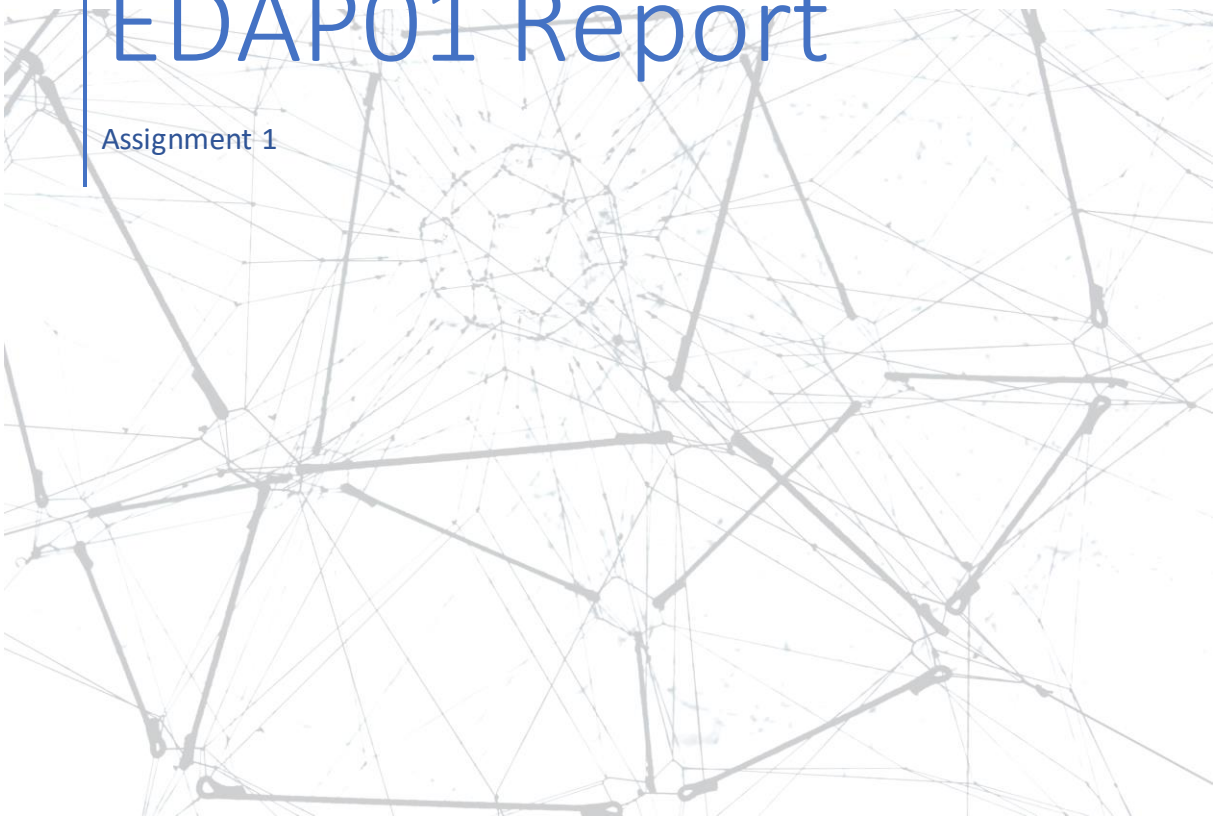
LTH | Computer science

EDAP01 Report

Assignment 1



Måns Alklint

2024-01-24

Index

1 Approach

The approach is to achieve 20 wins in a row. During development, certain things like depth for the search tree and the values in the preset score matrix was adjusted accordingly, ensuring that the algorithm won 20 times in a row against the server opponent.

1.1 Alpha-Beta Pruning

Alpha-beta pruning is an optimization technique applied to the minimax algorithm to reduce the number of nodes evaluated in the game tree. It maintains two values, alpha and beta, which represent the minimum score the maximizing player is assured of and the maximum score the minimizing player is assured of, respectively.

1.2 Function Minimax

This function defines the function responsible for the search algorithm to iterate through possible moves in the connect-4 game. The function works by recursion, which in short is a sort of a loop. The loop breaks as the search depth in the search tree reaches 0 or if the game is finished (i.e is win, tie or loss). The alpha and beta values during the search cuts off branches that are deemed irrelevant based on the current alpha and beta values. This allows for a more efficient search, reducing the number of nodes that need to be evaluated.

1.3 Function evaluation

The minimax-function uses a helper method to evaluate (give each node in the tree a stochastic value), as this is needed for determining if the move is considered good or bad. It assigns infinity if the player has won, negative infinity if the opponent has won, and computes a score based on the positions of the pieces on the board. This score is a preset represented as a shadowed matrix where each cell has its respective “score”-value.

1.4 Function checkWin

The evaluation needs this small helper method to check the board for positions that gives the respective player’s marker on the board its “score” (the stochastic value). For instance, if a player has a set of markers in a line with 3 as length, the next marker to be placed is deemed a “winning move”. This is done using loops that checks the board horizontally, vertically and askew.

2 Launching the application

The application's perquisites can be installed automatically using the following command:

```
pip install -r requirements.txt
```

Then the help commands for executing the application can be found using:

```
python game.py -h
```

3 Peer-review

A peer review was done with Christoffer Fjällborg as a partner.

Points discussed were the respective approach of evaluation. Since the partner never came to the point of having code that fulfilled the 20 wins in a row-requirement, I mentioned the use of a score matrix for heuristic evaluation which the partner then chose to utilize. The peer's solution is aside from that very well-structured and follows a clear algorithmic approach.

The partner also chose to explicitly infer datatypes as to avoid having mixed types (integer, boolean, string etc). More on this under “Technical differences”.

3.1 Peer’s Solution

The peer’s solution uses the minimax algorithm with alpha-beta pruning to search for optimal next moves in the board. The evaluation function assigns scores to game states based on the distribution of pieces on the board, using a predefined score matrix.

Potential areas for improvement include refining the evaluation function. The game should perhaps look out for the moves guaranteeing the win of a player. With some refinement and optimization in the evaluation function, it has the potential to achieve even better performance, especially in terms of winning against opponents.

The evaluation function (called “utility”) looks at the current state of the game board and assigns a score to it.

By looking at if the game is terminal, it decides whether the game is over. Then it checks who won or if it's a draw, and gives a high score for a win, a low score for a loss, and a neutral score for a draw.

In the non-terminal states, the game is still ongoing. It then looks at the current layout of the pieces on the board and calculates a score based on how advantageous it is for the player. It values certain positions on the board more than others based on a predefined scoring matrix. Higher scores mean the position is better for the player, while lower scores mean it's worse.

3.2 Technical Differences of the Solutions

Datatypes were inferred by the partner. Since python generally uses dynamical types, the partner chose to use infer this with each function that defines the returned value's type.

The partner chose to use the following class for defining his own state of the game, instead of using the inbuilt functions in the environment:

```
class State:
   env: ConnectFourEnv
   move: int
   board: np.ndarray
   result: float
   done: bool
```

3.3 Opinion and Performance

The major difference is the types which keep return values from getting mixed, which can be a problem in Python.

As for performance, the peer's solution seems to lag behind overall. It might be because the peer is using a maximum depth of '5' instead of '4', or maybe because the evaluation functions a bit differently. We could technically measure this with Python's built-in time functions, but it was just noticed by running the script.

Overall, the peers script looks to be a lot more robust and with improvements to the evaluation like implementing a better way of determining winning moves, it should perform great in 4 in a row-matches against the server opponent.

4 Interaction summary with an AI chatbot

4.1 Interaction prompts

After multiple attempts with ChatGPT 3.5, Claude and Bing AI, the code never ran. It needed extensive rework to complete the task. If I provided my code, like in a sense “training” the model the code ended up almost identical to mine. Instead, I ended up paying for a ChatGPT 4 license and this led me a bit further. The following prompts were given to ChatGPT 4.

Prompt 1:

“You are a student is taking the course "Artificial intelligence" Your task is to make this program work that wins the game "connect 4". For the program to work, the student_move should return a valid move (column to put marker between 0-6) that is the best. This can be done by: Writing an alpha-beta-pruning function for minimax. Writing an evaluation function that checks whether the move is good or not (give each move a score) This is the code that you are tasked to fix: [skeleton.py]”

As ChatGPT didn’t produce code to evaluate the board, it needed a prompt stating this.

Prompt 2:

“implement the evaluate_board and its logic”

The code was run and had an error. So, it was provided to ChatGPT.

Prompt 3:

“best_move = valid_moves[0]. This is a TypeError: 'frozenset' object is not subscriptable. Please provide a solution to this”

The code was run yet again, and there was a problem with ChatGPT assuming the env worked as a stack, as it tried calling .push() and .pop(). (Notice: ChatGPT consequently used the functions in the environment in a wrong manner and ended up fixing this manually)

Prompt 4:

"env.push(move) # Make the move

env.pop() # Undo the move

This should instead be done by:

env.step(move)

env.reset(board=theBoardItResetTo)"

After tweaking the code on some specific lines in the algorithm to make a deep copy of the environment to recursively call the minimax again with the next state of the environment without interfering with the original, the code ran but the algorithm was no match for the server opponent. However, in some cases the ChatGPT solution wins.

4.2 Differences

Both codes correctly implement the minimax algorithm with alpha-beta pruning in almost the very same manner.

My code is primarily focused on checking for win conditions (i.e., four in a row in any direction) for both the player and the opponent. If a win condition is found, it assigns an infinite score to represent an immediate win or loss, simplifying the evaluation to a binary outcome. Beyond checking for win conditions, it uses a predefined matrix to evaluate the board, where each position has a predefined score. This means that certain positions on the board are considered more valuable than others, regardless of the board's current state.

4.3 Performance

There are almost no significant differences in performance between my solution and ChatGPT's. However, important to note is that ChatGPT's evaluation isn't very good as it doesn't consider that some board placements could be better than other.

4.4 Conclusion

ChatGPT as it only is a language model faces limitations in performing logical operations due to its training on text datasets. While it is proficient in understanding and generating language-based outputs by its linguistic patterns, it has difficulties in deterministic reasoning and logical operations.