COMP 424 – Artificial Intelligence
FINAL PROJECT

HUS

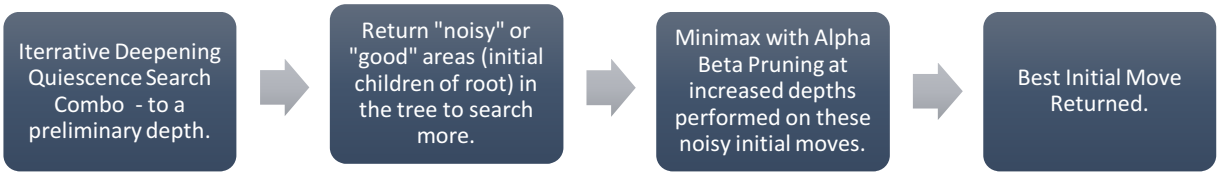
NIC YANTZI
260467234
WINTER 2016

Section 1. How does my program work?

Hus is an ancient game and a member of the Manacala family. Similar to games like chess, checkers, tic-tac-toe and even the game Go, Hus is a two player, zero sum, deterministic, discrete, perfect information, sequential game. Each move made takes the current state of the board and performs a transitions to a new board state. Because of this, a game of Hus and each of the possible game boards can be mapped to game tree. This tree can then be analyzed and explored using different artificial intelligence techniques. In Hus, each move is crucial to the game's development. Sowing, capturing and the general placement of seeds in an offensive or defensive position are crucial. In order to make intelligent decisions and choose the best possible move involves having the ability to look as far ahead into the game tree and evaluating game boards based at that future position. When designing the Hus Agent, a variety of techniques were used to allow my agent to produce the most intelligent decisions each and every turn.

Throughout the design and implementation of my agent, various methods and techniques were explored. In Section 4 of this report, the various methods that were tested and implemented are discussed in greater detail to show how the final version of my agent was constructed. On a broad level however, my agent combines minimax with alpha beta pruning. In my opinion the game of Hus can be viewed as a mountain. At the peak of the mountain the maximum branching occurs and the game is tied, on either side of the mountain the game is approaching either a win or a loss and the branching factor is greatly reduced. My agent invokes various policies based on the location on the mountain or the current state of the game. To split up the game into sets of game board states intervals of the seeds percentage owned were used. Having a policy for each one of these intervals allows for differentiation in the methods used that turn, to ensure maximum ability throughout the game. My agent also uses a combination of iterative deepening and quiescence search algorithms which allows the agent to search for "noisy" or "good" regions versus "quiet" or "worse" regions. Essentially the agent uses minimax, and goes to a preliminary depth of 4 or 5 and returns the scores associated with each initial move. Then, a selection of the top moves based on the initial minimax results,

are sorted based on the score in decreasing value. This rearrangement of the moves so that the best scores are searched again first has two effects. First, it allows for greater pruning during the alpha beta portion of minimax. Second, the reduction factor on the initial moves that undergo a second run of minimax allow for much greater depths and thus higher quality evaluations to be returned.



Another key part to my approach is the heuristics I chose to use for describing a “good” game board. In my final version I settled on two main heuristics when quantifying the value of a board. I attempted to research possible strategies for this game within literature, however any available literature did not contain specific strategy. Therefore, my first and main heuristic was the number of seeds that my agent has on his side of the board out of the total possible seeds in the game. After playing many games, I believe that the more seeds you have has a direct effect on your likelihood of winning the game. In every game played, the winner always captured more than ~75% of the seeds. In order to lose in Hus, a player must have either zero or one seeds in all of there pits on the board. Thus in order to force the opponent into losing, the best course of action is to try and take as many of the seeds as possible. Next a key part of my evaluation, something my agents determines on every board prior to checking anything else is whether that board is a winner or not. For winning boards, a valued of 10000 is assigned to that board. This score is the highest possible score returned and if my agent is able to pick a path leading to this board it will. Likewise, for losing boards, the smallest value, -10000 is returned which is a path the agent will avoid at all costs. Finally, my agent is able to keep track of the time used during a turn in order to avoid causing it to timeout. Any timeout can cause drastic damage as a random move is chosen. This timer combined with a buffer ensures that if the agent is close to the end of the allotted time for his turn, he returns the maximum move found so far that round. With this combination of techniques and heuristics, the agent is able to look deep into the game tree and choose the best move that will lead to winning the game.

Section 2: Theoretical Basis of the Approach

Explanation for the algorithms used in my Agent are found in other sections but the following is a brief summary of the research performed to decide the mix of algorithms used. Once my agent has constructed the game tree, different algorithms are used. “Searching for the best move in a zero-sum game is known as minimax search” (Platt, 255). In order to maximize the use of our computation however the agent must try and limit the number of nodes that it looks at. “The alpha beta pruning algorithm is the most commonly used procedure in game playing applications, where it serves to speed up game searching without loss of information,” it, “skips all [the] nodes that can no longer influence the minimax value of the root” (Pearl, 560). The only issue with relying solely on alpha beta pruning is that there is an “exponential gap in the size of trees built by best-case and worst-case Alpha-Beta. This led to numerous enhancements to the basic algorithm, including iterative deepening, transposition tables, the history heuristic…” (Platt, 255). Next after working on trying to sort the nodes in way such that this exponential gap is decreased if not eliminated to maximize the computational power on each turn I discovered an algorithm known as quiescence search where, “…the purpose of this search is to only evaluate "quiet" positions” (Chessprograming). My agent combines this method with iterative deepening and instead of looking for quiet positions, finds noisy or promising nodes that it can search further.

Section 3: Advantages & Disadvantages, Failure Modes or Weaknesses.

The following advantages and disadvantages of my agent are based on data generated while testing my agent against other Hus agents within class, as well as constructing a second Hus Agent of my own. This data allowed me to compare techniques to try and produce an optimal agent. Because of the quiescence/iterative deepening combo search my agent is able to search the tree at deeper levels then other agents who are just performing one round of minimax. This deeper search is caused by both enhanced alpha beta pruning as well as a reduction factor on the children of the root that undergo a second run of minimax. Next my agent is able to search deterministically on end game moves, rather then use a Monte Carlo

rollout strategy, my agent uses an evaluation function that can determine if a board is a winner or a loser. In section 4, this decision will be further discussed. My agent also has a timer and buffers implemented using Java's NanoTime function. This allows the agent to maximize computation on each turn while ensuring it doesn't timeout during the game. The buffer is equal to the time it takes to run the next DFS path on a potential move from the initial game board. If the buffer value is greater than the time left in the agents turn, the maximum move found so far is returned. Finally, the various policies implemented allows the agent to achieve the optimal depths at different points in the game, this allows for the agent to recover from a seed deficit and come back and win games. In terms of disadvantages, my agent's evaluation function and policy characteristics are based solely on what I could infer during testing and implementation. My agent could have been stronger if instead of these polices had been groomed. The optimal heuristics, and policies for each state could have been found from playing many games while using using policy improvement techniques such as the Policy Iteration Algorithm with Bellman's equation. If a competitor of mine is able to search from higher depths right away somehow without performing the iterative deepening quiescence combo search on the tree, then they might be able to recover more accurate information. My agent is assuming at the depth of this search that if a value of board at this point is bad it won't be better in the future which is not always true.

Section 4: Other Attempted Approaches and Implementations

Besides the methods described above I also tried to implement a few other techniques during the evolution of my agent. One method that I thought would be quite promising that ended up being removed was Monte Carlo Tree Search. This method tries to find optimal decisions based on random sampling within a game tree. One of the reasons MCTS is so powerful is the, "lack of need for domain-specific knowledge, making it readily applicable to any domain that may be modelled using a tree. Although full-depth minimax is optimal in the game-theoretic sense, the quality of play for depth- limited minimax depends significantly on the heuristic used to evaluate leaf nodes" (Browne 9). In a high branching game such as Hus where there is a lack of domain knowledge I thought MCTS would be quite powerful. I also found that

during tests of MCTS finding a balance between the depth of the minimax tree and the number of rollouts performed at each leaf node way key. The total number of rollouts performed at a level is equal to the number of leaf nodes at that level multiplied by the number of rollouts performed at each node. Thus MCTS performed at lower depths even in lower numbers outperformed MCTS done at higher depths with greater rollouts. This is due an increased perception or clarity. This result sense as minimax results performed at lower depths are more indicative of a move then more superficial ones. In the end after many tests (Section 6), I determined that my final version of my agent was more powerful. An agent that used MCTS as well as the methods my agent had in any combination through the various policy intervals would always lose. My intuition for this is that since my minimax methods were preforming at depths of 7 to 11, in order to make MCTS's results effective, the minimax depths used had to be decreased. Between the combination that my agent was able to perform greater search depths during this period, as well as the fact that my agent had two powerful heuristics in deterministically searching for end game scenarios as well as the powerful heuristic in the number of seeds, my agent was able to overcome any added benefit that MCTS would have added.

Another attempted design implementation that ended up directing my efforts in developing the combination of quiescence search and iterative deepening was an attempt to sort the nodes of each parent within the game tree based on their individual board evaluations. This implementation turned out to be too costly in terms of computation time and thus was removed.

I also experimented with a few other heuristics for evaluating game boards such as the number of pits that contained singletons (0 or 1 seeds), the number of vulnerable seeds on the following move open for capture as well as the ratio of seeds front to back. In the end I decided to just stick with the heuristic involving the number of seeds as I though this was the best indication of how a game was going. In all the games I played, no player ever won unless they had captured over ~70% of the seeds.

Last, I tried implementing another enhancement within alpha beta pruning called Aspiration Windows. Essentially the idea is to try and reduce the window or interval size

between your alpha and beta values. The idea is that given the shorter window more pruning will occur and thus the search will occur in less time. “Since using an aspiration window always bears the danger of having to repeat the search *(re-search),* in case the minimax value is outside the window, the effort for computing it may even become greater than that when using no window at all” (Kaindl, 1991). Since I am searching to such deep depths, there might actually be a large variation in the possible scores returned thus after weighing the pros and cons of this search method I decided to not use it within my agent’s policies for decision making as I felt given the chances for missing a better move was not worth the rewards of enhanced pruning.

Section 5: Possible Improvements

During the process designing an optimal agent, I tried to find the best combination of the methods and depths based on the various policy intervals implemented. Each test I ran I learned more about what worked and what didn’t. My decisions were based on running multiple games against my second agent and observing the number of wins and losses, whether my agent had any timeouts occur, as well as whether I was indeed searching the whole portion of the tree I had selected, or only a portion of this portion. I had to learn by trial and error so I was just comparing every iteration of my algorithm to previous iterations. In the end I am certain if I had developed learning algorithms that could have found the ideal mix of methods this new version of Hus agent would get better. Another potential idea for developing a stronger agent was including a full game tree in a .JSON data file that could be easily parsed to find optimal paths or sets of set initial and final winning variations of gameplay. Finally, some sort of agent that incorporated Markov Decision Processes I feel would be quite powerful. In the end all of these ideas would need to be run against the agent I have currently and they would need to be able to run within the memory and time limits.

Section 6: Data and Tables

Data and tables used to fine tune the selection of algorithms used in my agent follows on subsequent pages.

Works Cited

Browne, Cameron B., Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. "A Survey of Monte Carlo Tree Search Methods." *IEEE Trans. Comput. Intell. AI Games IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012): 1-43. Web. 04 Feb. 2016.

Kaindl, H., R. Shams, and H. Horacek. "Minimax Search Algorithms with and without Aspiration Windows." *IEEE Transactions on Pattern Analysis and Machine Intelligence IEEE Trans. Pattern Anal. Machine Intell.* 13.12 (1991): 1225-235. Web. 03 Apr. 2016.

Pearl, Judea. "The Solution for the Branching Factor of the Alpha-beta Pruning Algorithm and Its Optimality." *Communications of the ACM Commun. ACM* 25.8 (1982): 559-64. Web. 29 Mar. 2016.

Plaat, Aske, Jonathan Schaeffer, Wim Pijls, and Arie De Bruin. "Best-first Fixed-depth Minimax Algorithms." *Artificial Intelligence* 87.1-2 (1996): 255-93. Web. 20 Mar. 2016.

"Quiescence Search." *Chessprogramming* -. Web. 05 Apr. 2016.

Section 6: Data and Figures

Game Logs	Test Number	My Player					My Opponent					Other Notes on Game	Results (in terms of my Player)				
		Percentage Other	60% Of Seeds	70% of Seeds	80% of Seeds	90% of Seeds	Percentage Other	60% Of Seeds	70% of Seeds	80% of Seeds	90% of Seeds		W/5 when I go 1st	# of Turns in Game	W/5 when I GO 2nd	# of Turns in Game	Timeouts During Moves
	1	minimaxAB depth 5					minimaxAB depth 2						5	17	5	26	
	2	minimaxAB depth 8					minimaxAB depth 2						0		0		10/10
	3	minimaxAB depth 7					minimaxAB depth 2						5		5		10/10
621-630	4	minimaxAB depth6	minimaxAB depth 7				minimaxAB depth 5						5	21	5	55	
631-632	5	minimaxAB depth6	minimaxAB depth 7				minimaxAB depth 2						5	21		55	
633-636	6	minimaxAB depth 6	minimaxAB depth 7				minimaxAB depth6						5	61	0	62	
637-640	7	minimaxAB depth 6	minimaxAB depth 7		minimaxAB depth 8		minimaxAB depth6						5	61	0	62	
641-644	8	minimaxAB depth 6	minimaxAB depth 7	minimaxAB depth 8			minimaxAB depth6						5	61	0	62	
645 - 646	9	minimaxAB depth 6	minimaxAB depth 7	minimaxAB depth 8			minimaxAB depth5						5	21	5	55	
656-657	10	minimaxAB depth 6	minimaxAB depth 7	minimaxAB depth 8			minimaxAB depth6					added evaluation in minimax for end game board...	5	57	0	59	
658-659	11	minimaxAB depth 3 +MCTS(5)		minimaxAB depth 7	minimaxAB depth 8		minimaxAB depth6					tried putting MCTS first.	0	23	0	16	
660-661	12	minimaxAB depth 4 +MCTS(5)		minimaxAB depth 7	minimaxAB depth 8		minimaxAB depth6					tried putting MCTS first.	0	1	0	1	time outs on turn 1.
xxx-xxx	13	minimaxAB depth 7		minimaxAB depth 8	minimaxAB depth 3 +MCTS(5)		minimaxAB depth6					MCTS second. Still not helping.	0		0		
xxx-xxx	14	minimaxAB depth 7		minimaxAB depth 8	minimaxAB depth 4 +MCTS(5)		minimaxAB depth6					MCTS second. Still not helping.	1		0		
664-665	15	minimaxAB depth 6	minimaxAB depth 8				minimaxAB depth6						5	46	0	59	
702-711	16	minimaxAB depth 6	minimaxAB depth 7	minimaxAB depth 8			minimaxAB depth6					added a random number generator to my evaluation function. add a random number from 1-10 to each score.	4	range	4	range	none
712-721	17	minimaxAB depth 6	minimaxAB depth 7	minimaxAB depth 8			minimaxAB depth6					random number from 1-20	3	range	3		

	18	minimaxAB depth 6	minimaxAB depth 7	minimaxAB depth 8	minimaxAB depth6	random number from 1-5					
Other Game Tests					Various Observations						
955-956	19				changing bounds -- at minimax level 8 when reach 0.69. At 0.27seconds buffer, i am able to have a turn of 1999095714 nanoseconds, or 1.99 seconds. Going to try to change it to 0.275 seconds buffer						
Vs Classmate	20	MinimaxAB with quiescence search method. Had level 3, needed to increase this to level 5, else I was cutting out good branches!!!			Minimax 5-7 opponent. 6 or 8 games played. Once changes were made, won 4/4. Increased the quiescence search from 3 to depth 5 to get a better representation of scores to choose from.		Was losing to an opponent with lower depths in minimax then me. Confused as my random player had high depths, but didnt have the intervals on both sides.				
vs My Random Player	21				Random player has same qualities as me, except for the quiescence search. Playing with depths of 7,8,8,9,10,11. Opponent has depths of 7,8,8,8,8. Played 4 games, randomPlayer times out when he tries to do minimax at level 8 in L2. Quiescence search is helping me!						
vs My Random Player	22	Going to play same random player with the fixed depths so no timeouts occur. Playing 10 games to see results. Logs 1018-1027.			Even when im down, im able to recover as I am looking ahead further depth 9. More than my opponent can. I start to run out of time when i have 80.24% of the seeds on games i start. Also decrease the lower bound. Increase timeout amount slightly too, times out on 42.7% of seeds. When i go second i seem to be loosing						
various games	23	realized against some players and boards i was analyzing that if I tried depth 7 i would end up timing out. this is possibly one of the worst things one can do.			changed mainDepth to 6., and added another depthL1 and made that 7.	No longer timeout on the first point in the game. Able to beat myself without quiescence at minimaxAB depths 6,7,8.	With this change. No longer time out.				s