Rummy 500 With Symbolic AI Opponent

Matt Grzenda

Abstract

Rummy 500 is a classic card game played by at least 2, but up to 8 people. The goal of the Rummy 500 opponent was to create a machine opponent which used the same mechanisms as a human to play. This thesis serves to document the process of implementing a heuristic, rule-based opponent for this game.

Contents

1. Introduction

Rummy 500 is a classic card game that can be played with 2 - 8 people. The main goal is to score 500 points. It is a game where many heuristics can be applied to increase a player's chance at scoring points. The ability to play Rummy 500 using a set of predefined heuristics, or rules, is the trait which makes it ideal to model an opponent for the game with a symbolic, rule based AI architecture since all rule based systems contain rules (Davis & King, 21).

The process of implementing Rummy 500 comes down to multiple stages:

1. Modeling the physical components of the game, such as the deck, discard pile, players' hands, and players' melds.

2. Implementing operations to deal cards, deal a card onto the discard pile, and start the game.

3. Implementing algorithms to pick a card, check the player's hand for a meld, and discard a card from the player's hand.

4. Creating a function for a player to perform a "turn" or the sequence of picking a card, looking for melds and playing any if found, and discarding a card.

5. Defining a function for a round of Rummy 500 to be played. Where a round ends when either player runs out of cards or when the deck is empty.

6. Creating a basic game, where cards are drawn only from the deck, melds are sets of 3 or four cards of the same face value, and discarding is done randomly.

7. Adding heuristics to the simple game.

8. Adding the ability for the opponent to explain its actions.

Once models for the physical components of the game are created, multiple versions of the game can be made - each with a different set of heuristics used, and each with its own performance. In order to better understand how the game is making decisions, the AI architecture is explainable, meaning that an explanation is given for every decision the program makes (Samek & Müller, 2019).

Explainable AI has been growing in demand since the growth of convolutional neural networks (ConvNets) began. This is due to the nature of ConvNets not being able to explain what

logic was used to come to a conclusion (Samek & Müller, 2019). Since this implementation of Rummy 500 does not use a ConvNet, creating a way for every action to be explained was fairly straightforward. The explanation part of this project was to allow the user to see how the opponent is behaving when it draws a card, plays melds, or discards.

This thesis serves to document the process implementing the Rummy 500 opponent. Section 2 gives a background on the construction of card games, noteworthy card playing programs, the heuristic architecture, and an overview of Rummy 500. Section 3 summarizes the approach taken to model the opponent. Section 4 describes the knowledge representations necessary to create the game. Section 5 discusses the game playing framework. Section 6 conveys the results of multiple opponents playing one another. Section 7 gives a summary of the explainable nature of the program. Finally, Section 8 describes the extensions and elaborations that could be made if more time was available.

2. Background

The Construction of a Card Game

All card games are created by humans and all card games are played by humans. In these card games, humans develop heuristics to play the game in a way which maximizes their payment. The payment is defined as something of value in a game that an individual would want to possess. In card games, payments may include points, cards, the ability to lose cards, and, in the case of Poker, money (Von Neuman & Morganstern, 1947).

Considering how humans play games, why not construct a card game centered around the use of pre-programmed heuristics to play the game? Unless the implementation of the game happens to be able to learn new heuristics, it will only perform as well as the programmer who created it. However, depending on the heuristics that were encoded, the machine could still be a formidable opponent. The reasoning behind using a heuristic architecture in this implementation of Rummy 500 is for a simple, natural design of a machine opponent, while still being able to create a worthy opponent.

Classic Card Playing Programs

Game playing programs are created for a variety of reasons. Whether it is to see if it is possible for a machine to beat a human (Mitchell, 2020), or because elements of the game can be abstracted to other parts of society (Billings et al., 1998), people have been fascinated with creating AI games for decades. This of course extends to card games, to the point where many card playing applications are now available to buy on any app store.

Ginsberg's Intelligent Bridge Player (GIB) was a bridge playing program created in the late 1990s. Unlike the other bridge playing programs before it, such as Bridge Baron or Paradise, GIB does not use any human methodology to play bridge (Ginsberg, 1999). Instead GIB uses a brute force search to determine the best move in a given situation. To select cards, for example, GIB uses the Monte Carlo Card Selection Algorithm. This algorithm constructs a deal of the bridge game thus far, based on what is known and which cards are in play. Any unknown cards are randomly dealt out if necessary. Every possible move is calculated and then compared to find the best move on this particular deal. This move is then returned (Ginsberg, 1999). After the performance of the machine was published in a bridge magazine, GIB was invited to compete in the world bridge championships in France, where it finished in 12th place out of a possible 34th against human opponents (Ginsberg, 1999).

The significance of GIB lies in its achievements in performance and its radically different architecture. Each bridge playing program before GIB used a heuristic architecture to play bridge while GIB used a mini-max style tree generation algorithm to calculate how to respond to a certain scenario. Unless there is a significant jump in research for heuristic based bridge programs, GIB has set a new standard of card game architectures.

Another influential card game program was a Texas Hold'em opponent called Lokibot. It was created as a way to research how AI should respond to imperfect knowledge, multiple competing agents, risk management, deception, and unreliable information (Billings et al., 1998). Other programs before it had created simpler versions of poker to simply have a game that was playable, but the developers of Lokibot wanted the real-world parts of the game to be included (Billings et al., 1998). In order to account for this, Lokibot evaluates its hand every time a card is turned over during gameplay. To evaluate its hand, an enumeration technique is used which calculates how good its hand is when compared to every other possible hand. A percentile is calculated for the hand and based on the percentile, the program decides whether to check, bet or fold. The enumeration is also weighted to account for different opponents' playing strategies and also because not all hands are equally likely (Billings et al., 1998). To test out the overall performance of Lokibot, it played itself. The self play was tournament style and each "opponent" was a different variation of itself with varying skill level. Overall, the variation with the most strategies encoded to decide on moves won the most (Billings et al., 1998).

Lokibot's significance is its approach to hard problems to solve in AI. By taking real world behaviours that are hard to define in a general context and applying them to a well defined space such as Texas Hold'em, these problems can be examined more closely. Under this new light, dealing with imperfect knowledge, unreliable information, multiple competing agents and more is simply a probability computation. Each one of these behaviours also exists outside of Poker in the worlds of business and even warfare. What was learned by creating this Texas Hold'em game could be the basis for the stock trading, weather and political forecasting, and business transaction applications of tomorrow (Billings et al., 1998).

The Heuristic Architecture

According to Allen Newell and Herbert A. Simon (1957), "A process that may solve a given problem, but offers no guarantees of doing so, is called a heuristic for that problem." Based

on this definition, humans tend to be heuristic problem solvers. When a human encounters a complex problem, rarely is their first thought the most optimal solution to the problem. More often than not, their first solution is lacking in some way. However, this makes heuristics far from being useless. Humans tend to think of simple solutions, compared to the exhaustive alternatives. For example, search algorithms for data structures tend to be heuristic based since the exhaustive options are more computationally expensive (Kokash, 2005).

When a program is said to be based on rules or conditions, this program can be said to be based on heuristics since the programmer created a process which may solve a given problem. The Cyc project, for example, was, "an attempt to model the human consensus knowledge." (Yuret, 1996). Cyc used a very large knowledge base to try and accomplish this goal, with a large team hired to enter knowledge into the database. Cyc proposed to solve the problems of brittleness in programs by offering the common sense that humans had. Although Cyc did not accomplish its main goal to be able to derive a deeper meaning from a provided symbol, it opened up a treasure trove of research possibilities (Yuret, 1996). Flaws in the program created a demand for new approaches to issues, such as how to model symbols in other ways than deductive inference (Yuret, 1996).

Another noteworthy program which is based on a heuristic architecture is called Bagger. This program is designed to find the optimal way to put groceries in bags (Roma et al., 1993). Bagger does this by grouping all of the items by weight and then packing the items into bags one at a time starting with the heaviest items and working down to the lightest weight items (Roma et al., 1993). The significance of using heuristics here is the amount of computation time saved compared to other methods. An exhaustive method would try every possible combination of grocery items in bags, and keep track of the results. Once all the possible combinations of groceries are found, the best option is returned. The exhaustive method is much more expensive to compute than the heuristic approach due to the excess number of steps needed to compute it.

Heuristic architectures are not the perfect solution to any problem, but they are also not the worst solution. The goal of a heuristic architecture is to write a program which solves a problem the way humans do. Heuristic approaches to problems often lead to the most optimal solution as heuristics are improved and swapped for better heuristics. If any process that may solve a given problem is a heuristic, then all processes thought of by humans are heuristics.

Rummy 500

The rules for this implementation of Rummy 500 are based on those described in the Wikipedia 500 Rum article (2005). Rummy 500 is a classic card game played by 2 – 8 people. In a game with just two people, each person is dealt 13 cards to start. On each person's turn, they start by picking a card from either the deck or the discard pile. When drawing from the discard pile, the player is allowed to draw multiple cards. In order to legally draw from the discard pile, a person must be able to use at least one of the cards drawn. At this point, the person will look for a meld, or combination of cards, to play on the board to earn points. A legal meld in Rummy consists of at least three cards. The first type of meld is having three cards of the same face value (i.e. three kings). The second type of meld is a run of cards in increasing order of the same suit (i.e. 2, 3, 4 of hearts). Once a player has either produced a meld, played on another opponent's

meld, or realizes he/she can not make a meld, they put a card in the discard pile and their turn is over.

Rummy 500 is split into multiple rounds. A round starts once cards are dealt and ends when a player runs out of cards. At the end of each round, points are calculated based on the melds each person played. Point values for each card in a meld are as follows: cards 2 – 10 are worth 5 points; jacks, queens, and kings are worth 10 points; and Aces are worth either 5 points if played low, or 15 points if played high. A player's score is the points from their melds minus the sum of the cards left in their hand. Once a player reaches 500 points, the game is over and that player has one the game.

3. Approach

When asked the question, "What ways are there to implement a machine opponent for a game?" There are many answers. To determine which one to use, generally another question has to be asked: "how do you want the opponent to play?"

If the goal was to make an opponent which would always make an ideal move, one might base the architecture around generating every possible move for a given game state. All moves that can be made during gameplay form a tree, where gamestates are nodes and possible moves are links to other gamestates (Aziem et al., 2014). Searching this tree for the most optimal move can be done by any search algorithm, but is commonly done by the minimax algorithm with alpha-beta pruning. To begin, imagine two players, one called Max and the other called Min. Min is trying to score the least number of points possible, while Max is trying to score the most points possible. The minimax algorithm works by using a depth first approach to visit all nodes up to a certain depth of the game tree. At each node, the gamestate is evaluated. The moves linking gamestates together are then given scores based on how well the move benefits Max (Marsland, 1986). Moves that benefit Min will be scored low and moves that benefit Max will be scored high. Depending on the size of the game tree, this algorithm can get expensive. To accommodate, alpha-beta pruning is used to eliminate certain nodes and/or branches which are less beneficial to a player than a previously found move (Marsland, 1986). A variant of this algorithm was used in the Deep Blue Chess Playing Machine, created by IBM (Campbell et al., 2002). This machine shocked the world when it beat the world champion Gary Kasparov in 1997 (Campbell et al., 2002).

Although an opponent which generates a game tree will theoretically make the best move possible in any situation, it has its limitations. First, it only works for games where most knowledge is known. In chess for example, the only knowledge not known at any point of the game is what move a player will make (Billings et al., 1998). Which pieces are on the board, and their respective locations, are known at all times. In other games such as poker, many factors are unknown during gameplay. These include, which players have which cards, what a player will bet during a round, and how strong your hand is compared to everyone else (Billings et al., 1998). These unknown factors make generating a game tree difficult. Humans do not create a game tree when they play games. Instead, we have another approach.

Humans are heuristic problem solvers, meaning that for any given problem we use a set of heuristics to try and solve it (Newell & Simon, 1957). Games inherently present their players

with problems that need to be solved. Which move do you make to benefit the most? What are my opponents going to do next? For each of these problems, we create heuristics to determine some kind of answer for ourselves (Newell & Simon, 1957). The Rummy 500 Opponent follows a set of encoded heuristics to play Rummy 500 like a human would.

The heuristics used by the Rummy 500 Opponent are greedy heuristics. Following the definition of a greedy algorithm, a greedy heuristic is one which finds the best immediate or local solution to a problem (Black, 2005). This type of heuristic is mainly used in searching for melds to play. When a set or run is found, it is played as a meld no matter the point value, or if any other melds exist in the opponent's hand. Another type of heuristic is a random selection, which is where something is selected at random (Stone, 2009). The Rummy 500 Opponent uses a random heuristic to randomly discard at the end of each turn. Although there are other methods that could be used to discard, a random selection allows, in theory, for an equal chance of any card being discarded (Stone, 2009).

A greedy heuristic is not the only type of heuristic which could be used to search for melds. Another heuristic which could be used is the satisficing heuristic. This heuristic will continually search a given space until some aspiration value is met (Radner, 1975). A variant of this which allows for the possibility of an opponent's hand not containing any melds could be implemented in the Rummy 500 Opponent. The aspiration value would be a minimum score for a meld, but would be decreased after a search failed until it reaches 0.

Once an approach for heuristics is established, a way of organizing these heuristics is now needed. The Rummy 500 Opponent uses a forward-chaining rule base to accomplish this. Forward-chaining means that a rule is triggered when a change produces a situation which matches the conditional of a rule (Hayes-Roth, 1985). The "change" for the Rummy 500 Opponent is after either opponent playing takes a turn. On the next turn, the opponent will examine the cards in its hand and examine any melds that were played to see how the game state changed. An opponent will then take some action depending on the new state of the game. In other words, the rule base is a collection of *if* $\rightarrow$ *then* style rules organized by the part of the Rummy 500 they are relevant to. Rules for drawing a card, searching for and playing melds, and rules for discarding are organized respectively in their own spaces.

The main approach to implementing the Rummy 500 Opponent was to create a preliminary version of the opponent, called the base opponent, and then evolve the base opponent by adding more heuristics. The base opponent could only draw from the deck, play melds that were sets of three or four cards, and discarded randomly. Before heuristics could be added to the base opponent, the ability for the base opponent to keep score, take a turn, and play an entire round against itself was added. The first addition to the base opponent was the ability to play runs as melds. Next was the ability to continue on pre-laid sets, and then continue pre-laid runs to earn more points. Each opponent created is called a *variant* of the Rummy 500 Opponent. Variants are distinguished from each other based on which heuristics they can use during game play. The most advanced opponent variant to date plays Rummy 500 in the following manner. First the Rummy 500 Opponent draws a card from the deck. Next, the opponent looks for any sets in its hand and plays the first one it finds as a meld. Next, it looks for any runs in its hand to play as meld. Then, it looks to see if it can play a card on any premade sets and plays the card if applicable. The opponent then checks to see if it can play a card on any premade runs and plays the card if possible. Finally the opponent discards randomly onto the discard pile. Games are played between two opponent variants, where variants may be the same opponent.

4. Knowledge Representations

Representing any game is to represent the components which make up a game. This includes the physical components and the rules used to play the game. Since the rules depend on the physical components, the physical components of the game should be modeled first (Aartun, 2016). In the Rummy 500 Opponent, the physical components are the cards, the deck, the discard pile, and each player's hand. Since the Rummy 500 Opponent is written in Prolog, the representations used for each physical component are the data structures available in Prolog. **Figure 1** shows an example of each structure. A card is a Prolog fact consisting of the card's face value, suit and rank. Cards have two representations. The first representation is the internal representation as a Prolog fact. The Prolog fact representation makes for relatively easy comparisons of cards based on the suit, rank, and face value. However, this representation is not very human readable, so another human-readable representation is used for output. The deck, discard pile, and player's hands are sets of these cards. The melds played by a player are represented as a set of sets, where each set is a meld.

```
Card: card(ace, spades, 1)
Human-Readable Card: AS

Deck: [3H,8C,JH,8H,9D,QH,QC,AC,9H,JD,7C,KD,10S,6H,6S,KH,5D,5C,2H,JC,AS,10C,KS,10H]
Discard Pile: [3D]

Player Hand: [9S,4C,6C,6D,8S,QD,4S,2C,AH,KC,4H,2D,8D]
Player Melds: [[4C,4S,4H]]
```

Figure 1: Physical Components of Rummy 500

Once the physical components of the game are modeled, the rules must then be constructed. Every game has rules to define how it is played. However, implementing these rules programmatically may not be as straightforward as defining rules in the physical world. A rule generally consists of two sides: a condition which can be met, and an action to be taken if the condition is found to be true (Hayes-Roth, 1985). The Rummy 500 Opponent described in this paper uses this type of rule and matches a condition to perform a given action.

Encoding a rule comes down to encoding the condition and encoding the set of actions associated with the condition. The general structure is shown below in **Figure 2**, where n is the number of actions for a given condition. Each action may also consist of another condition to create an embedded rule (Hayes-Roth, 1985). Each rule is evaluated recursively for this reason.

```
If Condition1
      Action1
      Action2
      ...
      Action n
```

Figure 2: General Condition-Action pair

In the Rummy 500 opponent, this is the structure used to construct rules for every action taken by the opponent during the course of a game.

When it comes to designing a rule base, there are four basic parts of the architecture: rules, interpreters, translations and explanations (Hayes-Roth, 1985). The rules, as mentioned before, are a condition-action pair. The interpreter is the mechanism which matches patterns or symbols to determine if the condition of the rule has been met. If the condition is true, the interpreter responds by performing the set of actions associated with the rule. The interpreter follows the pattern of a *recognize-act* loop, where a rule is matched, the actions are performed, and then the rule selection continues as defined (Davis & King, 1984). Shown below in **Figure 3** is the general recognize-act loop that is used in each rule evaluation, where N, M, X, and Y are arbitrary values. The translations are used to rewrite rules for other purposes where the logic may be the same. A translation is used in the Rummy 500 Opponent to deal one card at the beginning of the game and for the opponent to draw a card during game play. A translation is used here because both rules need to take a card off the top of the deck, just in different contexts. Finally explanations are used to explain how the rule base came to a conclusion (Hayes-Roth, 1985). In the Rummy 500 Opponent, an explanation is given for every action taken during the course of a turn.

```
while(Condition)
       if(Rule1)
              Action1
              Action2
              ...
              ActionN

       if(Rule2)
              Action1
              Action2
              ...
              ActionM

       ...

       if(RuleX)
              Action1
              Action2
              ...
              ActionY
```

Figure 3: General recognize-act loop

The Rummy 500 Opponent examines rules based on the natural order of a Rummy 500 turn. Rules to draw a card would be examined first, rules for playing melds are examined second, and finally rules for discarding are examined at the end of the opponent's turn. Each section of the turn follows its own recognize-act loop to match a given rule. For each part of the opponent's turn, the rule base is examined in the order that the rules were put into the rule base. As an example let's say we are examining the Rummy 500 Opponent variant which can play sets and runs as melds. The opponent begins by searching its hand for sets first, plays a set if found, and then searches for runs and plays a run if found. The rules for finding sets are triggered first since they were encoded first. Rules for finding runs are triggered second since they were encoded second, and so on.

5. Game Playing Framework

The framework used to play a game can vary from program to program depending on the type of game and who developed it. Programs like Deep Blue that are designed to play professional chess players have a game playing framework designed around this. The framework used in Deep Blue revolves around making the perfect move in chess. Using a parallel search algorithm, it evaluates up to 330 million possible game states per second, and selects a move based on how beneficial it is to Deep Blue (Campbell et al., 2002). Other systems, such as LokiBot, have game playing frameworks designed around testing and gathering results. LokiBot's framework allows for its skill level to be adjusted based on how many strategies it is allowed to use. This allows for multiple different versions of LokiBot to play each other in a controlled environment to gather accurate results of performance (Billings et al., 1998).

The Rummy 500 Opponent's game playing framework is similar to LokiBot's. Like LokiBot's game playing framework, the Rummy 500 Opponent is designed to play different versions of itself, and where each version is differentiated by skill level. Where these two programs differ is how game play is carried out. In the Rummy 500 Opponent, all game play revolves around the concept of a turn, where in LokiBot full games of Texas hold 'em are played (Billings et al., 1998). A turn in Rummy 500 is defined as a player drawing a card, playing melds if possible and then discarding a card. Rummy 500 Opponent variants differ based on what each one is allowed to do on a given turn. Some variants only have the ability to play sets of three to four cards as melds, while other variants can play sets and runs of any length as melds. Other variants can play on pre-laid melds along with playing melds from their hand. All variants, from the most basic to the most complex, draw from the deck and discard randomly. Since these two parts of a turn are consistent across all opponent variants, constructing an opponent's turn follows a general process. First, call the rule which deals a card into the opponent's hand, then call the rules to find and play different types of melds specific to this opponent. Finally call the rule to randomly discard.

Once a turn for a Rummy 500 Opponent variant has been constructed, extensions can be made. When two opponent variants have the ability to take a turn, the most simple game play that can happen between them is for each opponent to only take one turn and then stop. From here,

turns can be repeated for a round of Rummy 500 can be played. During the round, two variants take turns against each other until one opponent runs out of cards or the deck is empty. A score is then calculated at the end of the round. The final extension is to repeat rounds and add onto opponent scores until one opponent reaches 500 points. This would be an entire game of Rummy 500 between two opponents.

6. Testing and Results

Testing the performance of the Rummy 500 Opponent was done by having multiple variants of itself play each other over ten rounds of Rummy 500. For the purposes of this experiment a "game" will be defined as 10 rounds of Rummy 500. Opponents were matched based on the goal of having every opponent play every other opponent. Whichever variant won the most rounds won the game and is assumed to be the better variant overall. **Figure 4** shows the results of the games. Similar to LokiBot's experiment, the opponent that had the most encoded heuristics performed the best (Billings et al., 1998).

Game Type	Round 1	Round 2	Round 3	Round 4	Round 5	Round 6	Round 7	Round 8	Round 9	Round 10	Winner
S vs R	S	R	R	R	R	S	S	S	S	R	TIE
S vs R+	R+	S	S	S	S	R+	R+	R+	R+	R+	R+
S vs S+	S+	S+	S+	S+	S+	S	S+	S	S+	S+	S+
S & R vs S	S & R	S & R	S	S & R	S & R	S & R	S	S & R	S & R	S & R	S & R
S & R vs R	S & R	S & R	S & R	TIE	S & R	S & R	R	S & R	S & R	S & R	S & R
S & R vs S+	S+	S+	S+	S & R	S+	S+	S & R	S & R	S & R	S+	S+
S & R vs R+	R+	R+	R+	S & R	S & R	R+	R+	S & R	S & R	R+	R+
S+ & R+ vs S	S+ & R+	S+ & R+	S+ & R+	TIE	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S	S+ & R+
S+ & R+ vs R	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+
S+ & R+ vs S+	S+	S+ & R+	S+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+	S+ & R+	S+ & R+	S+ & R+
S+ & R+ vs R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	S+ & R+	TIE	S+ & R+	R+	S+ & R+
S+ & R+ vs S & R	S & R	S+ & R+	S & R	S & R	TIE	S+ & R+	S+ & R+	S+ & R+	S & R	S+ & R+	S+ & R+

KEY:
R = Runs
S = Sets
"+" = playing on pre-made melds of the type it is applied to
(ex: R+ means that in addition to playing Runs, the opponent will also play melds on pre-laid Runs)

Figure 4: Rummy 500 Results

Based on the table, a positive correlation exists between the number of heuristics an opponent can use and their success against other opponents. The most basic opponents, those that only use sets or only use runs, did not win a game against any other opponent. A mid grade opponent, one that used both sets and runs only, won 50% of the games it was entered in. Finally the highest grade opponent, one which plays sets and runs along with playing on pre-laid melds, won all games against all opponents it was matched with.

7. English Explanations of Decisions

For every action taken by the Rummy 500 Opponent, a short piece of text is printed to the terminal describing the action. For example, whenever an opponent draws from the deck, a statement saying where the card was drawn from and the card drawn are printed to the terminal. An example output of a Rummy 500 round is shown in **Figure 5** below. Notice that on top of every action taken by the Rummy 500 Opponent, every card that the opponent interacts with is also shown. The reason for these English explanations is to show that the Rummy 500 Opponent is playing Rummy 500 honestly.

```
Deck: [8S,7D,3C,JH,JD,7H,4C,KC,10H,6S,4H,8C,QH,10S,JS,KH,AD,KD,2H,9D,QS,6H,2D,9S]
Discard Pile: [AC]

Player 1 turn!
HAND BEFORE TURN: [8D,10D,4D,AH,8H,JC,4S,5D,3D,2C,9C,10C,QC]
MELDS BEFORE TURN: []

DRAWING FROM DECK: [8S]
LOOKING FOR SETS...
SET FOUND: [8S,8D,8H]
LOOKING FOR RUNS...
RUN FOUND: [3D,4D,5D]
DISCARDING: [QC]

HAND AFTER TURN: [AH,2C,4S,9C,10D,10C,JC]
MELDS AFTER TURN: [[3D,4D,5D],[8S,8D,8H]]


Player 2 turn!
HAND BEFORE TURN: [KS,7C,AS,5S,9H,7S,QD,5H,6D,3H,6C,5C,2S]
MELDS BEFORE TURN: []

DRAWING FROM DECK: [7D]
LOOKING FOR SETS...
SET FOUND: [7D,7C,7S]
LOOKING FOR RUNS...
NO RUNS FOUND.
DISCARDING: [9H]

HAND AFTER TURN: [AS,2S,3H,5C,5H,5S,6D,6C,QD,KS]
MELDS AFTER TURN: [[7D,7C,7S]]
```

Figure 5: Explanations of Actions

Explainability in AI has been in increasing demand ever since the rise of deep learning and neural networks began in the early 2010s. The reason behind the demand is because the

architecture of a neural network is a “black box” in a sense (Samek & Müller, 2019). The basic unit of neural network is called a perceptron. A perceptron takes a series of numerical inputs, applies some function to them and gives a single numerical output (Mitchell, 2020). These perceptrons are then connected into a network so that they feed into each other and eventually give a final output (O’Shea & Nash, 2015). This type of AI is leading the charge in tasks like image recognition, facial recognition and self driving cars (Mitchell, 2020). However, due to their complexity, neural networks can not describe the logic used to come to a conclusion. The lack of explainability can lead to many ethical dilemmas, especially when government agencies use this type of AI for their tasks.

Imagine this: you are arrested for shoplifting days after a shoplifting incident happened. The only problem is that you have never been to this store. In the interrogation room you ask the police why they think it was you. The police claim that their newly installed facial recognition system matched a face shown in the security footage to your face. Eventually your lawyer arrives, gives the police evidence that you could not have committed the crime, all charges are dropped, and you are on your way. This entire event happened because of a mistake made by an AI system which has no way of explaining how it came to the decision that your face was in the surveillance video.

Imagine now that you had no alibi. Instead of the charges dropped, this case is taken to court and based on your resemblance to the person in the security tape and it being possible you could have committed the crime, you are found guilty and sent to prison. All over a mistake a computer made trying to recognize someone’s face. The scenario described is becoming more real everyday. Rekognition, a facial recognition system developed by Amazon, is currently being marketed to police (Mitchell, 2020). A similar situation happened to Nijeer Parks. He was falsely arrested for aggravated assault, using a fake ID, unlawful possession of weapons, possession of marijuana, leaving the scene of a crime, and resisting arrest. He was brought up on all these charges after a facial recognition system falsely identified him on the photo of a fake ID from the crime scene. After a year-long battle he was eventually proven innocent (General & Sarlin, 2021).

The neural networks used today must be trained manually by humans, which allows for bias to be present. In facial recognition, there is bias against people with darker skin tones (General & Sarlin, 2021). The ACLU demonstrated this in front of congress by testing Amazon’s Rekognition system on all 535 members of congress, looking for matches in a national database of people who have been arrested. 28 members were incorrectly matched, with 21% of the errors occurring on photos of African Americans (Mitchell, 2020). This bias can be present with any application which uses a neural network. In the medical field there is interest in creating machines to diagnose patients. Bias could lead to the wrong diagnosis based on the patient’s information (Samek & Müller, 2019).

The Rummy 500 Opponent does not raise the same ethical concerns as other applications of AI, but it is still necessary for it to explain all of its actions. Otherwise, how would you know if the opponent was actually playing by the rules of Rummy 500? Or if it was cheating at the game? The English explanations of the system were put in place to address these issues.

8. Possible Extensions and Elaborations

The Rummy 500 Opponent variant with the most heuristics currently has the ability to draw from only the deck, play sets and runs as melds, play on pre-laid melds of both types, and discard randomly. From here, the opponent can be improved upon.

In Rummy 500 it is legal to draw multiple cards from the discard pile if you can play at least one of them in a meld on your turn. One of the first additions to make is a heuristic that would check the discard pile at the start of the opponent's turn to see if any cards could be used in a meld. This addition to the program would yield more points, as cards that would normally be wasted in the discard pile could be used in melds to benefit the opponent.

Another addition which could be made would follow the previous addition to drawing from the discard pile. If a player can use cards in the discard pile to make melds, without picking up very many cards, a player would most likely draw from the discard pile on their turn. To combat this, probabilities could be calculated based on how likely an opponent could use a card which is to be discarded. This addition to the Rummy 500 Opponent would not help the opponent score more points, but inhibit the other opponent's ability to score points.

Finally, an opponent which could learn on its own will eventually behave uniquely. The Rummy 500 Opponent was designed to use heuristics to play like a human. The key human aspect that is missing from the opponent is the ability to be unique in how it plays. Humans develop heuristics by discovering something about the problem they face and then using this insight to help solve the problem (Romanyc & Pelletier, 1985). By giving the Rummy 500 Opponent this ability, it will behave more like a human. Reinforcement learning would be used to train the opponent, as humans learn by trial and error (Bianchi et al., 2007). First an evaluation function to determine how beneficial or damaging a current game state is for the opponent must be developed. Using the Q-learning algorithm, the current game state would first be evaluated by the evaluation function. Next, different actions would be simulated to move into another game state, which would then be evaluated. The penalties and gains of each state transition would be stored in some manner and from these values, beneficial state transitions would be learned (Santos et al., 2012). From here, the Rummy 500 Opponent would play many games to learn which decisions would be beneficial to itself.

9. Conclusion

The Symbolic AI Rummy 500 Opponent uses a rule based heuristic architecture to play Rummy 500 in a similar manner to a human (Simon & Newell, 1957). The approach to this project was done in stages, with the first stage being the creation of a simple Rummy 500 Opponent with the most basic heuristics used to play the game. The final and most sophisticated heuristic has the ability to play any meld. Knowledge was represented by Prolog facts and lists, as well as *condition-action* rules to encode the heuristics.

Once multiple heuristics were encoded to play Rummy 500, games to test performance of different opponents were conducted. These tests showed the strength of each different opponent. Unsurprisingly, the Rummy 500 Opponent variant with the most encoded heuristics performed the best. As more heuristics are encoded to create new opponent variants, this may not always be the case.

Explanations are given for every action taken by the Rummy 500 Opponent in an effort to make the AI more transparent and seem like less of a “black box” (Samek & Müller , 2019). This way any user can verify that Rummy 500 Opponent is being played by the rules.

The Rummy 500 Symbolic AI Opponent is still a work in progress. More heuristics can be added to make the opponent perform better during gameplay. More testing can be done as more opponent variants are developed. And eventually, this opponent which plays Rummy 500 like a human may finally be tested by playing a human.

References

1. Hofstadter, D. R., & Fluid Analogies Research Group. (1995). *Fluid concepts and creative analogies: Computer models of the fundamental mechanisms of thought.* Basic Books.

2. Davis, R., & King, J. J. (1984). The origin of rule-based systems in AI. *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project*.

3. Samek W., Müller KR. (2019). Towards Explainable Artificial Intelligence. In: Samek W., Montavon G., Vedaldi A., Hansen L., Müller KR. (eds) Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. Lecture Notes in Computer Science, vol 11700. Springer, Cham. https://doi.org/10.1007/978-3-030-28954-6_1

4. Von Neumann, J., & Morgenstern, O. (1947). *Theory of games and economic behavior* (2nd rev. ed.). Princeton University Press.

5. Mitchell, Melanie. (2020). *Artificial Intelligence: a Guide for Thinking Humans.* Pelican.

6. Billings, Darse & Papp, Denis & Schaeffer, Jonathan & Szafron, Duane. (1998). Poker as a testbed for machine intelligence research. Artificial Intelligence - AI.

7. Ginsberg, Matthew L. (1999). GIB: Steps toward an Expert-Level Bridge-Playing Program. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (*IJCAI '99*). 584 - 593.

8. A. Newell, J. C. Shaw, & H. A. Simon. (1957). Empirical explorations of the logic theory machine: a case study in heuristic. *Papers presented at the February 26-28, 1957, western joint computer conference: Techniques for reliability* (*IRE-AIEE-ACM '57 (Western)*).

Association for Computing Machinery, New York, NY, USA, 218–230. https://doi.org/10.1145/1455567.1455605

9. Kokash, Natallia. (2005). An Introduction to Heuristic Algorithms. University of Trento, Italy.

10. Yuret, Deniz. (1996). The binding roots of symbolic AI: a brief review of the Cyc project.

11. Roma, R. F. Gamble & W. E. Ball. (1993). Formal derivation of rule-based programs. *IEEE Transactions on Software Engineering*. 19(3), 277-296. doi: 10.1109/32.221138.

12. Wikimedia Foundation. (2021, March 13). *500 rum*. Wikipedia. https://en.wikipedia.org/wiki/500_rum.

13. Aziem, Mostafa Abdel., Elnaggar, Ahmed A., Gadallah, Mahmoud., & El-Deeb, Hersham. (2014). Comparative Study of Tree Searching Methods. *(IJACSA) International Journal of Advanced Computer Science and Applications*. 5(5), 68 - 77.

14. Marsland, T.A.(1986). A Review of Game-Tree Pruning. *International Computer Chess Association Journal.* 9(1), 3-19.

15. Campbell, Murray., Hoane, Joseph A., & Hsu, Feng-hsiung. (2002). Deep Blue. *Artificial Intelligence*. 134(1-2), 57 - 83. https://doi.org/10.1016/S0004-3702(01)00129-1.

16. Black, Paul E. (2 February 2005). "greedy algorithm". *Dictionary of Algorithms and Data Structures*. U.S. National Institute of Standards and Technology (NIST). Retrieved 3 May 2021.

17. Stone, P. (2009). The Logic of Random Selection. *Political Theory*, *37*(3), 375–397. https://doi.org/10.1177/0090591709332329

18. Radner R. (1975). Satisficing. *Marchuk G.I. (eds) Optimization Techniques IFIP Technical Conference*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-38527-2_34

19. Hayes-Roth, Frederick. (1985). Rule-based systems. *Commun. ACM*. 28(9), 921–932. https://doi.org/10.1145/4284.4286.

20. Aartun, Vemund Innvær. (2016). *Reasoning about Knowledge and Action in Cluedo using Prolog.* Master's Thesis. University of Bergen.

21. O'Shea, Keiron & Nash, Ryan. (2015). An Introduction to Convolutional Neural Networks. ArXiv e-prints.

22. Story & Video by John General, & Sarlin, J. (2021, April 29). *A false facial recognition match sent this innocent Black man to jail*. CNN. https://www.cnn.com/2021/04/29/tech/nijeer-parks-facial-recognition-police-arrest/index.html.

23. Romanycia, M. H. & F. J. Pelletier. (1985). What is a heuristic? *Computational Intelligence* 1(2), 47–58. http://www.sfu.ca/~jeffpell/papers/RomanyciaPelletierHeuristics85.pdf

24. Bianchi, R., Ribeiro, C., & Costa, A. (2007). Heuristic selection of actions in multiagent reinforcement learning. InIJCAI'07, Hyderabad, India.

25. Santos, M., Antonio, J., López, V., Botella, G. (2012). A heuristic planning reinforcement learning algorithm applied to role-playing game strategy decision systems. *Knowledge-Based Systems*. 32(1), 28 - 36. https://doi.org/10.1016/j.knosys.2011.09.008.