Alex Heffner

Dr. Chelberg

CS 4900/6800

4/30/2021

Euchre Artificial Intelligence Report

I. Abstract

For my final project I implemented an intelligent Euchre game. I created it to show different heuristics making decisions on the different task in Euchre. My program decides on weather or not to tell the dealer to pick up the card for selecting trump or not. My program also has three different heuristics to determine on what card to play once the selecting of trump is over. The first heuristic is picking a random card. The second heuristic is playing a card to take the trick if you team is not already winning. The third heuristic simulates playing a card and count the number of wins with that card and then actually plays the card with the most wins. In this project I mainly focused on implementing Intelligence on the playing card portion and not as much on the selecting trump. Through out the rest of my paper you will find our how each one of these heuristics performed statistically against each other.

II. Introduction

The reason I choose to do this project is that I have played a lot of euchre with my friends and I think it is a cool game to play. I also think that making an intelligent program for Euchre would be cool, because there is a lot of unknown variable making it a challenge. I think it would be cool to create an intelligent program that could be comparable to a “smart” or “experienced” partner in the game. With accomplishing this I could also make it so in the future if I ever wanted to make an online euchre game, I could possibly use something like this project for the automated players.

*How to play:*For this project I will implement a standard round of euchre. Euchre is play with a standard deck of cards excluding cards rank 8 and below (Not ace). There are 4 players in euchre. A random person will start for dealing. The dealer will deal 5 cards face down to each player. Each player can look at their cards. The deal takes the 4 cards left and flips the top one face up and put that in the middle. A player’s partner sits across from them (a player in between). The player to the left of the deal goes fist and chooses if they would like the suit of the card face up to be trump. If so, they will tell the dealer to pick up the card and the dealer will swap that card with another card in their hand and then put their other card face down on the pile and move the pile to the side. If the person does not want that suit to be trump the player passes and the option moves to the next player. If all players pass, then the person to the left of the dealer can call trump by declaring any suit except the suit of the card that was turned down. If the player does not want to call, then the player can pass on to the next person. If the pass gets back to the dealer again the dealer must call a suit.

When trump is deiced the person to the left of the dealer starts by playing a card. The next person to place a card down must place down a card of that same suit if they have that suit in their hand, otherwise they can play another suit or trump. The goal is for you or your partner to take the “trick” which is when one card from each person is laid down. For the basic scoring of this project the goal is get at least 3 tricks to win the round.

Rules of ordering. For the ranking of cards, the higher the rank the better. Ace being high. But when a suit becomes trump, that suit will “trump” any other card of another suit. A trump 3 will beat a Ace of non-trump. Also, when trump is called the jack of that suit becomes the all-time best card (now out of order) and the jack of the same color will be the second-best card and its suit will change from what it was to trumps suit. These are sometimes referred to as the bowers.

III. Technical Approach

When starting to create this project I first implement a working “game”. This was first done by create a class of a card and of a game. The card class entails a suit and a rank. The game class entails many things needed for the game, such as players hands, current trump, etc... After creating the classes, the first thing I worked on was to play the game just the first stage of selecting trump. This entailed a *start* function, a *play to trump* function, a *evaluate for dealer pickup* function, a *dealer picks up* function, a *card comparison* function, an *update bower function* and a *choose trump function*.

The *start* function initializes the dealer, each player’s hand, the kitty, and the card facing up.

The *play to trump* function was my main body for this first half and used many of the other functions listed. This function would go through the first round and play until trump was picked.

The *evaluate for dealer pick up* function would be a function that would tell a player if they should tell the dealer to pick up the card or not. This is a function where some intelligence is implemented. I currently have it only based off how many trump cards they would have. If a player has 4 trump cards, then the dealer will pick it up, but if a player has 3 trump and the dealer is on there team or they themselves are the dealer then they will pick it up.

The *dealer picks up* function swaps the cards.

The *card comparison* function is a function that compares cards to each other to see which one is better or worse.

The *Update bower* function changed the rank of the two highest jacks to be the left and right bower and it changes the suit of the left bower to trump.

The *choose trump* function is a function that gets called when the dealer turns down the card faced up and other players get the option of choosing trump. This is another intelligent function. The player will return a suit if the player has 3 trump cards of that suit, if not then it will return no suit. If the dealer is at this point, he will be stuck and forced to choose the one with the most trump.

For the second phase of the project, I implemented the rest of the game using the *play to end* function. This function would have each player play to their heuristic and play out the rest of the game. This second phase included functions named, *legal moves, rank legal moves, card is greater than, print game, shuffle1, change heuristic of players*, and *play to end simulation.*

The *legal moves* function return all of the legal moves a player could play.

The *rank legal moves* function would rank the best move based on each heuristic.

The *card is greater than* is a card comparison function that compares cards based on playing on table. This could also be where intelligence is implemented into the game. This function at the moment looks at what trump is and what rank the cards are. But there could be more things to be considered like if you play this card will you only be two suited making it easier for you to play trump in the future.

The *print game* function is a function that prints different aspects of the game.

The *shuffle1* function shuffle cards that are not seen by the current player. Used for guessing in simulation. \*NOTE: I used a built in library to shuffle the cards (#include <algorithm>) (std::shuffle(…))

The *change heuristic* function changes what heuristic the players are using, based on what parameters are put into it. (0 – 2) currently. This could be a place where you make new Artificial intelligence heuristics and test them against others.

The *play to end simulation* takes in a card to play and plays a simulation of that. It is almost identical to the play to end function.

There were three different heuristics that I used in the rank legal card’s function. The first one is a random pick of a card. This could represent a complexly new person or someone who is not paying attention to the game.

The second heuristic I used was a heuristic that would take play a card that would win the trick if your team were not already winning. If your team were already winning it would throw the worst card down. And for leading it would throw a good card.

The third heuristic would simulate playing each card in shuffled version of the other player cards (because those cards could be anywhere to the player) and then it would play out the game and count how many times the team won and lost and when it gets back to the original selection it will select the card with the most wins.

IV. Contributions:

I implement the whole project except for the shuffle function that I imported for shuffling from the algorithm library (#include <algorithm>). I also looked up information on how to seed a random time with this shuffle algorithm from stack overflow (link in references). The idea of shuffling and looking down the line for my third heuristic has been done before, but I did not look off of any particular site for any implementation examples or information.

V. Experimental Design

The way I validate my design was to simulate playing many games and having the heuristic play against each other and average the wins over all the games to get a percentage of how well that heuristic did against the others. This is implemented in my pmain and seq main. The pmain is the cc file that executes the program on multiple threads to save time and the seq is the basic sequential version. For the sequential version I have 50 games running due to the time it takes to compute. The pmain is currently implemented with 1000. This takes approximately 5 mins to run. When I calculated I went ahead and made it 10000 for a more accurate result.

VI. Experimental Results:

When using 10000 games I came up with the following results.

When using heuristic 0 (random) for both teams, team 1 won 50.87% of the time.

When using heuristic 1 for team 1 and heuristic 0 for team 2, team 1 won 58.48% of the time.

When using heuristic 1 for both teams, team 1 won 47.32% of the time.

When using heuristic 2 for team 1 and heuristic 0 for team 2, team 1 won 67.13% of the time.

When using heuristic 2 for team 1 and heuristic 1 for team 2, team 1 won 61.89% of the time.

Total time 54min 32.950s

When using 1000 Games I cam up with the following results.

When using heuristic 0 (random) for both teams, team 1 won 50.1% of the time.

When using heuristic 1 for team 1 and heuristic 0 for team 2, team 1 won 58.7% of the time.

When using heuristic 1 for both teams, team 1 won 48.2% of the time.

When using heuristic 2 for team 1 and heuristic 0 for team 2, team 1 won 64.8% of the time.

When using heuristic 2 for team 1 and heuristic 1 for team 2, team 1 won 62% of the time.

Total Time 5min 5.415s

See End of paper for more information on results.

VII. Conclusions

Looking at the results I would say my main three heuristics turned out well. I think that I would expect the random one to lose on average compared to the other two heuristics. I would not make any changes to this heuristic because by nature it is supposed to be a “bad” heuristic and not that smart when playing. I think it held up to its use in this project.

I think that the second heuristic performed decently well for always just taking the trick if it can. I think one reason this performed decently well is that it played the worst card in its hand when it knew it would not win. I think getting rid of the worse cards helped it in the future. There is more that can be added to this heuristic. There are times in euchre where playing the best card may not be the best move for a person and this could be programed into this heuristic or could be made into another one. Another way to make this heuristic better I could make it that when it throws a bad card the heuristic could consider the suit of the card and maybe if we throw away one bad card, we could be one suited which may help later for having and option to throw trump. I think that this heuristic could have a lot of potential to see how close you could get it to winning.

The third heuristic performed the best out of all three. This is probably due to the fact that it played out a simulated version of the game and plays out many games to see which option would have the most wins. I think that this strategy performed well, but this heuristic took the biggest chunk of time out of my program by far. This is due to the recursive nature of this heuristic. This is something to consider when playing thousands of games. I think that this heuristic turns out to be better performing but at a greater cost which is something to balance between when trying to consider different heuristics for this game. The second heuristic could be much quicker and still has an advantage over a new player (first heuristic). I do think that this this heuristic could be improved in multiple ways. I think one way it could be improved is to prune the search by adding restriction on which legal paths it takes. This can be shown in multiple different ways. One of which is that we can stop searching if a team has 3 tricks. We do not need to play out the rest of the round because we know who won. Another thing we could do to increase accuracy is to simulate multiple shuffled games. Right now, I have the heuristic only shuffling once and playing out that game but if we were to shuffle multiple times then we could possibly get a better average. This would increase the amount of time the heuristic would take but maybe with some of the other techniques used to reduce the time it could be something to consider doing in the future.

In the future if I had more time to work on this project, I would also implement a better trump choosing system. Although that was not what I choose to focus on in this project, it is still a large part of Euchre strategy and how the game turns out. Currently my strategy is just based off of the amount of trump cards a player has in their hand, but there are other things that could definitely be considered. One thing that is large that comes to mind is the rank of the cards. If you only have two trump cards but they are the left and right bower it is probably a good idea to tell the dealer to pick up the trump. The heuristic I have currently would pass up on that.

VIII. References:

“std::random\_shuffle,” *cplusplus.com*. [Online]. Available: <http://www.cplusplus.com/reference/algorithm/random_shuffle/>

. [Accessed: 01-May-2021].

laurentlaurent 78.9k6262 gold badges255255 silver badges387387 bronze badges, user703016user703016 35k77 gold badges8080 silver badges106106 bronze badges, Mehmet FideMehmet Fide 1, user195488user195488, madxmadx 5, Apollys supports MonicaApollys supports Monica 1, and OcezoOcezo 1, “How to shuffle a std::vector?,” *Stack Overflow*, 01-Apr-1960. [Online]. Available: <https://stackoverflow.com/questions/6926433/how-to-shuffle-a-stdvector>

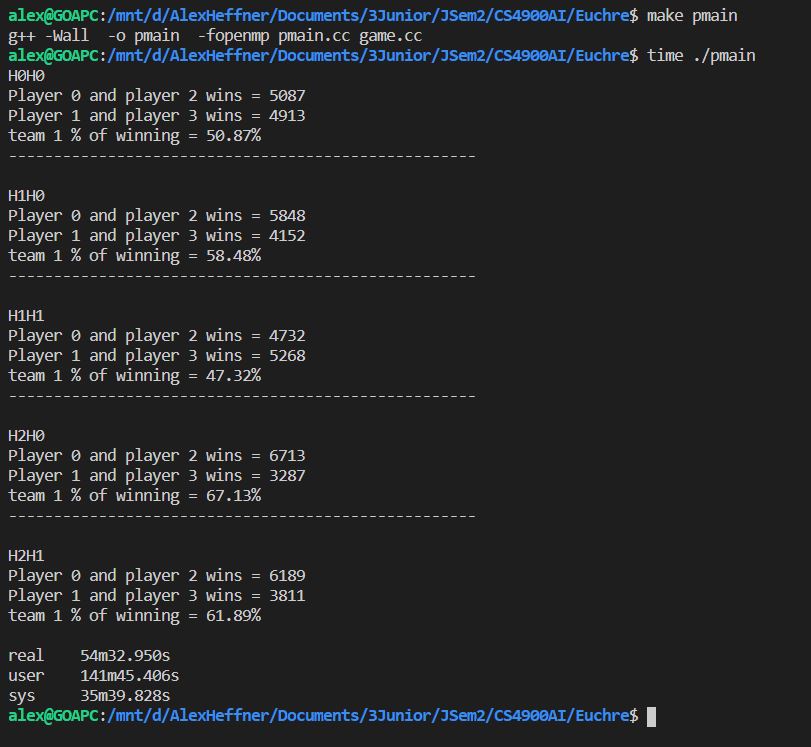
. [Accessed: 01-May-2021].

D. Bravender, “Over 1 billion tricks played - Information Set Monte Carlo Tree Search Euchre simulation database,” *Over 1 billion tricks played Information Set Monte Carlo Tree Search Euchre simulation database*, 09-Jun-2017. [Online]. Available: <https://dan.bravender.net/2017/6/9/Over_1_billion_tricks_played_-_Information_Set_Monte_Carlo_Tree_Search_Euchre_simulation_database.html>

. [Accessed: 01-May-2021].

IX. Appendix 1:

10,000 Game Evaluation



1,000 Game Evaluation

