

University of Puerto Rico

Campus of Mayaguez

Department of Electrical and

Computer Engineering

**AI Project: Brisca Agent**

By: Christian J. Lorenzo

Ibraim Figueroa

For: Professor Vega

Course: ICOM 5015

Date: June 17, 2017

**Abstract:**

When developing intelligent agents for games, adversarial search algorithms are considered. One of these algorithms is the Minimax algorithm which is a complete algorithm and optimal. The problem with this algorithm is the that it considers every possible future outcome of game which would take on almost all games a long time to finish if it can even finish such a long calculation. This leads to limiting the algorithm to go as far as to a limited number of plies and using that information to make the best possible move. Another way of making the algorithm more efficient while limiting the number of plies it can consider is by adding a heuristic function which analyzes aspect of the state the game is in to make a better-informed decision. Depending on how well the heuristic function is implemented is how much of an improvement there will be in the algorithm.

**Introduction:**

One of the areas of research and development of intelligent agents is in the field of games where competitors are trying to have the best outcome possible compared to others. For this reason, adversarial search algorithms were developed which would consider the different future possibilities of the games from the current state the game is. Example of this algorithms are Minimax and Alpha-Beta pruning which are both complete and optimal. They both create trees with the initial nodes or node being the current state and children nodes are future possible states. The problem with both these algorithms is the fact that by considering every possibility, the tree grows in most cases exponentially which would take too long for the computer to make the calculations. There are alternatives using these algorithms which make the calculations possible and in some cases, completely viable as how optimal they are. For Minimax, there is limiting the tree until a certain depth and/or limiting the branching out factor to a certain amount of plays. This makes the beginning of the game almost completely random but as the game progresses and the number of moves diminish, the algorithm will become more precise. Another alternative is the use of pruning which stops the expansion of those nodes that make no difference in the final decision. Finally, for Minimax one could apply heuristic function which finds the utility of each state by considering the states of the game and some of the conditions that are part of the environment of the game. By both limiting the tree and using heuristic function one could get even a more efficient variant of the Minimax algorithm.

For a game like Briscas, one could apply one these adversarial search algorithms like the Minimax algorithm but not completely since it would consider the 40 cards which would create a tree that would be too large to calculate. For this type of game, it would be better to apply a Minimax with a limited branching out factor and a limited depth since as the game progresses the tree will become smaller making it more efficient in its decision making. A project of Briscas applying the algorithm would require the game to only use the algorithm to make the decision for it to be a completely adversarial search program. The application of actual rules to the agent would take away from trying to make algorithm more efficient. Through means of changing the heuristic function or simulating different possibilities of the game. Apart from card games these algorithms have applications in the stock market and business decision making.

**AI agent using Minimax:**

For this project, we used the Minimax algorithm as the adversarial search algorithm for the Briscas card game.

* **Tree generator:**

We made the tree generator which expanded using the cards in the deck and as the game progress using the new deck which had less cards. The tree was made to expand exactly only two moves which would be four plies. In other words, the tree had a maximum depth of four and had a branching factor of three maximum. The tree generator would also be the function which applied a special value to the cards which would be used later in the heuristic function. The tree would also use backtracking to know which cards have already been expanded and with this information it would randomly choose a valid card to expand.

* **Heuristic function:**

For the heuristic function, we took every leaf node and assigned it a value which was calculated by means of subtracting special value of the opponent card and adding the special value of the player card which were all taken from the path that led to the leaf node from the root.

* Special value calculation rules:
  + It is taken out of the value of the card
  + Trump cards get an added value but at the beginning of the game it is taken down to a lower value so that they are retained.
* Heuristic Pseudocode:

function heuristic

q: queue

current: node

T : tree data structure

for(deck.size())

if(card.getClub == trump)

card.SpecialValue 🡨 card.getValue + 12;

else

card.SpecialValue 🡨 card.getValue;

current = root;

while (current.getLeft != null)

q.add(current.getLeft);

q.add(current.getMiddle);

q.add(current.getRight);

current = q.remove;

temp = current;

while(!q.isEmpty)

while(temp.getfather != null)

i = 1;

if(i%2 = 0)

current.SpecialValue += temp.getfather.SpecialValue

if(i%2 = 1)

current.SpecialValue += -temp.getfather.SpecialValue

temp = temp.getfather;

* **Minimax algorithm:**

After every leaf node was assigned their heuristic function value, the minimax function would first pass through every parent of the leaf node and calculate the minimum value of their respective leaf nodes. This minimum value is then assigned to the parent node. It would then run through the parent of the last nodes to be assigned a heuristic value and check the maximum value of their respective child nodes. It would repeat this steps until reaching up to the root of the three returning the card it had selected.

**Testing:**

To test the effectiveness of the agent a sample of 10 random games from a group of 20 games was taken and from that group of 10 games we found that the AI won 3 of the 10 games. This results in average of 30% of the game being won by the AI which is lower than expected. A way to improve the algorithm would be to make the heuristic function stricter (take in consideration more information) before choosing the card to play. To test this the one of the heuristic rules was changed to be less restricting which didn’t make much of a change since it got 4 out of 12 games. To test the generation of the tree and to ensure that minimax was working properly we used the debugger from the eclipse to analyze the nodes up to two moves ahead.

**Conclusion:**

Our project was partially successful as it could play the game following the minimax algorithm as planned but the efficiency of the agent was lower than anticipated. The most probable cause for this is our implementation of the heuristic function. To improve performance, we could alter the heuristic function to consider more variables other than the scores of the cards. We believe the minimax algorithm is one of the best algorithms for the agent in this project, because of the efficiency and simplicity in the implementation. In our proposal, we suggested using minimax with Monte Carlo but after careful consideration we opted for minimax because the one with Monte Carlo would take more time to implement and the improvements if any would be negligible.

Graphs:

Figure 1. Games played and their respective scores

Figure 2. Games without the heuristic that limited when trumps should be thrown