Developing an Adversarial Agent

Declan Simkins, Rebecca Walton, Jonathan Ismail
Macewan University

***Abstract*— A common approach in dealing with decision based game theory when creating a rational agent is implementing an effective Minimax algorithm. Our paper focuses on strategies of developing an agent for the game Kōnane otherwise known as Hawaiian checkers which is a two player, zero-sum strategic board game. The Minimax algorithm is a recursive algorithm best fitted for maximizing a players' move used in zero-sum games to denote minimizing the opponents maximum payoff. During the course of our work, we investigate different methods of implementation that increase the algorithms' effectiveness given a restricted time limit. The scope of our pa- per describes a detailed analysis of our Minimax algorithm which is represented by an evaluation function and illustrates the problems encountered along development and possible solutions that improve the effectiveness of the algorithm.**

I. Introduction

Kōnane is a two player strategy game with a 8x8 board filled with alternating patterns of black and white pieces. The goal of the game is to reach a state in which one player is unable to capture an enemy piece. Players have the ability to maneuver across the board by jumping over adjacent pieces thus removing opponents pieces. If desired, play- ers also have the ability to perform multiple jumps which may or may not benefit the jumping player. Therefore, to create an intelligent adversarial agent, one must develop a logical path in which the agent can employ a comprehen- sive technique in achieving a desired goal. The Minimax algorithm is a decision rule based algorithm formulated for two-player zero-sum game theory which minimizes the possible loss for a worst case scenarios as seen in Figure 1. The algorithm covers both cases in which players take alternate moves and extends to more complex decision-making games where uncertainty in present [1]. Considering the nature of Kōnanes' zero-sum environment, our team implemented the Minimax algorithm when developing our adversarial agent.

1. 1 Minimax

This algorithm uses an informed adversarial search that utilizes a cause-and-effect search that considers each possible action available at a given moment, then considers subsequent moves for each of those state in attempt to satisfy a goal condition. Our team chose to use this algorithm after dissecting the Kōnanes' deterministic environment. We utilize an informed search tree to choose a "best" move given the current state of the game and implement a static evaluation function that assigns a heuristic value to possible available moves. Each node or move represents a state in our game tree in which our Minimax algorithm behaves with the assumption that a desired move benefits on player and minimizes the opportunities of the opponent. The evaluation function our team developed was designed to prioritize speed over accuracy and when represented, focuses on current state and does not explore possible moves.

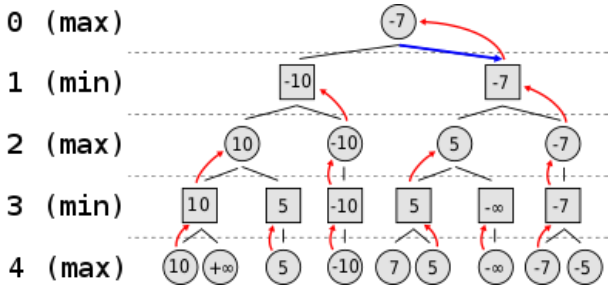

Fig. 1. The image above provides a depicted description of the Minimax algorithm

1.2 Testing

Our agents rationality and intelligence was measured by the success rate of its win-to-loss ratio. We began test- ing our agent by playing against a "random" agent who's

moves would be selected at random with no clear path to victory. Upon playing random agent, our team encountered several problems. The first of which was the time limit. Our agent had to have the ability, knowledge and skill to select a beneficial move within 10secs. The basic Minimax algorithm developed would be problematic in a time restricted scenario due to the mere fact that our agent couldn't foresee all the best possible moves within the given time. This restriction would lead to an outcome loss of 100 percent, given 5 simulated games.

```
void evaluate(node_s *node)
{
    static const char black = 'B';
    static const char white = 'W';
    int moves_b = available_moves(black, node);
    int moves_w = available_moves(white, node);

    switch (model->colour) {
        case 'B':
            node->heuristic = moves_b - moves_w;
            break;
        case 'W':
            node->heuristic = moves_w - moves_b;
            break;
    }


}
```

Fig. 2. The above provides the code for the evaluation function used in determining a heuristic for possible moves. The heuristic is the amount of available moves for the agent minus the amount of available moves of the opponent.

II. Implementation and Enhancements

In order to increase our win ratio, we would need to increase the intelligence of our agent. A traditional Minimax algorithm proved as insufficient in producing ideal moves that benefit our agent in achieving a desired final goal. One enhancement our team decided to implement was maximizing our search results for an ideal move given the time limit.

2.1 Alpha-beta Pruning & Iterative Deepening

We decided to enhance our Minimax algorithm by establishing an Alpha-beta pruning method which seeks to decrease the number of nodes that are evaluated by the traditional Minmax algorithm in its search tree. This enhancement would increase the effectiveness of possibly finding a "best" move by terminating the evaluation of a path that has been proven to be worse than a previously examined move. When this enhancement is applied to the Minimax algorithm, it allows for a greater search by pruning away branches that cannot possibly influence the final decision of our algorithm [2].

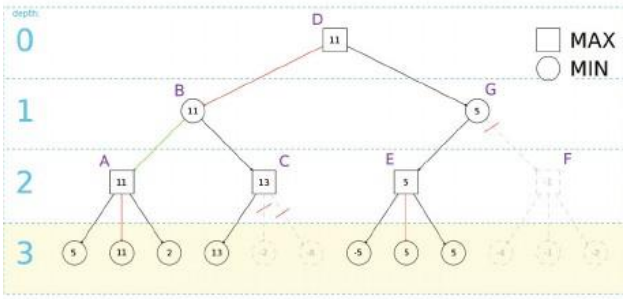

Fig. 3. The image above presents a visual description of the Minimax algorithm with Alpha-Beta Pruning and Iterative Deepening.

Furthermore, we enhance our search by applying an Iterative Deepening search method to our algorithm. Simply using the alpha-beta method, we noticed that weren't able to use partial results with confidence unless searching the full breath of our tree in a given the time limit. Therefore, using Iterative Deeping on our search, we were able to be conservative and set a depth-limit which guaranteed finding a move given a time limit. We used a separate thread to keep track of time, and when the clock reached 10 seconds, our algorithm would use the solution found in the previous depth limit. This showed to be efficient in finding effective moves, however, was problematic when evaluating visited nodes. The time limit meant there was a restriction on time spent searching the tree, and reevaluating visited moves would cost our agent in finding a "best" possible move. To solve this, our team created a Hash table that contained all the evaluated moves of a given current state. The algorithm would save time by checking the state of a move and cross reference the hash table to see if the move needs evaluation.

III. Conclusion

During development, we've observed the effectiveness of using the Minimax algorithm as a solution for developing an intelligent agent. We noticed that the algorithm, if

	Minimax Agent	+Hash-table	+ Alpha-Beta Pruning	+ Iterative-Deepening
Win Ratio against Random Agent	0%	20%	80%	100%
Time	< 10s	< 10s	< 10s	< 10s

Fig. 4. The table above describes the Minimax algorithm playing against a Random Agent who' s moves are chosen at random and the Win ratio of the enhancements of the algorithm throughout development. The ratio is based on 5 games played and the percentage of wins our created Agent has.

implemented correctly can result in an optimal move being performed in zero-sum games, which was demonstrated in our game agent for Kōnane. However, the specific nature of Kōnane adds elements that require a more complex algorithm then the traditional Minimax. Seeing as Kōnane introduces an element of time, we would require an agent to play optimally within a 10 second time limit, therefore as we established Iterative Deepening which sets a limit to our search and return the best move within the given time limit. The enhancements given to our algorithm saves time during our search and thus additionally increases its depth which leads to a greater chance of finding a true "best" move. Our algorithm does not always guarantee a win against its opponents, however provides sufficient strategies of playing intelligently for the game of K ō nane.

IV. REFERENCE

https://wikivisually.com/wiki/Minimax

https://en.wikipedia.org/wiki/Alpha–beta_pruning

https://www.ics.uci.edu/~rickl/courses/cs-171/2012-wq-cs171/2012-wq-cs171-lecture-slides/2012wq171-07-Games.pdf

