical user interface (GUI)

that allows the user to select the algorithm type (Minimax or Alpha-Beta) and the

difficulty level of the game. The GUI should provide an intuitive way for players to interact

with the game.

9. Measure Algorithm Performance (Bonus): Evaluate the performance of the Minimax and

Alpha-Beta pruning algorithms. Measure the time taken by each algorithm to make a

move and compare their efficiency. Create a graph to visualize and compare the results.

10. Write LinkedIn Post (Bonus): Share your project experience on LinkedIn. Discuss the

benefits and risks of using AI tools in the project. Include screenshots of significant

interactions, such as chat conversations with AI tools like ChatGPT.

11. Complete the Submission Folder: Gather all the project components for submission. This

includes the source code, a link to the GitHub repository, a link to the LinkedIn post (if

applicable), screenshots of conversations (if applicable), and the performance graph (if

applicable).

Remember to refer to the provided playlist and any other relevant resources to guide you

through the implementation of the Minimax algorithm, Alpha-Beta pruning, and other

aspects of the project.

Good luck with your AI course project!

act as a AI Course Professor and explain Minmax Algorithm

🔗 ↗ ↘

As an AI Course Professor, I'd be happy to explain the Minimax algorithm to you.

🔗 ↗ ↘

The Minimax algorithm is a decision-making algorithm used in games with two players,

typically called the maximizer and the minimizer. The algorithm aims to determine the best

move for the maximizer player while assuming that the minimizer player will also make

optimal moves. It works by exploring the game tree, which represents all possible moves and

resulting game states.

Here's a step-by-step explanation of the Minimax algorithm:

1. Game Tree Generation: The algorithm starts by generating the game tree, representing

all possible moves and resulting game states. Each level of the tree represents a player's

turn, and the branches represent the available moves.

2. Terminal State Evaluation: At the leaf nodes of the tree, the algorithm evaluates the game

states and assigns a utility value. A positive value represents a favorable state for the

maximizer player, while a negative value represents a favorable state for the minimizer

player. The utility values can be based on predefined rules or heuristics specific to the

game being played.

3. Recursive Evaluation: Starting from the leaf nodes, the algorithm propagates the utility

values up the tree towards the root. At each level, it alternates between maximizing and

minimizing the utility values.

4. Maximizing Player's Turn: When it's the maximizing player's turn, the algorithm selects

the move that leads to the highest utility value. It chooses the child node with the

maximum utility value among its children.

5. Minimizing Player's Turn: When it's the minimizing player's turn, the algorithm selects the

move that leads to the lowest utility value. It chooses the child node with the minimum

utility value among its children.

6. Tree Traversal: The algorithm continues traversing the tree, alternately maximizing and

minimizing the utility values, until it reaches the root node.

7. Final Move Selection: Once the algorithm reaches the root node, it selects the move

corresponding to the child node with the highest utility value. This move represents the

optimal decision for the maximizer player.

The Minimax algorithm guarantees that, assuming both players play optimally, the

maximizer player will make the best move possible. However, it is not always feasible to

explore the entire game tree due to its exponential size. Therefore, the Minimax algorithm

can be computationally expensive for complex games.

To improve the algorithm's efficiency, techniques like Alpha-Beta pruning are often used.

Alpha-Beta pruning eliminates the need to explore certain branches of the game tree that

are guaranteed to be worse than previously explored branches, further reducing the search

space.

I hope this explanation helps you understand the Minimax algorithm. If you have any further

questions, please feel free to ask!

tell me how to use the algorithm to make a connect4 game

🔗 ↗ ↘

To use the Minimax algorithm to create a Connect 4 game, you can follow these steps:

🔗 ↗ ↘

1. Set Up the Game:

• Define the Connect 4 board as a grid with 6 rows and 7 columns.

• Create a function to display the current state of the board.

• Implement functions to check for a win or a draw in the game.

2. Generate Possible Moves:

• Write a function to generate all possible moves for the current state of the game. In

Connect 4, this means identifying the available columns where a player can drop their

token.

3. Implement Minimax:

• Create a function, let's call it `minimax`, that takes the current state of the board, the

depth of the search, and a boolean flag to indicate if it's the maximizer's turn or not.

• Inside the `minimax` function, check if the game has reached a terminal state (win or

draw). If so, return the utility value (positive for maximizer, negative for minimizer, 0 for a

a draw).

• If the search depth is 0 or the game hasn't reached a terminal state, generate all

possible moves.

• Iterate over each possible move and recursively call `minimax` with the updated

game state and depth. Keep track of the best value found so far and the

corresponding move.

• If it's the maximizer's turn, update the best value if a higher utility value is found. If it's

the minimizer's turn, update the best value if a lower utility value is found.

• Finally, return the best value and the corresponding move.

4. Create the Main Game Loop:

• Start the game loop, where players can take turns making moves.

• For the AI player's turn, call the `minimax` function with the current board state and a

depth limit.

• Retrieve the move suggested by the `minimax` function and update the board

accordingly.

• Display the updated board and check for a win or a draw.

• Allow the human player to make a move by selecting a column to drop their token.

• Continue the game loop until a terminal state is reached.

5. Connect 4 AI:

• Set the depth limit for the `minimax` function based on the desired difficulty level of

the AI player. Higher depths may lead to more strategic and challenging gameplay.

• Adjust the utility values for win, loss, and draw based on their relative importance in

the game. Experiment with different values to achieve the desired AI behavior.

By following these steps, you can implement the Connect 4 game using the Minimax algorithm for AI decision-making. Remember to test and debug your code along the way to ensure it's working correctly.

Note: If you want to enhance the AI's performance, you can implement Alpha-Beta pruning as an optimization technique to reduce the number of nodes explored during the Minimax search.

Good luck with your implementation, and have fun playing Connect 4!