# CS221 Final Project Report
# Learn to Play Texas hold'em

Yixin Tang(yixint), Ruoyu Wang(rwang28), Chang Yue(changyue)

## 1 Introduction

Texas hold'em, one of the most popular poker games in casinos, is a variation of the card game. Each player seeks the best five card poker hand from the combination of 5 community cards and their own 2 hole cards. Since Texas hold'em is a game with imperfect information and stochastic outcomes, creating a rule-based playing bot is not only complex but also not effective.[1] In light of this, we developed a program that can play one on one Texas hold'em using artificial intelligence techniques. Our baseline model classifies cards and gives advice on the next action based on the current hand's winning probability. We then trained a model by employing reinforcement learning method, especially Q-Learning, to get the optimal policy at each state. After knowing the optimal policy, the action finally has been taken is determined by a non-deterministic betting strategy.

There are two ways to evaluate our program and we tested on both. The first method is asking a human volunteer to play against our AI player, and the other one is letting our program to play against other existing algorithms. Winning rate and average reward are two major metrics. We will describe both methods and their results determined by both metrics in the analysis section.

## 2 Background

The figure below shows the process of the game. The first example shows the case where the player bets after the 'turn' state but folds before the 'river' state so its reward is minus the total amount of money it bet. The player folds after betting because the opponent re-raised and then the player choose to fold. In the second example, the player did not fold in any action and the reward is based on the game result that determined in the end state. If the player wins, it is awarded the pot, so the reward will be the total amount of money that the opponent put in the pot. If the best hand is shared by both players, then the pot is split equally among them, which means that the reward for both players is $0.

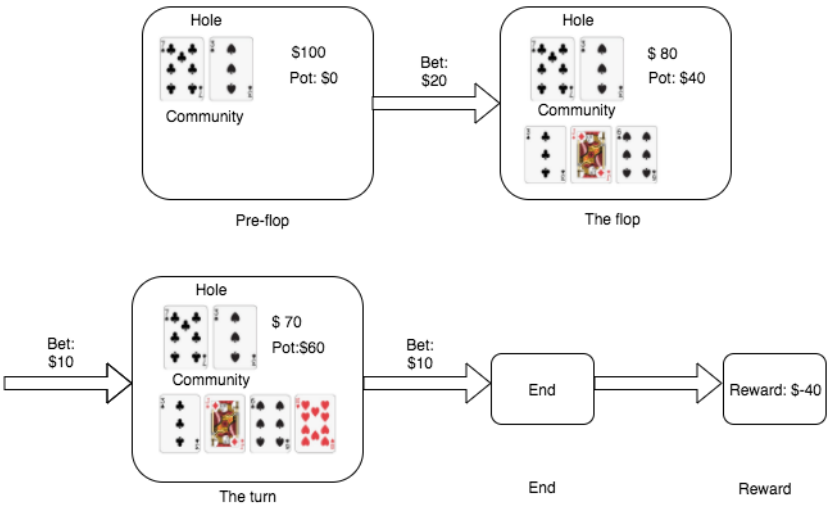


Figure 1: Example 1: Flow chart of a game.

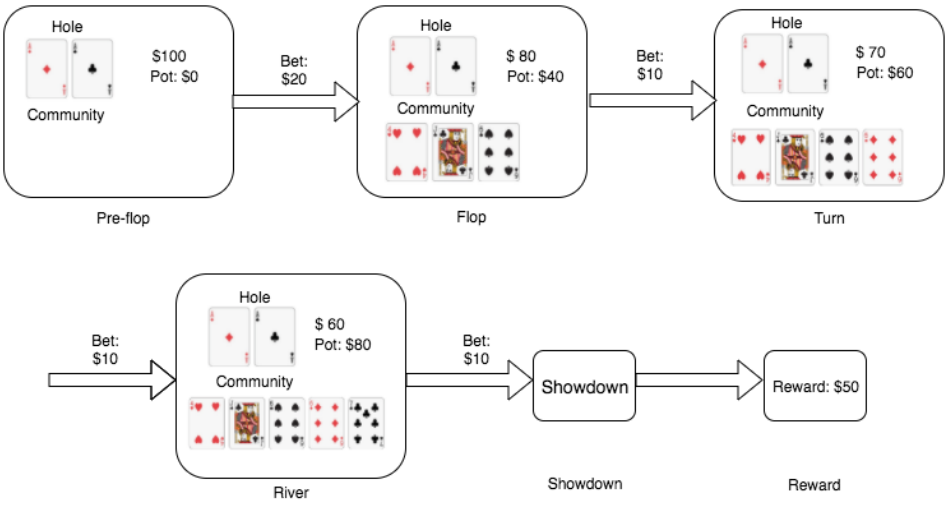


Figure 2: Example 2: Flow chart of a game.

# 3 Approaches

We built a user interface to make the game joyful and convenient to play. It shows table cards and user's hole cards throughout the game. It also asks the user to input his or her action every round and reminds the user about the real time pot size and the money he or she left. The program can also recognize any invalid input and tells the user what went wrong. An example is shown in the appendix. Deuces is a Python poker library which can print cards in iconic forms (i.e., it can print clubs, diamonds, hearts and spades) and classify a combination of cards into 7642 categories, and we imported it to improve our program's user experience and help with classification. Other than Deuces, we write the game engine by our own. Our program has the main python file (called controller.py) that controls the whole process of the game by asking for human player's actions and outputting its own actions, keeping a record for all players current status, and determining the winner and the way to distribute the pot in the end.

The betting strategies which computer take are based on the model we designed. We developed two models: baseline, and Q-learning. Our baseline model simply take actions by current hand's winning probability, while Q-learning model makes decisions based on state. The unbeatable oracle model is a cheat version of the game, in which computer takes advantage of knowledge about the human player's hand cards.

## 3.1 Baseline

Generally, our baseline model is based on conditional winning probability and matching this with pot odds to decide the amount we bet. For simplicity, we assume there is no restriction on the amount of each bet, which means no limit and no minimum requirement. And we will do our baseline model for only one on one game.

Before we start, the first thing we need to do is: given five cards, to classify its poker hand rank. Since it is verbose and time-consuming to check the cards with each class, we want to employ machine learning methods to train the computer so that it can give us an accurate answer faster. We tried logistic regression and decision tree solvers using Python sklearn. With basic functions, we got an accuracy of 0.501209 from logistic regression and an accuracy of 0.479585 from decision tree method. After changing the parameter 'min_samples_leaf' to 49, the decision tree method can reach an accuracy of 0.528427. Using random forest method, we got an accuracy between 0.57 and 0.59. The gradient boosting method can produce an accuracy of 0.613977.

After classify any five cards combinations to a certain class, we also need to make use of a probability table. This is the theoretical probability of getting a certain class:

| Poker Hand | number of hands | Probability |
|---|---|---|
| Royal Flush | 4 | 0.000000154 |
| Straight Flush | 36 | 0.00001385 |
| Four of a kind | 624 | 0.00024 |
| ... | ... | ... |

For any stages of the game, our program will always compute the amount of bet by balancing the probability above. For example, suppose we are in the stage of making the third bet, that is we already have 2 holes and 4 community cards. Then our program will classify each combination of five cards of the total 6 cards and select the highest ranked class. For instance, the highest class we can get from a 6 card deck is "Four of a kind". Then we can compute the winning probability according to the table above, which is

$$P_{win} = 1 - P(RoyalFlush) - P(StraightFlush) - 0.5 * P(Fourofakind)$$

The reason we take a 0.5 factor in the last term is because we want to make an unbiased estimation of the within-class rank. Then the amount we bet will be:

$$M_{Bet} = M_{\text{Total in Current Pot}} * P_{win}$$

The reason behind is that $\frac{M_{Bet}}{M_{\text{Total in Current Pot}}} = P_{win}$ yields the theoretical optimal strategy in idealized model.
Then if the opponent raise within the 1.5 times of $M_{Bet}$, we will call and otherwise we fold.

**while** *not at end of the bet* **do**
  Classify all 5 combination cards of your hand and community cards;
  Select the one of the highest class rank;
  Compute the winning probability $P_{win}$ based on this class;
  Bet with $M_{Bet} = M_{\text{Total in Current Pot}} * P_{win}$;
  **while** *Opponent stops raising* **do**
    **if** *opponent folds or raise within 1.5 * $M_{Bet}$* **then**
      call;
    **else**
      fold;
    **end**
  **end**
**end**

**Algorithm 1:** Baseline strategy

## 3.2 Q-learning

### 3.2.1 Model

Our model consists of two parts: (1) basic states which are determined by the number of community cards and (2) the rounds in between basic states. Since five community cards are dealt in four stages (pre-flop; flop; turn; river), and there are only two main decisions between stages (stay; quit), no matter how many betting rounds happened in between two stages, our basic model is based on only four stages. Major decisions made at these stages will lead to either the next stage or the end of the game. The main state diagram is shown below.

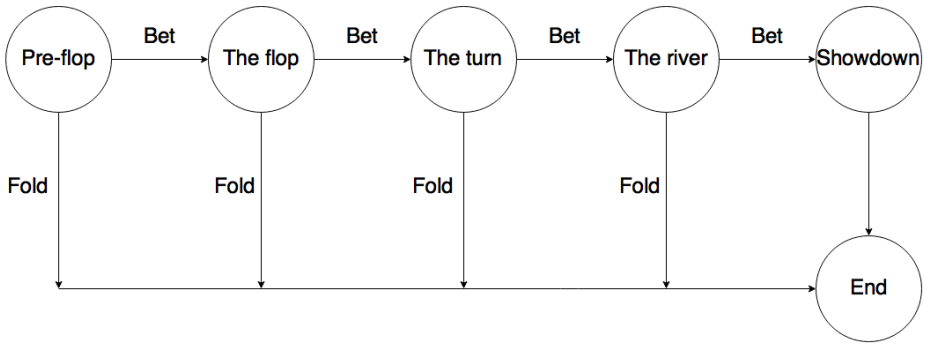


Figure 3: State diagram of basic model.

Current cards (hole cards plus community cards), money left and the minimum betting amount to stay active are three facts that could uniquely define a state. However, since there could be at most 7 cards in total (2 hand cards plus 5 community cards) and 10 discrete possible values for money left, there are more than $5^{12}$ possible states. The number of states is too big to reach all of them, even with the help of function approximation. So we decided not to use exact cards as a factor of state, instead we use the rank of cards, a derived version of winning probability which Deuces provides. There are 7642 possible ranks, and the reason that it is so small compared with card combinations is that in Texas hold'em only hand category matters when determining the winner, where there are only 10 categories. Mathematically, the new version of winning probability and rank are related in this way:

$$P_{win} = 1 - \frac{Rank}{7642}$$

$$Rank = (1 - P_{win}) * 7642$$

where higher rank (smaller number) means better hand. With this important trick, we can now represent states in three numbers neatly: [betMin, betMax, rank]. Here, betMin represents the minimum betting amount to keep the player on the table, and betMax represents maximum valid betting amount (which is equivalent to money left).

We adopted the reinforcement learning method, specifically Q-learning with function approximation, to train our model. The reason for choosing function approximation is that it has better generalization, which helps with the huge number of states we got. From lectures we know that:

$$w \leftarrow w - \eta[\hat{Q}_{opt}(s, a; w) - (r + \gamma \hat{V}_{opt}(s'))]\phi(s, a)$$

$$For \quad \hat{Q}_{opt}(s, a; w) = w \cdot \phi(s, a)$$

where $\phi(s, a)$ is a vector of selected features and the optimal action is the one gives best $\hat{Q}_{opt}(s, a; w)$. We firstly tried to set all factors in state and action as only features but found w hardly ever converged. Next, we tried to include non-linear relationships between those factors and succeeded this time. After eliminating some minor features, our final feature space has a dimension of 10. Below is a summary of state, action, reward and feature.
State: (betMin, betMax, rank)
Action: bet
Reward: 0 for intermediate states, determined at the end by rules
Feature: [$\text{betMin}^2$, $(\text{betMin} * \text{rank})^2$, $(\text{betMax} * \text{rank})^2$, $\text{bet}^2$, $\text{betMin}^2$*rank, betMin*betMax*rank, betMin* bet, betMin * betMax * $\text{rank}^2$, betMin * rank * rank, betMax * rank * rank].

### 3.2.2 Training Data

The training data was generated by using Monte Carlo simulation, in which we let our program automatically playing against itself and some other existing programs for roughly 16,000 times. Each piece of training data is a list of all (s,a,r,s') tuples in one round, and we generated 495,924 tuples from 160,000 rounds of games in total. Each state includes the minimum bet and the money the player left which is the maximum bet, and the rank of the player's current poker hand. Since we cannot evaluate

a poker hand before the total number of cards of the player (hole cards and community cards) reach 5, the poker hand rank for the pre-flop states is None. At every betting round, the player chooses whether to fold or bet with an allowed amount, and the money he or she put in is recorded as an action, and we define -10 as the fold action. Since this game only determines reward in the end, rewards at intermediate rounds (basically pre-flop, flop, turn and river) is 0 if the player decides to remain in the game and negative of total money the player has put if he or she decides to quit. The reward in the end (the showdown state) is distributed by Texas holdem's rule.

### 3.3 Betting Strategy

After completing Q-learning algorithm, we can obtain the optimal action given each state. Concretely, for each given state, say we now have hand cards: six of spades and ace of hearts, and four community cards: seven of spades, king of diamonds, and jack of clubs, we know the optimal amount of money we need to bet to enter next state (or directly quit). However, this is just a number instead of a strategy, since a strategy needs to deal with cases like what if opponent raise, re-raise, check or fold.

We considered two aspects when defining our strategy. First of all, we need our strategy to be non-deterministic. This is practically important since if we follow the optimal betting value on each turn and fold every time when opponent raises higher than the optimal value, once the opponent finds out this feature, it will always 'bluff' our program. Similarly, we cannot always be aggressive. Therefore, we need to insert some randomness into our strategy to decide when to play aggressively and when to play cautiously to not to let our opponent find out our real strategy. The second aspect is to decide quantitatively how we attribute the weights (or probability) of the fold, call, bet to make our program aggressive, modest, or normal.

Based on the above considerations, we designed the algorithm shown below.

**Result:** Betting Amount
Compute theoretical optimal betting value $X * P_{opt}$;
Setting a symmetric density function $f$ with mean 0;
**while** *Not at the last turn of flop* **do**
    Set $c_i$ and $\sigma_i^2$;
    Random generate a number $x$ from density $f$;
    Bet with $x\sigma_i^2 - c_i + X * P_{opt}$
**end**

**Algorithm 2:** Betting Algorithm

Putting the whole betting strategy into one Mathematical formula, we therefore set the amount we bet as any reasonable symmetric distributed density function $f(\frac{Y-(X*P_{opt}+C_i)}{{\sigma_i}^2})$. We use bias and variance to represent the personality of our AI agent. Concretely, a greater variance and more positive bias means a more aggressive player. We evaluated both algorithms with and without Betting Strategy Model, and we found that though it brings some noise to our AI, it has significant improved winning rate after the certain large number of games.

### 3.4 Oracle

Given future community cards and current opponent's cards, the "best" player should quit before the flop if found himself going to lose, and All-in if it has better hand category. In reality, we don't know about future and opponent's cards before the showdown stage. However, as both the dealer and player, the computer can cheat: it can randomly draw 9 cards before the game starts; assign itself and human player hole cards; at this stage, the computer has already known the winner, and it can act accordingly. The reward is maximized in this way. It could even deliberately assign itself better hole cards and always win, but this is no longer a game,a conspiracy instead. The difficulties to achieve the oracle model are that we only have too few information about the quality of our hole cards with 3 or 4 table cards, and we lack the information of opponent's hands even if we know our hand category (i.e., 5 table cards).

# 4 Results and Analysis

## 4.1 Human Evaluation

| # of rounds / Model | 50 | 200 |
|---|---|---|
| Baseline model | 0.58 | 0.44 |
| Baseline model with betting strategy | 0.47 | 0.54 |
| Reinforcement learning model | 0.66 | 0.61 |
| Reinforcement learning model with betting strategy | 0.59 | 0.68 |

Figure 4: The winning rate of our models with human players

We evaluated both our baseline model and reinforcement learning model with and without the betting strategy (3.3) against 10 volunteers. And we can find:

- Both baseline model and reinforcement learning model beats human with high probability.
- Reinforcement learning model has significantly better performance than baseline model.
- Adding betting strategy will decrease AI winning rate when total rounds are small, while significantly improve it when total rounds become large.

Remark: The main reason that all of our models can beat normal people is because models can get the exact conditional probability of cards while human judgment is based on past experience. Reinforcement learning is much better than baseline model since it incorporates reward, state, and actions together and makes approximation function and features space much more complex. The betting strategy can be viewed as introducing bias and variance to our optimal from a statistical perspective. In a short term, due to the bias, it lowers the winning rate of AI. However, in a long time, bias and variance are used to confuse human so that people cannot find the underlying pattern of our algorithm.

## 4.2 Compete with Other Models

Besides human evaluation, we also implemented a strategy using the idea from Deuces, a tool used for MIT Pokerbots Competition. Deuces handles 5, 6, and 7 card hand lookups. The 6 and 7 card lookups are done by combinatorially evaluating the 5 card choices. That is to say, for any cards combination, this strategy (we will call this Deuces-based strategy afterward) have the ability to compute a score to measure how good a player's hand cards are. Our Deuces-based strategy is implemented based on this scoring system and our baseline model.

We let our reinforcement model played against the Deuces-based model for a total of 1000 rounds, and kept winner history and winning amount history over the time.

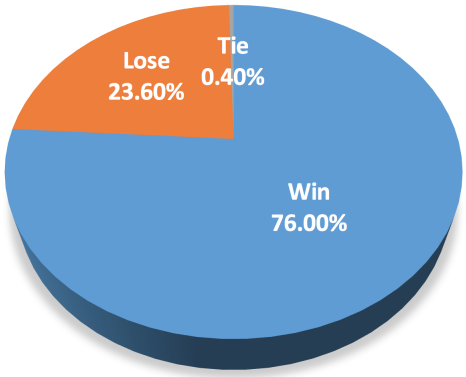


Figure 5: Win-Loss Pie Chart of the Q-learning model

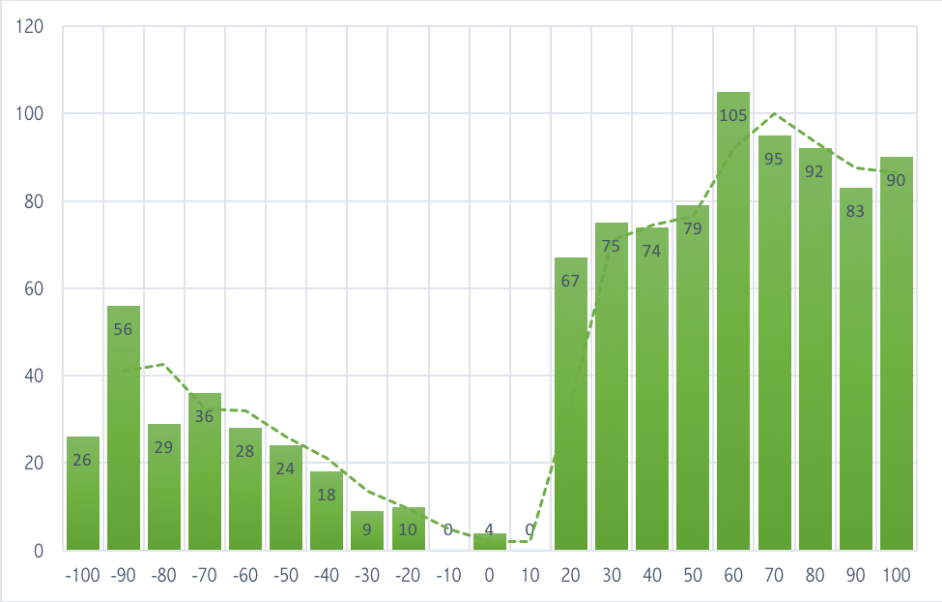


Figure 6: Net Profits Histogram of the Q-learning model

From the charts, we can conclude that our model beats Deuces-based model by a lot in terms of both winning rates and average winning amount. Concretely, in a total of 1,000 consecutive rounds (each round starts with 100 dollars), our model has won 760 rounds, with a net profit of 30,730 dollars, which was $30.73 on average per game!

The primary reason is that Deuces-based model uses only card evaluation rank, or equivalently, winning probability, while our model uses much more state features and complicated function kernels to represent states' weights.

# 5 Related Work

Except for the rule-based playing agent we mentioned earlier, there are other types of approaches used in creating Texas hold'em playing agent such as simulation-based and game theory-based, such as $\epsilon$-Nash equilibrium approach. Researchers at Carnegie Mellon University created several poker agents using the $\epsilon$-Nash equilibrium strategy.[2] University of Alberta computer poker research group have been studied in this area for more than twenty years and they also created a series of artificial poker players such as the Cepheus and Polaris, which are heads-up limit Texas hold'em agents, and the Hyperborean Ring which is a multi-player poker agent created by using counterfactual regret minimization.[3][4]

# 6 Future Work

For next steps, we may expand our feature space, and employ more AI techniques such as neural networks and genetic algorithms. We may also generalize our game settings such as allowing multiple rounds and multiple players.

# References


[1] Darse Billings. *Algorithms and Assessment in Computer Poker*. Ph.D. thesis. University of Alberta, 2006.

[2] Andrew Gilpin, Tuomas Sandholm. A Competitive Texas Hold'em Poker Player Via Automated Abstraction and Real-time Equilibrium Computation. *Proceedings of the 21st National Conference on Artificial Intelligence*, 2006.

[3] M. Bowling, N. Burch, M. Johanson, and O. Tammelin. 2015. Heads-up limit hold'em poker is solved. Science 347, 6218, August 2015, 145–149.

[4] Nicholas Abou Risk. *Using counterfactual regret minimization to create a competitive multiplayer poker agent*. Master of science thesis. University of Alberta, 2010.

# Appendix

```
-------------go Hands----------------------

player1 hand:
  [ A ❤ ] , [ 7 ❤ ]
AI's hand: Invisable to you

player1 bets smallBlind 10
AI bets bigBlind 20

How much do player 1 bet?  The money you bet must be at least 10. You have $90 left.
10
AI bets with 10
How much do player 1 bet?  The money you bet must be at least 10. You have $80 left.
10
-------------go Flop----------------------

The Flop is
  [ 3 ♣ ] , [ 4 ❤ ] , [ 7 ♠ ]
How much do player 1 bet?  The money you bet must be at least 0. You have $70 left.
20
AI bets with 30
How much do player 1 bet?  The money you bet must be at least 10. You have $50 left.
30
AI bets with 20
-------------go Turn----------------------

The Flop is
  [ 3 ♣ ] , [ 4 ❤ ] , [ 7 ♠ ] , [ 6 ♦ ]
How much do player 1 bet?  The money you bet must be at least 0. You have $20 left.
20
AI bets with 20
-------------go River----------------------

The Flop is
  [ 3 ♣ ] , [ 4 ❤ ] , [ 7 ♠ ] , [ 6 ♦ ] , [ J ♦ ]
How much do player 1 bet?  The money you bet must be at least 0. You have $0 left.
0
AI bets with 0


Player1 got:
  [ 3 ♣ ] , [ 4 ❤ ] , [ 7 ♠ ] , [ 6 ♦ ] , [ J ♦ ] , [ A ❤ ] , [ 7 ❤ ]
AI got:
  [ 3 ♣ ] , [ 4 ❤ ] , [ 7 ♠ ] , [ 6 ♦ ] , [ J ♦ ] , [ 7 ♦ ] , [ 3 ❤ ]


AI wins!!
-------------Conclusion----------------------

You lose pot -200
AI now has 200
```

Figure 7: An example of our user interface.