Fast and furious game : MonteCarlo drift
Pre-study and analysis report

Prateek Bhatnagar, Baptiste Bignon, Mikaïl Demirdelen,
Gabriel Prevosto, Dan Seeruttun–Marie, Benoît Viguier

10/15/2014

Abstract

Our project is called Fast and furious game playing, MonteCarlo drift. Our purpose is to create an Artificial Intelligence able to compete against humans using the MonteCarlo Tree Research.
We will only focus on two players games. Furthermore, we want to avoid games already resolved. We will choose something not studied entirely. We want to work on some new application. That is why we are interested by Arimaa.

For our game, we will need a program and statistics to make a good Artificial Intelligence. Each move should be calculated using a reliable method. MonteCarlo Tree Research is an algorithm able to take these optimal decisions. It has been used in the past for draughts, or chess. By exploring numerous possibilities, it will become possible to know what move is the better one. We will parallelize this algorithm in order to use it in a multi-core machine, to improve his efficiency. That MCTS algorithm is better than the classic Min-Max algorithm, that is why we will use it.

We will analyse parallelization methods, we will present it, and then we will choose the one adapted to our project. Thanks to the results of these latest methods, we will be able to choose a state resulting of the current move. Then we will explore the tree and with the same methods as before, we will figure out what the opponent will most probably do. The way we will be exploring the tree will only depend on the parallelization method. The first part of our project will be the analysis of latest thesis of technologies we will use, in order to choose the best one, and using it on the right environment, to improve his efficiency. In the next part, we will choose technologies we will need to achieve our goals, we will create a UML diagram to settle down our program.

Finally, in the last part, we will implement this program, and its documentation and test his executing on Grid5000, a cluster of multi-core machines. What is interesting in this project is we will create an Artificial Intelligence using technologies and methods fully optimized. Then we will create a program that can lead to true improvements for current algorithms applied to this game.

Contents

1 Presentation of our project

1.1 Generalities

Insert Text Here

1.2 Algorithm MCTS

Insert Text Here

1.3 Presentation of Arimaa

Arimaa is played on a board composed of 64 tiles, like a chess board. Like in chess, there are 6 types of pieces, but they are not those chess players are used to. From weakest to strongest, they are : rabbits (8 per player), cats, dogs, horses (2 of each per player), camels and elephants (one of each per player).

Figure 1: The different piece types in Arimaa.

Each player, starting with the gold player, places all of his pieces on the two back rows of his side. Then, the gold player takes the first turn. On his turn, each player disposes of four moves. He or she can use these moves on a single piece, or on how many pieces as they desire.

All pieces can move on an adjacent square (but not diagonally), except for the rabbit which cannot move backwards. A piece can instead use two moves to push or pull a weaker adjacent enemy piece, as shown in 2.

A piece sitting next to a stronger enemy piece is frozen. When a piece is frozen, it cannot move. As shown in figure 3, a piece cannot be frozen while there is an ally piece beside it.

There are four traps on the board. As shown in figure 4, any piece sitting on a trap with no ally piece next to it dies.

There are three ways to win the game :

Victory by reaching the goal You win the game if one of your rabbits reaches the other end of the board.

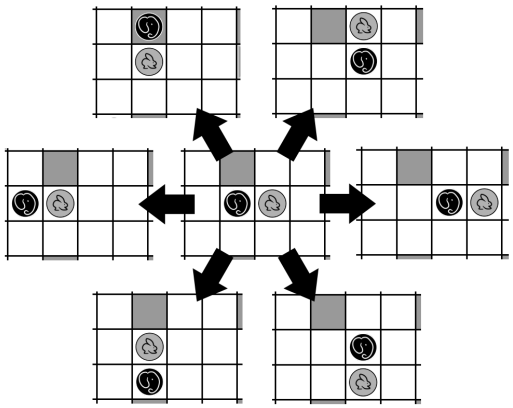

Figure 2: The different ways you can push or pull a weaker enemy piece.

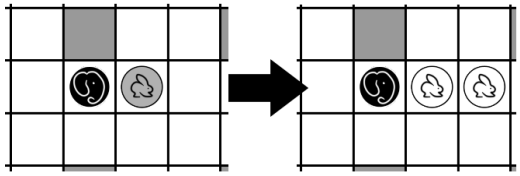

Figure 3: Example of the freezing mechanic.

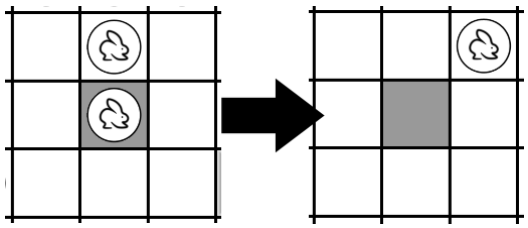

Figure 4: Example of the traps mechanic.

Victory by elimination You win the game if you eliminate all the rabbits belonging to the opponent.

Victory by elimination You win the game if the opponent can't make a move on his turn.

Victory by repetition If the same position happens three times in a row, the player that makes it happen the third time loses the game.

Insert Text Here

2 Strategies and state of the art

2.1 Strategy of root parallelization

Insert Text Here

2.2 State of the art

2.2.1 Arimaa

Insert Text Here

2.2.2 MCTS

Insert Text Here

3 Solutions and schedule of our project

3.1 Solutions we could use

Insert Text Here

3.2 Tasks' schedule

Insert Text Here

4 Conclusion

Finally, our project is about creating an Artificial Intelligence able to beat an human playing the Arimaa game.

Arimaa is a two players game. It has been designed to be difficult to foresee for computers, but easy to play for humans. In order to realize our project, we will need some concepts and technologies. We will use the MonteCarlo Tree Research algorithm, to take the best decisions[1] in our game. We will decide what move to play according to this algorithm figures. Because of the numerous moves possible, we will need to choose to develop the better ones, that will depend on the chosen parallelization tree. Any variation could totally change statistics. We already analyse the state of the art of Arimaa and the MonteCarlo Tree Research. Consequently, we will base our work on these thesis, to make it possible to use the best technologies with the best environment without doing what has already been done. We already know how to play this game. So it would be easier to create strategies for our Artificial Intelligence. We handle a task's schedule to control our requirements. We will be able to give back our work to the due time. The point of this part is mostly to learn to schedule a project, foreseeing surprising events, and answering the demand.

The point of this project is as well to test our program upon Grid5000, a powerful network of multi-core machines. Then, we will use the entire potential of our program in his maximal capacities, against a human.

References

[1] Jeremy Siek, Lie-Quan Lee, and Andrew Lumsdaine. The boost graph library - 1.56.0.