Lenni Martikainen - BSc (CS) - Python - English (can review in Finnish)

Problem:

AI that knows how to play Connect 4. As I'm familiar with many of the topics already, if time allows, I will extend the game to utilize boards of different kinds. This includes at least the game being playable on a (emulation of) cylindrical or toroidal board beyond the regular 7x6 planar board. The main problem is to develop an AI which makes good decisions in a performant manner.

Algorithms planned to use:

Min-max with Alpha-beta pruning. This is the main algorithm I seek to use, and efficient pruning is crucial to increase the possible depth that the AI uses for the algorithm. In the worst case scenario, min-max has time complexity of O(b^d), where b is the amount of branches (7 in the case for Connect 4), and d is the depth of the search. This simply indicates that we need to visit every node in order to find the best choice. With Alpha-beta pruning and good ordering, we can expect to cut the amount of calculations by a factor of up to 2^d, meaning much less calculations. In the worst case ordering, no improvement over naive min-max is achieved. In the perfect case, we can get O((p^d)^0.5)), indicating we can search about twice as deep with the same amount of computations. I'm seeking to push the program to get somewhere between the "good" and the best speed.

Data structures planned to use:

Initial versions will use Numpy arrays to store the board state and the calculations related to the strength of positions. If this proves too inefficient, I'll seek to use more efficient data structures, such as bitmaps, where swift operations such as bit shifting can be utilized.

Inputs and functionality + project core:

I will be implementing a Connect 4 game, a two-player game where the goal is to place markers in a 7x6 grid which follows gravity, and the players seek to form a row (horizontal, vertical or diagonal) of four of their own markers before their opponent does. The game can end in a player winning, or a draw.

I plan to have a few options for the game. These include modes, with at least solo player against the AI and the option for two players to play at the same time locally. If convenient, AI versus AI matches can be arranged. Board size I will keep fixed at 7 columns, 6 rows. AI difficulty will be scaleable, varying the search depth it employs in order to find the best move on its turn.

Sources:
Wikipedia and Stack Overflow alongside code/library documentations should get me far. If needed, I'll consult someone to help me with the code. As it was mentioned as allowed, I may employ an LLM to help me make a simple interface for the program.