Building a World-Champion Arimaa Program

David Fotland

Smart Games,
San Jose, CA, USA
Fotland@smart-games.com

Abstract. Arimaa is a new two-player strategy game designed by Omar Syed to be difficult for computer players. Omar offers a $10,000 prize to the first program to beat a top human player. My program, BOT_BOMB, won the 2004 computer championship, but failed to beat Omar for the prize. This paper describes the problems with building a strong Arimaa program and details of the program's design.

1 Introduction

Arimaa is a new two-player perfect-information strategy game designed by Omar and Amir Syed. Their goal was to design a game that was fun to play, and very difficult for a computer to play well. The game has free placement of pieces in the initial position to foil opening books. It has a huge branching factor and long-term strategy, which should make full width search impractical. The game ends with most of the pieces still on the board, eliminating any benefit from endgame databases.

Omar offers a prize of $10,000 for the first program that can beat a strong player (selected by Omar), in a multi-game match with long time limits. The first computer championship was in January, 2004, and was won by my program BOT_BOMB. The computer vs. human championship was played against Omar. He won all eight games, although none was easy, and the average length of the games was 55 moves. Typical Arimaa games last 30 to 40 moves.

Omar contacted me in January, 2003, and suggested I might want to write a program. While I agreed that Arimaa is a more difficult game for computers than chess, I felt I could win the contest. My opinion is that Arimaa is more difficult than chess, but still much easier than Go. It is more like Shogi or Amazons. Even though Arimaa is difficult for computers, Arimaa is a new game, and people are not very good at it yet.

I started in February, and by May BOT_BOMB was the highest rated player at the web site. This May version of the program is still available at the web site under the name BOT_ARIMAANATOR, and is rated about 225 points below the current version. This gain was due to the addition of a goal evaluation and search extensions, and adding the pin evaluation. I stopped working on it over the summer, and in September discovered that the human players had become much stronger. Several new strategic concepts were discovered, and the human players were not blundering away pieces. Several players were rated higher than BOT_BOMB. I worked on the program until the

H.J. van den Herik et al. (Eds.): CG 2004, LNCS 3846, pp. 175 – 186, 2006.

end of December, but was not able to close the gap. As in computer chess, the program is relatively stronger than people at short time controls. At 30 seconds per move average, the program's rating is about 100 points higher than the tournament version, with 3 minutes per move.

Arimaa can be played on-line at http://arimaa.com or using software available from Smart Games at http://www.smart-games.com.

2 Rules of Arimaa

Arimaa is played on an 8x8 chess board, and can be played with a standard set of chess pieces, although Arimaa renames the pieces Elephant, Camel, Horses (2), Dogs (2), Cats (2), and Rabbits (8), in order from strongest to weakest. Gold (white) starts by placing the 16 pieces in the two rows closest to him[1], in any arrangement, as his first move, then Silver (Black) places pieces on his side of the board. Human play has shown that there are several popular initial arrangements, with different strategic consequences. Silver can place his pieces to counter Gold's arrangement, which compensates for Gold's first move advantage. Because all pieces are placed at once, the first move for each side has a branching factor of almost 65 million, so making the first move part of the search is infeasible. There are over 10^15 possible opening positions, making an opening book infeasible.

After placement, Gold moves first. For each player, a move consists of 4 steps. Each step moves a piece one square horizontally or vertically, except that Rabbits cannot move backwards. The four steps in a move can be used to move one piece or multiple pieces. Any step but the first in a move can be a pass, but the move must change the board position. The goal of the game is to get one of your Rabbits to the 8th rank.

The board has 4 traps, at the 3-3 squares. After any step, if a piece is on a trap, and there is no friendly piece in one of the four squares next to the trap, that piece is captured, and removed from the board.

Stronger pieces can pull, push, or freeze adjacent weaker enemy pieces. To pull a piece, the player moves a piece one step, then uses another step to move the adjacent enemy piece into the square he just vacated. To push a piece, the player moves an adjacent enemy one square in any direction, then moves his stronger piece into the open square. An adjacent weaker enemy piece is frozen unless it has an adjacent friendly piece. Frozen pieces cannot be moved, although they can still be pulled or pushed.

Repeating a position a third time loses the game. If there are no legal moves available, the player to move loses. The only way to draw is for both players to lose all eight Rabbits.

[1] In this paper we use 'he' when 'he' or 'she' are both possible.

3 Why is Arimaa Hard?

The key issue for Arimaa and computers is the huge branching factor. Some positions have only one legal step (after a push), but most have between 20 and 25 legal steps. The four steps in a move lead to about 300,000 4-step sequences. Not counting transpositions, there are typically between 20,000 and 30,000 distinct four step moves.

Because the pieces move slowly compared to chess, an attack can take several moves to set up, so the program must look ahead at least five moves (20 steps) to compete with strong players. This is too deep for a simple iterative deepening alpha-beta searcher. Forcing sequences are rare, so deep searching based on recognizing forced moves (such as PN search or shogi endgame search) are not effective.

Sacrificing material to create a threat to reach the goal, or to immobilize a strong piece, is much more common than sacrifices for attacks in chess, so programs that focus on material are at a disadvantage.

Evaluation of an Arimaa position is difficult, and very unlike a chess evaluation. Control of the traps is important, but control of the center is not, since it is easier to move a Rabbit to the goal near the edge of the board than the center. Pieces can be immobilized for many moves defending a trap, or preventing a Rabbit from reaching a goal, which affects the balance of strength on the rest of the board.

3.1 Two Example Positions

Below I provide two example positions in the Figures 1 and 2.

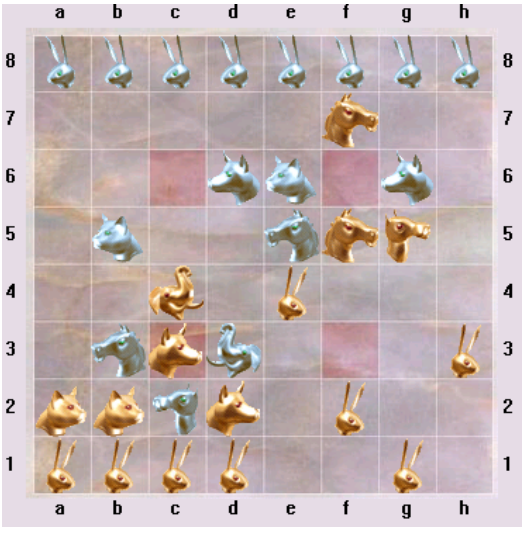

Fig. 1. Example position 1

In Figure 1, the Dog at c3 is on a trap, with the Elephant preventing it from being captured. The adjacent silver pieces are stronger than the Dog, so it cannot push them out of the way. The gold Elephant is pinned, defending the Dog. If it moves away, the Dog will be captured.

In Figure 2, the trap at f6 is dominated by strong nearby gold pieces. Once the gold Camel pushes the Cat at g6 out of the way, Gold will control 3 sides of the trap, and can start pulling pieces in. If the silver pieces move away to avoid being captured, Gold will be able to advance the Rabbit at h3 and reach the goal.

Silver has sacrificed two pieces to get an advanced Rabbit at g3. The gold Camel is frozen by the silver Elephant, so it cannot help defend the goal. If the gold Elephant moves away, Silver will capture the gold Camel in the trap at f6. Silver has a very strong position, and is threatening to reach the goal next turn.

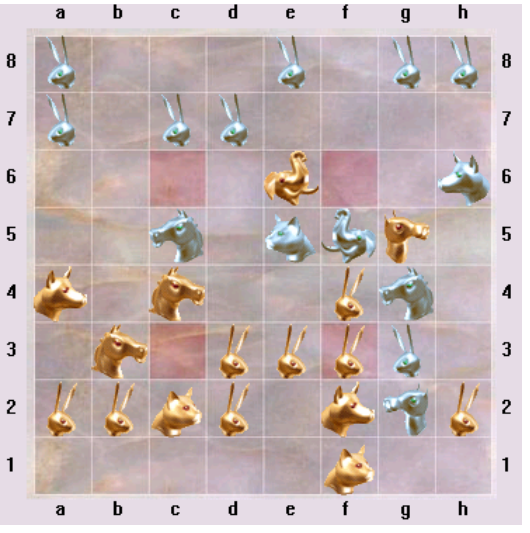

Fig. 2. Example position 2

4 The Program

My Arimaa program is called BOT_BOMB (which is the name used on the Arimaa web site, http://arimaa.com), but it also plays under the names BOT_SPEEDY, BOT_BOMBCC2004, and BOT_ARIMAANATOR. It is written in C++, derived from a chess program I wrote years ago. The rough specifications are:

4400 lines:	board representation and evaluation;
1800 lines:	search.

4.1 Board Representation

I use 64-bit bit-boards [5]. It was a good choice for a chess program, and even better for Arimaa, since the pieces move one space horizontally or vertically and there are many local evaluation terms. The bit-board class has members for logical operations, shifts, expand, count-bits, and iteration.

There is one bit-board for each piece type, one for empty squares, and one for frozen pieces. There is an array which gives the piece at each square, and another which gives the strongest adjacent piece to each square, by each color. Some flags track if there is a pull or push in progress. Additional board class members track the side to move, the step number, hash values, and the material balance. All of this data is maintained incrementally when a move is made, and copied and restored to take

back a move. The C++ operator "=" copies the core board data (373 bytes) using memcpy(), without copying any of the temporary data used during evaluation. This is much faster and simpler than writing an unmove function.

4.2 Evaluation

Below we discuss eight items of the evaluation function, viz., material, piece-square tables, Rabbit evaluation, mobility, trap evaluation, goal evaluation, pin evaluation, and center evaluation.

Material – In theory, material values in Arimaa should be completely relative, since if an Elephant is taken, the Camel becomes the new invulnerable piece. In practice, Elephants are never lost, so I use a fixed set of material values.

Rabbits	1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 12.0
Cat	2.5
Dog	3.0
Horse	6.0
Camel	11.0
Elephant	18.0

The rabbit value is adjusted depending on how many Rabbits are left. Once the last Rabbit is captured, the game cannot be won. You need two Rabbits to make goal threats on opposite sides of the board, and you need several Rabbits to defend your own goal. As the number of Rabbits goes down, the value of the remaining Rabbits goes up. The first captured Rabbit is worth 1.0, and the final one is worth 12.0.

Piece-square tables – This is a minor part of the evaluation, encouraging Rabbits to stay back to defend the goal, Dogs and Cats to stay near our traps to defend them, and the strong pieces to stay near the center and off of traps. Rabbits are encouraged to advance at the edges, and stay out of the center. For pieces, the values range from +0.1 to –0.5.

Rabbit evaluation – There is a bonus if a Rabbit has no enemy Rabbits on its file or adjacent files ahead of it (0.1 to 1.0 depending on how advanced it is). There is a bonus (0.1) to encourage Rabbits to have a piece to either side in the three points adjacent, behind or ahead. This tends to keep a solid wall of Rabbits and pieces across the board to make it harder to reach the goal. Advanced Rabbits are evaluated based on the relative strength of nearby pieces. This evaluation can be positive or negative, between about –0.3 and +5.0.

Mobility – Frozen pieces are penalized 0.02 for Rabbits, 0.1 for Cats and Dogs, 0.15 for Horses and 0.2 for the Camel. Mobility is very important for the stronger pieces. I found that if I make these penalties too large, the strong pieces get tied down freezing the weaker pieces, and become immobilized themselves.

The basic evaluation above is inadequate to prevent weak players from trapping the program's pieces. Weak players can also easily force a Rabbit to the goal. Searching

at least 16 steps is required, but is not possible due to the high branching factor. To make the program competitive with strong players, the evaluation must do a static analysis that replaces several ply of look ahead.

Trap evaluation – Trap evaluation statically evaluates how many steps (1 to 6, or more) it will take to trap each piece on the board, assuming no enemy defensive moves. For any piece that could be trapped in six or fewer steps, it statically estimates the number of enemy steps it takes to defend that piece. There are about 50 cases, evaluated with a decision tree, about 900 lines of code. The evaluation function combines the individual values to estimate the future material balance, and identify threats. The algorithm is run once for each trap, and looks at the pattern of nearby pieces.

Goal evaluation – Goal evaluation statically evaluates how many steps (1 to 8, or more) it will take each Rabbit to reach the goal, assuming no intervening moves by the opponent. This is a tricky 700 lines of code, since there are many cases. It allows the search to find goals four steps earlier, and enables a highly pruned search to find defenses against goal threats. When this was implemented, weak players could no longer sacrifice pieces to force a goal, and strong players complained that the program defended tenaciously against goal threats. The strong players shifted to new strategies that immobilized pieces, won material, and did not try for the goal until there was a large material advantage.

I test the goal evaluation with 50 problems that have a forced goal in 10 to 12 steps, taken from actual games. With the goal evaluation enabled, it solves all 50 problems in an average of 1.2 seconds and 190 K search nodes. With the goal evaluation disabled, and a 10 minute time limit, it only solves 21 problems in an average of 436 seconds and 94 M nodes. The test positions are available at http://www.smart-games.com/mate12.ZIP

Pin evaluation – It is possible to use one piece to pin two enemy pieces near a trap, giving you a material advantage on the rest of the board. You can also use several weak pieces to pin an enemy piece on a trap. The pin evaluator handles these situations. This is the most difficult part of the evaluation, since it is hard to judge the relative values correctly.

Center evaluation – Strong pieces (Horse, Camel, and Elephant) are encouraged to have access to the center. The value depends on the number of steps it would take each piece to move onto one of the four center squares. For the Elephant, it ranges from 0.35 for being on the center to zero for being 4 or more steps away. The peak value for the Camel is 0.10.

The Camel is encouraged to stay away from the enemy Elephant, unless that Elephant is immobilized, or the Camel is near a trap with several friendly pieces nearby. If the enemy Elephant is not pinned, the Camel is encouraged to stay on its own side of the board. If the Camel is advanced, the enemy Elephant gets a bonus for being behind it. The bonuses are in the range of 0.1, and are sufficient to prevent strong players from pulling a camel to their trap in the opening, but are not a general solution.

4.3 Search

The search is fail-soft alpha-beta negamax principal variation search (PVS, also called NEGASCOUT) [6] with iterative deepening and transposition table. Each iteration and call to negamax extends the search by a single step, which makes move generation simple. Because the side to move only changes every four ply, alpha and beta are only exchanged every four ply, and the code for PVS and cutoffs is a little different. PVS typically has some code that looks like:

```
if  (first move)
        score = −negamax(depth−1, −beta, −alpha);
else {
        score = −negamax(depth−1, −alpha−1, −alpha);
        if  (score > alpha && score < beta)
                score = −negamax(depth−1, −beta, −alpha);
}
```

Arimaa code looks like:

```
if  (step != 4)
        score = negamax(depth−1, alpha, beta);
else if  (first move)
        score = −negamax(depth−1, −beta, −alpha);
else {
        score = −negamax(depth−1, -alpha−1, −alpha);
        if  (score > alpha && score < beta)
                score = -negamax(depth−1, −beta, −alpha);
}
```

There can be no cutoffs in the first four ply so at the root, PVS is only used for iterations that search five steps or more. In negamax, PVS is only used at the 4^{th} step of each move. Using it at the other three steps only slows down the search. The first iteration searches three steps, since with extensions, many interesting lines go four or more steps, complete a move, and help sort the next iteration.

Null move [7] is used to prune uninteresting branches. A null move can happen on any step except the first in a turn, and causes a pass for all the remaining steps in that turn. The search depth is reduced by four steps. A null move is only tried if beta is low enough that a cutoff is likely.

If all remaining steps are by the same color, the position is evaluated to see if it can get an early cutoff.

4.4 Search Extensions

If the goal evaluation shows three or fewer steps remaining to the goal, and the player to move has that many steps left in his turn, the position is scored as a mate without further search. By an experiment it is shown that the goal evaluation is accurate up to

three steps. If there are four steps remaining, the search is extended one step to verify the goal.

If the opponent has a Rabbit four or fewer steps from the goal, the search is extended to find a defense. While searching for a defense, generated moves are pruned to only moves that are close enough to the Rabbit to affect the outcome. When there is a push, the next step is forced, so if that forced step is at depth zero, the search is extended one step.

4.5 Move Generation and Sorting

The move generator generates pulling and pushing moves first, then all others. Piece captures usually involve a push or pull, so these moves are more likely to cause a cutoff. The move generator never generates passes, although they are legal. Generating pass moves slows down the search, and does not seem to make the program stronger. The game ends with many pieces still on the board, so *zugzwang* is unlikely.

During a goal search, moves are sorted according to the distance from the piece moved to the Rabbit threatening to reach the goal. During the regular search, it first tries the move suggested by the transposition table, then two killer moves [1], then three moves from the history heuristic [2], then the rest of the moves. I tried using just the three history moves, and no killer moves, but the performance was worse, unlike the results in [3]. Perhaps this is because many Arimaa moves are quiet, so moves from one part of the tree are less likely to be effective elsewhere. Or perhaps it is because I only used the top 3 history moves, rather than doing a full sort.

4.6 Transposition Table

There is a 2 million entry transposition table [5] using Zobrist keys [4]. Half the entries are used for the primary table, and are replaced if the new value is closer to the root. Replaced entries are saved in the other half of the table.

5 Performance

On a 2.4 GHz Pentium, in the middle game, the program searches between 200K and 250 K nodes per second, where a node counts every time a move is made during the search. At 3 minute per move time control it completes 10 or 11 full steps of search, with extensions to a maximum of 14 to 18 steps.

The goal search can find 12 step forced goal sequences in under 1.5 seconds, and usually finds 20 step goal sequences within the 3-minute tournament time control. The program spends about 15 to 20% of the time generating moves, and 10 to 15% in search and move sorting. Most of the time is spent in the evaluation function. Early versions of the program with simple evaluations searched about 600 K nodes per second.

At the Arimaa web site, BOT_BOMB is currently rated 125 rating points below the top human. There are only 4 human players more highly rated. The next strongest computer opponent, BOT_OCCAM, is rated about 400 rating points below BOT_BOMB.

6 Future Work

The goal search, goal evaluation, and trap evaluation work very well, but there are still some bugs and missing cases to add. The evaluation of pieces stuck on traps or defending traps still has big problems, and leads to the loss of many games. Finally, the program has little sense of how to deploy its pieces, especially when one or more of the traps is tied up with strong pieces.

After the computer vs. human championship, the players discovered a way to sacrifice a piece to immobilize an Elephant forever, and easily win see Figure 3). Clearly mobility if the strong pieces needs more work.

The gold Elephant has just been immobilized, since it can not push anything or cross the trap. Silver can substitute weak pieces for the stronger ones involved, and get an advantage on the rest of the board. It is not easy to free the Elephant.

Fig. 3. An immobilized Elephant

7 Conclusion

Omar has succeeded in creating a game that is both fun for people to play and difficult for computers. In order to win you have to advance your Rabbits and pieces, but that puts them in danger of being pulled into traps in enemy territory. Pieces can be immobilized defending traps, giving one side a long-term strategic advantage.

A year ago it was not very hard to make a program to beat the strongest people, but the quality of top human play has improved faster than the computer programs. Any strong Arimaa program will have to evaluate accurately difficult strategic issues and have a deep selective search. Nevertheless, I have found that iterative deepening alpha-beta search is effective against most players. There does not seem to be a need for local search within the evaluation function, as in computer Go. But like Go, there are long-term positional effects that must be captured in a complex evaluation, that search cannot find.

References

1. S.G. Akl and M.M Newborn, The Principle Continuation and the Killer Heuristic, *ACM Annual Conference*, 466–473, 1977.
2. J. Schaeffer, The History Heuristic, *ICCA Journal*, Vol. 6, No. 3, pp. 16–19, 1983.
3. J. Schaeffer, The History Heuristic and Alpha-Beta Enhancements in Practice, *IEEE Transactions on Pattern Analysis and Machine Intelligence archive*, Vol. 11, No. 11, pp. 1203–1212, 1989.
4. Zobrist, A. L., A Hashing Method with Applications for Game Playing, *Technical Report 88*, Computer Science Department, University of Wisconsin Madison 1970, reprinted in *ICCA Journal*, Vol. 13, No. 2, pp. 69–73, 1990.
5. D.J. Slate and L.R. Atkin, Chess 4.5 – The Northwestern University Chess Program, in *Chess Skill in Man and Machine*, Springer-Verlag, 82–118, 1977.
6. A. Reinfeld, An Improvement on the Scout Tree Search Algorithm, *ICCA Journal*, Vol. 6, No. 4, pp. 4-14, 1983.
7. C. Donninger, Null Move and Deep Search: Selective-Search Heuristics for Obtuse Chess Programs. *ICCA Journal*, Vol. 16, No. 3, pp. 137–143, 1993.

Appendix: Championship Games

The following games can be played at http://arimaa.com/arimaa/gameroom. Follow the links to Computer Championship and World Championship.

The second best computer player was BOT_OCCAM, and the championship games are 5038 and 5039. In both games BOT_BOMB was able to pull the enemy Camel to its side of the board and get a very strong position in the opening, and games were over at move 30 and 33.

The human championship games lasted much longer, 38 to 96 moves, with a 56 move average. BOT_BOMB is able to defend its strong pieces in the opening, but

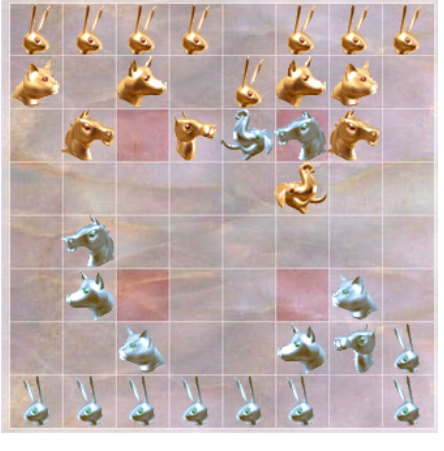

Fig. 4. After move 7

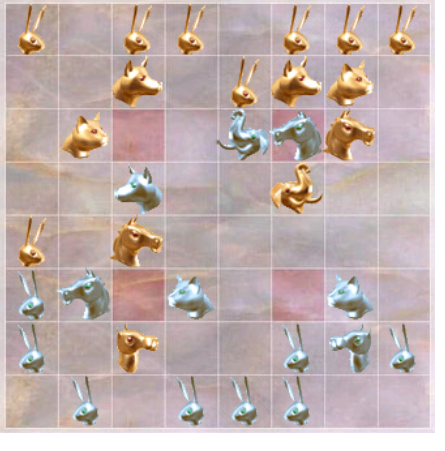

Fig. 5. After move 17

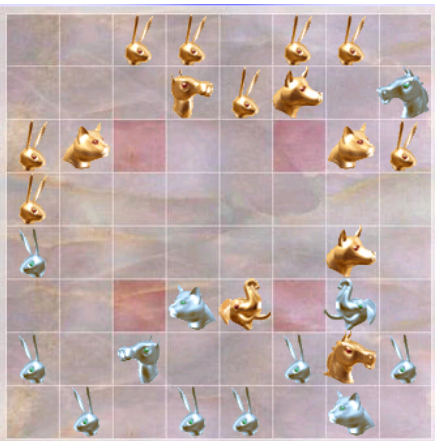

Fig. 6. After move 29

Omar was usually able to pull weaker pieces to his side of the board, trapping them or immobilizing other pieces. BOT_BOMB gave higher value to threatening Omar's strong pieces than to defending the cats, but the threats rarely led to any lasting advantage.

The fourth challenge match game, #5247: Move 7 (Figure 4), Omar (Gold) has pulled a Horse onto his trap, and immobilized BOT_BOMB's Elephant. Bomb thinks this is ok, since it takes more gold pieces to keep the Horse on the trap. But the gold Elephant can move away at any time, so BOT_BOMB's Camel can always be attacked, but Omar's Camel is free to move.

Omar launched an attack on the lower left trap, capturing a Dog, and leading to the position of Figure 5. Omar will capture the other Dog now. But Omar lets the Knight get off the upper right trap, and BOT_BOMB manages to immobilize Omar's Elephant at the lower right trap on move 29 (Figure 6).

Fig. 7. After move 36

Fig. 8. After move 51

Omar gives up the Horse to attack the lower left trap, but BOT_BOMB is able to immobilize Omar's Elephant defending the Camel at move 36 (Figure 7).

In spite of this disadvantage, Omar is able to trade a piece and capture several Rabbits, leading to this position at move 51 (Figure 8). BOT_BOMB was never able to get its Camel out to attack the upper right trap.

Now BOT_BOMB does not have enough pieces left to prevent the goal at move 67.