CS 374 Project 1

Angelica Munyao

March 29th 2019

1 Introduction

A game is "a physical or mental competition conducted according to rules with the participants in direct opposition to each other" according to Merriam-Webster's definition of the word. In the context of artificial intelligence, a game is an environment in which the involved agents, entities interacting with an environment, are in competition with each other and must consider each other's actions in determining their own as they seek out a particular goal within the environment. The success of one agent implies the failure of other agents (Russell and Norvig, 161).

In determining the solution to a game problem, one must first define its parameters then determine the elements of a potential strategy that would result in a successful game-play. After establishing these details, one can then look into optimizing the solution by tweaking the elements of the chosen strategy, or seek out a better strategy altogether.

2 The Problem: Nine Men's Morris

I worked on establishing a solution to a game known as Nine Men's Morris in Python. Nine Men's Morris is a two-player strategy board game whose details are as follows (Wikipedia):

Board The game board consists of a grid of three concentric squares with four lines bisecting the sides of the squares. There are 24 possible locations on which a player can place a game piece: on the corners of the squares and at the intersection of the bisecting lines with the squares. The game begins with an empty board as illustrated below:

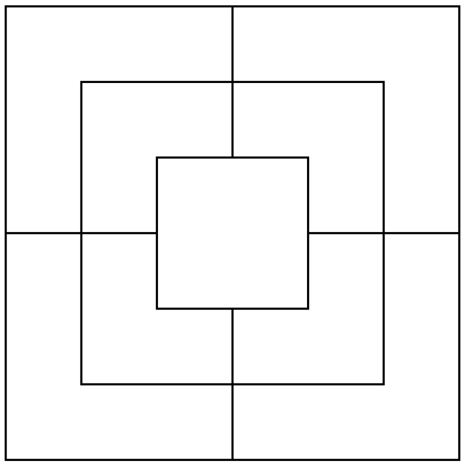

Figure 1: Nine Men's Morris Game Board

Game Pieces Every player in the game has 9 pieces to use at the beginning of the game. As the game progresses, pieces can be captured from the board by a player's opponent under certain conditions leaving a player with less than 9 pieces in total to work with for the rest of the game.

Mills When a player gets three pieces along the same line, which is on a square side or a bisecting line, on the board during their turn as a result of a move, they have what is known as a mill. This allows them to take off the board one of their opponent's pieces. Mills allow players to move towards a win in the game.

Game Play The game is divided into 3 phases:

Phase 1 Players begin with none of their pieces on the board and take turns placing their pieces on the board. They may place a piece on any vacant location.

Phase 2 Players have all their pieces on the board and can only move a piece to adjacent locations along the grid lines. During this phase, each player has more than 3 pieces on the board.

Phase 3 Players are down to 3 pieces and may move these pieces onto any vacant location on the board.

Mills Previously introduced, mills play an important role in moving a player towards a win. When a move results in creating a mill, it permits the player who made the move to take away an opponent's piece. However, once that mill has been used in this manner, it cannot be used again in another round unless one of the pieces is moved away then brought back to create the mill once more. A mill is only effective right when it is the result of a move made by the player during their turn.

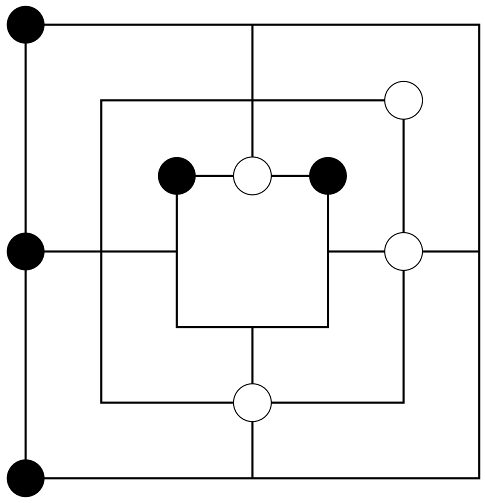

Figure 2: Nine Men's Morris Game Board in phase 2 with the black player having a mill. This allows them to capture one of the white player's pieces.

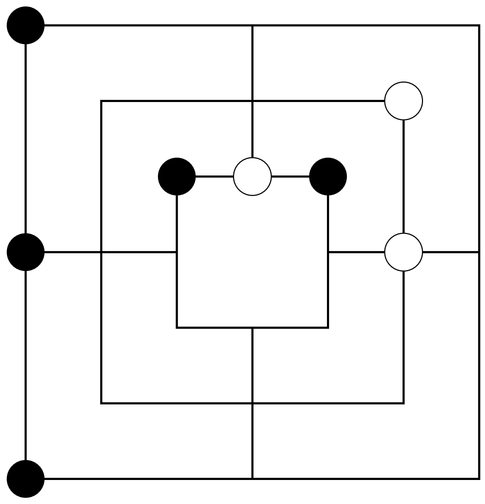

Figure 3: Nine Men's Morris Game Board in phase 2 with the black player having a mill and having taken a piece from the white player

Wins A player has won the game if their opponent has been left with only 2 pieces, or has no legal moves left to use. The latter case involves the opponent's pieces being blocked so they cannot move anywhere else on the board without violating the rules of the game.

Game Variations Nine Men's Morris has alternative versions, one of which is known as Six Men's Morris and has a smaller two-square grid that eliminates the possibility of mills along the bisecting lines:

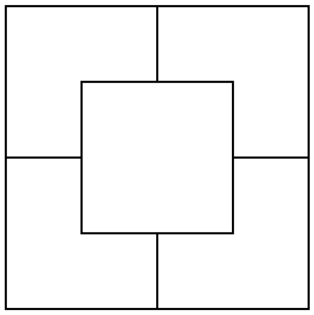

Figure 4: Six Men's Morris Game Board

The players have 6 pieces to use at the beginning of the game, and the game does not have the 3^{rd} phase of the original game. The variation came in handy to test out the possible solution with fewer conditions to consider.

3 Solution Approach

With Nine Men's Morris being a two-player game, the approach that came to mind is an agent algorithm known as the mini-max algorithm. It involves two agents competing against each other for the win. The agents are referred to as MAX and MIN. They take turns as the players of a game, choosing moves at each turn that result in the best outcome for them given their available options.

In determining the options available during a turn, the algorithm looks at all available moves and the result of making each move. For each result, which leads to the next turn, the same is done for the next player and their available moves. This goes on until we have all possible outcomes for completed games. From the very first turn until the final outcome options, the algorithm generates a tree structure representing the result of each available move at a turn. The turns are represented by the tree levels, and the nodes of the trees are the results of various moves made that lead to them. The nodes represent the state of the game at a particular juncture given certain conditions. The tree is known as a game tree and is illustrated below (Russell and Norvig, 162-164):

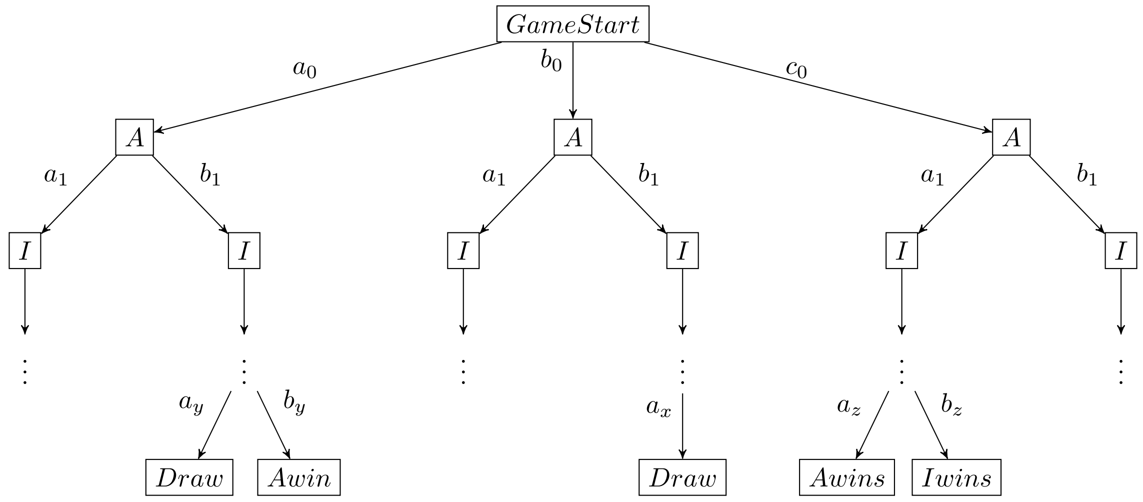

Figure 5: Game tree illustration. A represents a game result from the MAX player's move choice, while I represents a result from the MIN player's move choice. The lowercase letters represent move options available at a level. The MAX and MIN players take turns deciding their moves and eventually, a completed game is arrived at in a terminal node that indicates the final result of that game. One can determine the sequence of moves that resulted in a final result by following the moves up to the initial game state.

Using the generated values of the game tree nodes, the algorithm can determine for a player, which move to make at a particular point in the game. Given the available moves at a certain turn, the algorithm finds a value for each result gotten from a move that represents the best possible outcome of that branch of the tree for the player. The value of a completed game is referred to as the utility, and it represents the final result. For an unfinished game, the value is determined based on the values of the results from the succeeding turn. Consider the example below:

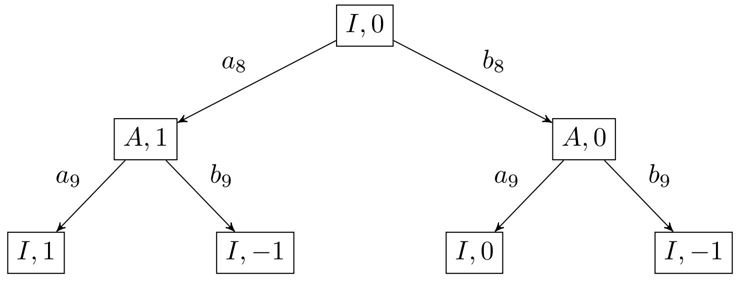

Figure 6: Value determination for nodes. MAX winning is represented by 1, MIN winning by -1 and a draw by 0. MIN seeks to minimize the value, while MAX seeks to maximize the value. Given the two options from MIN's turn at the 9th level, MAX's best result on one branch is a win, while on the other branch it is a draw. For MIN, given the two options from MAX's turn at the 8th level, their best result is a draw. Throughout a game tree, the mini-max algorithm determines the values of nodes from the bottom up, since only completed games have a definite utility that represents the outcome.

Once the values are determined for each node, choosing the best move for a player at a certain turn involves picking the move that maximizes or minimizes the value for a MAX or MIN player respectively. Since each player aims to get the best possible result for themselves, the values are determined assuming the player's opponent is also playing to get the best result for themselves as well (Russell and Norvig, 164).

The mini-max algorithm returns the best move for a player given a certain game state that is based on how far the game has progressed. It is referred to as mini-max, because the implementation involves accounting for both the MAX and MIN player. It is a recursive algorithm that goes down the game tree from a given game state node, alternating between players and their move options as the turns change, until it arrives at completed games with defined utility values. It can return the values up the tree and back to the game state it started from to provide the values necessary for choosing a move for the next game turn.
The algorithm pseudo-code is as shown below (Russell and Norvig, 165-166):

```
function minimax(game_state, player)
    if the player is MAX
        return a move that resulted in a game state with a value equal to maxvalue(game_state)

    else (the player is MIN)
        return a move that resulted in a game state with a value equal to minvalue(game_state)

function maxvalue(game_state)
    if the game state is a terminal node in the game tree and has a utility
        return the game state's utility

    initialize value to $-\infty$

    for every move available for MAX in this turn
        determine the game state as a result of the move
        update value with the maximum between value and minvalue(result_game_state)
    return value

function minvalue(game_state)
    if the game state is a terminal node in the game tree and has a utility
        return the game state's utility

    initialize value to $+\infty$

    for every move available for MIN in this turn
        determine the game state as a result of the move
        update value with the minimum between value and maxvalue(result_game_state)
    return value
```

Depending on the number of available moves for a player at each turn and the maximum number of turns for the game that determines the tree's depth, the algorithm can take a rather long time to find the solution of a game. Nine Men's Morris is a game with 3 different phases to it. If we consider the 1st phase alone, each player has 9 turns to place a piece on the board, and the move options available start from 24 possible locations for the 1st piece reducing up to a minimum of 7 possible locations for the 18th and last piece to be placed on the board. This could generate around $^{24!}/_{6!}$ different nodes, which is a lot of nodes for just the 1st phase of the game. The first test of a solution to Nine Men's Morris I ran was using a modification of the mini-max algorithm that made it significantly faster in solving a game. The improvement is known as mini-max with alpha-beta pruning.

3.1 Mini-max Algorithm with Alpha-Beta Pruning

An improvement on the mini-max algorithm introduces variables alpha and beta, to reduce the number of game states to consider from the game tree in deciding a move to make. It determines using the variables mentioned above, whether a certain branch would have an influence on the final choice of a move, and thus if it is worthwhile to expand down the branch. The process of taking out certain options from the tree given a criteria for usefulness is known as pruning, just like the process of trimming parts of a plant to promote healthy growth.

The alpha variable tracks the maximum value found for a game state as the algorithm recurses, while the beta variable tracks the minimum value found. Alpha starts at -infinity, while beta starts at +infinity. Their values are replaced as larger and smaller values are found respectively. Consider the example in the next page (Russell and Norvig, 167):

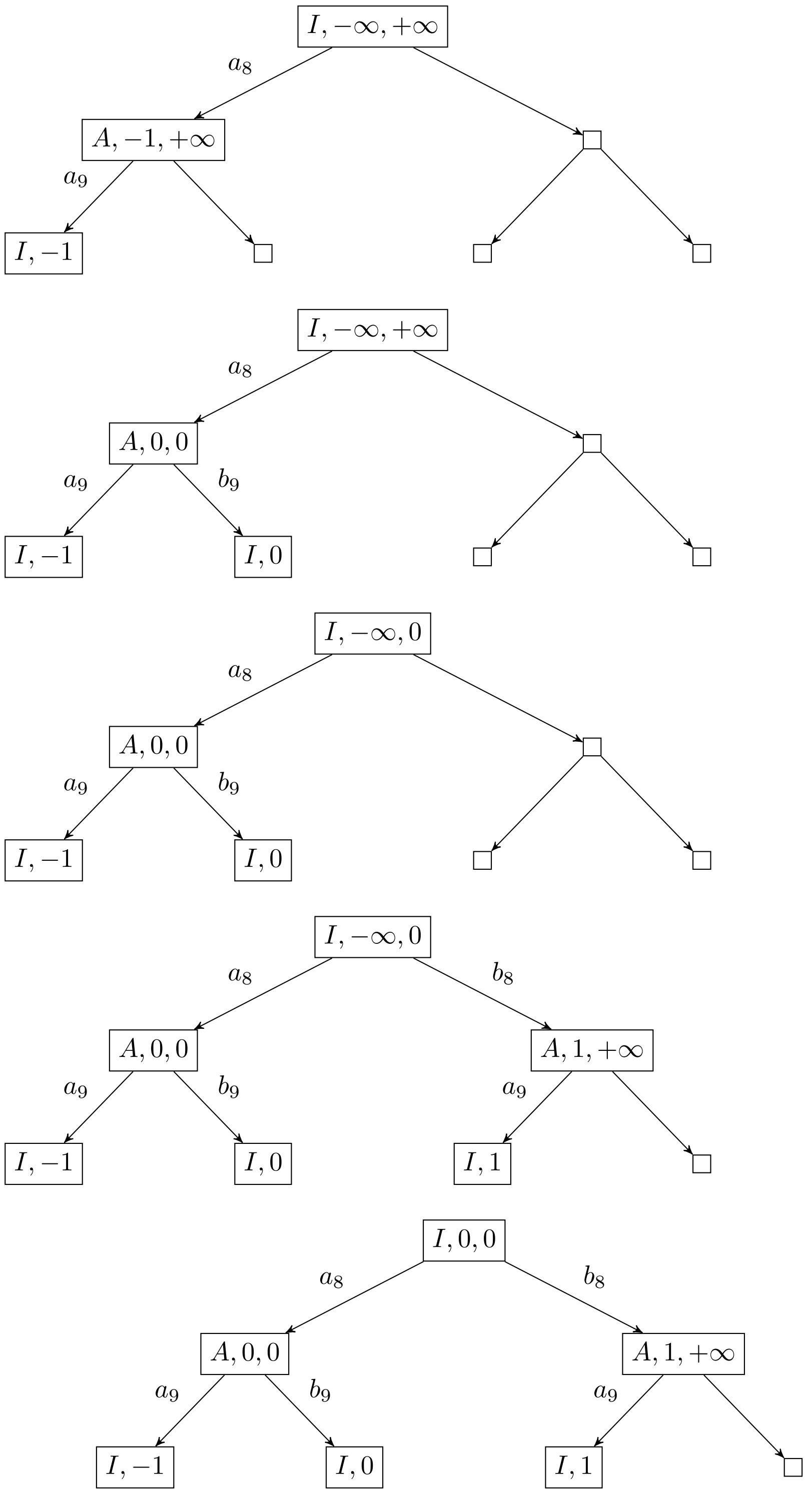

Figure 7: Alpha-beta pruning.

We determine the value of the MIN root of this tree without expanding every node under it by using the alpha and beta values. In going through the branch as a result of move a_8, we find the beta value of our root to be at most 0, since it is the smallest value found thus far. The MAX node that came from the move found a maximum value of 0 from its available options, updating its alpha value from the initial -1 to 0 as it went through the options as a result of moves a_9 and b_9. In exploring the branch as a result of move b_8

next, we find the first maximum alpha value for that MAX node to be 1. This value could only really remain as is since our value options are 1, 0 and -1, otherwise there would be room to improve the alpha value. However, we notice that 1 is greater than our minimum MIN root value thus far, so regardless of what other value options are under the MAX node, we know that our root's minimum value will not be affected, so we can conclude it to be 0 without exploring the other options under that MAX node. The pseudo-code for the algorithm is as shown below (Russell and Norvig, 168-70):

```
function minimax(game_state, player)
    if the player is MAX
        return a move that resulted in a game state with a value equal to
        maxvalue(game_state, $-\infty, +\infty$)

    else (the player is MIN)
        return a move that resulted in a game state with a value equal to
        minvalue(game_state, $-\infty, +\infty$)

function maxvalue(game_state, $\alpha, \beta$)
    if the game state is a terminal node in the game tree and has a utility
        return the game state's utility

    initialize value to -$\infty$

    for every move available for MAX in this turn
        determine the game state as a result of the move
        update value with the maximum between value and
        minvalue(result_game_state, $\alpha, \beta$)
        if value $\geq$ $\beta$
            return value
        update $\alpha$ with the maximum between value and $\alpha$
    return value

function minvalue(game_state, $\alpha, \beta$)
    if the game state is a terminal node in the game tree and has a utility
        return the game state's utility

    initialize value to $+\infty$

    for every move available for MIN in this turn
        determine the game state as a result of the move
        update value with the minimum between value and
        maxvalue(result_game_state, $\alpha, \beta$)
        if value $\leq$ $\alpha$
            return value
        update $\beta$ with the minimum between value and $\beta$
    return value
```

While this improvement is evident as so when used with games that had reasonably small game trees to begin with, running the alpha-beta mini-max MAX and MIN agents on my implementation of Nine Men's Morris resulted in a recursion depth error in Python, indicating that the game tree was far too deep for this to be a feasible solution to the game. Yet another improvement option was fortunately available for the algorithm.

3.2 Mini-max Algorithm with Alpha-Beta Pruning and Depth-Limiting

An improvement can be made on the alpha-beta mini-max algorithm to further minimize the time taken to determine the next move to be made in a game. Rather than have the algorithm go down a branch entirely to obtain utility values to return up the tree, it goes up to a certain defined depth and then estimates the value of game states rather than retrieving their actual values. As such, a means of estimating the value of a state must be established.

The estimation is done using an evaluation function, which considers various elements of game's state that contribute to the success of a player, and produces a value that represents a likely utility value. The choice of elements to consider greatly impacts the accuracy of the estimation and the algorithm's success overall. Note that when the actual utility at a completed game state is available, it will be used and not an estimation. Estimations are reserved for incomplete game states (Russell and Norvig, 171-172).

In determining an evaluation function for Nine Men's Morris, I considered game elements used by Petcu and Holban in their paper on Nine Men's Morris evaluation functions. They considered various elements of a game state related to the creation of mills, number of available pieces and the availability of moves for a player's opponent. Moreover, they considered these elements and the effect of the phase in which a game was on their relevance to the value estimation. My initial attempt at an evaluation function took into consideration:

- The number of mills already created by a player,
- The number of potential mills that required only one available move to be created,
- The phase of the game based on the number of pieces not on the board but available for a player.

These elements were used to determine the estimated value of a game of Six Men's Morris for both players, then the two estimates were compared to see which was greater, as the value estimated the number of moves available that could result in a mill. The player with the greatest estimated value determined the estimated utility value for that game state.

While it was sufficient to get some result, it was not a satisfactory solution that led to a completed game, and it made great assumptions regarding the relevance of the estimated values independent of an opponent's compared estimated values. Modifications were made to the elements considered for the evaluation function, as well as the final means of value calculation. I then took into consideration:

- The number of pieces available for a player that were not on the board,
- The number of pieces available for a player that were on the board,
- The number of possible mills for a player that would require a single move to create,
- The number of mills already created by a player,
- The number of pieces blocked for a player's opponent
- The phase in which the game is.

Elements were looked into individually to determine an estimated value for each. Using the estimates of both players, the estimated value for each element was determined by the difference between player values divided by their sum. All element estimated values were combined through a weighted sum and averaged to get a game's estimated value.

Aside from the evaluation function, a depth cut-off value was required to know when to apply the evaluation function. This was a fixed depth value, with the depth of a node being passed to the next level and incremented accordingly as the algorithm explored the game tree. When it came to finding the utility value of a game state, the availability of a definite utility value was first checked. If no such value was found, the node's depth is checked to see if it is at or beyond the depth cut-off value, then the evaluation function is applied accordingly. The modification to the alpha-beta mini-max algorithm is shown below (Russell and Norvig, 171-174):

```
function minimax(game_state, player)
    if the player is MAX
        return a move that resulted in a game state with a value equal to
        maxvalue(game_state, $-\infty, +\infty$, 0)

    else (the player is MIN)
        return a move that resulted in a game state with a value equal to
        minvalue(game_state, $-\infty, +\infty$, 0)

function maxvalue(game_state, $\alpha, \beta$, depth)
    if the game state is a terminal node in the game tree and has a utility
        return the game state's utility

    if the game state is a non-terminal node and the depth cut-off has been reached
        return the game state's estimated value

    initialize value to -$\infty$

    for every move available for MAX in this turn
        determine the game state as a result of the move
        update value with the maximum between value and
        minvalue(result_game_state, $\alpha, \beta$)
        if value $\geq$ $\beta$
            return value
        update $\alpha$ with the maximum between value and $\alpha$
    return value

function minvalue(game_state, $\alpha, \beta$, depth)
    if the game state is a terminal node in the game tree and has a utility
        return the game state's utility

    if the game state is a non-terminal node and the depth cut-off has been reached
        return the game state's estimated value

    initialize value to $+\infty$

    for every move available for MIN in this turn
        determine the game state as a result of the move
        update value with the minimum between value and
        maxvalue(result_game_state, $\alpha, \beta$)
        if value $\leq$ $\alpha$
            return value
        update $\beta$ with the minimum between value and $\beta$
    return value
```

4 Result

As previously mentioned in section 3.1, an attempt to use mini-max with alpha-beta agents was unsuccessful. The same was the case even when I tested the agents on a Six Men's Morris game. However, utilizing depth-limiting with a simple evaluation function that compared an evaluation result for both players to determine the estimated utility resulted in some success, with the game playing up until a point at which both players were stuck making the same moves despite better moves being available. Nonetheless, it indicated promise in the depth-limiting approach to a solution. In fact, the simple evaluation function for Six Men's Morris permitted the Nine Men's Morris game to play up until a certain point despite considering elements of a variation of the game.

After refining the elements considered for the evaluation of a game state for both Six Men's Morris and Nine Men's Morris, the agents came up with moves slower than before, but each player seemed to choose more defensive moves, aiming to block their opponent's attempt at creating a mill right from the first phase of the game. However, within the second phase, players got stuck making the same moves somewhat ending the game in a draw; there were other move options available that may not have initially seemed optimal, but could in future result in a more favorable situation for a player. However, it is uncertain by observation of the game state whether it would really improve the situation of a player in the next turns, given that the opponent would also aim for a favorable situation for themselves. In Nine Men's Morris, the agents were not necessarily more defensive than they were offensive players. The players also got stuck making the same moves at some point in the second phase despite having more options for mill creation.

A possible reason for the observed behavior is the nature of the estimation calculation. For every element of a game state, I estimated the advantage a player has over their opponent. However, this did not necessarily take into account the contribution of an element to a player's advance towards success. As such, I sought to include this consideration, looking at what each element implied independent of an opponent's estimated value. This was included in the final weighted sum calculation. However, the resulting play was the similar to the previous for both Six Men's Morris and Nine Men's Morris regardless.

In the initial tests and adjustments of the evaluation, what remained constant was the depth cut-off value at 6. If the depth of a node was greater than or equal to 6, it would apply the evaluation function. As such, this was the other parameter available for modification. I considered the time it took for players to make their moves and how optimal they played at each phase. The depth cut-off was adjusted for values between 4 and 7. A depth cut-off at 7 resulted in moves that took far longer than when the depth cut-off was less, especially for Nine Men's Morris. The table in the next page presents the results of adjusting the depth cut-off values to the game-play. SMM represents the values for Six Men's Morris, while NMM represents the values for Nine Men's Morris.

Depth Cut-Off	First MAX Move	First MIN Move	Game Play
7	SMM: 72.675s	SMM: 23.735s	In both phase 1 and 2, the players sought to block one another affording few opportunities to attempt creating mills. Not only were potential mills with two aligned pieces blocked when possible, but also pieces that could move towards potential mills. The game ended in a draw with repeated moves of the same game pieces for both players.
	NMM: 535.291s	NMM: ?s	The game took a very long time to determine the next move after MAX.
6	SMM: 4.355s	SMM: 5.778s	In phase 1, the players sought to create mills while blocking one another. By phase 2, the players had very limited options in moves, and the game ended in a draw with repeated moves of the same game pieces for both players.
	NMM: 42.809s	NMM: 64.756s	In phase 1, the players sought to create mills while blocking one another. In phase 2, MIN managed to establish a strategic position with 2 or so mill opportunities so that MAX could only block one at a time, allowing MIN to create a mill and take away pieces. The game ended with MIN winning after 56 moves.
5	SMM: 0.753s	SMM: 1.332s	In phase 1, the players sought to create mills while blocking one another. In phase 2, MIN managed to establish a strategic position with 1 mill. MAX did not seem to play as optimally as they missed opportunities to block MIN moves. The game ended in a draw despite MIN having two opportunities to create mills and win the game.
	NMM: 4.303s	NMM: 10.372s	In phase 1, the players initially sought to create mills more than they blocked one another. In phase 2, MIN managed to establish a strategic position with 2 or so mill opportunities allowing MIN to create a mill and take away pieces. MAX had blocking opportunities but missed them. The game ended with MIN winning after 44 moves.
4	SMM: 0.196s	SMM: 0.316s	In phase 1, the players sought to create mills while blocking one another. In phase 2, MIN managed to establish a strategic position with 1 mill. MAX did not seem to play as optimally as they missed opportunities to block MIN moves. The game ended in a draw despite MIN having the opportunity to create a mill and win the game.
	NMM: 0.551s	NMM: 1.556s	In phase 1, the players sought to create mills while blocking one another. In phase 2, MIN managed to establish a strategic position with 2 or so mill opportunities allowing MIN to create a mill and take away pieces. MAX had blocking opportunities but missed them. The game ended in a draw despite MIN having the opportunity to create mills and win the game.

There seemed to be a flaw in the logic of my Nine Men's Morris game implementation that gave the MIN player an advantage over MAX, primarily because MAX was not playing that defensively. In fact, with the starting player switched to MIN, MAX won after the same number of moves as MIN in the test; the player that went second always won with the same number of moves for a certain depth cut-off. However, with MAX as the first player and MIN player having a depth cut-off of 5 and MAX of 6, the game ended in a draw and MAX was in a significantly more favorable position than before. With MAX player having a depth cut-off of 5 and MIN of 6, the game ended in a draw faster with more pieces blocked for each player.

Depth Cut-Off	First MAX Move	First MIN Move	Game Play
MAX Agent: 4	SMM: 0.094s	SMM: 0.531s	In phase 1, the players sought to create mills while blocking one another. In phase 2, MIN managed to establish a strategic position with 1 mill. MAX missed some opportunities to block MIN, but still blocked some moves. The game ended in a draw despite MIN having the opportunity to create a mill and win the game.
MIN Agent: 5	NMM: 0.188s	NMM: 4.344s	In phase 1, the players sought to create mills while blocking one another, although MAX missed some opportunities to block MIN and MIN captured a MAX piece. In phase 2, MIN captured several of MAX's pieces, but eventually got stuck repeating the same moves. The game ended in a draw despite MIN having the opportunity to create mills and win the game.
MAX Agent: 5	SMM: 4.355s	SMM: 5.778s	In phase 1, the players sought to create mills while blocking one another, with MAX primarily attempting mills. By phase 2, the players had limited options in moves, and the game ended in a draw with repeated moves of the same game pieces for both players.
MIN Agent: 4	NMM: 2.094s	NMM: 0.547s	In phase 1, the players sought to create mills while blocking one another, but MAX played more defensively and successfully created some mills. In phase 2, MAX established a strategic position with about 2 mill options, reducing MIN to 3 pieces. With MIN in phase 3, they played defensively and the game ended in a draw with players moving around the board as if MIN was avoiding giving MAX the advantage of being in phase 3.

In Six Men's Morris, the game play seemed much more optimal until the depth cut-off was at 5 and below. Although the games ended in draws, those with a depth cut-off of 5 and below got stuck while still having move options that could have them in a better position than they were. With different depth cut-off values for the players, the game still ended in a draw, but the player with a greater depth cut-off value had more of an advantage than in games with the same cut-off value for both players. Given the defined evaluation of the game in section 3.2, a depth cut-off of around 6 resulted in a reasonably fast and optimal solution for Six Men's Morris. Nine Men's Morris most optimal play came from a depth cut-off of 5 for MAX and 4 for MIN, as if providing a handicap for MAX.

5 Conclusion

The alpha-beta mini-max algorithm with depth-limiting proved to be a useful solution strategy for Nine Men's Morris and especially so for the Six Men's Morris alternative with depth cut-offs of 6 and 7 for both players. The aspects of the game considered for an evaluation of a game's state were more sufficient for the smaller, less complex version of Nine Men's Morris. The inclusion of value estimation for MAX and MIN players to the alpha-beta mini-max algorithm is similar to a human player's strategy development process in the sense that various elements of a game state are factored in to determine the next best move to make. However, further improvements on the decision-making process would be necessary for more optimal game-play, especially for the full Nine Men's Morris game that has more factors to consider in identifying an optimal move. The depth cut-off value seemed to determine the likelihood of an estimated value and the resulting move selection to be optimal. This was more evident within the 2^{nd} phase of the game, which requires a more complex strategy, than in the 1^{st} or last phase. With a more reliable evaluation system that may look a few more turns ahead for potentially advantageous moves, the depth cut-off value would probably have less of an impact on the game play.

6 Citations

1. "Game." Merriam-Webster, Merriam-Webster, www.merriam-webster.com/dictionary/game.

2. "Nine Men's Morris." Wikipedia, Wikimedia Foundation, 17 Mar. 2019, en.wikipedia.org/wiki/Nine_men%27s_morris.

3. Petcu, Simona-Alexandra, and Stefan Holban. "Nine Men's Morris: Evaluation Functions." Faculty of Electrical Engineering and Computer Science at Stefan Cel Mare University of Suceava, 2008.

4. Artificial Intelligence: a Modern Approach, by Stuart J. Russell and Peter Norvig, 3rd ed., Prentice Hall, 2010.