Arimaa AI

Eric Thomas, Hristo Asenov, Paul Abbazia

The game Arimaa was developed specifically to challenge the limitations of current AI paradigms. While computing power has advanced to the point that many games can be brute forced, or at least allow for an exhaustive search of good moves in a lookup table, Arimaa foils these approaches through an extremely high branching factor of a given turn, with the observed average over a database of games at 276,386.[1] Additionally, as a relatively recently created game, Arimaa does not have the same kinds of established move books or opening/end game databases that exist for other board games. It is likely such books would not be nearly as effective in Arimaa, due to the simple piece movement system in Arimaa and limited chances to take enemy pieces. Arimaa pieces may move one step in any direction, with the exception of rabbits who cannot move backwards. On any given turn, a player may make up to four steps per turn and the steps can be broken up to be used by multiple pieces. Pieces can only be captured by being isolated on one of the four designated trap squares. Players have limited control over opponent pieces; stronger pieces can push, pull, or freeze weaker ones.

The challenge to developing a successful Arimaa AI then lies in either significantly speeding up computation, or, as hoped by the creator of Arimaa, bring about smarter, and more general means of AI development. In an effort to entice AI developers, the Arimaa creator has offered a $10,000 prize to anyone who can develop a bot that can defeat three selected human players in an official Arimaa match. It was our intention to try both smarter and faster techniques than attempted in other publicly available Arimaa AIs. Our approach for Arimaa begins with the application of a heuristic to narrow our initial search space. We compute the average strength of the pieces over the board, pick the 'strongest' central position, and then search all moves that are possible in the surround area[2]. It is our hope that this narrowed search will allow negascout, an optimized variant of minimax, to advance to deeper plys. In our evaluation of board states, we favor advancing pieces to the other side of the board, taking enemy

1 **Analysis and Implementation of the Game Arimaa.** *Chris-Jan Cox 2006.*

2 The area is a 4x4 grid. Positions off the board are not considered.

pieces, and moving pawns forward. Our final piece of the puzzle is the implementation of Zobrist hashing. We hash our board states to avoid generation of repeat moves, as well as to avoid having to reevaluate nodes of the minimax search tree.

In order for Arimaa bot to be recognized in the Arimaa community, the bot must be able to communicate with the Arimaa game room server. The server sends our program the turn number, whose turn it is, and the current board state. From there we generate the best possible move and send the move back to the server. A summary of our design is described in the diagram below.

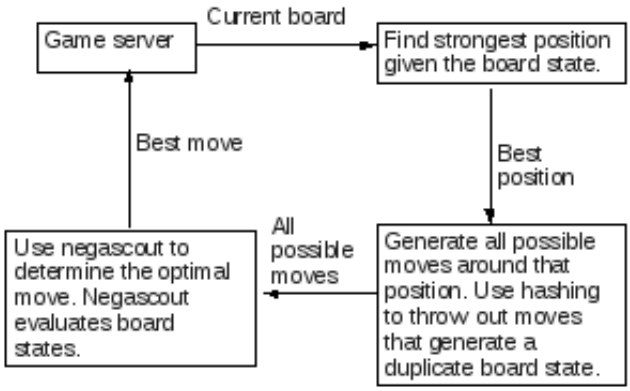

Illustration 1: A flow chart representing our design.

There are several additional changes algorithms we would have liked to implement if we had more time. A more robust evaluation function would be desired, however we lack the experience of playing the game at a competitive level that would allow us to make informed decisions about strategy. The application of additional formalism to move generation could have greatly cut down on the branching factor by generating only moves that meet some criteria of success. We would have also liked to generate virtual play-books as the AI participated in more games. By letting the AI play against itself or others, it could store the moves made in each game. By running an offline Monte-Carlo simulation, chances of success could be attributed to moves that come up during normal games, allowing the AI to quickly make good choices and then negascout at deeper ply depths.