# Comp 424: Artificial Intelligence
# Final Project

Geoffrey Saxton Long
Student #260403840
Geoffrey.Long@mail.mcgill.ca

April 2, 2016

# 1 Introduction

Hus is a game that lends itself well to AI development. It is from a family of rather well studied Mancala games, although Hus itself is rather unstudied. There are several variations of Mancala, but they all share the basic principles. Specifically that the board has holes which are ordered into rows; the game is played with indistinguishable tokens (henceforth referred to as seeds); each player owns a fixed and equal number of holes on the board; a move involves taking the seeds out of a selected hole and placing them one-by-one in subsequent holes (this is often referred to as sowing); the sowing may or may not terminate with capture conditions which allow a given player to seize another's seeds; and the goal of the game is to capture the most (or all) of the seeds [**?**]. In Hus, there are 4 rows of 8 holes a piece, and each player owns the 2 rows closest to them. Each player begins their turn by choosing a hole with more than one seed. If they cannot do this, they have lost the game. All of the seeds are taken from the hole and sowed one-by-one in a counter clockwise manner starting with the hole directly after the initially chosen hole. If the sowing ends on a previously occupied hole, then the seeds are taken. If the previously occupied hole is in an inner row, then the opponent's seeds on the same column can also be seized. If there are seeds to be sown, then the player continues as before.

As can be seen, Since the problem has a discrete state space, the search will be over a finite set of different game states. The changes between states in this state space is deterministic. This means that for each player action, we know exactly what the outcome will be. We might not know the full effect of the move due to the execution time constraints, but we know the immediate effects and how this alters the game flow over a finite span. This lowers the complexity and ensures that we will know the exact state of the board given a set of moves. This allows us to plan more effectively and with more certainty. The static and observable nature of the environment affords us similar benefits. Since the information on the game is perfect, planning can be performed with greater certainty.

The game rules themselves also help to make planning easier. There is a finite set of moves that can be applied at each turn, each with the same basic operator. Each player has the same simple moves and has the same simple goal, so it is easy to see the relationship between the players. The immediate cost of a player's action can be easily observed by counting the number of stones gained or lost. The simple rules, few operators, and observable cost all make it easier to derive a well rounded set of heuristics for estimating the quality of a move. The heuristics (discussed in Section **??**) are really what drives the AI agent to the goal quickly.

As can be seen, Hus is a good candidate for AI development. The remainder of this paper discusses an approach taken towards a suitable AI agent.

## 2 Background

A theoretical basis for the approach was derived by parsing articles on AI approaches in the Mancala family of games. Although no papers were found on Hus specifically, Mancala and several of its derivatives did have AI research. Since the Mancala games all have similar mechanics (described in Section **??**), many of the approaches are extensible to Hus. Specifically, an algorithm that works well for Mancala can be expected to work for Hus as well. This extensibility is mostly due to the simple and shared game operators. In any Mancala variant, a player will take seeds and distribute them evenly amongst the subsequent holes. The effect of a given action is rather similar across the entire family of games. Also the branching factor, representation, goals, and states are also quite similar among the family. Since the state, operators, goal, and cost are similar along with the environment, we can expect similar algorithms to perform similarly. That being said, variants that have mechanics that more similar to Hus's offer better indications.

Amongst all the strategies employed, MiniMax variants were the most common. Specifically MiniMax with alpha-beta pruning which relies on an evaluation function consisting of a linear combination of weighted features. Minimax variants were used in nearly all of the papers read on Mancala games.

Abayomi et al. studied a series of supervised machine learning methods in "An Overview of Supervised Machine Learning Techniques in Evolving Mancala Game Player". They focused on developing AI for the variant Awale. The majority of the methods were employed using minimax combined with some sort of machine learning. The machine learning was used to classify moves into two classes, one which helped the player and another that helped the opponent. The move was chosen which was the farthest away from the separating hyperplane. The classification was learned several ways including case-based reasoning using a perceptron; features using linear discriminant analysis; and linear discriminant functions learned with a perceptron. They also employed evolutionary computation to learn the evaluation function using both co-evolution and genetic algorithms to mixed success [**?**]. In a similar paper Randle et al. studied unsupervised methods in "An Overview of Unsupervised Machine Learning Techniques to Evolve Mancala Game Player". They initially attempted retrograde analysis, which takes a bottom up approach to scoring, but this was deemed too expensive. Then, similar to Source [**?**], they used learning to create a binary classifier separating good and bad strategies. Again, the move was chosen to maximize the distance to the hyperplane. The machine learning used was the Aggregate Mahalanobis Distance Function (ADMF), and it was used with minimax with a depth of 6. This method appeared to outperform the supervised methods seen earlier [**?**]. In "Searching & Game Playing: An Artificial Intelligence Approach to Mancala", Gifford et al. were able to achieve a 100% success rate by varying weights on a set of parameters used for an alpha beta minimax evaluation function. They found that small changes in certain weighted heuristics can make a big impact on the outcome of the search [**?**]. Other authors chose to use iterative deepening with memory enhanced test driver (MTD(f)), which is a variant of

alpha-beta minimax which searches using only zero-window windows. MTD(f) is often considered one of the more speed optimized versions of minimax. It was used for BAO by Donkers et al. in "Programming Bao" [**?**]. It was also used by Irving et al. to solve Kalah(6,5) in the paper "Solving Kalah" [**?**]. As is shown, minimax and its variants work well for Mancala style games.

In addition to the overall strategy, the papers also showed possible evaluation functions and heuristics. Although a majority of the features are not directly applicable to Hus, many could be adapted to optimize a Hus evaluation function. A list of relevant and possibly adaptable features is shown below. Note that some of the features have been altered slightly to align more closely with Hus's objectives:

- The number of pits that the opponent can use to capture seeds [**?**]
- The number of pits that we can use to capture seeds [**?**]
- The number of pits on the opponents side with enough seeds to reach to our side [**?**]
- The number of pits on our side with enough seeds to reach the opponents side [**?**]
- The number of pits with more than X seeds on the opponents side [**?**]
- The number of pits with more than X seeds on our side [**?**]
- The current score of the opponent [**?**]
- The current score of our player [**?**]
- The number of empty pits on the opponents side [**?**]
- The number of empty pits on our side [**?**]
- total number of seeds per side [**?**]
- number of holes in the front row that are filled [**?**]
- total number of counters in the back row [**?**]
- number of holes in the front row that are under attack [**?**]
- number of opponent holes that can be attacked [**?**]
- How far ahead of my opponent I am ("good heuristic") [**?**]
- How close I am to winning (¿ half) [**?**]
- How close opponent is to winning (¿ half) [**?**]

# 3 Algorithm

As was stated in Section **??**, one of the more common approaches was MiniMax with Alpha-Beta pruning. This algorithm was a logical choice for my AI agent because it is proven for this problem, well studied, and rather easy to implement. The most difficult part of the problem would be in crafting the evaluation function.

## 3.1 MiniMax

## 3.2 Evaluation Function

# 4 Testing

# 5 Results

# 6 Conclusions