Group: Robosquad
Phase 2 Document

Team Members: Himanshu Londhe
Swapnil Bhosale
Pratik Jogdand

Our original Approach which was to implement Reinforcement Learning to the trick taking game was changed due to the time constraint for the project. These are the following **challenges** we faced in implementing Implementing Reinforcement learning using the Partially Observable Markov Decision Process (POMDP).

- OpenAI gym is one of the libraries available to implement a POMDP.
- The OpenAI gym library requires the programmer to create a custom environment and setting reward functions.
- We found this task task challenging by itself. We tried to implement POMDP using OpenAI and created a custom environment. We even set up random bots to play with our agent. But when we started defining our agent, we couldn't define a finite state space for the given problem.
- Using our approach we reached a state space size of ~132,000 and with permutations the total size of the state space went above 6.6 Million.
- The size of the possible states in a card trick taking game makes the learning phase tediously time consuming. Due to insufficient time available, we were not able to experiment enough using our approach.
- Additionally, the computational power required for the learning for so many phases isn't available to us at the moment.
- Given more time and resources, we can implement this, but for the time being we are switching our approach.

Given the above circumstances and with our thorough introspection, we saw it fit to make the agent play as a human would. In this case the human in question is the programmer. To make the agent play as a human would, we chose to implement a **rule-based agent.** These rules written for the agent encapsulates perfectly the thinking of a human when it plays cards with an additional advantage of memorizing the cards that are being played so far. Given that the programmer's strategy is optimal, it should win the tricks most of the times (not proven). And if all the agents playing the game use this strategy using the same rules, we achieve Nash equilibrium, as no agent will benefit by changing the strategy. But equilibrium is possible only when the programmer has used an optimal strategy. In conclusion, the agent will only be as good as the programmer.

The rules of how a card is chosen is explained in the code with comments. The rules consider the past played hands and the cards played in the current trick in the given game. Additionally the following pseudo code gives the description of the ***play_card()*** function and how the rules help the agent choose the card to play in ever one of thirteen rounds for the game:

```
def(play_card):
        if agent is the first player:
                if agent has the 2 of Clubs it is the lead:
                        play the 2 of Clubs
                else if the agent has any ace:
                        play the ace.

                else if agent's any of the highest card from any of the suits has all the   greater
                valued cards already played in the history then:
                        play that card.
                else:
                        play the lowest card among all the suits.

        if agent is 2nd or 3rd player:
                if agent has the ace from the lead-suite:
                        play the ace.

                else if  agent has higher card from the lead-suit than the ones already
                played in the current round and all the cards greater than the
                chosen card have already been played in previous tricks then:
                        play that card.

        else if agent has card from the lead-suite:
                        play the lowest available card from the lead-suite.

                else:
                        play the lowest available card among all the suits.

        if agent is the last player:
                if agent has a card from the lead-suite which is greater than all the
                cards which were already played:
                        Select the minimum out the cards that are greater than all the
                        cards which are played already.
```

```
else if agent has the cards from the current suite:
    play the lowest available card from that suit.

else:
    play the lowest card among all the suits.
```