Algorithmic Economics Final Project

Rishi Shah, Yash Kolli, Tarun Donipati

Our Game: Leduc Poker

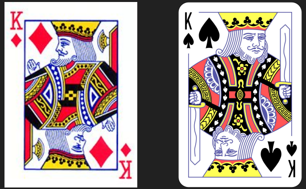
An Introduction to Leduc Poker

- Zero-sum two player imperfect information game
- A game of chance and strategy
- Explores RL capabilities to learn from a game where it doesn't know what the other player's strategy or move

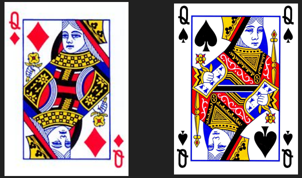
Rules of the Game

- As mentioned before, this is a simplified version of Texas Hold'em Poker
- There are only 2 players, and a card deck containing only 6 cards:
  - 2 Jacks, 2 Queens, and 2 Kings
- There is an initial ante of $1 and a fixed bet size of $1
- Players are dealt one card and there is one community card
- Player 1 can either Bet or Check
  - If Player 1 Bets, Player 2 can either Call or Fold
  - If Player 1 Checks, Player 2 can either Bet or Check
    - If Player 2 Bets, Player 1 can either Call or Fold
- 5 possible game endings:
  - Bet-Fold, Bet-Call, Check-Check, Check-Bet-Fold, and Check-Bet-Call

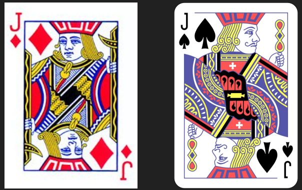
Ranks

1: 

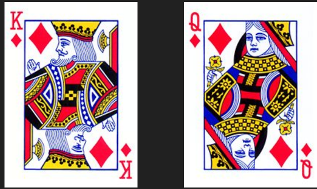
King Pair
Probability: 1/15

2: 

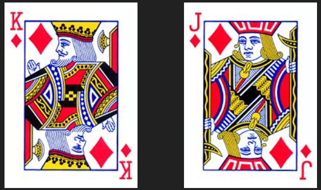
Queen Pair
Probability: 1/15

3: 

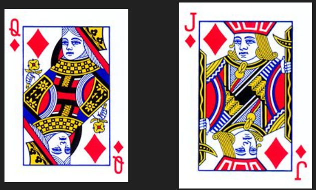
Jack Pair
Probability: 1/15

4: 

King-Queen Pair
Probability: 4/15

5: 

King-Jack Pair
Probability: 4/15

6: 

Queen-Jack Pair
Probability: 4/15

Demo of the Game

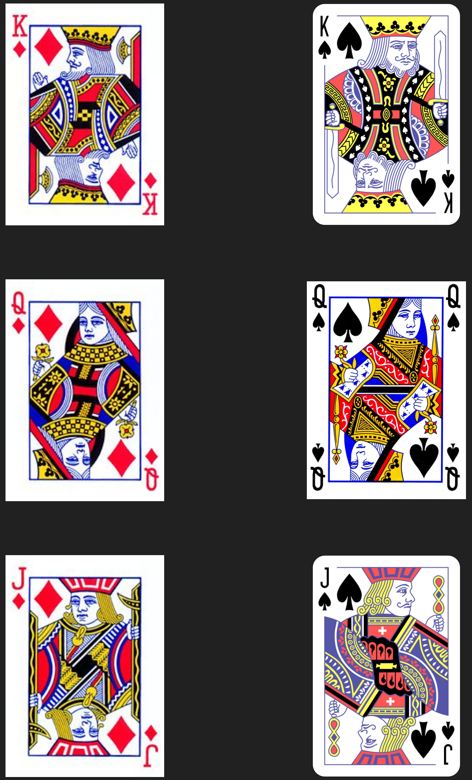
Player 1:

Starting Balance:
$100

Player 2:

Starting Balance:
$100

Deck:



Demo of the Game

Pot Size: $2

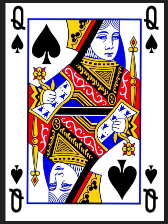
Player 1:

Balance: $99

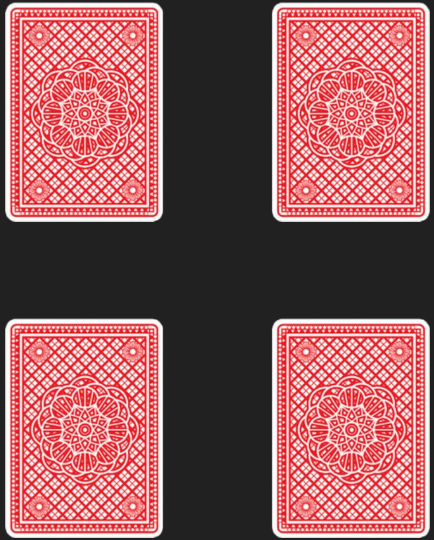
Player 2:

Balance: $99

Community Card:



Deck:



Demo of the Game

Pot Size: $2

Player 1:

Balance: $99

Player 2:

Balance: $99

Community Card:

In the 4 remaining cards, there are:

- 2 Jacks
- 1 Queen
- 1 King

Player 2:

- Wins with a Queen (25%)
- Ties with a King (25%)
- Loses with a Jack (50%)

Expected Value if we bet and Player 2 bets:

E[H] = (.5)*2 + (.25)*(0) + (.25)*(-2) = $0.50

Demo of the Game

Pot Size: $3

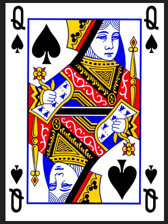
Player 1:

Balance: $98

Player 2:

Balance: $99

Community Card:



We will bet as this is a Positive Expected Value Play.

Player 2 can either Call or Fold.
If Player 2 calls, they most likely have a Queen or King.

Player 2 will rationally fold if they have a Jack.

Demo of the Game

Pot Size: $4

Player 2 Calls our bet.

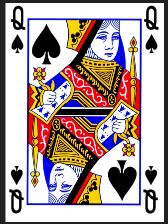
Player 1:

Balance: $98

Player 2:

Balance: $98

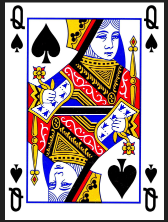
Community Card:



Now we will showdown.

Demo of the Game

Community Card:



Move to Round 2!

Player 1:

It is a tie! There was a 25% chance of this happening.

Balance: $98

Player 2:

Balance: $98

Pot Size: $4

$2

$2

Player 1:

Balance: $100

Player 2:

Balance: $100

Example of Leduc Poker in our code

```
**************** Game 1: ****************
Player 1 Card: J
Player 1 Balance: 1000

Player 2 Card not shown
Player 2 Balance: 1000

Community Card: J


Initial bets of 1 made

Player 1 Balance 999
Player 1, do you choose to check or bet
Press 0 for check, 1 to bet

 1
Player 2 Balance 999

Player 2 bets
Player 1 Wins
Player 1 Card: J
Player 1 Balance: 1002

Player 2 Card: K
Player 2 Balance: 998

---------------------------------------------
```

```
**************** Game 2: ****************
Player 1 Card: Q
Player 1 Balance: 1002

Player 2 Card not shown
Player 2 Balance: 998

Community Card: Q


Initial bets of 1 made

Player 1 Balance 1001
Player 1, do you choose to check or bet
Press 0 for check, 1 to bet

 1
Player 2 Balance 997

Player 2 bets
Player 1 Wins
Player 1 Card: Q
Player 1 Balance: 1004

Player 2 Card: J
Player 2 Balance: 996

---------------------------------------------
```

Leduc Poker (and Poker games in general) are Imperfect Information Games

- Unlike perfect information games, where all information is available to both players, imperfect information games require thought and strategy to predict players' moves and our reaction to their moves
- A rational player would follow a strategy which gives him the highest probability of winning. Yet, this strategy depends on the opponent's moves as well, which we must perceive as random or following some probability distribution, as we don't know what they hold

RL Algorithm: Modified Epsilon-Greedy

We used a Modified Epsilon-Greedy algorithm. To do so, we initialized a dictionary of probabilities for each player, represented below:

```
s1 = {
        'KK0': .5, 'KK1': 1, 'KK01': .5, 'KK10': 1, 'KK11': 1, 'KK00': 1, 'KK010': 0, 'KK011': 1,
        'QQ0': .5, 'QQ1': 1, 'QQ01': .5, 'QQ10': 1, 'QQ11': 1, 'QQ00': 1, 'QQ010': 0, 'QQ011': 1,
        'JJ0': .5, 'JJ1': 1, 'JJ01': .5, 'JJ10': 1, 'JJ11': 1, 'JJ00': 1, 'JJ010': 0, 'JJ011': 1,
        'KQ0': .5, 'KQ1': .5, 'KQ01': .5, 'KQ10': 1, 'KQ11': .5, 'KQ00': .5, 'KQ010': 0, 'KQ011': .5,
        'QK0': .5, 'QK1': .5, 'QK01': .5, 'QK10': 1, 'QK11': .5, 'QK00': .5, 'QK010': 0, 'QK011': .5,
        'KJ0': .5, 'KJ1': .5, 'KJ01': .5, 'KJ10': 1, 'KJ11': .5, 'KJ00': .5, 'KJ010': 0, 'KJ011': .5,
        'JK0': .5, 'JK1': .5, 'JK01': .5, 'JK10': 1, 'JK11': .5, 'JK00': .5, 'JK010': 0, 'JK011': .5,
        'QJ0': .5, 'QJ1': .5, 'QJ01': .5, 'QJ10': 1, 'QJ11': .5, 'QJ00': .5, 'QJ010': 0, 'QJ011': .5,
        'JQ0': .5, 'JQ1': .5, 'JQ01': .5, 'JQ10': 1, 'JQ11': .5, 'JQ00': .5, 'JQ010': 0, 'JQ011': .5,
    }
```

```
s2 = {
        'KK0': .5, 'KK1': .5, 'KK01': 1, 'KK10': 0, 'KK11': 1, 'KK00': 1, 'KK010': 1, 'KK011': 1,
        'QQ0': .5, 'QQ1': .5, 'QQ01': 1, 'QQ10': 0, 'QQ11': 1, 'QQ00': 1, 'QQ010': 1, 'QQ011': 1,
        'JJ0': .5, 'JJ1': .5, 'JJ01': 1, 'JJ10': 0, 'JJ11': 1, 'JJ00': 1, 'JJ010': 1, 'JJ011': 1,
        'KQ0': .5, 'KQ1': .5, 'KQ01': .5, 'KQ10': 0, 'KQ11': .5, 'KQ00': .5, 'KQ010': 1, 'KQ011': .5,
        'QK0': .5, 'QK1': .5, 'QK01': .5, 'QK10': 0, 'QK11': .5, 'QK00': .5, 'QK010': 1, 'QK011': .5,
        'KJ0': .5, 'KJ1': .5, 'KJ01': .5, 'KJ10': 0, 'KJ11': .5, 'KJ00': .5, 'KJ010': 1, 'KJ011': .5,
        'JK0': .5, 'JK1': .5, 'JK01': .5, 'JK10': 0, 'JK11': .5, 'JK00': .5, 'JK010': 1, 'JK011': .5,
        'QJ0': .5, 'QJ1': .5, 'QJ01': .5, 'QJ10': 0, 'QJ11': .5, 'QJ00': .5, 'QJ010': 1, 'QJ011': .5,
        'JQ0': .5, 'JQ1': .5, 'JQ01': .5, 'JQ10': 0, 'JQ11': .5, 'JQ00': .5, 'JQ010': 1, 'JQ011': .5,
    }
```

Each key represents the current state of the game. The first card is the player card and the second card is the community card. 1 represents a bet/call and 0 represents a check/fold.

For example, ‘KQ1’ represents that the player has a K, the community card is a Q, and that player 1 bets.

Note that each state in the dictionary is mapped to some probability of that player winning in that state.

RL Algorithm: Modified Epsilon-Greedy

Every time an action has to be made by a player, we look up the probability of the player winning in the current state by either betting/calling or checking/folding.

Our optimal action is the higher of these probabilities. We choose optimal action 1-epsilon of the time, and choose the suboptimal action epsilon of the time. Note that if the probabilities of winning are the same, then we will randomly choose an action for the player.

When the player wins we add to the winning probability alpha*gain. When the loses, we update the winning probability subtract from the winning probability alpha*loss. Then we normalize.

Pseudocode

```
Let α = learning rate

Let ε = P(Suboptimal Choice)

p = Runif(0,1) # Random number between 0 and 1

if(p<ε)

        choose suboptimal choice

else if(p>ε)

        choose optimal choice

else

        Choose randomly

If choice wins:

        P(optimal choice winning) += α*(amount gained in win)

else if choice loses:

        P(optimal choice winning) -= α*(amount lost in win)

else

        P(optimal choice winning) remains same
```

Results

Player 1 First Move:

	Bet	Check
KK	0.739918	0.273473
QQ	0.696825	0.316528
JJ	0.404552	0.620174
KQ	0.769367	0.222816
QK	0.529727	0.478377
KJ	0.876921	0.118065
JK	0.884070	0.110177
QJ	1.000000	0.000000
JQ	0.947217	0.050157

Player 1 Second Move:

	Call	Fold
KK	0.501947	0.516718
QQ	0.330671	0.694659
JJ	0.160806	0.834749
KQ	0.815781	0.178787
QK	1.000000	0.000000
KJ	0.527302	0.443995
JK	1.000000	0.000000
QJ	1.000000	0.000000
JQ	1.000000	0.000000

Results

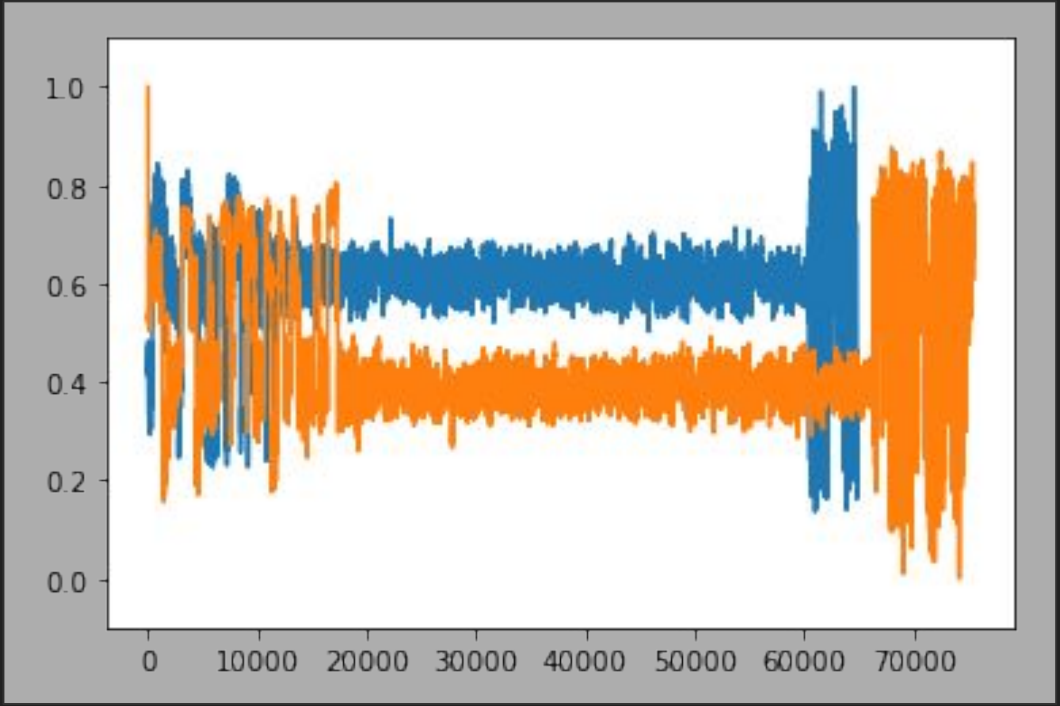
Player 2 Move If Player 1 Calls:

	Call	Fold
KK	0.497208	0.519357
QQ	0.544967	0.485169
JJ	0.688859	0.336812
KQ	0.086793	0.917304
QK	0.081736	0.922090
KJ	0.605381	0.402388
JK	0.000000	1.000000
QJ	0.000000	1.000000
JQ	0.000000	1.000000

Player 2 Move If Player 1 Checks:

	Bet	Check
KK	0.564954	0.449366
QQ	0.592357	0.399021
JJ	0.478451	0.548856
KQ	0.904764	0.099708
QK	0.000000	1.000000
KJ	0.754679	0.267175
JK	1.000000	0.000000
QJ	1.000000	0.000000
JQ	1.000000	0.000000

Changing Probabilities Over Time



On the left, we have the example of Player 1 being dealt a King and the community card also being a King.

The hand is therefore KK (best hand in the game).

As shown, the probability of winning from betting (in orange) eventually overtakes the probability of checking (in blue) over time.

This is logical, as the hand is guaranteed to win against any hand.

A better RL Algorithm for Imperfect Information Games: CRM (Counterfactual Regret Minimization)

- Used in imperfect information games
- Counterfactual: relating to what has not happened
- Counterfactual regret is the opportunity cost of not picking another action
- Goal of algorithm is to iterate to minimize the counterfactual regret, such that we pick the action with the lowest opportunity cost
- Key differences from modified epsilon-greedy:
  - While epsilon-greedy looks for the action with the highest probability of winning the hand, CRM chooses the action would minimize potential dollars lost
  - CRM is better for imperfect information games because it is optimal to be risk-averse than risk-loving

Implementation & Pseudocode

Algorithm 1 Counterfactual Regret Minimization (with chance sampling)

Initialize cumulative regret tables: $\forall I, r_I[a] \leftarrow 0$.
Initialize cumulative strategy tables: $\forall I, s_I[a] \leftarrow 0$.
Initialize initial profile: $\sigma^1(I,a) \leftarrow 1/|A(I)|$

function CFR(h, i, t, π_1, π_2):
if h is terminal **then**
  return $u_i(h)$
else if h is a chance node **then**
  Sample a single outcome $a \sim \sigma_c(h,a)$
  return CFR(ha, i, t, π_1, π_2)
end if
Let I be the information set containing h.
$v_\sigma \leftarrow 0$
$v_{\sigma_{I \to a}}[a] \leftarrow 0$ for all $a \in A(I)$
for $a \in A(I)$ **do**
  if $P(h) = 1$ **then**
    $v_{\sigma_{I \to a}}[a] \leftarrow \text{CFR}(ha,\, i,\, t,\, \sigma^t(I,a) \cdot \pi_1,\, \pi_2)$
  else if $P(h) = 2$ **then**
    $v_{\sigma_{I \to a}}[a] \leftarrow \text{CFR}(ha,\, i,\, t,\, \pi_1,\, \sigma^t(I,a) \cdot \pi_2)$
  end if
  $v_\sigma \leftarrow v_\sigma + \sigma^t(I,a) \cdot v_{\sigma_{I \to a}}[a]$
end for
if $P(h) = i$ **then**
  for $a \in A(I)$ **do**
    $r_I[a] \leftarrow r_I[a] + \pi_{-i} \cdot (v_{\sigma_{I \to a}}[a] - v_\sigma)$
    $s_I[a] \leftarrow s_I[a] + \pi_i \cdot \sigma^t(I,a)$
  end for
  $\sigma^{t+1}(I) \leftarrow$ regret-matching values computed using Equation 5 and regret table r_I
end if
return v_σ

function Solve():
for $t = \{1, 2, 3, \ldots, T\}$ **do**
  for $i \in \{1, 2\}$ **do**
    CFR($\emptyset$, i, t, 1, 1)
  end for
end for

Referenced and adapted Pseudocode from: http://modelai.gettysburg.edu/2013/cfr/cfr.pdf

This code was originally created for a different Poker variant (Kuhn Poker), but it was straightforward to change it to work for our game

Essentially, the algorithm recursively references future states from a given state (playing all possibilities of the game from the given state) and decides what the move with the least loss is.

Results

As shown in these results, the CRM-trained algorithm makes much more logical decisions.

Always choosing to bet when the hand is KK (best possible hand) is a sound decision, and being less likely to initially bet for other hands is also rational.

History		Check/Fold	Bet/Call
QJ	:	0.819	0.181
JQ	:	0.844	0.156
QQ	:	0.000	1.000
JK	:	0.826	0.174
KK	:	0.000	1.000
KQ	:	0.999	0.001
KJ	:	0.998	0.002
JJ	:	0.000	1.000
QK	:	1.000	0.000
KJ0	:	0.999	0.001
KJ1	:	0.170	0.830
KQ0	:	1.000	0.000
KQ1	:	0.177	0.823
QK0	:	0.999	0.001
QK1	:	0.150	0.850
JK0	:	0.839	0.161
JK1	:	0.999	0.001
QJ0	:	0.829	0.171
QJ1	:	0.993	0.007
JQ0	:	0.846	0.154
JQ1	:	0.996	0.004
QQ0	:	0.000	1.000
QQ1	:	0.000	1.000
KK0	:	0.000	1.000
KK1	:	0.000	1.000
JJ0	:	0.000	1.000
JJ1	:	0.000	1.000
QJ01	:	1.000	0.000
JQ01	:	0.999	0.001
QQ01	:	0.500	0.500
JK01	:	1.000	0.000
KK01	:	0.500	0.500
KQ01	:	0.320	0.680
KJ01	:	0.328	0.672
JJ01	:	0.296	0.704
QK01	:	0.310	0.690

RL agents competing under different conditions

Balances after 100000 rounds when Player 1 always goes first

```
Player1 = CRMPlayer(1000, '', 0)
Player2 = CRMPlayer(1000, '', 0)

competeCRM_v1(100000, Player1, Player2)
```

```
Player 1 balance: -774.0
Player 2 balance: 2774.0
```

Balances after 100000 rounds when both players alternate:

```
Player1 = CRMPlayer(1000, '', 0)
Player2 = CRMPlayer(1000, '', 0)

competeCRM_v2(100000, Player1, Player2)
```

```
Player 1 balance: 992.0
Player 2 balance: 1008.0
```

As shown on the right, going first in the game is a major disadvantage as player that goes second always has more information when making a choice.

When players alternate, the game is fair and win-rate is much less.