Technical Report

Lucas BESSON - Nathan BOSCHI - Robin HALM - Mathys NOURRY - Tom SIOUAN

SAE 1.2.6 - Développement IHM – Nine Men’s Morris Game

# Nine Men’s Morris Algorithms

## First algorithm – Minimax algorithm

### Presentation

#### Placing pieces

During the initial phase of the game, the algorithm checks which mills the player can complete. If there is a mill that can be completed (line of three pawns), the algorithm completes it by placing the third piece in the mill. If there are no mills that can be completed, the algorithm checks if the opponent can complete a mill. If so, the algorithm blocks the opponent's attempt by placing a piece in a location that disrupts the potential mill. If neither player can complete a mill, the algorithm places a piece randomly on the board.

#### Moving pieces

During the second phase of the game, the algorithm uses the minimax algorithm to determine the best move. The minimax algorithm considers all moves for the current player and the opponent, up to a certain depth limit. The algorithm evaluates each move by assigning a score to the resulting board position. The score is based on factors such as the number of pieces in mills, the number of potential mills, and the number of pieces that are threatened by the opponent. The algorithm then chooses the move that leads to the board position with the highest score. If the depth limit is reached, then victory is not close. The AI chooses a piece randomly and moves it.

#### Removing pieces

If a mill (line of three pieces) is formed, the AI looks for the piece that reduces the opponent's chances of winning. To do this, it chooses a piece that would allow the player to form a mill in the next turn. If the other player cannot form a mill in the next turn, the AI chooses a counter randomly and removes it.

### Technical review

The minimax algorithm is a widely used technique for developing artificial intelligence in board games like Nine Men's Morris. In this algorithm, the computer evaluates all moves that it and its opponent can make and chooses the move that leads to the best outcome for itself. The algorithm works by recursively exploring the game tree, which represents all possible moves and their outcomes, and assigning a value to each node of the tree. The value of each node is determined by the minimax function, which alternates between maximizing and minimizing the node values depending on whose turn it is. This process continues until a predetermined depth or until a terminal node is reached, at which point the algorithm chooses the move that leads to the optimal outcome.

The performance of the minimax algorithm can vary depending on several factors such as the complexity of the game, the branching factor of the game tree, and the depth of the search. In terms of code volume, the algorithm can be implemented in a relatively compact form, typically requiring a few hundred lines of code (here 200 lines of code without utility methods). However, the memory usage can be quite significant, especially for games with large search spaces. The algorithm stores the game tree in memory, which can lead to memory overflow issues for deep search depths. To solve this problem, we have limited the search depth to avoid big and infinite sets.

The time required for the algorithm to make a move also depends on the game's complexity and search depth. For Nine Men's Morris, the average time required for the algorithm to make a move can range from a few milliseconds to several seconds, status of the game and the available computational resources. In practice, minimizing the search depth is a common way to speed up the algorithm's execution time. This approach sacrifices accuracy for speed, but it can be effective for games that require quick decisions.

Overall, the performance of the minimax algorithm can be optimized by carefully tuning the search depth and by implementing various optimization techniques such as alpha-beta pruning, transposition tables, and iterative deepening. These optimizations can significantly improve the algorithm's performance while reducing the memory usage and execution time.

### Performance evaluation

## Second algorithm

### Presentation

### Technical review

### Performance evaluation

# Entry files