Artificial Intelligence and Knowledge Engineering
Assignment 2

Jan Jakubik, Piotr Syga

March 25th, 2024

1 Goals

This assignment is a practical introduction to game playing algorithms for two player full information games – MiniMax algorithm and it's improvement through alpha-beta pruning

2 Introduction

Below you will find information which may be helpful in completing the task list. We assume that after finishing the tasks, student has a grasp of theoretical aspects of the assignment.

2.1 Definitions

Definicja 1 (Decision tree)**.** *Decision tree is a directed tree in which every node represents a problem state and every edge represents an action or a choice, leading to another state.*

Decision trees can model problems in which actions are taken by selecting one from a set of possible options.

In case of two-player games, decision trees are often employed to represent possible moves. Every node represents the state of the game, and every edge represents a player move, which results in a node transition. The degree of a node corresponds to the number of possible moves, and might vary between game states. When the player makes a move, we move along an edge to another node. Leaf nodes represent possible endings of the game. Leaf depth, which corresponds to the length of a game in rounds, may be variable.

Definicja 2 (Zero sum game)**.** *A game is an ordered triple* $G = (N, \mathcal{S}, U)$*, where* N *indicates the number of players,* $\mathcal{S} = S_1, S_2, \ldots, S_N$ *is a set of finite sets of player strategies* 1 *to* N*, and* $U = u_1, u_2, \ldots, u_N$ *is a set of payout*

functions which assign result values to every combination of player actions. Specifically, $u_i : S_1 \times \cdots \times S_n \to \mathbb{R}$ *is the payout of player* i *assuming given player strategies.*
If $\forall s \in S_1 \times \cdots \times S_n, \sum_{i \in N} u_i(s) = 0$*, the game is a zero-sum game*

2.2 MiniMax algorithm

MiniMax is a game tree search algorithm designed for zero-sum, nonrandom, full information two-player games (eg., checkers, chess, go). The algorithm minimises a maximal possible loss the player may incur given the enemy strategy. MiniMax algorithm is based on tree search in which every node represents a game state and every edge represents a player move. Leaf nodes represent the victory or loss of a player. In practice, due to exponential growth in number of nodes with increasing depth, we use a predefined search depth limit and a heuristic evaluation of the game state when this limit is reached.

Steps of MiniMax:

1. Start the search from the tree root.
2. If the current node is a leaf or the depth limit is reached, return a node evaluation. For a leaf, this is the game result, at the depth limit it is a heuristic evaluation of the game state.
3. Otherwise, calculate the values of all child nodes of the current node by recursively running MiniMax (starting with step 2).
4. Assign the maximal or minimal value from its children to the current node, depending on which player the node represents.

MiniMax can be optimised using alpha-beta pruning, which stops the recurrent search of certain parts of the tree once we can establish further search in a particular branch is unnecessary.

2.3 Alfa-beta pruning

Alpha-Beta pruning is an extension of MiniMax, which allows the search to skip unnecessary evaluations and thus shortens search time. Alpha-beta pruning eliminates the branches which we can certainly say are sub-optimal because all moves they could potentially contain cannot improve the current node node value. The algorithm can significantly reduce the number of nodes visited during search

Steps of the algorithm:

1. Start MiniMax search in the tree root.
2. Initialise the alpha and beta values to plus and minus infinity respectively.
3. Perform steps 4-6 if the current node is the maximising player, 7-9 if it is minimising.

4. For every child of the current node, recursively evaluate with alpha-beta (from step 3).
5. If the value returned is greater than alpha, update alpha with that value.
6. If beta is lower or equal than alpha, stop the search and return alpha.
7. For every child of the current node, recursively evaluate with alpha-beta (from step 3).
8. If the value returned is lower than beta, update beta with that value.
9. If beta is lower or equal than alpha, stop the search and return beta.

3 Tasks

Halma is board game for 2-4 players using an 16×16 board. The game uses pieces in different colours, one colour per player. The game rules may be found, e.g., in: `https://en.wikipedia.org/wiki/Halma`

Using the information provided in this list and in the lecture, write a program that plays Reversi. The program should accept inputs in form of 8 lines of numbers – 1 is a white player piece, 2 is a black player piece, 0 is an empty square. Assume there is an even number of pieces on the board and the whites are moving. The program should output the final board state, the winning player, the runtime and the number of evaluated nodes.

Scoring:

1. Defining the board state and a function that generates possible moves from a given board state (10 points)
2. Building a set of game evaluation heuristics. Each player should have at least 3 different strategies. (20 points)
3. Implementation of MiniMax algorithm (30 points)
4. Implementation of alpha-beta pruning (40 points)
5. Modification of the program to play a single move of a given player, in order to allow two computer players implemented by different students to play against each other. (**bonus** 20 points).

Prepare a report which includes the theoretical description, a formal definition of the game state and the decision tree and your concept of a solution. The report should include references to sources and any libraries or external code used in the implementation. In the summary describe any problems you've encountered while solving the task. The report should be sent 24h before presentation of the assignment in class.

4 Literature

- Halma
- 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