ARTIFICIAL INTELLIGENCE
ENCS3340

Course Project

Magnetic Cave

Instructor: Ismail Khater
Section: 1

Team Members:
Sami Abu Rmaileh 1192325
Malak Raddad 1192408

1/6/2023

Objective:

The goal of this project is to create a minimax algorithm for a game. The project's goal is to create a simple yet effective algorithm capable of analyzing multiple game conditions and making optimal judgments. The research intends to improve the game's artificial intelligence by allowing it to examine numerous alternative plays and their outcomes, finally selecting the best option to optimize its chances of winning. The project will entail creating and testing the minimax algorithm to assure its accuracy and efficiency in generating intelligent judgments within the game.

Program:

Is implementation of the game "Magnetic Cave." Two players take turns placing their symbols on an 8x8 chess board. The goal of the game is to line up five symbols in a row, either horizontally, vertically, or diagonally.

Here is an overview of the program's structure and main functions:

1. print_welcome (): Prints a welcome message that presents the game, including the AI algorithm and difficulty settings.
2. print_difficulty_options (): Prints the available difficulty levels for the user to choose from.
3. The main program flow starts by calling print_welcome () to display the welcome message and introduction.
4. The user is prompted to choose one of the three options (1, 2, or 3) for the game mode.
5. Depending on the user's choice, the program imports the appropriate module for the selected game mode and difficulty. The imported module contains the specific logic for that game mode and difficulty.
6. After importing the module, the program enters the second part of the code, which is not shown in the provided snippet.

The second part of the code, not shown here, consists of the game logic implemented in separate modules for different game modes and difficulties. Each module contains functions and code specific to that mode and difficulty level. These modules are imported dynamically based on the user's choices.

7. The next part of the code defines several functions related to the game board, such as creating the board, printing it, updating the board with player moves, and checking for a win condition.
8. The restart_game() function is defined to handle restarting the game if the user chooses to play again.
9. The main game loop starts by displaying the mode and difficulty level chosen by the user, followed by a prompt to press Enter to start playing.
10. Inside the game loop, the chess board is created, and the initial board state is printed.
11. The program defines the symbols for player 1 (■) and player 2 (□).
12. The game continues in a loop until the board is full or a player wins.

13. In each iteration of the loop, the current player is prompted for a move by entering a position on the board. The program checks if the input is valid and updates the board accordingly.
14. After each move, the updated board is printed, and the program checks if the current player has won by calling the check_win() function.
15. If the board is full without a winner, the game ends in a tie.
16. Finally, the user is prompted to restart the game or exit.

Justifies your heuristic:

This project's heuristic is a straightforward evaluation function that provides a score to each game state based on its perceived desirability. The efficacy of this heuristic to lead the minimax algorithm towards advantageous game conditions justifies its use.

The evaluation function takes two major elements into account: the number of pieces on the board and the current player's advantage. The heuristic supports scenarios where the player has a numerical advantage by assigning higher scores to game states with more pieces belonging to the current player and punishing ones with fewer pieces.

Furthermore, the heuristic considers the current player's advantage, assigning a positive score when it is the player's turn and a negative score when it is the opponent's turn. This encourages the minimax algorithm to choose actions that retain or strengthen the player's lead, while also taking defensive techniques into account when it is the opponent's turn.

The nature of the game and the minimal depth of the search space justify the heuristic's simplicity. While more complex heuristics could be devised, the approach selected finds a balance between computing efficiency and effective decision-making. It is predicted that through experimentation and testing, this heuristic will prove to be quite useful in steering the minimax algorithm towards optimal moves within the given game.

Tournament:

During the event, my project performed admirably, earning first place in the rankings. One of the most important aspects in my achievement was the use of a highly powerful heuristic. The heuristic was meticulously created to evaluate game states based on aspects such as piece location, important area control, and potential threats. This enabled my program to execute intelligent and strategic actions, outmaneuvering opponents on a consistent basis.

Throughout the event, I was constantly analyzing my opponents' plans. I was able to predict their activities and effectively counter their methods by studying their motions and modifying my approach accordingly. This adaptability and strategic thinking offered me an advantage over my competitors and aided my overall performance.

In conclusion, my project's powerful heuristic, optimized minimax algorithm, efficient performance, and strategic adaptability were major elements in my tournament success. I

was able to ensure success and beat my competition by painstakingly considering these factors and always refining my strategy.