Report Assignment 1

Filippa Hansen, fi6368ha-s, EDAP01

February 7, 2023

1 Description of the solution

The implementation is based on alpha-beta pruning. When it is the students turn to make a move the function student_move is called with the current state of the board as an input. The minimax function is then called with a copy of the state, the depth (that is set to 5), alpha, beta and the boolean isMaximizingPlayer as inputs. Alpha is originally set to negative infinity, beta to infinity and isMaximizingPlayer to True.

The minimax function has three cases: isMaximizingPlayer is True, isMaximizingPlayer is False and if depth is zero or one of the players has made a move that results in a win. The function game_over evaluates all the vertical, horizontal and diagonal four-connected segments. If one of the four-connected segments equals a 'winning score' for one of the players, meaning that one of the players has four in a row, the function returns True, otherwise False.

Each time that the minimax function is called recursively the depth is subtracted by 1, meaning that if none of the players has won the game after 5 moves the recursion stops and returns the best move with regard to the score. The board in the input of minimax within the function is a copy of the board that the function was called with, where one move has been added in order to evaluate the value of that move. The minimax function returns a list of the best value of the score so far for each call of the function along with the move that resulted in that value.

The evaluate function returns the total score of the board. This score is calculated by checking all the four connected segments on the board and adding a score depending on what values that are in each segment. The evaluate function returns the total score of all four connected segments. The minimax function will return the column that gives the best value out of all the available columns. The evaluate function uses the get_vertical_score, get_horizontal_score and the get_diagonal_score functions just as the game_over function does. What differs between them is that the evaluate functions purpose is to return the total score of the board while the game_over functions purpose is to check if there is any four-connected segment that results in a win or a loss.

Below is the implementation of the minimax function and the alpha-beta pruning.

```
def minimax(board: np.array, depth: int, alpha, beta, isMaximizingPlayer: bool) -> list:
    """
    Minimax algorithm with alfa-beta pruning.
```

```
Returns the value of the board and the best move.
"""

if not depth or game_over(board, True) or game_over(board, False):
   return evaluate(board)

if isMaximizingPlayer:
   bestVal = [0, np.NINF]
   for col in get_valid_cols(board):
      testBoard = copy.deepcopy(board)
      testBoard = add_move(testBoard, col, isMaximizingPlayer)
      _, val = minimax(testBoard, depth-1, alpha, beta, False)
      if val > bestVal[1]:
         bestVal = [col, val]
      alpha = max(alpha, bestVal[1])
      if alpha >= beta:
         break
   return bestVal
else:
   bestVal = [0, np.inf]
   for col in get_valid_cols(board):
      testBoard =  copy.deepcopy(board)
      testBoard = add_move(testBoard, col, isMaximizingPlayer)
      _, val = minimax(testBoard, depth-1, alpha, beta, True)
      if val < bestVal[1]:
         bestVal = [col, val]
      beta = min(beta, bestVal[1])
      if alpha >= beta:
         break
   return bestVal
```

2 How to launch and use the solution

To launch the solution it is important to first make sure that all the required packages are installed. If any of the required packages are missing they can be installed using pip install "the required package". Packages that are required are gym, random, requests, numpy, argparse, sys and copy. To launch and use the solution against the server run the script from the terminal and end the request with '–online'. The optional arguments are:

-h, –help show this help message and exit
-l, –local Play locally
-o, –online Play online vs server
-s, –stats Show your current online stats

The status of the board will be printed after every move.

3 Peer-review (1-2 pages)

I did my peer review with Daniel Björklund, with stil-id: da1452bj-s.

3.1 Peer's Solution

Daniel has as the assignment instructions suggested implemented a solution based on a minimax function that uses alpha-beta pruning. The minimax function is called from the student_move function with the input arguments "alpha", "beta", "maximize", "depth" and "state", where he has chosen the values of alpha and beta to 100000 and -100000, maximize as True and depht to 5. "State" is the current state of the board.

If the depth is equal to zero he returns the evaluation function. If the depth is not equal to zero he adds all possible moves/children to a matrix with the same size as the board and adds these matrices to a list. When all possible moves are added he iterates through the possible matrices where he calls the recursive maximize function for each move and a value is returned and the best value is kept using alpha-beta pruning.

3.2 Technical Differences of the Solutions

Daniel's solution differs a bit from my solution even if the overall idea is similar. He has chosen to write the code in a quite different way. The value of the board is only evaluated in the evaluation function and will only do so once the depth equals zero. Which means that if one of the players wins the recursion will continue until the depth is zero.

Like my solution, Daniel evaluates the score of the board by examining the vertical, horizontal, and diagonal segments. However, his approach differs in that he does not consider the presence of an empty spot next to the player's moves. He verifies if there are two or three 1's or -1's in a row but does not check if the adjacent spots are unoccupied. In my opinion, if these spots are already taken by the opponent, this state should not be assigned a high score, as the possibility of these segments resulting in a win is absent. Additionally, he returns the value as soon as a segment of two or three in a row is detected, leading to only four possible scores and the highest score may not necessarily be the best option if the overall board value is not considered. In contrast, I would suggest adding onto the value instead of returning it immediately upon finding any of the cases.

Furthermore, I noticed that Daniel has written the diagonal, horizontal, and vertical evaluations within the evaluation function, duplicating the code four times, once each for two in a row by the student, three in a row by the student, two in a row by the opponent, and three in a row by the opponent. The code would be easier to understand if these calculations were performed in separate functions and written only once each for diagonal, vertical, and horizontal cases. This would allow him to concentrate on the calculation of values for each case in the evaluation function.

3.3 Opinion and Performance

- Which differences are most important?.

In my opinion the difference where he doesn't take in consideration the spots next to the 2 in a row or 3 in a row is the most important. This will result in a high score even if the current state of the board isn't the best. Another thing that is important is that he needs to implement a solution to when one of the players has won the game even if the depth hasn't reached zero. Otherwise the recursion won't stop, this will save time, but the implementation would work if a the value for when a player gets four in a row is a lot higher than the scores for the other cases.

In my solution I increase the value of the board if the move is made in the centered column, since this increases the chances of getting four in a row since the options are wider. This was not considered in Daniels code.

- How does one assess the performance?

  The performance can be assessed by checking the longest streak, and how long it takes for the student player to make a move. Unfortunately there is something strange going on in the code that I haven't been able to localize, the student makes an illegal move some of the times. And the times that the student doesn't make an illegal move the game is only won by having 4 in a row vertically, or lost. I have not managed to get a 20 streak yet.

- Which solution would perform better?.

  Considering the comments above my solution currently has a better performance.

4 Paper summary AlphaGo

The paper "Mastering the Game of Go with Deep Neural Networks and Tree Search" by Silver et al. presents a novel algorithm for playing the ancient and complex game of Go, which was considered one of the most challenging games for computers to win due to its large search space and complex evaluation function. The authors propose using a combination of deep neural networks and a tree search algorithm to build an AI system that can play Go at a superhuman level.

The deep neural network in AlphaGo was trained on a large dataset of expert human games, and it was used to evaluate the state of the board and predict the best move to make at any given time. The tree search algorithm then used this information to search through possible future moves and find the best sequence of moves that leads to a win. The authors note that the combination of deep neural networks and tree search algorithms allowed for more accurate and efficient evaluation of the game state, leading to improved gameplay compared to previous computer Go programs.

In a significant demonstration of the power of their algorithm, AlphaGo was able to beat the European Go champion in a tournament, and it outperformed the best existing computer Go program. This was a major milestone in the field

of artificial intelligence and demonstrated that deep neural networks and tree search algorithms can be combined to create powerful AI systems that can excel at complex, strategic games like Go.

The authors conclude that the approach used in AlphaGo can be applied to other games, as well as to other problems where the search space is large and the evaluation function is complex, such as in finance and logistics. They also note that the development of AlphaGo highlights the potential for machine learning algorithms to advance our understanding of complex systems and help us solve difficult problems in new and innovative ways.

In summary, this paper presents a significant contribution to the field of artificial intelligence and provides a promising solution for solving complex, strategic problems. The success of AlphaGo in mastering the game of Go serves as a testament to the power of deep neural networks and tree search algorithms, and it opens up new possibilities for applying these techniques to other challenging problems in various domains.

4.1 Difference to the own solution

The main difference between AlphaGo and the Alpha-Beta Pruning algorithm is that AlphaGo uses Deep Neural Networks to make predictions and Monte Carlo Tree Search to search for the best moves, while the Alpha-Beta Pruning algorithm only uses a Minimax function to evaluate the game tree. AlphaGo was trained on a large dataset of Go games to learn the strategy of the game, while the Alpha-Beta Pruning algorithm does not require any training. It is also capable of adapting to new situations and making novel moves, while the Alpha-Beta Pruning algorithm only evaluates predetermined game states and moves.

4.2 Performance

In general, AlphaGo is a much more complex system that is designed to play the ancient and complex game of Go, while the minimax algorithm with alpha-beta pruning is a simpler solution that is typically used for solving games with smaller search spaces and simpler evaluation functions.

For Connect Four, a solution with the minimax algorithm and alpha-beta pruning is likely to be more efficient and faster than AlphaGo, as Connect Four is a relatively simple game with a small search space and a simple evaluation function.

If computational resources and time are limited, a solution with the minimax algorithm and alpha-beta pruning might be the better choice. If the goal is to develop a highly competitive AI system that can play Connect Four at a superhuman level, a more complex solution like AlphaGo might be required.