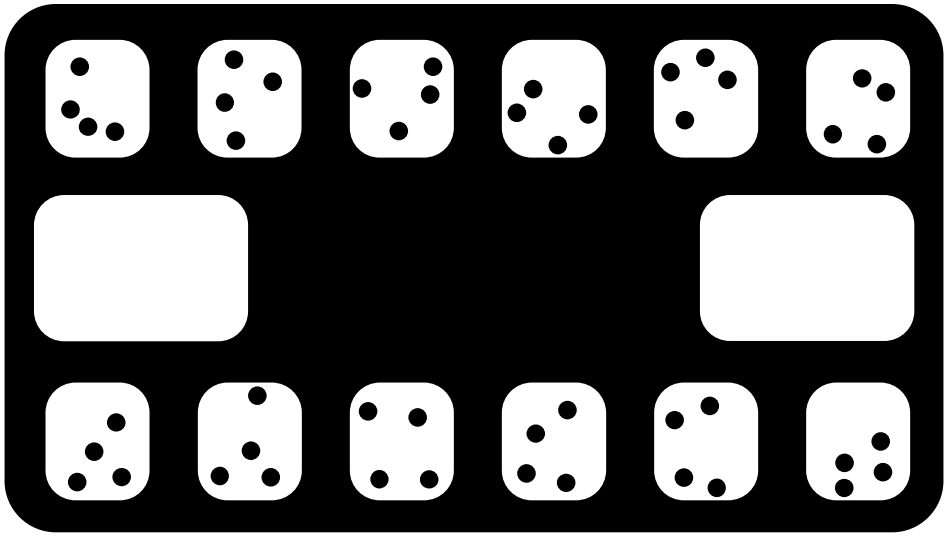
Board Game Assignment

March 2024

Introduction to AI
02180

Irene Valentina Berganzo s223230
Jiwoo Eom s231567
Sara Riegels Trangbæk s174932
Tobias Sætervoll Vangsy s234113



Contents

1 Choice of game - Mancala

1.1 Game instructions

In this section the rules of Mancala will be explained. There might be different versions and rule sets out there, however these are the rules that will be used in this project. Mancala is a game for 2 players. The game consists of 1 Mancala board and 48 plastic balls. 12 pockets are located at the Mancala board and there are two end zones called "Mancala store".

Initial game state
The initial game state is shown in figure 1.

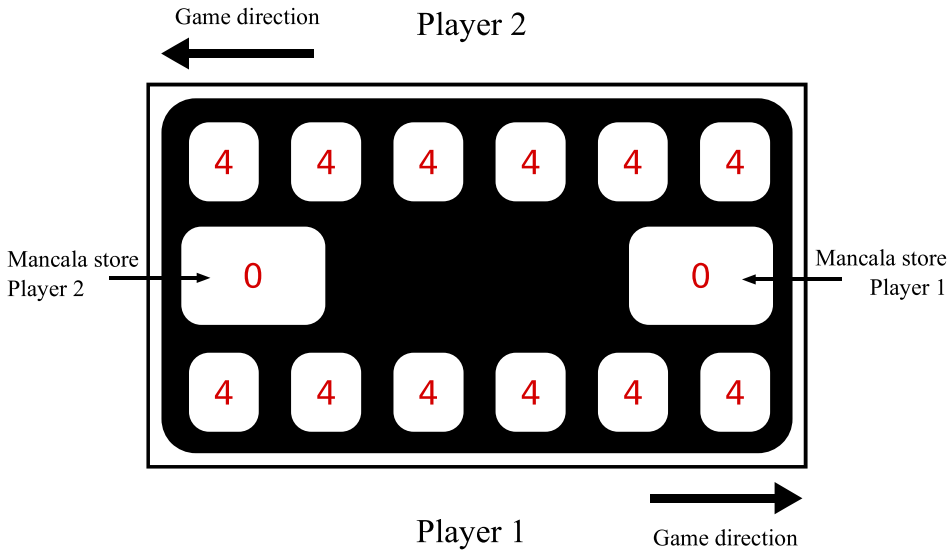


Figure 1: Initial game state

Game objective
The player with the most plastic balls in their Mancala store at the end of the game wins.

How to play

1. Player 1 picks up all the pieces in one of the pockets on their own side.
2. Then the player goes counter clockwise while dropping one ball in each pocket. The player gains a point(ball) when they reach their own Mancala store. The player doesn't drop one into the opposing players Mancala store.
3. If the player only has one ball left and drops it into an empty pocket on their own side, they obtain all the pieces on the opposite side and the ball they dropped into the empty pocket.
4. If the last ball is dropped in their own Mancala store, the player takes an extra turn.

5. The obtained pieces are transferred to their Mancala store
6. The game is over when all pockets are empty on one side of the Mancala board.
7. If a player still has balls left on their own side when the game ends, they are transferred to their own Mancala store.
8. Pieces are counted and the player with most pieces wins

Figure 2 shows the Mancala board when the game has ended, which is a terminal state. In this game player 1 wins, because the balls that are left on their side are transferred to their Mancala store, resulting in player 1 having 26 balls.

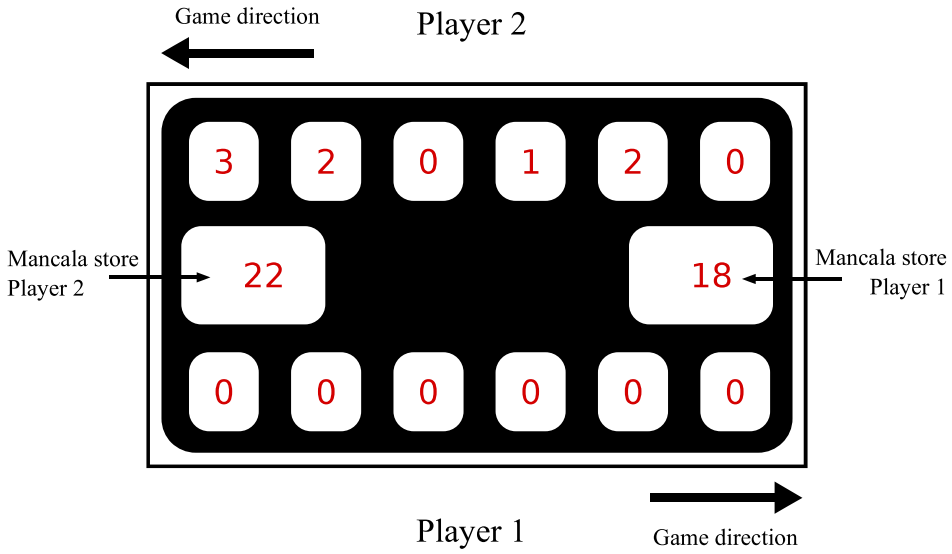


Figure 2: Terminal game state

2

2.1 Approach for developing an AI opponent

In this section the development approach will be explained. First the tactics used by human players will briefly be explained before diving deeper into the available methods for implementing AI.

How do humans play - Reasonable moves
This is highly dependent of each players game knowledge and intuition. Good players try to think ahead of time while other players go for short term gains. It is often seen that humans tend to go for the pockets with the most balls in. Some players try to calculate the possible moves on the board, however this is a time consuming task for most and they might make

errors. This is where AI comes in.

How does AI play - Calculated moves
By implementing various search algorithms, the outcome of countless moves can be evaluated in an instant. Depending on the size of the state space of the game this might take more or less computational power, and can in some cases require intelligent solutions to evaluate game positions without searching through/to the end of every branch.

Game theory
In game theory all actors are assumed to be rational, logical and are aware that the other agents are trying to maximize their outcome, and that agent 1 knows that agents 2 knows that agent 1 is trying to maximize their outcome and so forth. Thus game theory is a mathematical way to analyze strategic behavior.

Mancala is a turn based game with perfect information, meaning that both players have access to all information about the game state. It is a zero-sum game, meaning that a gain for one player results in an equal loss for the other. Furthermore the game is fully deterministic, which will be important to keep in mind when deciding on which algorithm to use. Since Mancala is a deterministic game, there will be a path leading to a solution in the state space. Thus the solution can be defined by a sequence of actions, which we call a sequential plan.

If the Mancala stores are not taken into account, the state space with only the 12 pockets can be calculated knowing there can be 0 or 4 balls in each pocket and there are 12 pockets:

$$Size(Statespace) = 5^{12} = 244.140.625 \tag{2.1}$$

The search tree with only the initial node expanded is shown in figure 3.

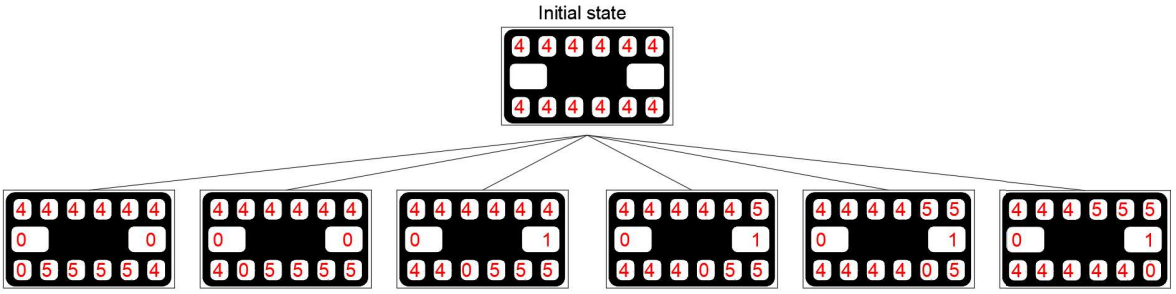


Figure 3: Graph search with only initial node expanded

This shows that in the first rounds the branching factor is 6, and for the rest of the games the branching factor will have a factor between 1 and 6.

2.2 Pseudocode

Initially the tree search problem can be formulated as follows:

- S_0: Initial state
- Actions(s): Return all possible actions in the state "s". Eq return legal actions which is to take the balls on their own side in one of the pockets that are not empty.
- Results(s, a): Returns new state reached from previous state "s" by performing action "a"
- Terminal test(s): Return true if current state is a terminal state. E.g. return true if either side of the board is empty.
- Goal test(s): Return true if state is a leaf node and results in a win

The minimax algorithm will be applied to solve the search tree. The state space is too large to be completely traversed, thus a search depth will be applied. The way the best moves will be evaluated in the algorithm is by implementing an evaluation function that calculates the difference between the amount of pieces in Player 1 and Player 2's Mancala store at some specified search depth. Thus the evaluation function for a certain state "s" can be written as follows:

$$eval(s) = score_2(s) - score_1(s) = \textit{Mancala store AI(s) - Mancala store Player 1(s)}$$

An example of how the minimax algorithm with this evaluation function works is explained in figure 4. The blue numbers are the results of the evaluation function.

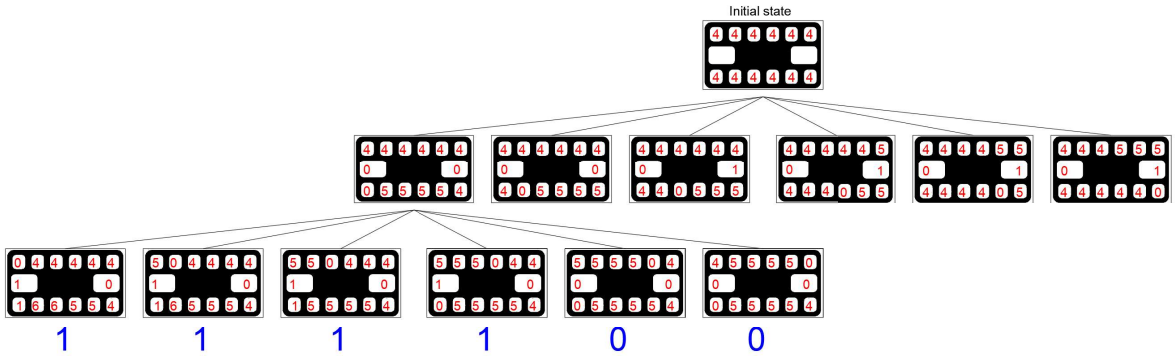


Figure 4: Minimax example with evaluation function at search depth 2 with only left node expanded

In the example above, if the first player (Min) decides to perform the action resulting in the left most node, then the AI Agent (Max), which is trying to maximise it's outcome, can ensure an evaluation score of 1. If some of the other nodes were expanded instead, it would be seen that the first player (Min) can ensure that the evaluation score would be 0.

2.3 Benchmarking

Benchmarking against random agents
Initially the algorithm will be benchmarked against a player that makes random moves, which will be called Random Agent. The AI Agent uses the minimax algorithm with Alpha-Beta pruning. The benchmarking is done with various search depths. All search depths are tested with 100 games.

Table 1: Benchmarking against Random Agents with variable search depths

Search depth	AI wins	Random wins	Tie	run-time
1	87	11	2	0.2 sec
2	91	8	1	1.0 sec
3	92	6	2	2.7 sec
4	94	4	2	5.8 sec
5	97	2	1	12.5 sec
6	98	2	0	30.4 sec
7	97	3	0	76.0 sec

From the results in table 1 it can clearly be seen that the AI agent performs better than an agent making random moves. It is also noticed that the value of the search depth has an impact. The increasing search depths also (almost consistently) result in a higher win-rate for the AI. The downside to the high search depth is the computational time which increases rapidly. In this case an appropriate search depth would be 5, since the computation time is manageable and the AI wins as many times as for search depth 6 and 7.

Benchmarking against human agents
Another approach to the benchmarking is the testing against humans. Amongst the group members, the game has been testet and the result can be seen in Table 2.

Table 2: Benchmarking against human agents

Amount of games	AI Agent wins	Human Agent wins	Tie
30	19	10	1

During the test, it was noticeable the difference between the skilled players who knew the game since childhood, and the newly introduced participants.From the results in table 2 it can be seen that the AI agent performs smart on the game, as it was an experienced player.

2.4 Parameter testing

As stated before, an evaluation function has been implemented in the AI in order to ensure the quality of the state on the game, and therefore guide the AI to the decision-making process in 2.2 (defined as "*evaluate_ board(board)*" in the code). This way, the AI will be able to follow strategic choices. The search depths tested are from 1 to 7, which determines how many moves ahead the AI looks for decision of the next move in table 1. This two parameters could be adjusted to alter the performance of the AI. It was also tested against both, a random agent and a human agent.

The Minimax algorithm has been implemented with the alpha-beta pruning, which increases the efficiency of the Minimax by itself, as stated in 2.3.

2.5 Adjustments to AI

Upon reviewing the benchmarks against human players, it's evident that it's relatively easy to discover a specific sequence of moves that guarantees victory for the human player every time. By following the sequence: a3 - a6 - a2 - a6 - a1 - a6 - a3 - a6 - a3 - a2 - a6 - a1, player 1 can secure a win consistently. This outcome assumes that player 1 (the human) initiates the game. To enhance fairness and unpredictability, a potential modification could be to introduce randomness in determining which player, whether the AI or the human, starts the game as player 1.

2.6 Future Work

Optimization of Search Algorithms
While the current implementation utilizes the minimax algorithm with alpha-beta pruning, further optimizations could be explored to enhance its efficiency. Techniques such as iterative deepening, transposition tables, or parallelization could be implemented to improve AI's performance without changing the underlying algorithm.

Enhanced Evaluation Function
Developing a more sophisticated evaluation function could significantly improve the AI's strategic decision-making. This could involve incorporating domain-specific knowledge, such as recognizing advantageous board configurations or considering potential future moves beyond immediate outcomes.

Exploration of Alternative Algorithms
While minimax with alpha-beta pruning is a common approach for game-playing AI, exploring alternative algorithms such as Monte Carlo Tree Search (MCTS) or reinforcement learning could offer new insights and potentially superior performance. These methods may require different data structures and computational techniques but could lead to more adaptive and robust AI opponents.