1 State of Art on MCTS

1.1 History of computers vs Humans

In 1997, Deep Blue a computer built by IBM won a six games match against the current chess world champion Garry Kasparov. Humans got beaten on Chess, but remain undefeated on Go, therefore the research has switched to that game. Until 2002, methods based on decompositions and positions evaluations were used in order to solve such games. From 2002 to 2005 the Monte Carlo algorithm was used in order to find the best moves. Since 2006, it's implementation in a tree (MCTS) has been developped, rocketing the results in term of Artificial Intelligence on Go. On june 5th, 2013, Zen a Go programm defeated Takuto Ooomote (9 Dan) with a 3 stone handicap.

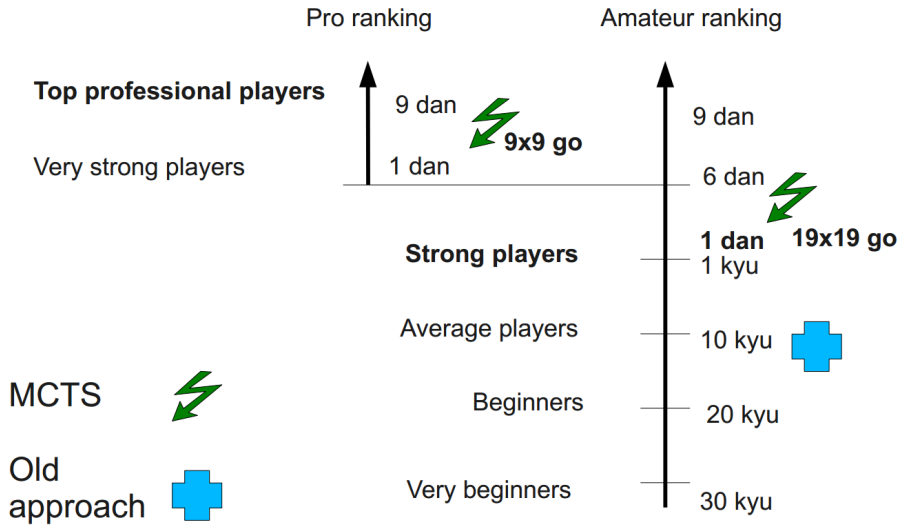

Figure 1: Comparison between Go algorithms and human skill (*2011*).

The ranking system of the game of Go is the following : kyu are for students ranks, Dan are masters ranks. Beginners start at 30 kyu and advance downward the kyu grades. Once one ranks over 1 kyu, he will receive the 1st Dan (as the black belt in Judo) and will move upward through the Dan ranks until the 9th.
However MCTS wasn't the first method applied in order to solve Arimaa, the $\alpha\beta$ method was used first. At the moment the top programms (Bomb by David Fotland : 2002 to 2008, Clueless by Jeff Bacher 2009) are ranked about 1800 elo. For comparison, strongest humans players are rated around 2450 elo. (source : Method of MCTS and the Game Arimaa, Tomas Kozelek, 2009, Master's Thesis).

The Elo rating system is a method for calculating the relative skill levels of players in competitor-versus-competitor games such as chess. It use also used for Arimaa. Beginers rank around 1200 elo, experts around 200 elo and International Masters over 2400 elo.

1.2 The minimax algorithm

The minimax algorithm is a way of finding an optimal move in a two player game. In the search tree for a two-player game, there are two kinds of nodes, nodes representing *your* moves and nodes representing your *opponent*'s moves.

Figure 2: Nodes representing your moves are generally drawn as squares, these are also called MAX nodes.

Figure 3: Nodes representing your opponent's moves are generally drawn as circles, these are also called MIN nodes.

The goal at a MAX/MIN node is to maximize/minimize the value of the subtree rooted at that node. To do this, a MAX/MIN node chooses the child with the greatest/smallest value, and that becomes the value of the MAX/MIN node.
Note that it's typical for two player games to have different branching factors at each node. The move I make could make restrictions on what moves are possible for the other player, or possibly remove restrictions. Note also that in this example, we're ignoring what the game or the probem space are in order to focus on the algorithm.

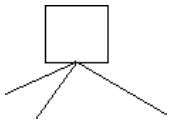

Figure 4: When we start the problem, all minimax sees is the start node.

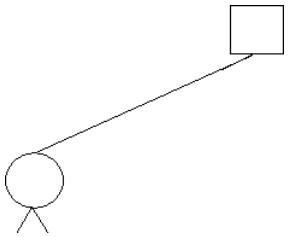

Figure 5: It begins like a depth first search, generating the first child.

So far we've really seen no evaluation values. The way minimax works is to go down a specified number of full moves (where one "full move" is actually a move by you and a move by your opponent), then calculate evaluation values for states at that depth. For this example, we're going to go down one full move, which is one more level.

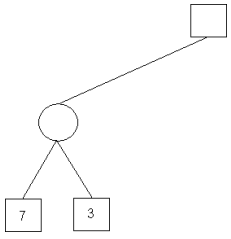

Figure 6: we generate the values for those nodes.

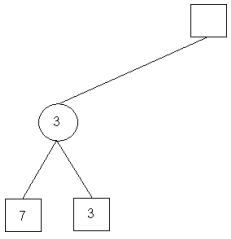

Figure 7: It chooses the minimum of the two child node values, which is 3.

The max node at the top still has two other children nodes that we need to generate and search.

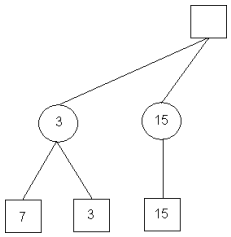

Figure 8: Since there is only one child, the min node must take it's value.

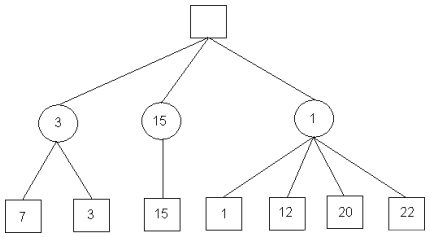

Figure 9: The third min node chooses the minimum of it's child node values, 1.

Finally we have all of the values of the children of the max node at the top level, so it chooses the maximum of them, 15, and we get the final solution.

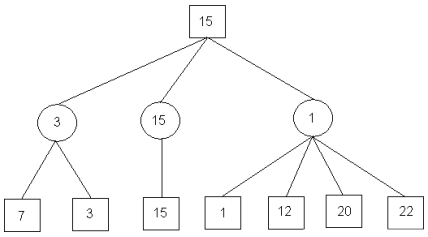

Figure 10: Final tree.

What this tells us is that we should take the move that leads to the middle min node, since it'll lead to the best possible state for us one full move down the road.

1.3 The $\alpha\beta$ pruning

The $\alpha\beta$ method is a search algorithm that decrease the number of leaf that will be explored by the minimax algorithm. That way, the size of the tree will be smaller, the algorithm will be able to dive further and the time spend on more interesting subtree is greater.
If the leaf's position is less interesting than its parents, the algorithm won't explore anyfurther.