ESPADINHA - Spades Card Game AI

Pedro Guerreiro
78264
Instituto Superior Técnico
afonso.guerreiro.pedro@gmail.com

João Esteves
78304
Instituto Superior Técnico
jpp49@hotmail.com

Manuel Santos
78445
Instituto Superior Técnico
manuel.jss@hotmail.com

ABSTRACT

Computing an AI to compete in real life card game competitions has been a great challenge over the past years. In specific the Spades game must perform well not only in the card to choose decision but also in the bidding part. On top of these two pillars of the game, the agent must have in consideration its partner. This paper will give an overview over the different heuristics and algorithms implemented, comparing all its features.

1. INTRODUCTION

ESPADINHA is a program implemented in Python that solves the card game Spades. The game is divided in two parts: the bidding part and the decision making part. The bidding part of the game is done with a heuristic function. For the decision part the program has three different bots: random bot, rule-based bot and a custom bot. The implementation supports a human player to play the game.

2. PROBLEM DEFINITION

The goal of ESPADINHA is for the game to perform as well as possible, since the players want the opponent to give them a real challenge. The strongest and most complete program will have a higher chance to win the competition. One of the reasons why computing a good AI for Spades is difficult is that the information of the game is partially observable (the agent only knows its own cards).

Since the game has a high number of states (all possible combination of cards to play) we could not implement a learning algorithm like Q-Learning because it would not be possible to compute such load of information. That said, we decided to separate the problem in two features. The bidding and the card decision making.

3. GAME DESCRIPTION

In Spades there are two teams of two players. The goal of the game is for a team to achieve 500 points.

The bidding is where each player guesses the number of tricks they can do. Each team add their total number of bids. This is the number of tricks the team need to make to get a positive score. If a team does not make the number of tricks bid, they will lose the bid times ten. For example, if a team bids 7 tricks but only make 5, the team loses 70 points. Although if they make the total of tricks bet, they win 70 points. If the team exceeds the bid, they gain a bag. For each additional bag, it is awarded a point. When the total number of bags reached 10, the team loses 100 points. The nil bid is different from the others. If a player bids 0 and makes one or more tricks, the team will lose 100 points. If he makes no tricks the team wins 100 points.

After all the bidding is done the players start to play. The card must be the same suit as the person who started. If the player does not have that suit, he can trump with the spades suit. If someone plays a spade, that spades trumps all other cards except higher spades. The winner of each trick leads on the next.

3.1 Bidding

The bid decision will affect directly the score of the team. Since the bid of the player influence how the game will be played, we decided to model the problem as a heuristic function to choose the perfect amount of tricks to bet.

The bidding function looks at the higher cards in a players hand (Aces and Kings) and for each high card it increments a trick to bid. After that, we look at the trump cards (spades) and calculate the number of tricks that can be trumped.

This function is used by all players except for the random player.

3.2 Card Decision Making

After the bidding ends, a dealer is chosen randomly and the game starts. ESPADINHA has three different types of player: random player, rule-based player and a custom player. These algorithms are described below.

3.2.1 Random Player

The first attempt and the easiest way to model this problem is to implement a random player. Knowing that the random player can only choose a playable card (e.g. assisting clubs) and both the bet and the card to play is pure random, we verify that the bots will play infinitely since they never reach 500 points. This happens because the number of bets almost always exceeds the number of tricks won by the team. When we decrease the bet to a random number between 1 and 4 the bots end about 50 percent of the games played with an average of 70 games played.

3.2.2 Rule-Based Player

The Rule-Based player uses the heuristic described above for the bidding part. The card decision is implemented with

a series of rules similar to what a human would do. (e.g. play the highest card to win a trick; save high cards when the trick will go to the other teammate; trump a hand with a low spade). The player also has in consideration the cards previously played (has intern memory).

3.2.3 MCTS Player

On an early stage a Monte Carlo approach was prototyped for one of the card decision algorithms, even though this was not implemented it is still a relevant approach to be described in this section. The Monte Carlo Tree Search algorithm consists of 4 basic steps: Selection, Expansion, Simulation and Back-propagation. In the context of the Spades card game the Simulation step can be exploited in a way that prunes the number of simulations required to achieve a good average utility on the result. This can be done by having beliefs on what cards are in what players hands at any given time. Since the player can see the cards that have been played and his own cards, he can update his belief in every trick on others' hands, significantly pruning the search tree. The MCTS player would have better future prediction then the other implemented agents making it "realize" good moves that are not visible for the current trick in play like for example playing one card that does not win the trick, but takes all the cards from one suit from its own hand, making him able to cut tricks using trump cards on the next rounds.

MCTS Belief.

For the MCTS algorithm, the belief relates each player with the probability of that player having a certain card on his hand. It is updated every time the MCTS player needs to start the algorithm since it's his time to play or when it's expanding the search tree so that his belief keeps truthful to the simulated game. The probability of each player having each remaining card is done by knowing if they do possess a certain suit or not and by looking at the most likely number of high cards on his hand. The latter can be obtained by looking at the bid made by that player and to the cards that he has already played.

MCTS Node Utility.

The utility function used for MCTS would attribute a value to a node in the tree after a simulation had reached the end of 1 game (13 tricks) returning the max value (260) possible if the team got the exact number of tricks they had bet before and it would be slightly decreased by over winning (260 - 2 * over win) and highly decreased by under winning (260 - 10 * Bet). The max value is equal to 10 * max bet possible even though 26 is an impossible to reach bet, this was chosen in order to keep utility values always positive.

3.2.4 Custom Player

The custom player is modeled in terms of beliefs. For each time the players has to choose a card, it will be calculated a belief for all the cards in the players hand. The card with the highest belief is the best potential card to maximize the teams score.

This player has record of all the cards that have been played and has in count if an opponent can trump a suit or not. This allow the player to be more careful when an opponent can trump a certain suit or when high cards are yet to be played.

3.3 Agent Sensors

The Sensors available for every implemented agent are :

- Table - This sensor allows for the agent to see what cards are currently laid down on the table.
- ScoreTable - The agent can check how the score, bags and tricks are in both teams, at any given time.
- Hand - Sensor that shows the agent what cards he currently has in his hands.
- Team - The agent can also check what other player is part of his team.
- Card.Value - Represents the value of the card from 2 to 14 (2-2; Ace-14)

3.4 Agent Actuators

- Bet - This actuator allows for the agent to bet for a certain number of tricks he can win during a round.
- LayDownCard - This actuator allows for the agent to lay down a card on the table during his turn.

4. EVALUATION

In this section we evaluate each agent in terms of number of games played in order to reach 500 points. We also check the win rates of teams with different algorithms over 1000 iterations.

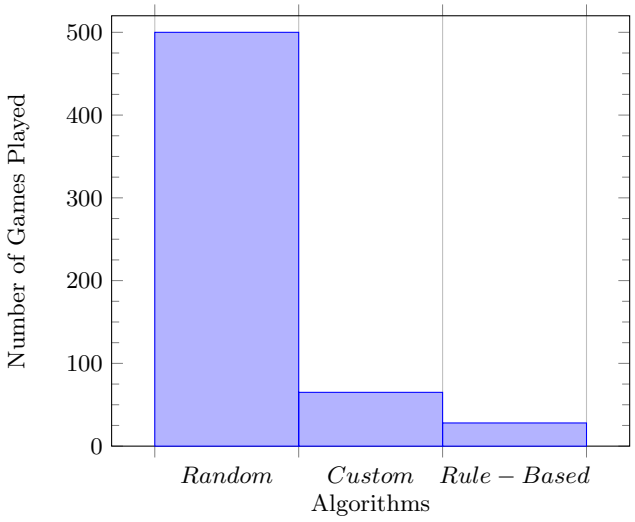

Figure 1: Games played per algorithm

5. CONCLUSIONS

Based on the presented graphs it is visible that the rule-based player was the most successful one, even though when the play style was checked the custom agent was clearly playing better in terms of winning tricks, witch lead us to the conclusion that our bidding heuristic might not be suited for our custom player and that it should be adapted in some other way. An MCTS Agent would most probably hold better results if well implemented but the bidding problem

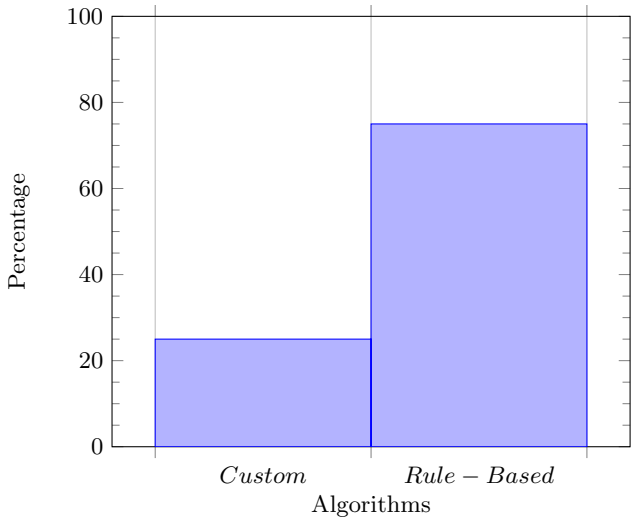

Figure 2: Win rate between Custom and Rule-Based

would have to be solved at the same time as the card decision while beliefs were being updated, causing the problem to be too complex so we eventually abandoned this idea. For future work, the heuristic that calculates the bet number should be done via MCTS while simulating playouts with the cards in hand. Then, the belief that originated that bet would be passed into the second MCTS that would solve the card decision problem, providing the agent with enough information to play according to the team's bet.