AI for Perudo

Submitted by:
Ana Hernandez 202058338

Supervised by: Prof. Michael Cashmore

Department of Computer Science
University of Strathclyde

AI for Perudo

Submitted to the faculty of the Computer Science
Department of the University of Strathclyde in partial
fulfillment of the requirements for the Degree of

Master of Science

in

Artificial Intelligence and Applications.

Department of Computer Science

University of Strathclyde

Declaration

I declare that the work contained in this thesis is my own, except where explicitly stated otherwise. In addition this work has not been submitted to obtain another degree or professional qualification.

_

Abstract

Games have underpinned a huge amount of the research and advancement that AI has undergone from the outset. Since then, AI has been widely used in a variety of games to create a range of intelligent players using different techniques. This has lead to the creation of a vast number of algorithms used for Game Theory. This paper will focus on the comparison of rule-based agents and search space agents in the game of Perudo and how these perform within different aspects of the game.

Contents

List of Figures

Abbreviations

MCTS	**M**onte **C**arlo **T**ree **S**earch
UCB	**U**pper **C**onfidence **B**ound
AI	**A**rtificial **I**ntelligence

Chapter 1

Introduction

Perudo is in many ways an interesting research topic within AI for games. It involves uncertainty, probability, partial observability, non-perfect information, bluffing, guesswork and a set of defined rules. All interesting topics in the context of AI.

The main objective of this paper is to research the application of Artificial Intelligence in Perudo - a less known Peruvian dice game. Within the many different approaches in which AI can be applied to this game, this project will focus on two techniques: rule based agents and search space agents. The difficulty will lie on building an agent that can work around uncertainty and uses probability for advanced guesswork.

What are the most salient features of the game of Perudo? What defines a better player? How can AI tackle this? This paper discusses the software used to create several artificial players and their performance.

Two different rule-based agents and two different search space agents were created, adding to a total of four different strategies. Experiments were carried out to compare their performances in different scenarios and some trial and error methods were put in place to tune parameters for some of these. The results provided insight on the importance of seating, probability and observability. Towards the end of experimentation, the clear winner amongst the players was found to be one of the search space agents using Monte Carlo Tree Search Algorithm with an adapted rollout method.

This paper is organized as follows. First, it will provide an explanation of Perudo and its rules and guidelines, followed by a brief description of the agents and environment characteristics of the game. Then, it will outline the methodology and approaches that will be applied to create different bots. After this, some experimentation will be carried out, the results of which will then be analysed and discussed. Finally, a conclusion of the project's approaches and results.

Chapter 2

Background

2.1 Perudo Game and Rules

This ancient dice game is played with 2 to 6 players and is based on probability and bluffing. There are many variants of the game, however the following rules are the ones used in the Perudo brand game (which defines the basic, common rules). The rules in *Perudo Rulebook* [2] were used as a source.

• To start the game, each player rolls a die; the player that rolls the highest number will start.

• Each player will be given a cup with 5 dice. All players roll their five dice and look only at their own. The first player makes a bid on the total count of a particular value considering all the dice on the table - most of which are unknown to this player.

• Then, the next player will be the one strictly to the left of the current player. The bidding will continue around the table, increasing the bid by a higher count or a bigger dice value, or both. For example, a call of "six fives" can be followed by "six sixes" or by "seven twos" but not by "six threes" or by "five sixes".

• Aces are jokers and are counted at the end of each round no matter what bid is made, as they count for every value. Thus, a call of "six threes" will be true if there are six (or more) dice with a value of either three or ace.

• Bids don't necessarily need to be +1 the previous bid - the bid can be raised as high as is desired. Jumping to +3 substantially increases the stakes for the next player and can be strategically used to make someone lose a die before the round reaches the risk-taking player again, thus saving them from losing a die.

• If a player believes the current bid is too high, they play **"dudo"** - which means "I doubt" in Spanish. All players reveal their dice. Dice of the same value as the last bid are counted and added to the number of aces - as aces are jokers. If the call was, say, "seven twos" and there are fewer than seven dice showing either two or ace, then

the player who made the bid loses one die. If there are seven (or more) twos and aces showing, then the player who called **"dudo"** loses one die.

• The player who loses a die starts the bidding process in the next round.

• A player who loses their final die is out of the game. The player to the immediate left starts the bidding in the next round.

• The last player with a die or dice is the winner of the game.

2.1.1 Aces

• Once the bidding has started, another option is available: calling **"aces"**. This bid calls only for the aces/jokers in a round. This means that the probability of the player's roll being true is split by half. To exemplify: when we count a normal bid of 'six fours', we are counting the dice with a value ace and a value of four; but when we call 'six ones', we are only counting those dice with value ace. Therefore, the probability of 'six ones' being true is half that of 'six fours'. To adjust for this, when one wants to make a bid of aces, the number of dice being bid can be up to halved that of the previous bid. So a bid of "eight sixes" can be followed by a call of "four aces" (or "five aces", etc.). Fractions are always rounded up, so a call of "eleven threes" becomes an aces bid of at least "six aces".

• Following a call of **aces**, the next player can either raise the quantity of aces or switch to another number by doubling the quantity of aces called, and adding one. So following a call of "four aces", the next bid must be at least "five aces" or at least nine of any other number. Of course, a call of dudo can also be made.

• A call of **aces** cannot be made on the first bid of a round.

2.1.2 Palafico

• Any player who loses his or her fourth die is declared **"palafico"**. During the bidding in the round immediately following this loss, other players are only allowed to raise the quantity of dice, not the value (e.g. an opening call by a palafico player of "two threes" can only be followed by "three threes", "four threes" etc.)

• During a **palafico** round, aces do not act as jokers; that is, they do not stand for any other number but themselves (number 1).

• Only a **palafico** player can make an opening call of "aces".

• A player can only be **palafico** once in the course of a game: the round immediately following the loss of the fourth die.

• A player with only one die left who has already been **palafico** is allowed, during any subsequent **palafico** rounds, to change the dice value during the bidding (e.g. by calling say "two fours" after a call of "two threes"). Subsequent players then follow the new value that has been called.

• When there are only two players left the **palafico** rules do not apply. Even if both players have only one die left, the dice number can be changed in the bidding and aces still only count for themselves.

2.1.3 Calza

• This is an optional rule for advanced players and is not applicable to a **palafico** round or when there are only two players left.

• A call of **"calza"** (Spanish for "perfectly fitting" or "flush") declares that the last bid that has been made is exactly right (e.g., following a bid of say "thirteen fours", there are exactly thirteen dice showing four or ace).

• A call of **calza** can follow any bid and can be made by any player, except for a player whose turn immediately follows a calzo bid (i.e. there can be no two consecutive calzo bids). Bidding immediately ends when **calza** is called and all the dice are revealed. If the player calling **calza** is incorrect, he or she loses a die. If the call is correct and there is exactly that number of dice, then the player gains a die from the bag. Note that no player can have more than five dice.

2.1.4 Pass

• A player with 5 dice has the right to skip their turn if their dice are of a certain combination. A player has a **pass** when their dice are either: all different from each other (e.g. an ace, a two, a three, a five and a six), all exactly the same (e.g. five threes) or any combination of two of a kind and three of a kind (e.g. two fours and three sixes). A player can pass only once per round.

• A player can bluff a **pass** without owning one.

• If a player calls a **"pass"**, the next player can either raise the most recent bid or doubt the pass, but they cannot doubt the most recent bid. For example, if player A bids, followed by player B **passing**, player C would have to either doubt player B's **pass** or increase player A's bid.

• Note that in the context of a pass, the joker/ace doesn't "equal" other dice. For example, if a player owns four fives and an ace, this isn't a **pass**. It must be five equal dice (e.g. five threes - without aces, or five aces).

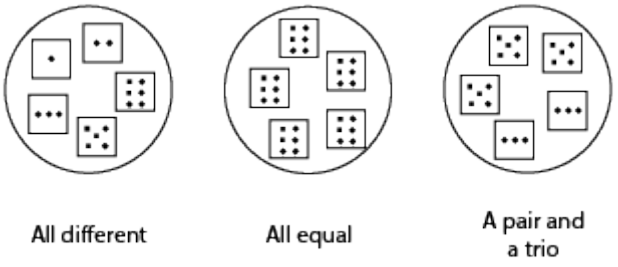

FIGURE 2.1: Roll for a Pass

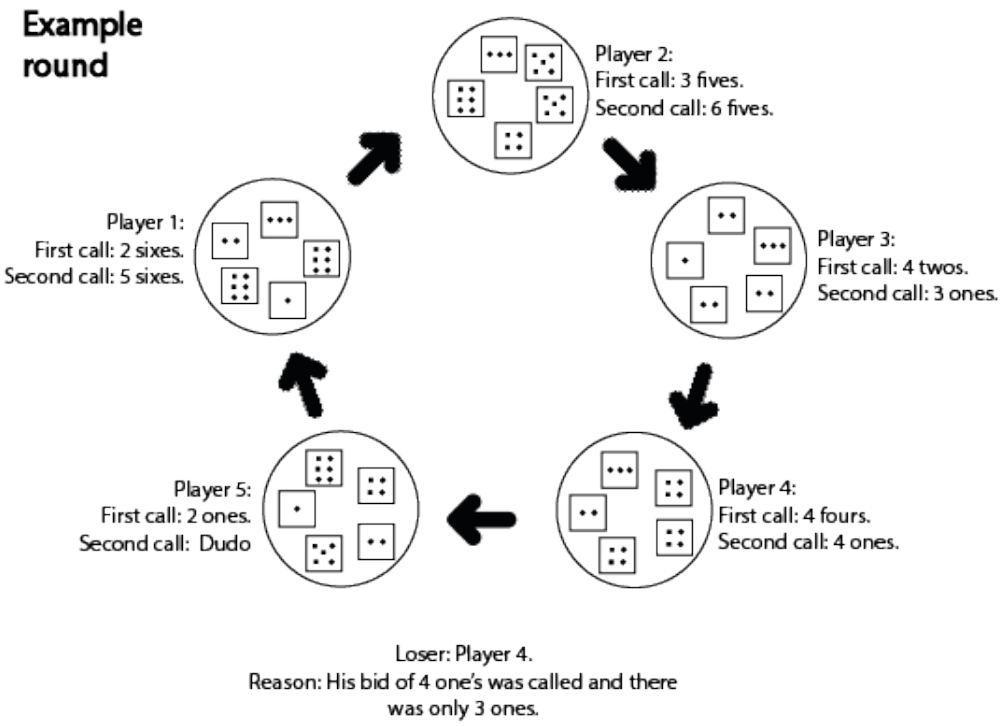

FIGURE 2.2: Example Round

2.2 Available Software and Research

Tabletopia, perudo.com and funnode are some of the only online Perudo playing platforms, which allow playing only with other human opponents. The market for online Perudo isn't big, not to mention AI Bots.

Some research was found on Perudo strategy in *Computer Strategies to the Perudo Game*[1] which studies computer bluffing strategy focusing on Perudo, and *Game Design Through Self-Play Experiments*[4], which explores the application of game design in Perudo. Nonetheless, the resemblance of game strategy amongst Perudo and Poker is substantial. Finally, *AI Poker Agent based on Game Theory* [3] was used for research on strategies, dealing with uncertainty and bluffing techniques.

Chapter 3

Problem statement

This section is focused on describing the problem to tackle and some of it's difficulties. Firstly, the agent and environment characteristics are analysed and later, the key features for this game's strategy are discussed.

3.1 Agent and Environment

There will be two types of agents created: model-based agents which will use solely rules and guidelines for it's decision making process, and goal-based agents, which will make use of search-based strategies to predict future outcomes as a consequence of their decision.

The agent's environment can be defined as follows:

• The environment is **partially observable** as each player can only see their own dice. At the start of the game, each player's partial observability is equal as they hold all their dice. However, as the game continues, players with a larger number of dice will have a higher observability percentage than the rest.

• Perudo is played with 2-6 players, therefore this project is focused on the design of a **multi-agent** system for players to be able to play online themselves against multiple AI's.

• Games can be based on strategy where fortune plays no role and decisions are made in a deterministic way, like chess. Other games can be based on chance where the results of the game are decided by stochastic factors, like gambling. However, Perudo is a **mixture between deterministic and stochastic**, each player depends on their stochastic dice roll which they then create a deterministic strategy with, like Poker.

• The agent only requires memory of the current round, and when that is finished no past memory is required for a new round. The experience is **sequential** but is restarted in every round.

• The environment does not change while the agent is deliberating their bid, which means it is **static**.

• Due to the finite number of percepts and actions we can classify this environment as **discrete**.

3.2 Game strategy factors

Perudo is based on three important factors: probability, bluff and guesswork. The probability features of this game will be more intuitive to program, whereas bluffing and guessing are highly complex features to indulge in an agent. The key points to focus on when shaping the strategy of the agent will include: probability, bluffing, fixed seating, partial observability and greediness.

3.2.1 Probability

The game strategy is highly dependant on probability, however, over reliance on probability will almost certainly result in a loss [4]. The most probable distribution is a binomial centred on the total nº of dice divided by 6 (possibilities of a dice). In the case of the first round with a game of 6 players, the center would lay on 5 dice of each dice value. However, bids should hint what dice values are unbalanced - the more a value is mentioned in a round the more likely it's count is above average. Although, players can bluff.

3.2.2 Bluffing

Bluffing will allow a player to decrease its chance of losing when having a weak roll. On top of this, reading the room to guess the predominant dice is an important factor of the game, but also to be an astute enough player to not be bluffed easily. Finding this equilibrium will be of increased complexity. Also, if opponents bluff but the player doesn't, this creates a disadvantage where they can almost accurately guess the player's dice increasing their knowledge of the round, while the player will believe they do too, although some information might be fake.

3.2.3 Fixed Seating

For an entire game of Perudo, the players seating order is fixed. A player only interacts with the players strictly to their left and right. If a player is sat between two very good players, winning will be harder, and viceversa. This will encourage using two different methods of evaluation, one with random seating every round and another trial of full games, these will provide insight on different aspects of each player.

3.2.4 Partial Observability

Hidden information poses a challenge within AI. At the start of the game, all players own the same quantity of dice, and hence, the proportion of observability is equal around the table. As the game evolves, players will have an observability proportional to the dice they hold.

Players that hold the most dice, and hence are winning, will consequently have a higher partial visibility ratio of the entire environment space than a player who is losing - which provides them with more information to make reasonable bids. Therefore, once a player loses a die, their probability of winning the game is very much reduced [4].

Not only the player has no observability other than their own dice, but the information they receive (bids from opponents) can be true or fake. This essentially entails decision making with partial observability of the environment and uncertainty on the honesty of the opponents calls.

When applying Monte Carlo Simulation to some of the designed agents, the uncertainty of opponents dice will complicate the simulated roll outs of the game, as the opponents will bid depending on their rolls which are unknown to each player and therefore, computationally expensive to simulate for all possible scenarios.

3.2.5 Greediness

One could claim trying to keep the bids as low as possible would result in a lower risk strategy, and therefore applying a greedy approach would keep the probabilities of losing die to a minimum. However, the game goes around in rounds and will come back to the same player until someone calls 'Dudo'. When the game is just starting, jumping the quantity of the bid by a lot can increase the players risk by a considerable amount, but also being greedy can result in the game going around and reaching the current player again in a riskier stage of the bidding process. A trade-off between greediness and riskiness will have to be achieved when bidding.

3.2.6 Evaluation of a strategy

As the experimental phase will be based on games of Perudo between AI agents, the concern of clashing strategies may arise. If all agents use the same code to play, how will we be able to, first, effectively evaluate its performance, and second, if this AI Player were to be used for human vs AI games, make sure that using the same strategy doesn't result in a poorer experience for the human player.

Chapter 4

Methodology

This system was created on a PC with Intel Core i7-7660U CPU and 16GB RAM, Windows operating system and Python language.

4.1 Approach

As mentioned in the problem statement, Perudo has many factors to tackle when it comes to designing an AI Player. Exploring multiple approaches and scenarios will result in either a clear strategy winner, or a defined winner for each different stage of the game - in which case, an ensemble player which uses a different player depending on the game phase would be the better option for a final optimal AI Perudo player.

All the different strategies and parameters will be used and experimented with to gather data and come to conclusions on their performance. *Computer Poker* [6] paper was used as a source to decide which algorithms to use for this project.

Search-based agents and rule-based agents are the focus area in this project. In total, four different strategies will be created: two of them are rule-based agents, and the other two are search-based agents. One of the rule-based agents will have a simpler decision making path, with the purpose of using this one as a base for comparison with other models. Similarly for the search spaced agents, one will use random rollouts and the other will use a more informed search tree.

Player	**Name**
Rule Based Agent 1	Probabilistic Agent
Rule Based Agent 2	Observer Agent
MCTS 1	MCTS Random Rollout
MCTS 2	MCTS Observer Agent Rollout

(4.1)

4.2 Rule-Based Agent

An artificial agent is defined as a rule-based agent if its behaviour is encoded by means of rules. These methods are commonly used for logic and real-time decisioning systems.

In the case of a Perudo game, several if-then statements will cover the entire search space for all the different scenarios the player can be found in and will decide on an action accordingly. These flowcharts are very straight forward and involve no learning or training from the agent.

Testing a rule-based agent against a human would take far too much time compared to an agent system. When confronting two agents using the same strategy, their strategies won't clash as their decisions will depend on their stochastic rolls, however, as would be expected after many rounds, the performance would eventually result in something close to an equal number of losses for both. Therefore, a ground base agent will be created for the sole purpose of comparison and evaluation.

Some inherent bias might be present in these agents, due to the fact that these are built with a structure of if-then conditions based on a human's opinion and updated using the performance and feedback of confronting itself against only one other agent.

4.2.1 Probabilistic Rule-Based Agent

This model's strategy is purely probabilistic and will be used solely for comparison against other models. It would be expected of this model to lose when competing against more complex and developed agents when several simulations are run.

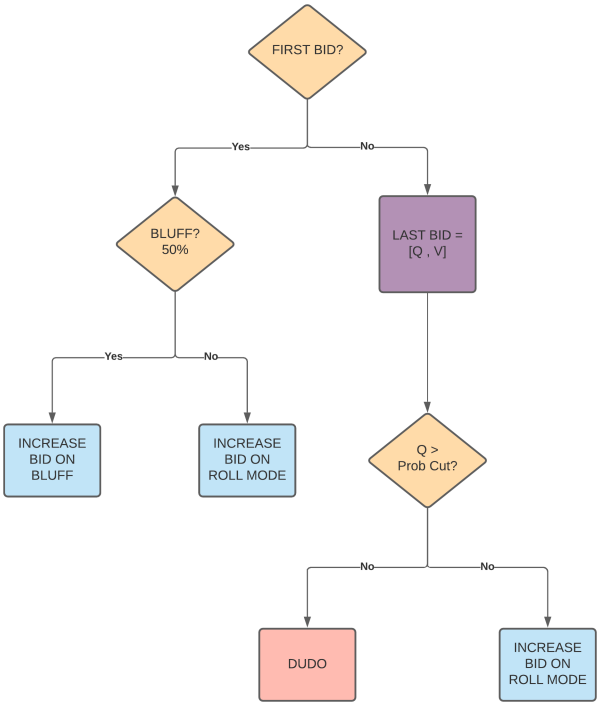

FIGURE 4.1: Probabilistic Agent Flowchart

The actions flowchart of this agent is very simple. The previous bid is taken in and analysed; if the quantity is above the probability cut (Equation 5.1), the agent will call 'Dudo'. If the quantity of the previous bid is not above the probability cut, then they will increase the bid by 1 on the most common dice value of their roll. Their own roll is considered for increasing bids, however when calling 'Dudo' they are basing this purely on probability. This player won't store bids or call calza.

$$P(Jokers) = \frac{Dice}{6} \qquad P(Vals) = \frac{Dice}{3} \tag{4.2}$$

It can be assumed that the performance of this player will be higher when there are several dice in play as, due to the rule of large numbers, the count of each dice value will converge to near equal numbers for all dice values. However, when the player is closer to the final and fewer dice are on the table, this can be a poor strategy as it will call dudo on bids above average, even when the player owned all the dice needed for this bid to be true.

4.2.2 Observer Rule-Based Agent

This agent is designed to take in some more information for its decision making process. The main differences from the probabilistic agent are the calculation of binomial probability, the use their roll to their favour and their observation of other players' bids to conclude on which dice values are unbalanced. For instance, if a dice value has not been mentioned at all in this round, it can be deduced that it will probably have a smaller count than probability would suggest.

As a ground base, the player will have a risk equation that focuses on the likelihood of the previous bids, to be as mathematical and reasonable as possible. The risk equation output - which is based on a binomial probability of the previous bid's success - will define the game phase the player is currently at. There are four phases; risk free phase, safe betting phase, danger phase and dudo phase.

Each one will differ in terms of greediness, riskiness, proneness to bluff and amount of observation towards opponents. Towards the end of the game, given the small sample size of dice, probability is less reliable and guesswork is a more powerful skill.

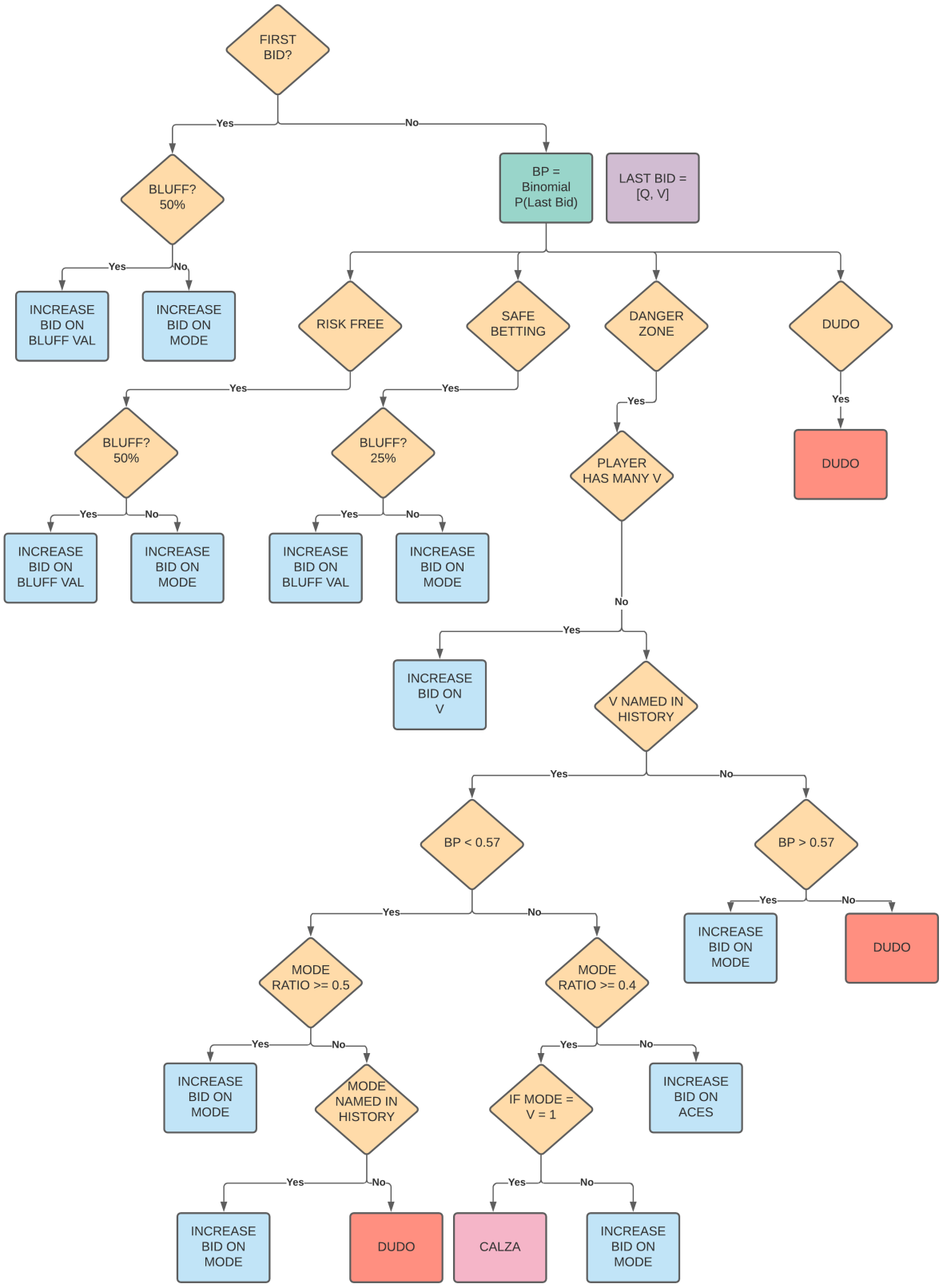

FIGURE 4.2: Observer Agent Flowchart

4.2.2.1 Binomial Distribution Calculations

To calculate the probability of the **previous player's bid being true**, the following equation was used:

$$P(X \geq A) = 1 - P(X < A) \tag{4.3}$$

where:

$P(X \geq A)$ = Probability of at least A (Bid Quantity) of a dice value V
$P(X < A)$ = Probability of less than A (Bid Quantity) of a dice value V

With this we want to know how likely it is for there to be at least A (bid quantity) of a given dice value V. To do this, we will calculate the probability of its complement, that is, the probability of there being less count of V than A, and then subtract this from 1.

Using the cumulative probability of all the different events that achieve X < A, and applying the correct probability of success and failure for each bid, a probability formula can be created for this to be calculated in every instance.

$$P(X < A) = \sum_{k=0}^{A-1} \binom{n}{k} ProbSuccess^{k} ProbFailure^{n-k} \tag{4.4}$$

In the case of aces, the binomial probability is the following:

$$P(X < A) = \sum_{k=0}^{A-1} \binom{n}{k} \left(\frac{1}{6}\right)^{k} \left(\frac{5}{6}\right)^{n-k} \tag{4.5}$$

In the case of any other value, the binomial probability has different cases of success and failure, as when we are counting 2, 3, 4, 5 and 6 we will also count the 1s - which make 2 out of 6 dice events successful and 4 out of 6 unsuccessful.

$$P(X < A) = \sum_{k=0}^{A-1} \binom{n}{k} \left(\frac{2}{6}\right)^{k} \left(\frac{4}{6}\right)^{n-k} \tag{4.6}$$

The formulas above show the probability that X, which is the count of a given dice value for a certain round, is smaller than A, which is the quantity of the most recent bid. n is the total number of dice and k is the count for a given dice value in this round. These variables are then fed into the binomial cumulative probability equation.

The graphs below shows an example of the probabilities this would result in if we were to apply this on the first round of a 6 player game.

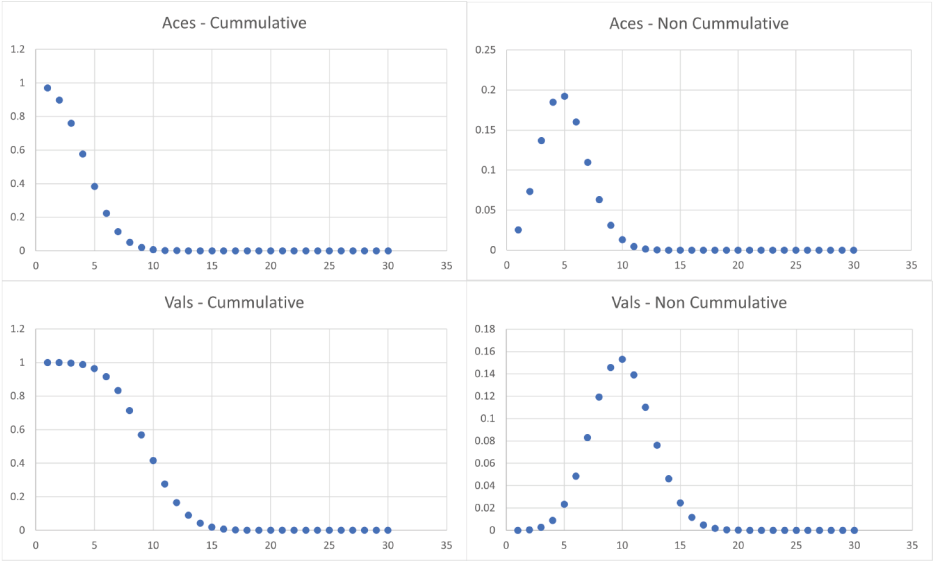

FIGURE 4.3: Binomial Probabilities

The cumulative probability would assist in the decision making for calling 'Dudo' and the non-cummulative probability would aid on the decision making for calling 'Calza'. These values will change for every round as the total number of dice will change too, therefore a function has been integrated into the agent to calculate the probabilities for each scenario individually.

These values will aid the agent in making an informed decision, creating a threshold for which the agent will not continue bidding over. This parameter will be tested and tweaked in the analysis section for higher efficiency.

4.2.2.2 Use of its own roll

When the previous bid is in a compromising probability of success - that is, when the previous bid forces the current player into a position where they are more likely to lose - the roll must be used to provide some more insight into the chances. The player's roll will always be considered when bidding, unless they bluff, which is 1 in 2 times in the first bid, 1 in 4 times during the course of the game, and never in the compromising final calls.

For example, if the previous bid results in a compromising probability of success but the bid is based on a value which the observer player's roll has an abundance of, then this could result in the conclusion that this value's count might be above average in this round. This will also be verified with previous bids to make sure this dice value has been mentioned in the round bidding history.

4.3 Monte Carlo Simulation

Tree search algorithms are very popular amongst AI for game strategy. Algorithms like Minimax or alpha beta prunning will always lead to the optimal result - supposing your opponent plays rationally too. These algorithms explore the entire search space to be able to provide the path to the goal. For games that have a large search space - such as Perudo - the full tree expansion could be immense, so limiting the depth of the search space by using an evaluation formula which judges the effectiveness of an action would result in a more computationally friendly and less time-consuming approach.

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm which blends the classic tree search implementations along with some reinforcement learning principles. In MCTS, exploration of other moves and evaluation of their optimality is carried out periodically by simulating the aftermath of some different actions. Converging to optimality will always be possible, as this technique will converge to minimax output after many rollouts, however the computational cost will be too high. To balance these two, some tweaking of the 'rollouts' parameter will be used, which defines the number of simulations it will carry out from the root node before it picks the best move. The higher the number of rollouts, the longer the computational process but the closer to optimality and viceversa.

Below is a flowchart of the MCTS algorithm where 'ni' is the number of node visits, 'rollouts' is the number of simulations and a 'leaf node' is a node which either hasn't been expanded or is terminal.

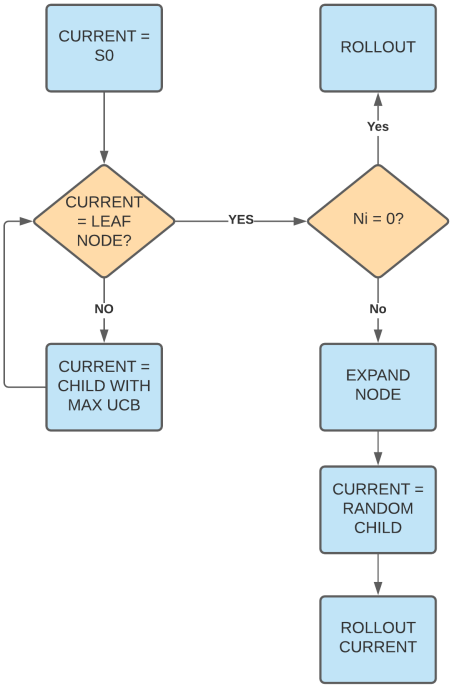

FIGURE 4.4: Monte Carlo Tree Search Flowchart

4.3.1 UCB Formula

$$UCB = vi + c\sqrt{\frac{\ln N}{ni}} \qquad \text{(MCTS Player UCB formula)}$$

where:

vi = Average node value
ni = Node visits
N = Parent visits
c = Scalar

Rather than performing exploration by simply selecting an arbitrary action, chosen with a probability that remains constant, the UCB algorithm changes its exploration-exploitation balance as it gathers more knowledge of the environment [5]. The returned score of this algorithm will be used to assess and compare the possible moves. The first half of this equation defines the exploitation and the second half defines the exploration of a node - which is at the same time controlled by the hyper parameter 'c'. The higher the 'c' value, the higher the encouragement for the exploration of less visited ones, and vice versa.

If an action remains unexplored, the second half of the UCB equation tends to infinity. As the selection process is based on picking a child node with the highest UCB score, this will encourage these unvisited nodes to be visited first. For each time a node is visited again, the second half of this equation decreases, putting more weight on the first half of the equation - the exploitation.

A zero-sum game implies that the loss of a player is the gain of the opponents, which is the case of Perudo. For rollout purposes, it is important to take into consideration that the opponents' goal is for the MCTS player to lose and the other way around.

The use of the UCB formula is to pick the best move to explore considering its visits and its value. Although, when we are a few states away from the current state in the tree search, we need to consider the fact that this would represent the opponent's turn and not their own, and they will consequently not be picking the child with the highest UCB value, as this scoring function revolves around the MCTS player (positive if they win and negative if they lose). Therefore we need to assume opponents will choose the move that minimises the value of the chosen node, and only the MCTS player will choose the move that maximises this one.

$$UCB = -vi + c\sqrt{\frac{\ln N}{ni}} \qquad \text{(Opponent UCB formula)}$$

4.3.2 Rollouts

For every rollout, each opponent is assigned with a random roll and a simulation of a round is executed with these dummy rolls. When any player calls dudo, the MCTS players' actual roll for the round and the generated dummy rolls are counted for evaluation. The reason why new dummy rolls are generated in each round is so that a player's strategy is not fixated on one possible roll (which is quite unlikely to turn out to be the real rolls for the opponents), and to pick the overall best move regardless of opponents' rolls.

4.3.3 MCTS - Random Rollout Policy

This agent will use the Monte Carlo Tree Search simulation with a random rollout policy, which means that when it runs a simulation from a node until a terminal node, it will use random actions from the possible actions available. The performance of this model will highly depend on the rollout parameters, as the higher the number of rollouts, the closer to optimality the result will get.

Nonetheless, considering the wide range of possible moves within a state of this game, when picking a random action, 'Dudo' or 'Calza' are unlikely to be picked. Therefore, the rollouts are very unlike real games of perudo due to their long depth.

4.3.4 MCTS - Observer Agent Rollout

This agent will work in a similar way to the one above, however instead of using random actions for a simulation to reach from a node to a terminal node, it will use the observer agent's decision making process as a sub-agent. This means that, when it 'would be' the opponents' turns, it will call the observer agent; otherwise, it will use random actions for itself. When this method is used, the observer agent class will be fed the dummy roll to return a bid accordingly.

The advantage of this agent is that for every layer in the tree in which the player isn't the MCTS player, it will not explore all the possible actions. Only a likely response from the opponent (given their pseudo-roll) will be explored, which will result in a tree with much less exploration, focusing the rollouts on the MCTS player's decisions and not the opponents'.

Chapter 5

Experiments

This chapter will give an overview of the different experiment set ups and players strategy. The reasoning behind using so many different players is to highlight each players strengths and find the possibility of combining these strengths into a merged optimal player.

The playing order can affect the results of the game as a player will interact only to the opponents strictly adjacent to him. Following a good player will result in a harder chance to win and vice versa. Therefore performance will be measured with random and fixed seating to first, evaluate the overall best player, and second, to observe the interactions between players.

Experiment 1	Rule-Based Agent vs Probabilistic Agent
Experiment 2	Parameter Tuning for MCTS
Experiment 3	Comparison of all Agents

(5.1)

5.1 Experiment 1 - Probabilistic vs Observer

In this experiment we will analyse the strategy of both rule-based agents, comparing their results and performance. Firstly, we will compare their losses when carrying out rounds. These results will highlight the overall performance of each strategy without considering the changes in partial observability along a full game - as all tested rounds will involve all players with all 5 dice. Afterward, we will carry out a similar experiment with full games, which reflect a higher impact on the seating order and the importance of losing dice for a players chance of winning.

5.1.1 Rounds

After confronting one probabilistic agent against an observer agent, the results pointed to a vast majority of losses for the probabilistic agent. More concisely, almost 1400 losses out of 2000 for the probabilistic agent and around 600 for the observer. Meaning less than one third of the games were lost by the observer agent.

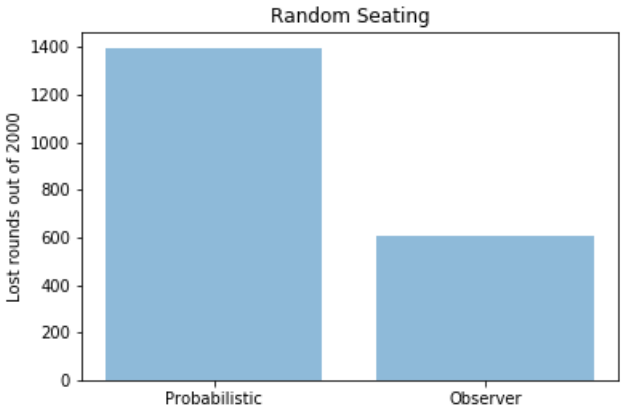

FIGURE 5.1: Experiment 1a - Probabilistic vs Observer - Rounds

Due to the strategy of the probabilistic agent, all bids from the observer agent that would pass the probability cut threshold would be called, therefore it would make sense that the reason of loss for the probabilistic agent will not be composed by a large number of high bids, and will mostly be the result of calling 'Dudo' too soon. On the contrary, the Observer agent is more lenient towards higher bids and is riskier. We can expect that the reason of loss will be higher for high bids than for early calls. However these results are not so accurate for the observer agent, as the probabilistic will never make a bid above the probability cut it will be unlikely for the observer to gather many 'Dudo too soon' losses, as the opponent will never make high bids to trigger these calls.

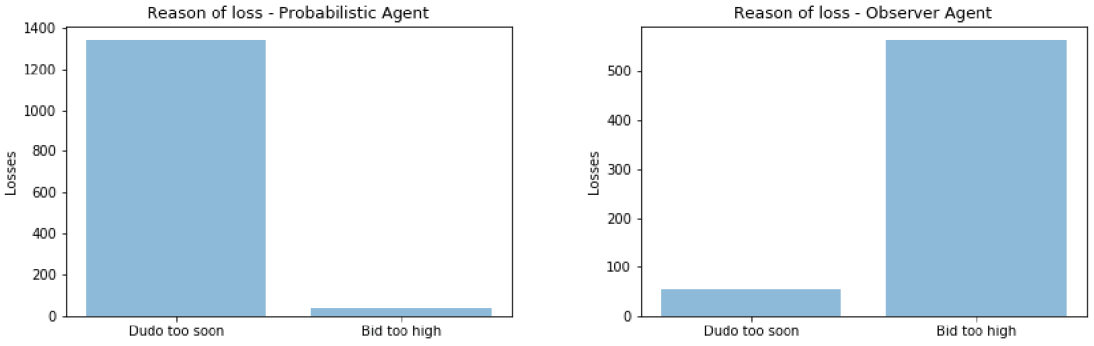

FIGURE 5.2: Experiment 1a - Reason of loss

5.1.2 Full Games - Fixed seating

In this experiment, the repercussions of losing a die and the effect of fixed seating are visible. For a player to win a full game of Perudo, they need to maintain at least one die by the last round. As observed above, the probabilistic is twice more likely to lose a round, considering they all have 5 dice. However this ratio will change when the players have different numbers of die - as this is a huge disadvantage.

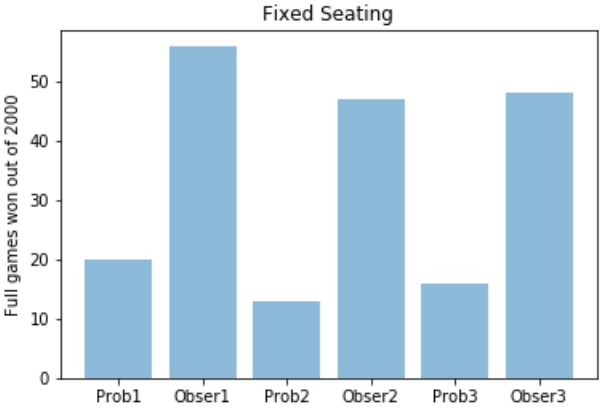

FIGURE 5.3: Experiment 1c - Probabilistic vs Observer - Games

As can be observed in the picture above, for every 2000 games, the probabilistic agents would win an average of around 15 games and the observer agents around 50. In this experiment, seating is fixed for the period of a full game, and then players are shuffled. These visualizations should reflect the likelihood of winning a full game dependant for each strategy. Seating will affect for every game individually, but each game will have reset seating order.

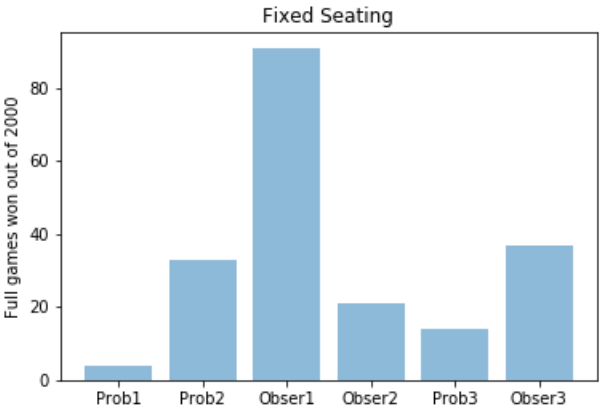

FIGURE 5.4: Experiment 1d - Probabilistic vs Observer - Games

On the contrary, in the experiment above, the seating is fixed for all games and rounds carried out. This reflects the importance of seating order, as sitting after two weak players will result in a much higher chance of winning and vice versa. Observer player 1 is at a great advantage with this seating order as its placed behind two probabilistic agents. Due to their strategy, they will only bid rational bids within probability which reduces the chances of placing Observer 1 in a compromising bidding phase. On top of this, after Observer 1 is Observer 2 which is much more lenient when it comes to calling 'Dudo'. This means Observer 1 is places behind players which keep their bidding low and in front of a player which will not call 'Dudo' swiftly.

In conclusion, this experiment highlights that over reliance on probability does not result in a strong strategy in Perudo, but it is a powerful tool. The observer agent uses probability to consider their risk but also checks other factors before making a final call.

5.2 Experiment 2 - Parameter Tuning for MCTS

Monte Carlo Tree Search algorithm has a few parameters to look into to achieve high calibre results. These include the number of rollouts, the scalar used in the UCB formula and the scoring function to assign a value to a rollout.

To evaluate all some different combinations of scalars and scorings, 6 different experiments will be carried out against the probabilistic agent as a comparison base, both players with 5 dice and the MCTS player set to 1000 random rollouts.

5.2.1 Scalar and Scoring Function

$$UCB = vi + c\sqrt{\frac{lnN}{ni}} \qquad \text{(MCTS Player UCB formula)}$$

where:

vi = Average node value
ni = Node visits
N = Parent visits
c = Scalar (often 2 or $\sqrt{2}$)

The tuning of the scalar (c) in the UCB formula and the scoring function are somewhat related. As explained in the methodology section, the scalar is used to either encourage exploration over exploitation or the inverse. However, if the values which are backpropagated after each rollout are much higher or lower than the backpropagated visits, this will silence the exploration side of the UCB function by making this value much smaller in comparison to the exploitation one. In essence, if the scoring system isn't growing or decreasing at a similar rate than the visits, the exploration values can be lessened.

If an MCTS algorithm is strongly focused on exploration, it will eventually converge to minimax algorithm but, as discussed previously, minimax algorithm would not be computationally friendly or time effective for a game like Perudo due to its large number of game states. On the other hand, an MCTS algorithm which is mostly focused on exploitation will use up most of it's rollouts by picking the node with highest value over and over again - without exploring other options that could score higher if given the opportunity of investigation. Therefore, the scalar and thus the scoring system are a spectrum. One end of this spectrum involves little to no exploration and results in non-optimal choice from the player, and the other end of the spectrum is a high computational cost and a time consuming method to reach the optimal path. The difficulty lies in finding the sweet spot in between, in which the computation cost and time is feasible but the result is close to optimality.

5.2.1.1 Experiment 2a

The first experiment for parameter tweaking was to compare both common scalar values with a very simple scoring system. The loss of a die results in a -1 score, the gain of a die is defined as a +1 and anything else scores 0. This was the most intuitive way of scoring as the values represent the gain and loss within the game.

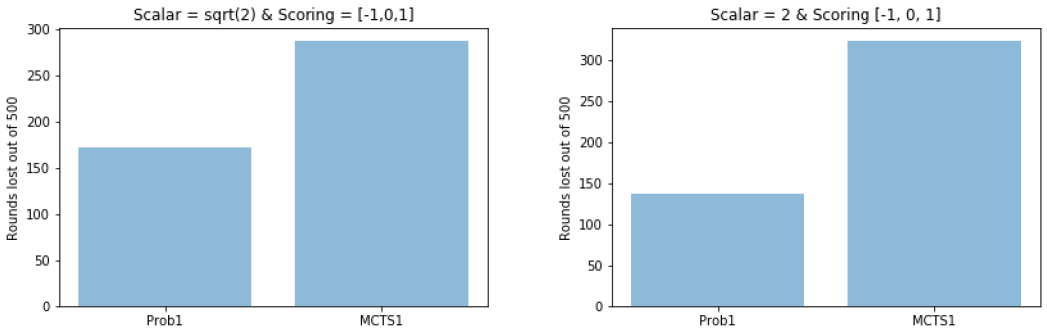

FIGURE 5.5: Experiment 2a - Scoring = [-1,0,1]

As can be seen in Figure 6.6, with either of the scalar options this scoring system wasn't very successful, making the MCTS player lose around 300 dice while the probabilistic lost near 150. Note that the missing dice are dice to add up to 500 were won by the MCTS agent by calling 'Calza'. However, the scoring still points for a clear loss for the MCTS agent.

By observing the game bids, it was realised the agent called 'Calza' often, winning a dice less than half of the times. Meaning the scoring for calza was enhancing the effectiveness of this option. Therefore, changing the scoring to penalising harsher the loss of a die was considered, making 'Calza' a riskier move statistically - hoping to reduce the number of calls of this one.

5.2.1.2 Experiment 2b

Taking the conclusions gathered from Experiment 2a, the score for losing a die was lowered from -1 to -10, with the intention of highlighting the importance of losing a die.

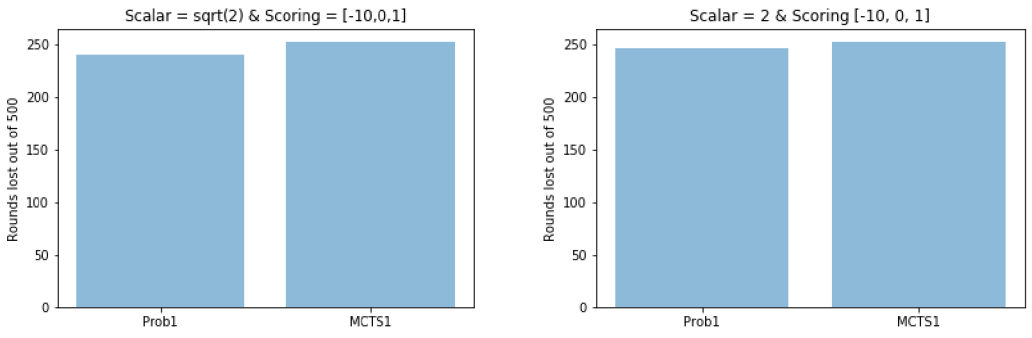

FIGURE 5.6: Experiment 2b - Scoring = [-10, 0, 1]

The results for 2b were deemed more successful than those for 2a, however still almost equal than the probabilistic agent - which was designed with a very simple method to compare with complex models. Regarding the complexity of MCTS and it's well known performance for games in AI, it should be expected to win.

As explained in section 6.2.1, increasing or decreasing the value of a the scoring function above or below the range of -1 and 1 can affect the exploration vs exploitation trade-off. If the score is -10, the exploration side of the formula will be lessened and silenced, as this one increases at a smaller rate and will have a much lower value in comparison - which means it will have a much smaller effect on the UCB score for the tree search process.

5.2.1.3 Experiment 2c

Calza is an unlikely move to win from as the chances are slim, however it should be used for when 'Dudo' will almost likely result in a loss and increasing the bid too. It should be called when no other options seem feasible, and not often with the goal of gaining die constantly.

Experiment 2b used -10 as the penalisation of losing to stop the agent from calling 'Calza' rather lightly. However as explained above, increasing the scoring to one with such high value compared to the visits would defeat the exploration vs exploitation trade-off. Therefore, this new experiment was carried out by making the score of gaining a die 0, letting the agent stop focusing on earning die and avoiding losing die being it's biggest concern.

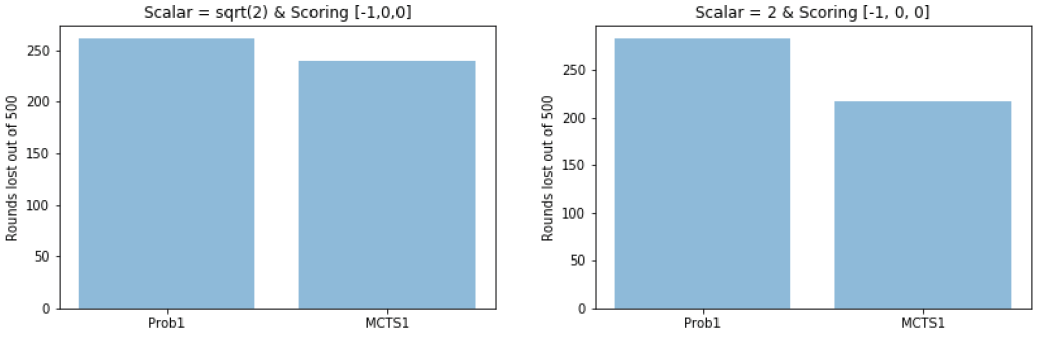

FIGURE 5.7: Experiment 2c - Scoring = [-1, 0, 1]

As can be seen on Figure 6.8, this achieved much better results than 2a and 2b. More so for a scalar of value 2 than for $\sqrt{2}$, so this combination will be used for the rollout analysis which will be the second part of this Experiment.

5.2.2 Rollouts

Using the best combination of scalar and scoring function, the rollouts parameter will be optimized in this section. This parameter and it's relation to efficiency will be associated with the size of the search space. For each rollout, a pseudo-roll is generated for every opponent to simulate a round. For the rollouts to attempt to cover a large enough sample of the search space - we need to consider the number of combinations with repetition that all rolls combined can achieve.

The number of dice in the game is correlated to the time taken for a rollout, as the higher the number of dice, the higher the number of bids needed to reach a terminal node and therefore the deeper the tree search. On another note, the time taken for the MCTS player to make a decision is also to be considered as the experience of playing must be alike to a human's game. Factors like the number of dice in the game and the number of moves from the players given state will affect the time taken to make an agent's decision, however this should be capped to a maximum of 30 seconds to trade-off between experience and performance.

The possible dice combinations will be higher the more dice there is at stake, however random sampling can be highly efficient and will reduce the computational time to execute in comparison to the entire search space.

The number of k-element combinations of n objects, with repetition is:

$$CR\binom{k}{n} = C\binom{k}{n+k-1} = \binom{n+k-1}{k} = \frac{(n+k-1)!}{k!(n-1)!} \tag{5.2}$$

where:

k = Number of dice in all pseudo-rolls
n = Possibilities of a die

5.2.2.1 Random Rollout

Since random rollout policy doesn't take in each players individual roll to take action the distribution of the dice to each player is irrelevant. The main concern is the total count of each dice value for the purpose of scoring and defining a loser or winner of each round.

Below is an example of the calculations for a game of 6 players with 5 dice each. If we exclude the dice of the MCTS Player which are known, the calculation would be the following:

$$CR\binom{25}{6} = C\binom{25}{6+5-1} = \binom{6+25-1}{25} = \frac{(6+25-1)!}{25!(6-1)!} = 142506 \tag{5.3}$$

This value is quite high as there are many dice on the table at this stage of the game.

However the number will decay quickly as the game evolves and dice are eliminated. Below are examples of the combinations with repetition with different number of opponent dice:

$$CR\binom{20}{6} = 53130 \qquad CR\binom{15}{6} = 15504 \qquad CR\binom{10}{6} = 3003 \tag{5.4}$$

If the number of rollout for a game with X opponents dice was set to $\mathrm{CR}\binom{X}{6}$, the entire search space would be explored and consequently the results would be reaching optimality. However, the higher the number of dice, and thus the number of rollouts, the longer the time of computation. For the sake of sustaining an enjoyable experience for the opponents, the waiting time between players turns should not be above 30 seconds.

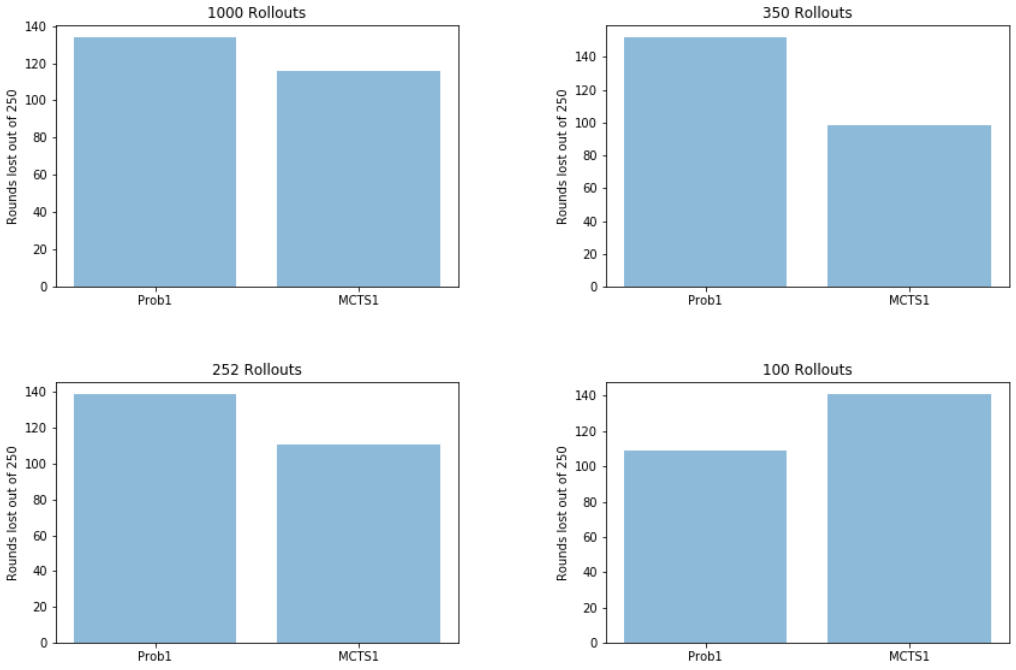

FIGURE 5.8: Random Rollouts Simulation

As can be seen on Figure 6.9 the closer the rollouts were to the number of combinations, the better the MCTS performance. Due to the low number of opponents dice, the rollouts are low and the recorded time of response of the MCTS player was always lower than 30seconds. It would make sense that the optimal number would be the number of combinations if we were feeding the rollouts each possible dice combination - however the rolls are set randomly, so not a rollout per roll is carried out. Therefore, a slighly higher number of rollouts would make up for this imbalance in tested rolls as can be seen on the graph of 350 rollouts in comparison to 252 - which is the combinations with repetition of 5 dice. To achieve high performance, the rollouts should be set to slightly higher than the number of combinations.

Calling the MCTS Player in a game with more players would take far more time as the combinations for more dice is higher and therefore, to reach optimal results, the rollouts should also be increased. However, as mentioned above, there is a threshold in computing time that should not be passed to keep the experience of playing human-like.

Using manual trial and error, 10000 was the maximum number of rollouts with 25 unknown dice that would return a bid in just under a minute. The algorithm for the MCTS player was changed for it to do as many rollouts as combinations (with an increase of 10%) of unknown dice are on the table but capped at 10000, which would results in a lower performance for those rounds with plenty dice but a computational return within the accepted range.

5.2.2.2 Observer Agent Rollout

In contrast, this rollout makes use of each players dice for their each opponents decision making process that leads to their bids. Meaning the expansion of each opponents leaf node will highly depend on their roll. Therefore, the order amongst the die within each players roll doesn't matter, yet the order of each full roll within the game will. As an example, the simulation would be different if all players switched rolls but remained in their seating order.

For each player, the number of combinations with repetition disregarding order would be calculated as shown in equation 6.3 - where dice would represent the number of dice of this player. These numbers are calculated for reference but they will be capped at whatever value maximises efficiency within a time range of 30 seconds.

These rollouts are expected to be more efficient as will only explore more likely scenarios than just random - using a rational player to guide the opponents moves. Therefore the number of rollouts can also be lower than those for the random rollout as this last one requires more exploring.

$$CR\binom{dice}{6} = C\binom{dice}{6+dice-1} = \binom{6+dice-1}{dice} = \frac{(6+dice-1)!}{dice!(6-1)!} \tag{5.5}$$

To calculate all possible combinations around the table - all this combinations are multiplied to each other.

$$TotalCR = CR_1\binom{dice_{Player_1}}{6} \times ...CR_n\binom{dice_{Player_n}}{6} \tag{5.6}$$

For example, in another round with 6 players which hold 5 dice each, the number of combinations would be the following.

$$TotalCR = CR_1\binom{5}{6} \times CR_2\binom{5}{6} \times CR_3\binom{5}{6} \times CR_4\binom{5}{6} \times CR_5\binom{5}{6} \tag{5.7}$$

$$TotalCR = 1016255020032 \tag{5.8}$$

However, for 3 players which hold 5 dice each:

$$TotalCR = CR_1\binom{5}{6} \times CR_2\binom{5}{6} \tag{5.9}$$

$$TotalCR = 63504 \tag{5.10}$$

A similar simulation than with the random rollouts was carried out, by exploring if going above, below or exactly on the number of rollouts has much impact on the performance.

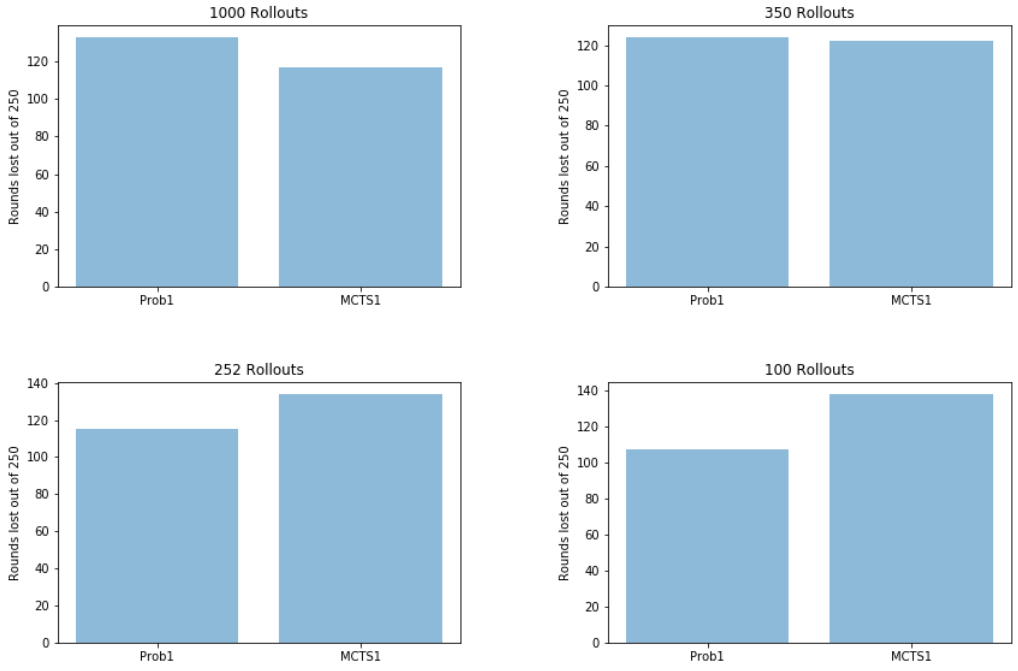

FIGURE 5.9: Observer Rollouts Simulation

There is a clear trend of increased performance as rollouts go up in this experiment. One thing to keep in mind is, this MCTS Player with observer agent rollout shapes their strategy according to what they assume their opponent will respond to each of their bids, however it is using the observer agent for this analysis and the opponent is the probabilistic agent - this means its guesswork of the opponents future bids are inaccurate. However, the results speak for themselves and it seems this rollout isn't so overfitted and can deal with opponents with different strategies. Two things that would be insightful are, making the same experiment with an observer agent as an opponent instead, and increasing the rollouts again to check if the performance keeps rising.

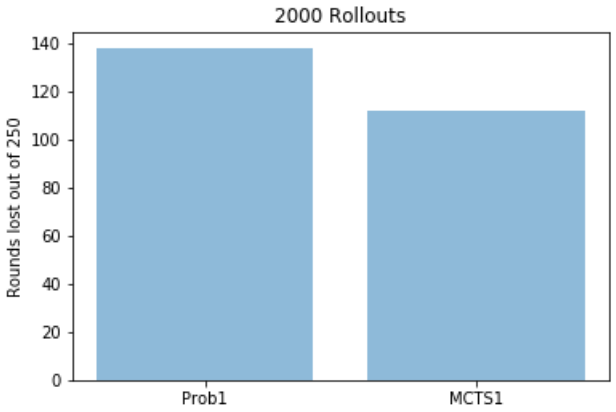

FIGURE 5.10: Simulation with Increased Rollouts

We can observe from Figure 6.11 that performance did not rise noticeably and the computational time doubled due to the doubling in number of rollouts.

Below in Figure 6.12 we can observe a similar display of experiments than in Figure 6.10, although this time with an observer agent as an opponent instead of a probabilistic one. This was expected to have a better result as the MCTS player would accurately be doing the rollouts by using the observer agent as a subagent, however this did not seem to happen. If anything can be concluded from this is that the MCTS with Observer Agent Rollout is not overfitted to play only against an observer agent but will have sometimes even better results with different opponents.

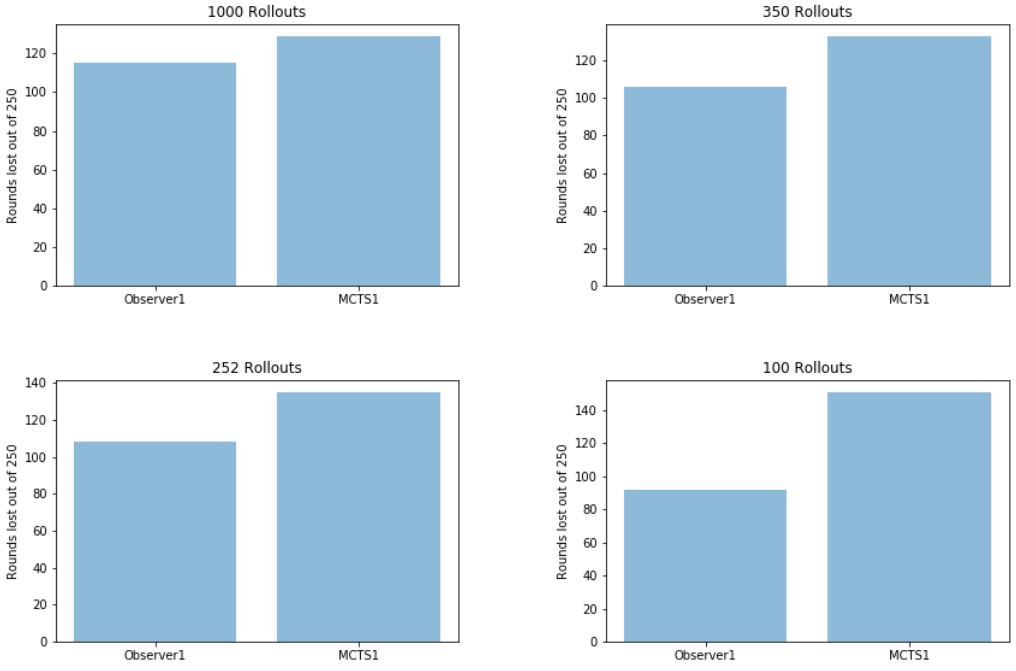

FIGURE 5.11: Observer Rollouts Simulation with Observer Opponent

5.3 Experiment 3 - Comparison of all Agents

In this experiment we will compare all agents in two different scenarios. First, 400 rounds with random seating will be carried out, this will provide some insight on the overall performance of each strategy, regardless of seating and partial observability - as each player in all of these 400 rounds will have 5 dice. Then 200 full games will be conducted, which will highlight the abilities of each player when partial observability is not distributed equally and when seating can affect a players luck.

5.3.1 Rounds

In Figure 6.13 we can observe the number of lost rounds per player. Note that the lower the score the better the performance. The observer rule-based agent seems to have recorded the least number of losses, followed by the MCTS Players with Observer Rollout, the MCTS Player with Random Rollout and lastly, the Probabilistic Agent. The fact that the probabilistic agent has the lowest performance is satisfactory as it was designed in a simple manner solely to compare against other more complex models and it would make sense that these models outperform it.

Comparing both MCTS Players, The MCTS with Random Rollout had the number of rollouts capped at 10000 and the MCTS with Observer Agent Rollout was capped at 3500 - due to having more time consuming rollouts. However, the performance is still better for the second. This could be explained by saying that even using fewer rollouts, these rollouts were more realistic by using a subagent for opponents calls instead of random, making these rollouts much more informative and less futile.

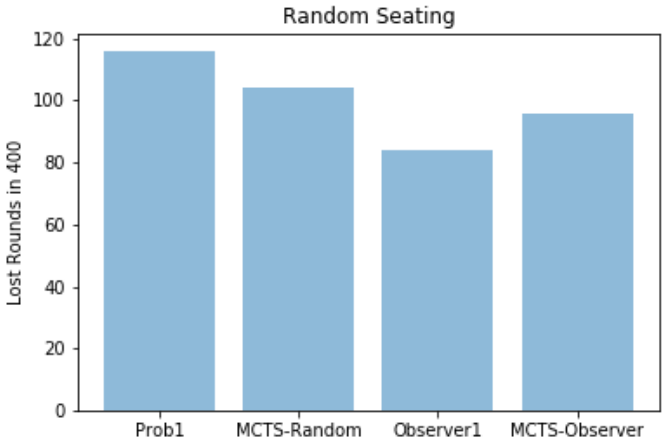

FIGURE 5.12: All vs All - Rounds

5.3.2 Games

In this experiment, 200 Full Games were carried out with 4 players, each player using one of the four designed strategies. During a game, seating was fixed, and after every game, seating was shuffled, and so on.

After carrying out 200 Full Games simulations, these results changed slightly. In Figure 6.14, it can be appreciated that the MCTS Observer Agent has taken the first place, followed by the Observer Agent and the MCTS Random Rollout Agent - with very similar scoring, and lastly, the probabilistic Agent.

This could insight that the Observer Agent's strategy was overfitted to rounds with 5 dice, and it's performance decreased when observability lowered. However, the MCTS Observer Rollout's strategy is affected less by partial observability due to the simulations per combination of dice.

This experiment proves that the chances of winning a round are far higher than those of winning a game. A round can be won by luck, however, winning a game requires a sturdier strategy. Games can also be won with weaker strategies due to a combination of factors like luck or fixed seating around worse players.

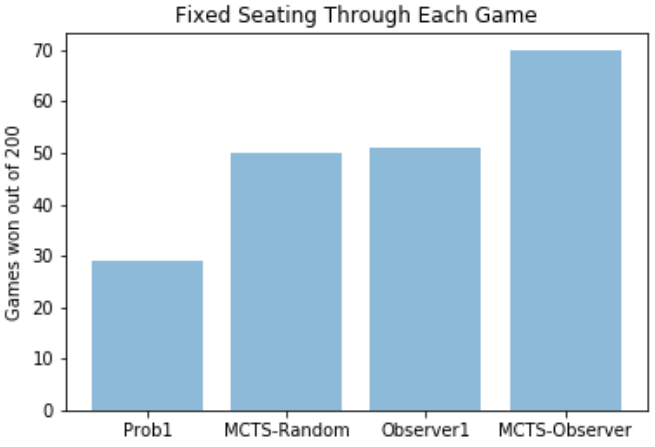

FIGURE 5.13: All vs All - Games

Chapter 6

Conclusions and Recommendations

The main goal of this paper was to apply Artificial Intelligence in Perudo and experiment with different approaches. Perudo not being a perfect information game accounts for much of the complexity encountered when creating an artificial player.

The Observer Agent's performance was surprising due to it's reduced complexity in terms of training, ease of debugging and computational time and power. The flowcharts were designed with a human-like strategy for a real game of Perudo and was continuously updated with the use of trial and error when competing versus the Probabilistic Agent. Overall, considering the complexity of the model, it's performance and it's computational cost, this method is highly successful. However, there might be some inherent bias due to the strategy being based on a single human's strategy - maybe more experimentation should be carried out versus human players to truly evaluate these.

Monte Carlo Tree Search with Random Rollout did not achieve a performance as high as expected, scoring equally than the Observer Agent on the final experiment (Figure 6.14). One factor that could have affected the performance on both MCTS players was the fact that each rollouts pseudo-roll was created randomly, instead of creating all the possible combinations and carrying out a rollout with each one of these. This would've resulted in a more real-like study of the search space - whereas doing random pseudo-rolls will have resulted in multiple trials with certain repeated pseudo-rolls while other combinations weren't trialed at all. Another approach that could have been worth exploring is using previous bids from opponents to create more informed guesses of the opponents rolls instead of doing all possible combinations - this would reduce computing time and would focus the rollouts in a more likely scenario to happen.

Coming back to the relevant problems to tackle in regards to each strategy, most of these agents were quite resistant to opponents bluffing as only the Observer Agent took into some consideration the previous bids in a round, only after considering binomial

probabilities and their own roll. Although overall, if opponents bluff - these players will not be hugely affected. On another note, the MCTS Players don't tend to bluff as their simulations don't acknowledge that bluffing could alter their opponents decision making process. This will be a weakness when confronted to human players as it makes the agent very predictable.

Partial observability and seating were proven to be a key factor during experimentation, given that results experienced substantial variations when carrying out games when compared to rounds. With this being said, the MCTS players can show a higher performance than their opponents when it comes to full games. This could be pinned down to both factors, observability and seating. On the other hand, the observer agent and probabilistic agent tend to perform worse as their dice decrease.

Concerning probability, both rule-based agents used this as a ground base for decision making. However, the MCTS Players did not directly use probability calculations as such, but generated random pseudo-rolls that eventually lead to a similar distributions of dice values than probability would have suggested.

Finally, in terms of evaluation of the strategies comparing all versus all could lead to non-informative results. Using ELO score, which is common rating system used for calculating the relative skill levels of players in zero-sum games. This method is commonly used in chess and could provide different insightful results, as there is a chance that some of these four players have clashing strategies that affect the performance results.

In conclusion, winning can be fueled by luck but the better the strategy the more wins will be recorded over time, and this is why many simulations are needed to properly evaluate a strategy. The MCTS with Observer Rollout being the best player of the four does not mean it couldn't keep improving. One of the biggest limits within this strategy is the time taken for decision making, as if rollouts were increased the performance would also. Therefore an ensemble player could be created to play using the Observer Agent (Rule-based) for phases where risk is low, and call the MCTS Player with Observer Agent Rollout with a higher number of rollouts for bids within the risky phase. This would reduce the overall time of execution of a game, although the time taken for riskier bids will be higher.

Bibliography

[1] Norbert Boros and Gabor Kallos. Bluffing Computer? Computer Strategies to the Perudo Game. https://sciendo.com/article/10.2478/ausi-2014-0018, Last accessed on June 30, 2021.

[2] Pressman Toy Corporation. Perudo Rulebook. https://cdn.1j1ju.com/medias/f4/4f/09-perudo-rulebook.pdf, Last accessed on August 2, 2021.

[3] Laasya R Dr. C. Nandini, Sachin Kumar. AI Poker Agent based on Game Theory. https://www.researchgate.net/publication/340626329_AI_Poker_Agent_based_on_Game_Theory, Last accessed on July 3, 2021.

[4] Alasdair Macleod. Game Design Through Self-Play Experiments. https://dl.acm.org/doi/pdf/10.1145/1178477.1178572?casa_token=QNPbZf4u86kAAAAA:N0XbBc03DfQ6MjlRpcd39oxcyzLRHh-GkASyKjp4Vy6alGIjJvABkAr0dYDsoYE8xgPEwkSDRU8, Last accessed on June 29, 2021.

[5] Steve Roberts. The Upper Confidence Bound (UCB) Bandit Algorithm. https://towardsdatascience.com/the-upper-confidence-bound-ucb-bandit-algorithm-c05c2bf4c13f, Last accessed on July 31, 2021.

[6] Jonathan Rubin Ian Watson. Computer Poker: A review. https://www.sciencedirect.com/science/article/pii/S0004370211000191, Last accessed on June 31, 2021.