Solving Nine Men's Morris as an Adversarial Problem

[*]Minimax variations to solve Nine Men's Morris (Group 02 - Project A2)

Ângelo Teixeira (up201606516)
FEUP
Porto, Portugal
up201606516@fe.up.pt

Henrique Lima (up201606525)
FEUP
Porto, Portugal
up201606525@fe.up.pt

Margarida Silva (up201606214)
FEUP
Porto, Portugal
up201606214@fe.up.pt

***Abstract*—Adversarial search is a common strategy for solving board games of two opponents. In this article we explore in detail the characteristics of the game *Nine Men's Morris*, and formulate them accordingly. The aim is to find the best ways of solving the game according to certain criteria such as the quality of the solution and time of execution. We have found other attempts with variations of *Minimax* algorithms and intend on developing some of our own, as future work.**

***Index Terms*—Adversarial Problems, Artificial Intelligence, Nine Men's Morris, Minimax algorithm, Alpha-Beta Pruning**

I. Introduction

Regarding board games, Adversarial Search is the most used approach for solving. *Nine Men's Morris* is an ancient board game which can be perceived as an Adversarial Search Problem. There are different variations of the Minimax algorithm, and possible parameters can be the state evaluation and the depth of the search. We aim on concluding on the most adequate variations to this specific case study.

In this document we present a thorough description of *Nine Men's Morris*, together with a detailed formulation of its characteristics, and how we tackled the problem, in order to create an Artificial Intelligence capable of playing the game and hopefully winning versus a human player.

We also present some experiments, regarding the parameters variation and their impact on the game winning rate and time consumption for the AI. Furthermore, we will also mention some related work done in the area and present possible future approaches.

II. Problem Description

Nine Men's Morris is a strategy board game for two players dating at least to the Roman Empire. The game has also been called cowboy checkers and is sometimes printed on the back of checkerboards. Nine men's morris is a solved game one in which either player can force the game into a draw.

The board consists of a grid with twenty-four intersections or points. Each player has nine pieces, or "men", usually coloured black and white. Players try to form 'mills'three of their own men lined horizontally or vertically. When they form such a line, they are able to remove any piece from the adversary. A player wins by reducing the opponent to two pieces (where they could no longer form mills and thus be unable to win), or by leaving them without a legal move.

The game proceeds in three phases:

- Placing men on vacant points
- Moving men to adjacent points
- (optional phase) Moving men to any vacant point when the player has been reduced to three men

A. *Phase 1: Placing pieces*

The game begins with an empty board. The players determine who plays first, then take turns placing their men one per play on empty slots. If a player is able to place three of their pieces on contiguous points in a straight line, vertically or horizontally, they have formed a mill and may remove one of their opponent's pieces from the board and the game, with the caveat that a piece in an opponent's mill can only be removed if no other pieces are available. After all men have been placed, phase two begins.

B. *Phase 2: Moving pieces*

Players continue to alternate moves, this time moving a man to an adjacent point. A piece may not "jump" another piece. Players continue to try to form mills and remove their opponent's pieces as in phase one. A player can "break" a mill by moving one of his pieces out of an existing mill, then moving it back to form the same mill a second time (or any number of times), each time removing one of his opponent's men. The act of removing an opponent's man is sometimes called "pounding" the opponent. When one player has been reduced to three men, phase three begins.

C. *Phase 3: "Flying"*

When a player is reduced to three pieces, there is no longer a limitation on that player of moving to only adjacent points: The player's men may "fly" (or "hop") from any point to any vacant point.

III. Problem Formulation

In this section we formulate *Nine Men's Morris* as an adversarial problem regarding the state representation chosen, considerations on initial state, goal testing, operators available and respective pre-conditions and effects, concluding on the state evaluation metrics.

A. State representation

The state representation used features the following information:

- *Active player* - Boolean value indicating which player should play next.
- *Number of Turns* - Integer value for each player indicating the amount of moves executed.
- *Player's Next Action* - enumeration value indicating the behaviour of each player's next turn. The enumeration includes the following:
  1) PLACING_PIECES
  2) MOVING_PIECES
  3) FLYING
- *Current Board* - (see below).

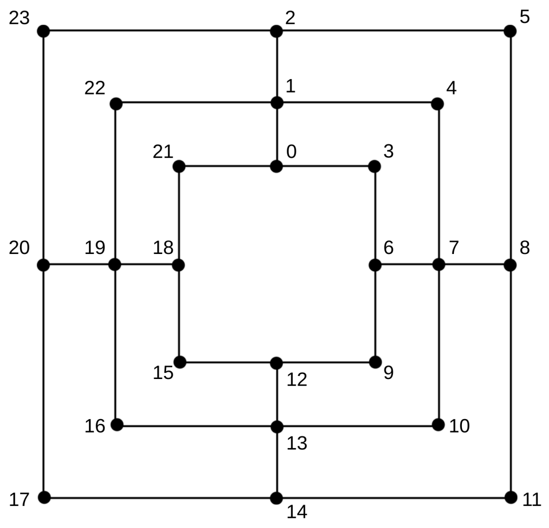

Fig. 1. Nine Men's Morris Board

The *Current Board* is represented through an array of bytes. Each byte is assigned to a node with a numeric value, where 0 represents an empty state, 1 represents a black piece and 2 represents a white piece. The array contains the nodes in the order according to the numbers in Fig. 1. This allows us to check for formed mills quite easily without sacrificing memory or time efficiency. Mills are then formed when:

- For a node of index i and $i \bmod 6 = 0$,

$$v_i = v_{i+1} = v_{i+2} \qquad (1)$$

  where v_i is the value of the node of index $i \bmod 24$.
- For a node of index i and $((i/3) \bmod 2 = 1$,

$$v_i = v_{i+3} = v_{i+6} \qquad (2)$$

  where v_i is the value of the node of index $i \bmod 24$. The calculation assumes an integer division (e.g. $5/3 = 1$).

B. Initial State

When the game starts, the initial *Active Player* is chosen by the user. The *Number of Turns* starts at 0 for both players and the *Player's Next Action* starts at PLACING_PIECES for both as well. Finally, the *Current Board* is initialized with all its bytes set to 0, as all the nodes are empty.

C. Goal Testing

The game ends when one of the two players loses one of their pieces off the board during the third phase (*"Flying")*, by which point the winner is the adversary player.

D. Operators

The operators include the following:

- Add piece - Add a player's piece to an empty node
- Move piece - Move a player's piece to an empty node adjacent to the current one
- Fly piece - Move a player's piece to any empty node
- Remove piece - Remove a player's piece from a node, making it empty

The availability of the operators depends on the *Player's Next Action*, which also depends on the current game phase (see *Problem Description*).

E. State Evaluation - Utility Function

As we have noted, one of the most valuable moves in the game is the formation of mills. That way, we have decided that the most valuable boards are the ones that can have a mill produced in the next play. Opposed to counting the current mills in the board, this strategy is better, because current mills have no benefits, as the players can only remove an opponent's piece immediately after forming a mill.

Another factor to take into account regarding a board's value is the difference of each player's number of pieces, as the goal of the game is to leave the opponent without legal moves, which happens more easily if the opponent has less pieces to work with.

To sum it up, the board's value can be calculated using the following expression:

$$Val = f_a * A + f_b * B \qquad (3)$$

where

- f_a = weight of parameter A (0, 1)
- A = number of mills that can be formed in the next play, by putting, moving, or flying a piece, depending on the game phase by the White player, subtracted by the Black player ones.
- f_b = weight of parameter B (0, 1)
- B = $N_{pieces_{white}} - N_{pieces_{black}}$

The **White** player will be trying to **maximize** the state values, while the **Black** player will trying to **minimize** the state values.

IV. Algorithms

A. Minimax

```
function MINIMAX($node, depth, maximizingPlayer$)
    if $depth = 0$ or node is a terminal node then
        return the heuristic value of node
    end if
    if $maximizingPlayer$ then
        $value \Leftarrow -\infty$
        for each child of node do
            $value \Leftarrow max(value, minimax(child, depth - 1, false))$
        end for
        return $value$
    else                                        ▷ *Minimizing Player*
        $value \Leftarrow \infty$
        for each child of node do
            $value \Leftarrow min(value, minimax(child, depth - 1, true))$
        end for
        return $value$
    end if
end function
```

B. Alpha Beta Prunning improvement

```
function MINIMAX_2(
        $node, depth, \alpha, \beta, maximizingPlayer$
)
    if $depth = 0$ or node is a terminal node then
        return the heuristic value of node
    end if
    if $maximizingPlayer$ then
        $value \Leftarrow -\infty$
        for each child of node do
            $value \Leftarrow max(value, minimax(child, depth - 1, \alpha, \beta, false))$
            $\alpha \Leftarrow max(\alpha, value)$
            if $\alpha \geq \beta$ then
                break
            end if
        end for
        return $value$
    else                                        ▷ *Minimizing Player*
        $value \Leftarrow \infty$
        for each child of node do
            $value \Leftarrow min(value, minimax(child, depth - 1, \alpha, \beta, true))$
            $\beta \Leftarrow min(\beta, value)$
            if $\alpha \geq \beta$ then
                break
            end if
        end for
        return $value$
    end if
end function
```

V. Related Work

Adversarial Search is a very common approach taken to board games. In this strategy, a tree of possible game states is generated and the next state is chosen by alternately maximizing and minimizing the utility function of a board, according to the respective player's turn.

A similar approach, Negamax, takes advantage of the $\max(a, b) = -\min(-a, -b)$ rule. By doing so, it always maximizes the result of the current level nodes, but recursively calls the negamax function negating it, which simplifies the minimax code, by taking the condition to maximize or minimize depending on the depth of analysis.

Alfa-Beta pruning is an optimization to the last two, in which the search nodes which can't possibly be better solutions (regarding the already explored nodes) are spared. This reduces the search space, and allows for a faster and more efficient way of executing the initial algorithm.

Regarding our specific case study, hence the game is ancient, there exist some approaches to it already using Adversarial Search. For instance, Gasser [3] has used the *Alfa-Beta* variation and concluded knowledge-based approaches are not beneficial. Andaloro [2] has most recently experimented using a *Monte Carlo Tree Search* on the solving of the problem, concluding on the lack of advantages compared to a pure MiniMax.

VI. Game Implementation

To implement the game logic, we created the Game class (*Game.js*), which handles to game cycle and manages the state and player actions with the help of Player and State classes. The Player class is used as an abstract class which is extended by the different types of players (Human, Random, Minimax). Each Player class extension implements its own way of playing: The human asks for the play and reads from the STDIN, validating the given slot. The random, selects a random move from the list of valid moves for the current state. The Minimax executes the minimax algorithm (with Alpha-Beta pruning) on the valid moves array, choosing the best option, depending on the depth of the algorithm, which is a property of the MinimaxAIPlayer object, so that we can have multiple MinimaxAIPlayers with different depths which translate to different levels of difficulty.

The State class contains the board information, players' information, as well as "mutation functions" such as placing, moving, flying and removing pieces from the board, which are called by the Players, returning a new state (the child state).

VII. Minimax

VIII. Experiments and Results

In order to evaluate performance of Minimax and its variants, we have carried out several testings on samples of 20 games. All the results presented represent the average of results. The adversaries used in this simulations were:

- R - Random
- M_1 - Minimax with depth = 1

- M_2 - Minimax with depth = 2
- M_3 - Minimax with depth = 3
- M_3 H_2 - Minimax with depth = 3 and a modified heuristic counting $0.2 * NumberofPiecesInBoard + 0.8 * ExistenceofMills$
- M_3w/p - Minimax with depth = 3 and without alfa-beta pruning optimization

A. Time and Explored Nodes

Hereby follows an analysis centered on solution quality measures, such as the time taken for solution choosing, for game winning and the explored nodes throughout the game for each play.

As seen in the table below and in Fig. 3 , as expected, more advanced adversaries take more time to win the game (due to its more complex state analysis for choosing) despite being able to win earlier in the game for doing better game choices. In the case of $M_3 + H_2$, there is an apparent sign that winning being quicker, hence the average time for winning the average number of moves are inferior. Nonetheless, this may not be sufficient to conclude on which heuristic is the best.

.	Random	M_1	M_2	M_3	$M_3 + H_2$
Time to win	60	21	355	550	430
Moves to win	43	16	13	12	10

Moreover, as seen in 2, the number of expanded nodes of a random player is far from being constant or even linear. Hence such player only chooses a random move from the set of valid moves, this also enables the analysis of the ramification degree throughout the game. This is due to the 3 different modes by which the game might be segmented, $PLACING$, $MOVING$ and $FLYING$, and in which different rules ought to be taken. We can conclude than on the initial stage the number of valid moves is very high, hence in the $PLACING$ phase one can put its peaces on whichever empty position.This number decreases as the game evolves to the $MOVING$ phase and continues decreasing throughout. In the end, with the entering on the $FLYING$ phase, this number quickly escalates hence there are no more restrictions to moving pieces rather than not moving to an occupied piece.

To conclude this topic, as can be seen, the use of Alfa-Beta pruning significantly reduces the number of explored nodes. An example of Minimax with depth 3 is at Fig. 3, and clearly evidences the advantages of such. This difference is even greater than it might seem in the graphic hence the scale had to be adjusted to logarithmic for better data perception.

B. Victories/Losses statistics

After analyzing all possible permutations of adversaries for plays, we are able to conclude on a few notions.

On one hand, we observe that, generally speaking, the results of a gameplay between adversaries of the same kind tends to a 50% chance of win/loss. I.e., the chance of a random AI player winning over another is nearly the same, and would theoretically tend to 50% with the sample size increase. An example of such phenomenon is present in 4.

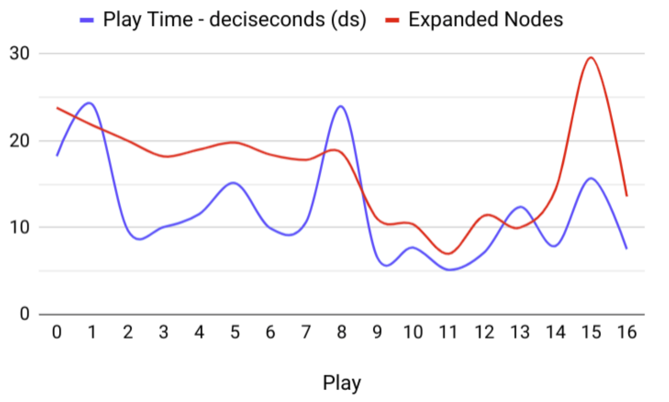

Fig. 2. Metrics for Random Player

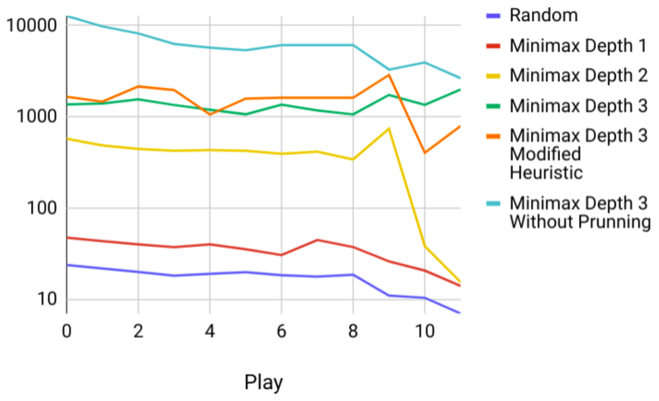

Fig. 3. Explored Nodes Throughout the Game (logarithmic scale)

Furthermore, it is also possible to state the clear likelihood for Minimax adversaries to win over the random AI by far, as expected. While the Minimax approach with depth 1 still looses 5% of the times(Fig. 5) , M2 and M3 with 100% of the times. There is also evidence for the likelihood of M3 winning over M1 (70%), but not as prominent. Finally, to our surprise. There seems not to be great advantage in using M2 over M1 or M3 over M2 in terms of winning rate. This might be explained due to the minor difference of depth levels, which is higher for the M3-M1 previous case.

Despite the apparent promising results for $M_3 + H_2$, we stated a tie in win/loss rate when compared to M_3. This means, despite tending to wind games quicker and in less moves, statistically does not overpower M_3 significantly.

1) Random VS Random:

IX. Conclusions and Future Work

Nine Men's Morris is a very ancient game, with several strict rules and with several previous attempts, throughout history. The knowledge of the game's specificity are key for devising the best of solutions.

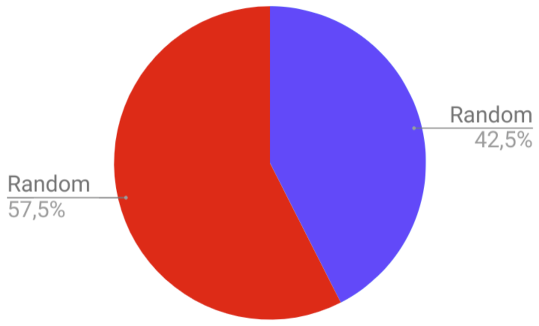

Fig. 4. Random Vs. Random

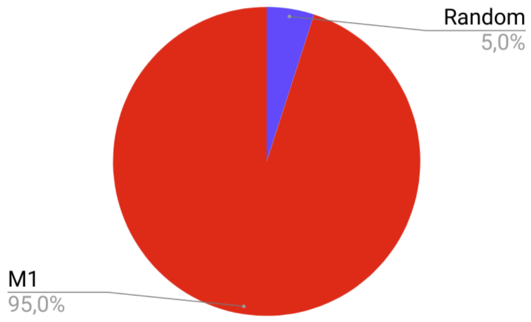

Fig. 5. Random Vs. Minimax with depth = 1

We conclude on the improvement of solutions with the depth used for minimax, and a considerable improvement on performance using alfa-beta pruning. We have also tested a different evaluation function than the naive number of mills counting, but haven't fond significant evidence of it being superior than the previous.

As future work problem formulations and paradigms could be further tested and compared to this adversarial search approach, such as more Machine Learning based approaches, such as Monte Carlo Search Trees.

References

[1] Stuart Russel and Peter Norvig, Artificial Intelligence: A Modern Approach, Third Edtition, Pearson Education Inc., 2010, ISBN: 978-0-13-604259-4.

[2] Andaloro, Sergio. "Monte Carlo tree search applied to Nine Mens Morris." (2016).

[3] Gasser, Ralph. "Solving nine men's morris." Computational Intelligence 12.1 (1996): 24-41.

[4] Heineman, George et al. (2008). "Chapter 7:Path Finding in AI". Algorithms in a Nutshell. pp. 213217. ISBN 978-0-596-51624-6.