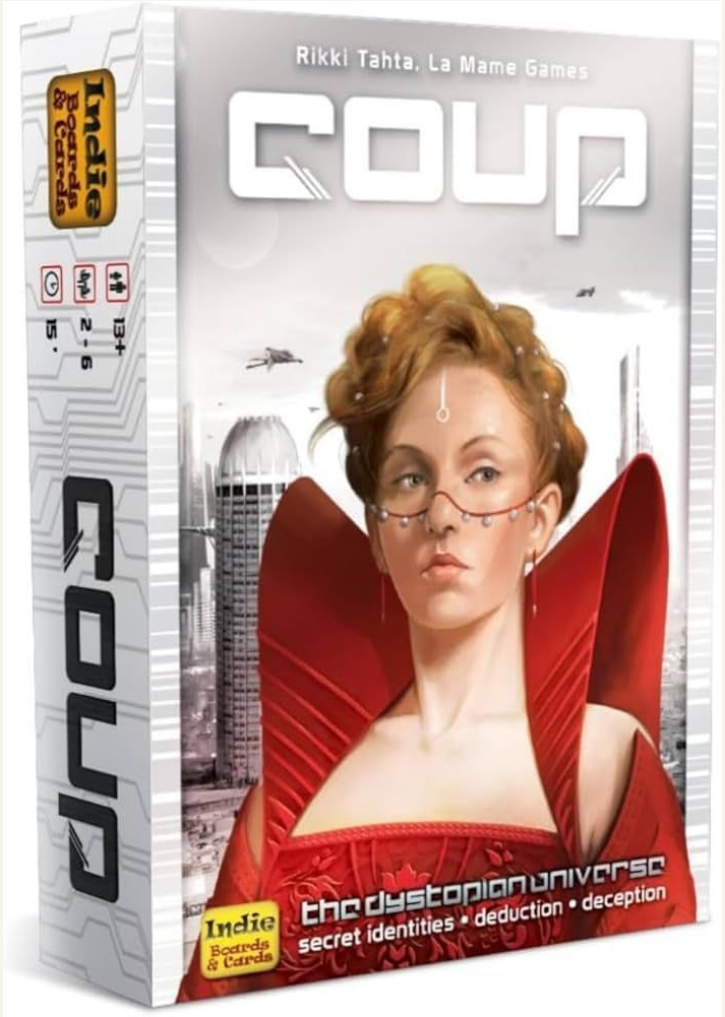
Coup

AI Bluff Bot Project

Tejaswi Tripathi
tt507@scarletmail.rutgers.edu
05 / 12 / 2025

What is Coup?

- A game with unique cards, as well as coins
- Goal: be the last one standing by removing other players from the game
- Everybody has 2 cards and loses them over the course of the game
  - If you lose both cards, you're out
- Everybody starts with 2 coins and can have 10 at maximum
  - You can take coins from the main treasury or from other players



Actions

Income

- Take 1 coin from the treasury.

Coup

- Target player loses a card.
- Costs 7 coins.

Influences

Each influence determines legal actions.

- There are 3 of each influence in the deck (15 cards total)
- Player can bluff an influence

Duke
Tax (take 3 coins from treasury).
Block foreign aid.

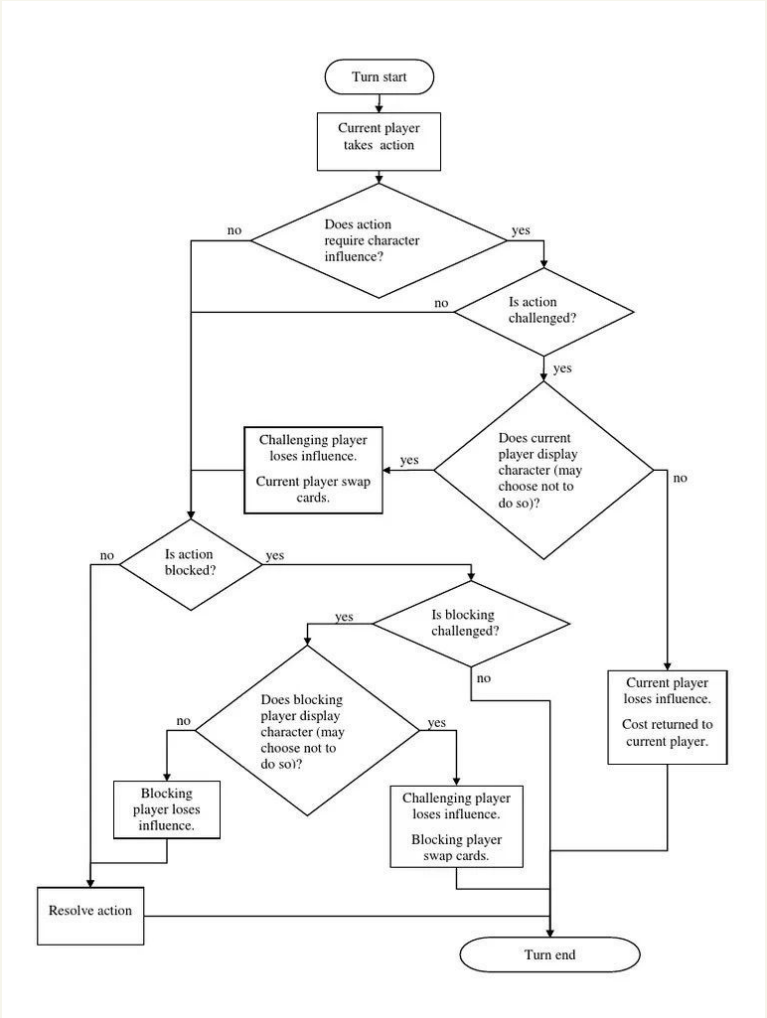
Captain
Steal 2 coins
Block Steal
Foreign Aid

Ambassador
Exchange
Foreign Aid
Block Steal

Assassin
Assassinate (costs 3 coins)
Foreign Aid

Contessa
Block assassinate
Foreign Aid

Flowchart



Prior Research

Stanford Paper

- A group of students in Stanford attempted to solve Coup using Monte Carlo Tree Search and Sparse Sampling
- They assigned discrete rewards to each possible outcome
- Assumed the bot knew the opponent(s)'s strategies
- Removed the "Ambassador" influence
- Primarily implemented forward search, then in order to beat that baseline, implemented forward search with discrete state filtering (bot knows opponent's strategy)

Ryan Campbell

- Built a Q-Learning neural network
- Bot learns the opponent's strategy throughout the game
- Made 1 network for decision-making (choosing an action)
- Created multiple agents and trained them against different types of players to assess performance under varying conditions
- Displayed frequency of each action throughout the games
- Bot does not take as input the game history, only considers current state

Prior Research (Limitations)

Stanford Paper

- Assigning discrete rewards to each possible outcome is taxing and can lead to inaccurate results
- The bot may not know the opponent(s)'s strategies
- Removed the "Ambassador" influence, which is a problem

Ryan Campbell

- Made 1 network for decision-making (choosing an action) — in reality there are more networks at play (4 specifically)
- Bot does not take as input the game history, only considers current state, which is something we can improve upon
- The bot is trained on static strategies rather than dynamic, something we can also improve upon?

Basic Goals

Q-Learning

Only 1 reward: victory / defeat
Bot learns immediate rewards of each action over time
Victory state is like a moving target that the bot is trying to hit
Uses Bellman Equation to predict rewards

Consider Game History

Bot will take as input the entire game rather than current state

4 Networks instead of 1

Action selection, block selection, challenge selection, and card selection

More Performance Metrics

Use action frequency graphs, but also include which cards the bot preferred to keep
Graphs for each network → how often blocking was used, how often challenging occurred, etc

Main Goal

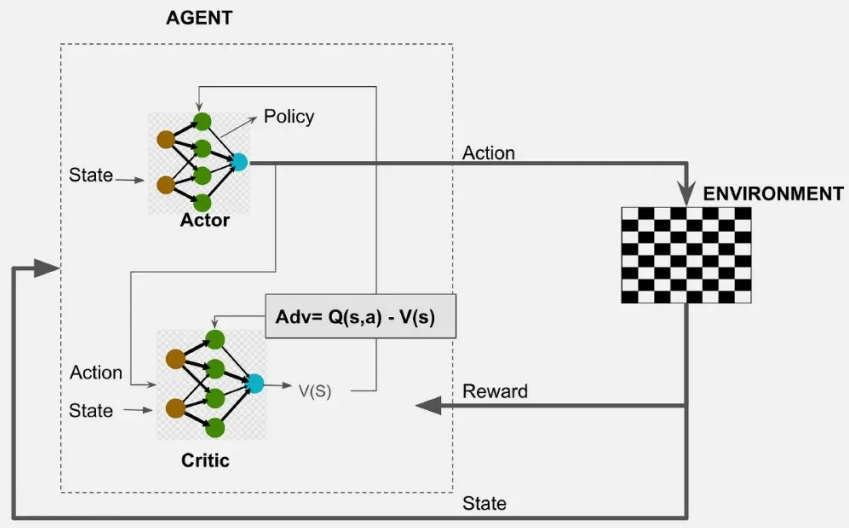
To build a bot that can **determine the optimal strategy against various opponent strategies;** essentially, solving Coup.

→ *How to do this?*

I. Opponents that change their strategies mid-game
   A. We can have the opponent stick to one strategy for the first half of the game, then randomly switch to another strategy in the second half
   B. As the bot gets more advanced, we add more players (4 maximum) and more strategies

Actor-Critic

- Main algorithm I'll be using is the Actor-Critic
- Prior research made use of a more straightforward Q-Learning network in which the bot learns the opponent's strategy over the course of the game
- This makes sense in games where each action has a more obvious immediate reward, but not in the case of Coup
- Here, since the only reward is victory, it makes more sense to have an "actor" play through an entire game or a series of games, then calculate rewards *backwards* by having a "critic" replay the games and evaluate the actor's performance
- Then, the critic relays updated parameters to the actor, and the actor's performance improves
- We can measure performance through increasing win rate



Adjustments & Updates

Neural Network

Originally, we were using a simple DQN; now, we are using an RNN to efficiently take into account game history.

Intermediate Rewards

The rewards structure is as follows:
+0.5 if opponent loses card at any point
-2.5 if agent loses card
+0.05 * number of coins gained
-0.05 * number of coins stolen from
+10.0 for victory
-10.0 for loss

Opponents

Each opponent will bluff with probability $\exp(-x/7)$, where x = the number of turns so far in the game. If they bluff, they take the maximum-utility action; the utilities have been predetermined. Otherwise, they take the maximum-utility action based on their cards.

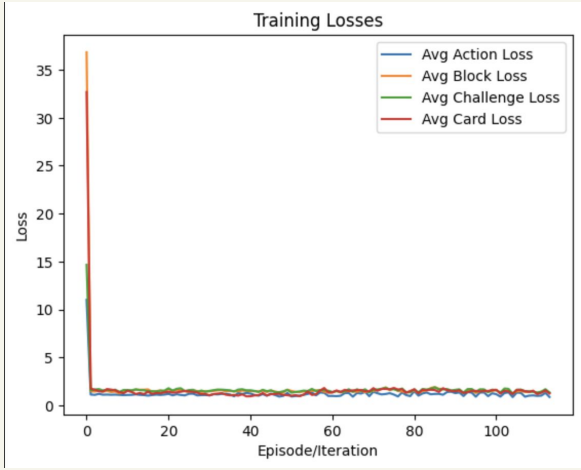
New Architecture

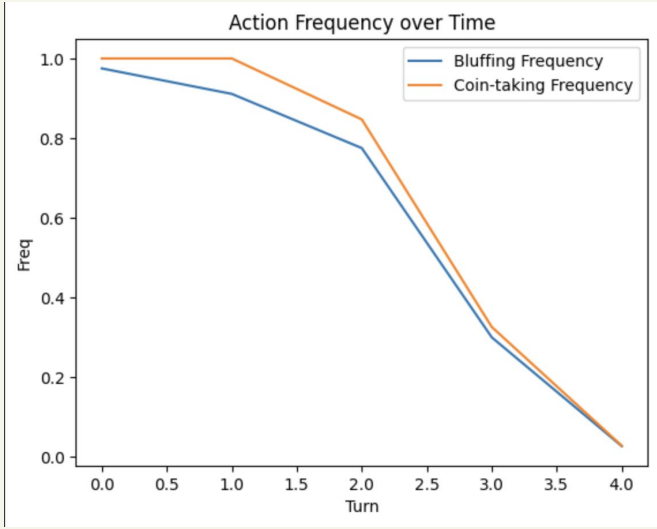
Originally, the neural network architecture followed a pyramid structure with a hidden size of 128. Now, it's a cylindrical structure with a hidden size of 2048. This leads to roughly 25 million parameters.

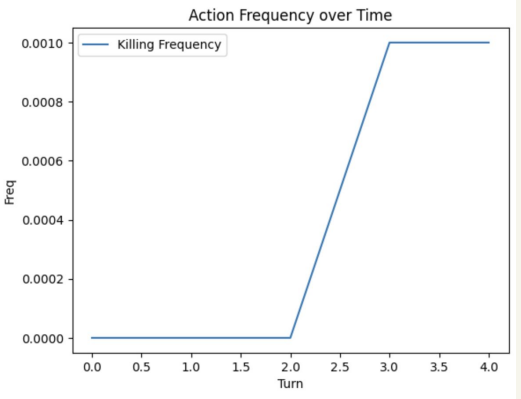
Results & Statistical Analysis

In a 4-player environment, the agent managed a win rate of **92.2%**. Analyzing the agent's playstyle, it discovered that the dominant strategy was outpacing the opponent through coin-taking, then using killing moves only at the end of the game.

Moreover, it generally mimicked the opponents' bluffing patterns by *always bluffing* at the start of the game, then relying on its own cards as the game went on.







Statistical Analysis (contd.) — Archetypes

Thief

Thief — the player who biases stealing. The RNN agent won against this archetype **99.8%** of the time.

The RNN agent was targeted **26.2%** of the time.

Greedy

Greedy — the player who biases using action “tax” (taking 3 coins). The RNN agent won against this archetype **99.9%** of the time.

RNN was targeted **18.5%** of the time.

Assassin

Assassin — the player who biases assassination. The RNN agent won against this archetype **74.3%** of the time.

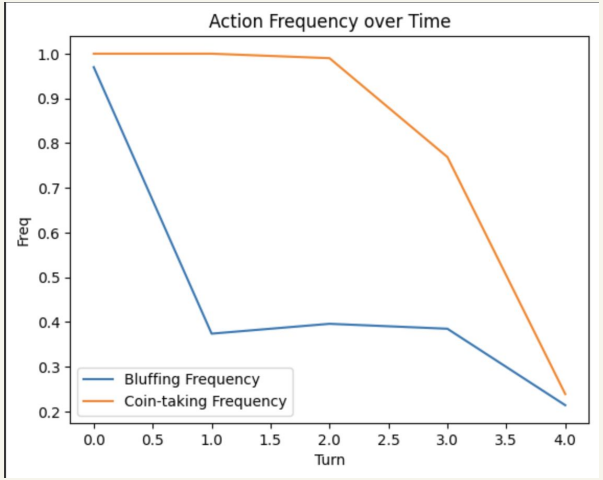
RNN was targeted **19.2%** of the time.

/ All results were in 4-player environments. */*

Dynamic Strategy

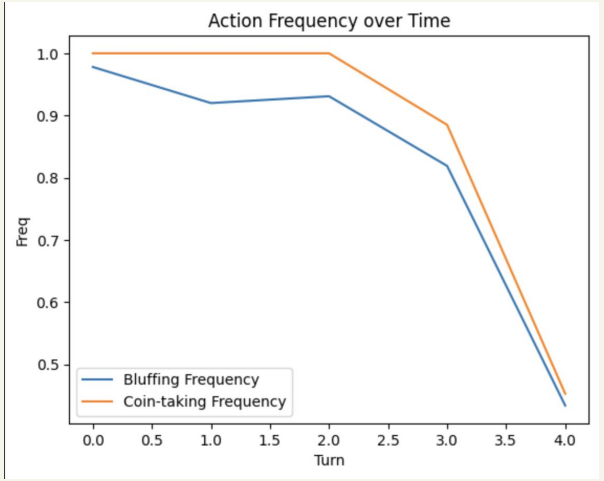
We tested the agent against an archetype that swaps strategies mid-game from the thief to the assassin. To implement this, after roughly 4-5 turns in the game, action utilities are adjusted to bias the other archetype.

Results:
The RNN agent won **92.7%** of the time. Interestingly, the agent dramatically reduced bluffing much more quickly than with static strategies:



We also tested the other way around, against one that swaps strategies mid-game from the assassin to the thief.

Results:
The RNN agent won **99.8%** of the time, still mainly using the same strategy as it was before:



Commentary & Future Work

Rationality & Generalizability

- Through generating and evaluating large quantities of simulation data, the agent was able to predict causal relationships between actions and the environment. For example, the agent knew to prioritize coin-taking and block actions to prevent losing coins.
- The bot also demonstrated generalizability when it won against various archetypes and dynamic strategies
- For future work, one can train the agent against humans or against itself. This would make the agent smarter (ideally) and further promote generalizability.

Adequacy & Conservation

- We observed AI principles of adequacy and conservation when the agent reduced bluffing as it was dealing with assassins. In such scenarios, bluffing is much more risky as it is easier to be knocked out of the game.
- It's possible that the agent would be more flexible to switch its own strategy if it were dealing with smarter opponents. For example, the opponents rarely targeted the agent. In real-life Coup, there are many player-to-player interactions that can change which player is targeted.

Commentary & Future Work (contd.)

Overall

- This was a tremendously fulfilling project that taught me the basics and techniques of reinforcement learning
- We also learned about how complex social deduction truly is
- What might help for future work is training this agent against some of the current benchmark algorithms of reinforcement learning, such as Atari