Texas Hold'em Roadmap

1. Idea & Problem Definition

- **Motivation**: Youtube Games

    1. Probability simulation (Monte Carlo equity).
    2. Decision-making under uncertainty.
    3. Reinforcement learning in adversarial settings.

- **Goal**: Build a **heads-up Texas Hold'em poker engine** where:

    1. The game rules and mechanics are faithfully simulated.
    2. Multiple AI agents (rule-based, heuristic, reinforcement learning) can play against each other or a human.
    3. The system can train and improve its strategy over time through **self-play and imitation learning**.
    4. Results are **logged, visualized, and analyzed** for research purposes.

---

2. Phase 1 – Core Game Engine

Idea

Start with the foundation: poker cards, hand rankings, and game flow.
The engine must be deterministic, fair, and easily extendable.

Implementation

- **Card & Deck System**

    - `Card` class (rank, suit, Unicode symbols).
    - `Deck` class with shuffling and `deal()` method.
    - `ALL_CARDS`: full 52-card deck.

- **Hand Evaluation**

    - `_best5_score()` computes the best 5-card hand from 7 cards.
    - Supports straights, flushes, full houses, etc.

    - Special handling for **Ace-low straights (A-2-3-4-5)**.

- **Game State Management**

    - `TexasHoldemGame` → manages blinds, hole cards, betting rounds, showdown.
    - `Session` → runs multiple hands in sequence.

Validation

- Unit tests in `test_poker.py` verified:

    - Deck completeness (52 unique cards).
    - Hand evaluator correctness (e.g., straight flush beats quads).
    - Stack and betting logic (bets, folds, all-ins).

Outcome

Stable, testable poker engine serving as the base for AI experimentation.

---

3. Phase 2 – Heuristic AI

Idea

Before deep learning, implement a **rule-based AI** to simulate realistic play.

Implementation

- `HeuristicAI`:

    - Computes **Monte Carlo equity** of its hand vs. random opponents using thousands of simulations.

    - Compares equity to **pot odds**:
        - Equity > odds → raise or bet.
        - Equity ≈ odds → call.
        - Equity < odds → fold (unless cheap).

    - Uses configurable thresholds (0.10 preflop margin, 0.07 postflop margin).

Validation

- Unit tests confirmed:
    - Equity of pocket aces > 70% preflop.
    - Flopped set has > 80% equity.
    - Flush draw equity reasonable (25–70%).

Outcome

Baseline AI that mimics a **beginner-to-intermediate player** with probabilistic awareness.

---

4. Phase 3 – Reinforcement Learning Agents

Idea

Enable agents to **learn strategies automatically** rather than rely only on heuristics.

Implementation

- **NumPy REINFORCE Agent** (`RLAgent`):
    - Policy represented by a lightweight MLP (`MLPPolicy`).
    - Uses REINFORCE algorithm with a moving baseline for variance reduction.
    - Learn by updating weights after each hand based on the final reward.

- **PyTorch A2C Agent** (`TorchA2CAgent`):
    - Actor-Critic model with separate **policy head** (π) and **value head** (V).
    - Supports entropy regularization (encourages exploration).
    - Stores trajectories for gradient updates.
    - Implements a tiny **replay buffer** for stability.

Validation

- Tests confirmed:
    - Forward passes generate logits and probability distributions with correct shapes.
    - Action masks applied correctly.
    - Agents produce valid decisions (fold, check, bet, raise, jam).

Outcome

Two families of RL agents:

- **NumPy REINFORCE**: fast, educational, simple.
- **Torch A2C**: scalable, more advanced RL architecture.

---

5. Phase 4 – Dataset Generation & Imitation Learning

Idea

Train policies by **imitating heuristic play** before exposing them to self-play.
This reduces random exploration and speeds up convergence.

Implementation

- `generate_heuristic_dataset`:
  - Runs hands between two heuristic AIs.
  - Captures state vectors (`encode_state`), legal action masks, and labels.
  - Maps continuous bet sizes into **discrete action categories** (check, small bet, medium bet, pot, jam).

- Training Functions:
  - `train_imitation_numpy` → trains NumPy MLP using cross-entropy.
  - `train_imitation_torch` → trains Torch model with CE loss.
  - Supports plotting train/val curves.

Validation

- Dataset unit tests:
  - State vector shape matches `FEATURE_DIM`.
  - Labels in range [0,4] (5 actions).
  - Encoded states are **consistent** given the same input.

Outcome

Trained models achieve **>60–70% validation accuracy** in choosing actions that match heuristic AI.

---

6. Phase 5 – Hyperparameter Tuning & Self-Play

Idea

Improve learning by optimizing hyperparameters and exposing RL agents to **self-play tournaments**.

Implementation

- **Grid Search**:
    - Exhaustively tested hidden sizes (64,128), learning rates (1e-3–1e-2), epochs (2–3).

- **Bayesian Optimization**:

    - Starts with random samples → refines with local Gaussian perturbations.

    - Tracks best-performing model.

- **Self-Play Training**:
    - Torch A2C vs Heuristic AI across hundreds of episodes.
    - Logs cumulative rewards, pot sizes, winners.
    - Option to save tournament logs to CSV for post-analysis.

Validation

- Training logs show:
    - Improvement in win rate over episodes.
    - Exploration (via entropy) gradually replaced by exploitation.
    - Position bias handled by rotating button seat.

Outcome

Agents evolve beyond static heuristics, discovering **bluffing and aggressive strategies** via self-play.

---

7. Phase 6 – Analytics & User Interface

Idea

Provide visibility into gameplay and AI decisions.

Implementation

- **Logging & Rewards**:
  - `log_action`: records every move (equity, pot size, draws, overcards).
  - `fill_rewards_for_hand`: backfills final rewards per player.

- **Analytics**:
  - `analytics_report`: computes VPIP, PFR, AF, and total reward.
  - `plot_basic_charts`: cumulative winnings & action histograms.

- **CLI Features**:
  - Interactive mode → human vs AI.
  - "Advice Mode" → shows equity, pot odds, suggested bet sizes.
  - Rules walkthrough for beginners.
  - Configurable session length.

Validation

- Tests ensured logging consistency and reward backfilling worked.
- CSV exports opened correctly in Excel/Google Sheets.

Outcome

End-to-end system where humans can play, observe AI advice, and analyze results.

---

8. Phase 7 – Comprehensive Testing

Idea

Guarantee correctness through structured unit testing.

Implementation

- `unittest` framework covering:

  - Core: Card, Deck, HandEvaluator.
  - AI: Heuristic decisions, RL forward passes.
  - Mechanics: Betting, action mapping.
  - Dataset: State encoding consistency.
  - Logging: Rewards filled properly.

Outcome

High confidence that the poker engine and AI agents function correctly under multiple scenarios.

---

9. Deployment & Where Implemented

- Built in **Python**, tested locally with `numpy`, `torch`, `pandas`, `matplotlib`.
- Runs as a **CLI application** (interactive play).
- Data exports allow external visualization in Tableau/Excel.
- Modular design supports further integration (e.g., research experiments or GUI expansion).

---

10. Final Outcomes & Learnings

- Delivered a **full poker AI framework** supporting:
    - Deterministic rules engine.
    - Heuristic AI, NumPy RL, Torch RL.
    - Self-play and imitation learning
    - Analytics and visualizations.
- **Key Learnings**:
    - Monte Carlo simulations are powerful but computationally expensive → batching improves efficiency.
    - Imitation learning stabilizes RL training by providing a starting policy.
    - Self-play naturally develops aggressive strategies unseen in heuristics.
    - Logging + analytics is as critical as the gameplay itself for research.