Assignment 2: Monte Carlo Tree Search Poker Bot

Due: May 7, 2025

Overview

In this assignment, you will build an AI bot that plays heads-up Texas Hold'em (2 players: your bot vs. one opponent) using Monte Carlo Tree Search (MCTS) to decide whether to fold or stay in a hand.

The bot's decision is based on a single rule:

"Does my hand have at least a 50% chance of winning?"

You will use MCTS to simulate possible opponent hands and future community cards to estimate this probability. This project teaches decision-making under uncertainty—a core AI concept—by applying MCTS to handle hidden information (the opponent's cards).

Game Rules and Simplifications

- **Players:** 2 (your bot vs. one opponent).
- **No Betting:** Decisions are limited to "fold" or "stay."
- **Card Dealing Phases:**
  - **Pre-Flop:** Each player receives 2 private cards (hole cards).
  - **Flop:** 3 community cards are revealed.
  - **Turn:** 1 additional community card is revealed.
  - **River:** 1 final community card is revealed.
- **Decision Points:** Before each phase (Pre-Flop, Pre-Turn, Pre-River), your bot has 10 seconds to decide whether to fold or stay.
- **Hidden Information:**
  - Your bot can see its own hole cards and any revealed community cards.
  - The opponent's hole cards remain hidden unless both players stay until the River.
- **Win Condition:** If neither player folds, the highest-ranking hand wins (standard Texas Hold'em hand rankings).

Bot Requirements

Your bot must:

- Implement Monte Carlo Tree Search (MCTS) to estimate winning probability at each decision point.
- For each decision:
  - **Random Rollouts:**
    - Simulate random possible opponent hole cards.
    - Simulate random future community cards.
    - Play out to showdown randomly.
  - **Selection Policy:**
    - Use UCB1 (Upper Confidence Bound 1) to guide exploration during simulations.
    - Track wins vs. losses.
    - Calculate win probability as:

$$\text{win probability} = \frac{\text{wins}}{\text{simulations}}$$

  - **Decision Rule:**
    - Stay if win probability $\geq 50\%$.
    - Fold if win probability $< 50\%$.
- Complete all simulations and decision-making within the strict 10-second limit at each decision point.

Technical Guidelines

- **Card Representation:** You may represent cards however you like (e.g., strings "AS", tuples (`'A'`, `'♠'`), integers 0–51). Choose a format that supports fast simulation and evaluation.
- **Deck Management:**
  - Implement shuffling and drawing mechanics.
  - Ensure no duplicate cards are drawn.
  - Simulate from the remaining deck correctly.
- **Hand Evaluation:**
  - Implement a function to correctly determine hand rankings based on Texas Hold'em rules.

  - Be able to rank all hands properly (e.g., royal flush > straight flush > four of a kind, etc.).

- **Programming Language:** You may use any language you prefer.
- **Optimization:**
  - Maximize the number of simulations completed in 10 seconds.
  - Write efficient code (avoid unnecessary recalculations and redundant data structures).
- **From Scratch:** You must implement all components yourself (deck, evaluation, MCTS rollout).

Clarification on MCTS Requirements

You are expected to:

- Use random rollouts to complete games after random simulations of hole cards and board cards.
- Use UCB1 during simulation selection to balance exploration vs. exploitation.
- You do **not** need to build a full deep search tree (no opponent modeling, no betting rounds).
- Just simulate possible worlds and estimate win probability.

Submission Instructions

Submit a link to a single public GitHub repository containing:

- Your bot implementation (`PokerBot.py` or equivalent).

Hints and Best Practices

- **Maximize Simulations:** More rollouts = better win probability estimates.
- **Hand Evaluation:** Test your hand evaluator separately with known cases.
- **Deck Management:** Carefully track which cards have been drawn.
- **Timing:** Monitor elapsed time carefully and stop simulation when reaching 10 seconds.
- **Debugging:** Try simple known hands first (e.g., pocket aces) to check if win probabilities look reasonable.
- **Code Efficiency:** Precompute things where possible and use fast data structures.