Qi Nan Jin – 260531346
COMP424 – Artificial Intelligence
Friday 8th of April, 2016

# AI agent with Alpha-Beta Pruning and Empirical Heuristics

# for the Game of Hus

*Project Report*

## Introduction

In the context of this class project, an AI agent is developed for the game of Hus. The main goal of this project is to apply theories of artificial intelligence learnt in this course to practice and create an AI agent that out performs other agents. This report will first describe the basic design of this AI. It will then explain in details the tools and methods used to build and optimize the agent. It will finally present the full structure of this agent and discuss about its strengths and weaknesses.

## Design

The backbone of this agent is a standard MinMax search algorithm with Alpha-Beta pruning [1], in pseudo-randomized order. The main design tasks are to determine the maximum number of search depth within the memory and time constraint, and to optimize the search heuristic. These design tasks are mainly achieve through exhaustive simulation using different parameters, the details of which will be discussed in the next section.

The algorithm is run within a standard Java executable object [2],[3] with a slightly stricter time constraint. An alternative move found by a greedy algorithm is chosen if the main algorithm takes too long. The motivation of this extra feature is discussed in more depth in the following section.

A key factor of this design is a scoring system that acts as the search heuristic. The scoring system computes a weighted sum of different aspects of the game after each move. The selection of factors to include in the scoring as well as the optimization of the weightings are discussed in details in the next section.

## Methods

The main design tasks for this agent are all achieve through analysis of empirical data from numerous simulations. This includes the determination of search steps as well as the optimization of the scoring system.

### ***Search Steps***

After a standard Alpha-Beta pruning algorithm is implemented, 100 games are simulated for each number of steps until the agent start to run into significant number of time-out moves.

Based on the simulation, negligible number of timeout moves are observed for under six steps. A seven-step search produces approximately 1 timeout move every 15 games. An eight-step move, however, produces on average at least 1 timeout move every game.

The simulations for any given number of steps are run against the same algorithm with one step less. For example, the agent with 7 search steps is played against the same agent with 6 search steps; the agent with 1 search step is played against the random player. This comparison simulation confirms that more search steps is advantageous.

Based on these simulation results, it is determined that the algorithm will run with 7 search steps.

## ***Scoring System***

The following aspects of the game after each given move are considered for the scoring system: number of seeds, number of occupied pits, number of opponent's occupied pits, number of possible moves, number of opponent's possible moves (negative score), number of possible captures (attack moves), number of empty pits in the outer row (defense moves).

To determine which of these aspects are to be included in the scoring system, all of them are first included in the system with a weighting of 1 as a baseline reference. Each parameter is then varied individually and 1000 games are simulated against the baseline using a greedy MinMax search. Only parameters that significantly affect the winning percentage are included in the final scoring system. Figure 1 illustrates an example of effects of number of possible captures on winning percentage.

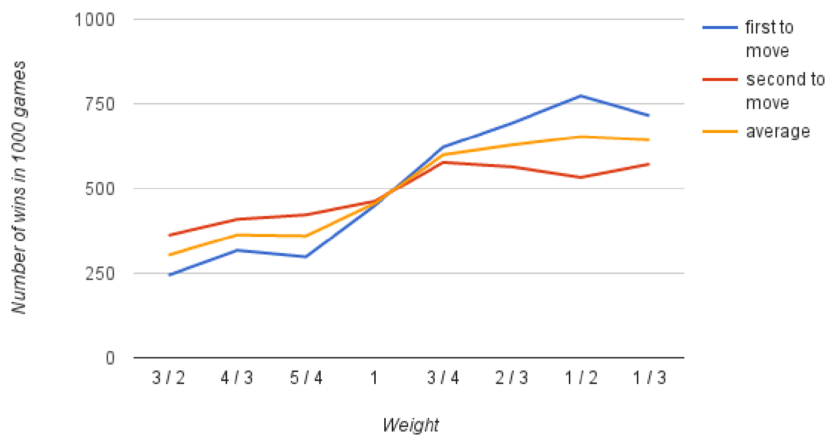

Figure 1: Effect of varying the weight of potential captures on winning percentage

Based on the simulations, the following 4 factors are included in the final scoring system: number of seeds, number of moves, number of opponent's moves (negative score), and number of potential captures.

Once these factors included in the Alpha-Beta search heuristics, more simulations are run to fine tune the weighting of each of these parameters in the full algorithm. Again, starting with a weighting of 1 for all parameters as baseline, each parameter are varied individually and 20 games are simulated against the baseline. If the new weighting provides a winning percentage of over 70%, then the weighting is kept as the new baseline. The simulations continues for each parameter until it is optimized. The final set of optimized weightings are then used as the scoring system for the search heuristic.

## Agent Structure

The final AI agent has the following structure:

```
if first move of the game {
    return move (pit 23)
}
run greedy to get back up move
start Java executable thread with 1800ms time limit{
    run alpha-beta pruning with depth 7
} catch (timeout exception){
    return backup move
}
return best move
```

The scoring system for search heuristic is determined as the following:

when moving first: $score = 4 \times (seeds - 48) + 3 \times moves + \frac{1}{2} \times attacks - 3 \times moves_{op}$

when moving second: $score = 3 \times (seeds - 48) + 3 \times moves + attacks - 4 \times moves_{op}$

Where *attacks* is the number of potential captures in the next move and *moves$_{op}$* is the number of opponent's moves.

## Discussion

### Strengths and weaknesses

The main strength of this agent is the scoring system acting as the search heuristic. This scoring system is optimized based on exhaustive simulation and uses multiple parameters, instead of simply using the number of seeds or moves.

From the starting set of factors, the ones that have less impact on the winning percentage against baseline are eliminated. Intuitively, these discarded factors may be considered "absorbed" into the final metrics. E.g. *number of occupied pits* can be closely correlated with *number of moves*. One factor that is surprisingly included in the final scoring system is *number of potential captures (attacks)*. It significantly impacts outcome and seems to be more sensitive when the player moves second. One possible explanation for this effect is that a large capture on the next move can turn a game around from a losing situation.

This scoring system also takes into consideration whether the player is first or second to move. The optimal set of parameters puts more emphasis on the number of seeds when player plays first and values less the number of potential captures. However, when player moves second, minimizing the opponent's moves and maximizing potential captures have more importance. This feature is in fact implemented to accommodate a potential weakness of the agent. Based on the simulations, the agent strongly favors games where player moves first, given the same scoring system. Therefore, a different scoring system is used when player moves second to make the agent more balanced.

Another strength of this agent is the use of a back-up move generated by a quick greedy algorithm when the Alpha-Beta pruning takes too long. Based on simulation results, it is observed that a random move played due to the agent's timing out is gravely disadvantageous. The player will lose the game almost every time if a random move is played in the middle of the game. However, a greedy move significantly increases the chance of winning whenever the agent times out.

### ***Potential Improvements***

One of the main remaining issues of this agent is the fact that it still favors games where the player is first to move. It appears that the agent performs much better when player is in a winning situation, whereas its capability of turning a losing situation around is limited. One potential improvement to this issue would be to make the agent recognize winning and losing situations (based on board state) and adjust the weightings of the scoring system accordingly, in order to choose more aggressive or conservative moves.

## **Conclusion**

This AI agent is built on standard search methods. More specifically, a typical MinMax search with Alpha-Beta pruning is employed as the main algorithm. An optimized heuristic, as well as the number of search steps, are found through empirical data from exhaustive simulations. An extra backup greedy algorithm acts as a catching net if the agent runs for too long. Testing results show that this agent will beat the random player 100% of time, and beats the greedy algorithm over 98% of the time.

## **References**

[1] Russell, Stuart J., Peter Norvig, and Ernest Davis. *Artificial Intelligence: A Modern Approach.* Upper Saddle River, NJ: Prentice Hall, 2010. p.167-170
[2] Oracle. "Package java.util.concurrent." Java.util.concurrent (Java Platform SE 7 ). Accessed April 08, 2016. https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html.
[3] Lea, Douglas. *Concurrent programming in Java: design principles and patterns*. Addison-Wesley Professional, 2000. p.279-285