Artificial Intelligence and Knowledge Engineering
Assignment 2

Jan Jakubik, Piotr Syga, Daria Dziubałtowska

March 17 2025

1 Excercise goal

The purpose of the exercise is to practically learn algorithms for playing two-player games - the Minimax algorithm and its improvement using alpha-beta cuts. By working through this list, the students should develop their problem-solving skills and gain a deeper understanding of the underlying principles of zero-sum games.

2 Theoretical Introduction

Below you will find information to help you complete the task list. It is assumed that upon completion the student has mastered the theoretical issues. In this section you will find information useful during solving the stated problems. It is assumed, that after finishing the exercise students are fully knowledgeable about theoretical concepts described below.

2.1 Definitions

Definition 1 (Decision tree)**.** *A directed tree in which each node represents a state and each edge represents an action or a choice that leads to the next state is called a decision tree.*

A decision tree can be used to model problems in which decisions must be made, considering many possible options.

For games, decision trees are often used to represent possible moves in a game. Each node in the tree represents the state of the game, and the edges correspond to possible moves or decisions the player can make. Thus, it is easy to see that the degree of a vertex corresponds to the number of possible moves, and therefore does not have to be constant at one level of the tree. When a player makes a move, we move towards the leaves of the tree to the node corresponding to the new state of the game. The end of the game occurs when reaching a leaf.

The length of the path (corresponding to the number of rounds) can be different for different leaves.

Definition 2 (Zero-sum game)**.** *An ordered triple is called a game* $G = (N, \mathcal{S}, U)$*, where* N *denotes the number of players,* $\mathcal{S} = S_1, S_2, \ldots, S_N$ *is a finite set of strategy sets for the respective players* 1 *up to* N*, and* $U = u_1, u_2, \ldots, u_N$ *is a set of the utility (gain) functions for the specific players that assigns a value to each of the possible movements' combination, i.e.,* $u_i : S_1 \times \cdots \times S_n \to \mathbb{R}$ *is the gain of the player* i *while all the players use their respective strategies* S_j*.*
If $\forall s \in S_1 \times \cdots \times S_n, \sum_{i \in N} u_i(s) = 0$*, then we call a game, a zero-sum game.*

2.2 Minimax algorithm

The Minimax algorithm is a game tree search strategy that is used to make decisions in two-player zero-sum games such as chess, checkers, and go. The algorithm seeks to minimize the maximum possible loss a player can suffer, given the opponent's moves. The Minimax algorithm is based on viewing the game tree, where each vertex of the tree represents a game state, and the edges between them represent the players' moves. The leaves will be the vertices representing the end of the game won by one of the players. In practice, due to the number of vertices increasing exponentially with depth, we use a depth limit and heuristics for evaluating the game state, allowing us to evaluate the unfinished game if we reach this limit.

The main steps of the algorithm involve:

1. Starting from the root, search the game tree.
2. If we have reached the leaf (or the set depth), we return a value for the player who is currently making a move. For the leaves, it is the score of the game, whereas reaching the search depth limit - returns the value of the evaluation heuristics.
3. Otherwise, in a given state, we calculate the value of each of the node's successors by recursively running the Minimax (starting from the 2 step).
4. We assign the current vertex the maximum or minimum value among its descendants, depending on which player this vertex represents.

The Minimax algorithm can be optimized using alpha-beta pruning, which involves stopping the recursive browsing of a certain part of the tree when certain conditions determine that further browsing no longer makes sense, and the result will not be improved.

2.3 Alpha-beta pruning

The alpha-beta pruning algorithm is an extension of the Minimax algorithm, which allows you to skip considering unnecessary nodes in the game tree and

thus reduce search time. The alpha-beta pruning algorithm works by eliminating branches of the tree that are certainly not optimal, as they contain only moves that will not improve the value function for a given node. This algorithm significantly reduces the number of nodes that must be browsed when searching the game tree.

The main steps of the algorithm:

1. Execute the Minimax algorithm on the game tree, searching it recursively from the root.
2. Store two values: alpha and beta, initially set to minus and plus infinity, respectively.
3. For the current tree vertex, if it is a maximizing level vertex, follow steps 4.-6. Otherwise, follow steps 7.-9.
4. For each descendant of the vertex, call the alpha-beta cutting algorithm recursively (from step 3).
5. If r, the value returned by the algorithm is greater than alpha, update the alpha value to r.
6. If the beta value is less than or equal to alpha, stop browsing the descendants and return the alpha value.
7. For each vertex descendant, call the alpha-beta cutting algorithm recursively (from step 3).
8. If r, the value returned by the algorithm is lower than the beta value, update the beta value to r.
9. If beta is less than or equal to alpha, stop browsing the descendants and return the beta value.

3 Tasks

Originally, game was designed on a 5x6 board, but more and more games are being played on a 10x10 board. Each player has his own color. At the beginning, the board is completely covered with disks, so that black disks are placed on black fields, and white disks are placed on white fields. A black player starts the game. Players move alternately. The move can only be made to a orthogonally adjacent space occupied by the opponent's disk. Once the move is made, the opponent's disk is "clobbered"and removed from the board, and the player's disk takes its place. The game ends when there is no more moves for any of players. The winner is the player that moved the last.

Using the information in this list and those given in the lecture, write a program that plays Clobber (board size nxm=5x6, can be modified). The program should, on the standard input, accept a starting board (e.g., the default starting

board for the Clobber game), allow the selection of heuristics, and adjust the parameter d which is the maximum search depth of the decision tree. The board representation should has the form of m lines consisting of n space-separated marks – B denotes player one's disk, W denotes player two's disk, _ denotes an empty field. Player ones's turn follows. After starting the game, the program should:

- Standard version – play the entire game using the Minimax algorithm from the perspective of player one. Player two always makes the optimal move (both players use the same heuristics, so player one's prediction of the opponent's move is always accurate).
- Extended version (for additional 20 points) – program will allow for the game to be played between two separate agents (the opponent does not always have to act optimally, players can change strategy).

The standard output should return the board representation of the finished game. Then, the information about the game's winner and the number of rounds should follow in the next line. The standard error output should return the number of decision tree nodes visited and the running time of the algorithm.

Grading:

1. Defining the game state and the function that generates possible moves for the current state and player. (10 points)
2. Building a set of heuristics to assess the state of the game for each player, each player should have at least 3 different strategies. (20 points)
3. Implementing the Minimax algorithm for *Standard version* of the program. (30 points)
4. Implementing the alpha-beta pruning for *Standard version* of the program. (40 points)
5. Modification of the program to the *Extended version* which will allow the game to be played between two programs (e.g., two executions of the same application) and the use of heuristics that adaptively change the player's strategy to achieve victory. (**additional** 20 points).

Prepare a report including a theoretical description of the method, a precise formulation of the problem with constraints, game state, the decision tree, and a description of the solution idea with a brief example. In the report, indicate the source materials used and briefly describe the libraries used in the implementation. In the report summary, describe the implementation problems encountered in performing the task. Send the report to the instructor at least 24 hours before handing in the list.

4 Bibliography

- Clobber
- Clobber - opis zasad
- J.F. Nordstrom – Introduction to Game Theory
- M. Maschler et al., Game Theory
- N. Nisan et al., Algorithmic Game Theory
- M. J. Osborne, An Introduction to Game Theory