Texas Hold'em AI Agent Development Report

FAI 2025 Spring Semester Final Project - B11902080

1 Introduction

This project develops and evaluates nine distinct AI approaches for Texas Hold'em poker in a heads-up, 20-round tournament environment with 1000 initial chips and 5-second decision limits. The implemented agents range from rule-based systems with opponent modeling and probability-based methods using Monte Carlo simulation, to machine learning approaches including neural networks, MCTS, and reinforcement learning, alongside experimental behavioral strategies like all-in aggression and chaos injection. The goal is to comprehensively compare algorithmic paradigms under real-time constraints against diverse baseline opponents.

2 Agent Implementations and Technical Analysis

2.1 Probabilistic, Mathematical and Rule-Based Agents

2.1.1 Basic Rule Player (`rule_player.py`)

Approach and Strategy: implements a fundamental rule-based system with dynamic baseline opponent detection, adaptive short-stack strategies, and opponent modeling for tournament-style play. This agent addresses critical shortcomings in naive rule-based strategies through intelligent stack-size adjustments and baseline-specific counter-strategies. **Core Design:** Features opponent classification system detecting tight, loose, aggressive, passive, and balanced baseline types through action frequency analysis. Implements three distinct strategic phases: push-fold strategy for stacks <150 chips, short-stack strategy for 150-300 chips, and normal play for larger stacks. Uses dynamic threshold adjustments based on detected opponent type with baseline-specific modifications for fold thresholds, raise frequencies, and bet sizing multipliers. **Strategic Framework:** position-aware preflop ranges with mixed strategies, postflop decision trees incorporating pot odds and opponent tendencies, and intelligent bet sizing based on value/bluff situations. Game phase recognition (early/middle/late) with corresponding aggression adjustments, conservative baseline adaptation multipliers (0.8x vs tight, 1.2x vs loose), and enhanced hand strength evaluation combining rank values, positional bonuses, and suited/connected card premiums.

2.1.2 Probability-Based Agent (`pb_player.py`)

Approach and Strategy: implements a mathematical framework using Monte Carlo simulations, expected value calculations, and adaptive opponent modeling to make optimal betting decisions. This agent treats poker as a probabilistic optimization problem, employing statistical modeling to estimate win rates and calculate risk-adjusted expected values for each action. **Core Architecture:** Features Monte Carlo simulation (800 iterations) with opponent-aware hand range selection, comprehensive preflop evaluation matrix incorporating suited/connected bonuses, and dynamic risk management based on current profit and opponent strength assessment. Uses tuned hyperparameters ($\mu = 3.8$, $\sigma = 18.0$) for conservative expected profit modeling and implements three-tier opponent classification (weak/medium/strong) affecting strategy adaptation. **Mathematical Framework:** Employs advanced EV calculations combining immediate pot odds, fold equity estimation, and long-term game theory considerations. Features normal distribution modeling for game rate calculations, risk factor adjustments (0.5-1.5x) based on stack position and opponent strength, and adaptive bet sizing with conservative multipliers against strong opponents. Integrates statistical confidence intervals and variance calculations for robust decision-making under uncertainty. **Adaptive Learning:** Implements real-time opponent assessment through action frequency analysis, performance tracking with automatic strategy adjustment based on recent results, and dynamic risk tolerance modification responding to win/loss streaks.

2.1.3 Probability-Based Agent V2 (Submitted) (`pb_player_v2.py`)

textbfApproach and Strategy: implements a streamlined version of the probability-based framework, prioritizing computational efficiency and practical performance over complex modeling features. **Key Optimizations:** Increases Monte Carlo simulation count to 1500 iterations for improved accuracy while simplifying opponent hand selection. Uses refined hyperparameters ($\mu = 4.2$, $\sigma = 16.5$) representing more aggressive profit expectations. Replaces complex opponent strength modeling with basic fold tendency tracking (tight/normal/loose classification with 0.7x-1.3x fold multipliers). **Enhanced Decision Logic:** Implements three-tier raise sizing strategy (0.3-0.5x, 0.5-0.75x, 0.75-1.0x pot) based on hand strength, position bonuses for button play, and sliding window performance adaptation over 5-game periods. Eliminates opponent strength penalties that caused overly conservative play in the original version.

2.1.4 Monte Carlo Rule Player (`monte_rule_player.py`)

Approach and Strategy: combines rule-based decision-making with optimized Monte Carlo simulations and advanced opponent modeling to create an adaptive tournament-focused AI. This agent integrates mathematical precision with exploitative strategy adjustments based on opponent pattern recognition. **Core Architecture:** multi-layered opponent modeling tracking 15+ behavioral metrics including VPIP, aggression factors, bet sizing patterns by street, and exploitable tells. Implements dynamic all-in threshold adjustment based on hand classification (premium/strong/good/marginal/weak), game phase recognition, and stack pressure calculations. Uses optimized Monte

Carlo equity calculations (40-80 iterations) with timeout protection and intelligent strength blending combining rule-based evaluation, Monte Carlo equity, and contextual adjustments. **Strategic Framework:** Employs pattern classification system detecting six opponent archetypes (maniac, tight-aggressive, calling station, loose-aggressive, nit, balanced) with confidence-weighted exploitative adjustments. Features dynamic strategy modification based on recent results, desperation factors, and blind pressure calculations. Implements position-aware play with stack-size dependent threshold adjustments, progressive aggression scaling with tournament progression, and meta-game adaptation through sliding window analysis of opponent tendencies.

2.2 Machine Learning Agents

2.2.1 Neural Network Learning AI (`nn_player.py`)

Approach and Strategy: implements a simplified feed-forward neural network with online learning capabilities, combining engine-based hand evaluation, rule-based logic, and adaptive neural network predictions. This agent focuses on practical learning with lightweight architecture. **Core Architecture:** SimpleNeuralNetwork with 15-dimensional feature encoding (hole cards, community cards, game state, pot odds, opponent modeling) processed through a single hidden layer (15→20→1) using ReLU activation and sigmoid output. Features Adam-like optimization with clipped gradients. **Learning System:** Implements experience replay with deque-based storage (150-hand limit), blends three evaluation sources with dynamic confidence weighting: 60% engine evaluation, 30% neural network prediction, 10% rule-based fallback when engine available. Uses round-based saving strategy with training triggers every 8 hands and minimum 15-hand datasets for network updates. **Decision Logic:** Conservative decision-making with pot odds consideration, big-bet handling thresholds (70% stack), and opponent modeling through aggression factors and VPIP tracking. Incorporates position awareness, stack-size adjustments, and adaptive bluffing based on opponent tendencies and board texture analysis.

2.2.2 Monte Carlo Tree Search Agent (`mcts_player.py`)

Approach and Strategy: implements a neural network-guided Monte Carlo Tree Search for poker decision-making, combining deep learning policy/value estimation with systematic tree exploration. This agent uses a dual-head neural network (policy + value) with Adam optimization and comprehensive state encoding. **Core Architecture:** PolicyValueNetwork with 50-dimensional input encoding game state, hole cards, community cards, pot odds, opponent modeling statistics, and board texture features. The network outputs both action probabilities (3-action policy head) and position evaluation (value head) through shared hidden layers (50→128→64→3+1). **MCTS Implementation:** UCB-based node selection with prior probabilities, dynamic expansion using neural network guidance, and temperature-controlled action selection. Uses 40 simulations per decision with 1.5-second time limits and c_puct=1.0 for exploration-exploitation balance. **Learning System:** Comprehensive opponent modeling tracking VPIP, aggression factors, and showdown statistics. Implements experience replay with deque-based training data storage (200-sample limit), game-based saving strategy for persistence, and conservative early-game strategy during first 50 games. Features adaptive temperature scheduling and optimized memory management for computational efficiency.

2.2.3 Reinforcement Learning Agent (`rl_player.py`)

Approach and Strategy: implements a Deep Q-Network (DQN) with experience replay and target networks for poker decision-making, treating each betting decision as a reinforcement learning problem with state-action-reward optimization. This agent uses temporal difference learning and neural function approximation. **Core Architecture:** Deep Q-Network with 4-layer architecture (50→128→64→32→3) using Leaky ReLU activation and Adam optimization, paired with separate target network for stable Q-learning. Features 50-dimensional state encoding including hand evaluator integration, board texture analysis, opponent modeling, and game phase recognition. **Learning System:** Implements epsilon-greedy exploration (ϵ =0.15, decay=0.996) with experience replay buffer (3000 capacity), batch training (size 24) every 8 steps, and target network updates every 50 steps. Uses gamma=0.95 discount factor and sophisticated reward shaping combining intermediate action rewards, final hand outcomes, and stack change normalization. **Decision Framework:** Intelligent raise sizing based on pot odds and game phase, masked Q-value selection for valid actions only, and adaptive exploration rates increasing in late-game scenarios. Incorporates opponent action tracking, position-aware play, and game-based saving strategy.

2.3 Experimental and Behavioral Agents

2.3.1 All-In Monster (`all_in_player.py`)

Approach and Strategy: implements an extreme pressure strategy by going all-in every single hand, regardless of card strength or game situation. This agent serves as a pure game theory experiment exploring maximum aggression and opponent fold frequency exploitation. **Core Design:** Always taunts and attempts to raise the maximum possible

amount (entire stack) when raising is available, otherwise calls any amount or fold. The strategy is completely deterministic, no hand evaluation or situational analysis occurs.

2.3.2 Aggressive Player (`impulsive_player.py`)

Approach and Strategy: implements a psychologically-inspired AI with dynamic aggression levels, tilt mechanisms, and opponent modeling. This agent explores behavioral poker strategies by simulating adaptive pressure tactics. **Core Design:** Base aggression level of 0.8 with tilt factor that increases after losses, position-aware play favoring big blind aggression, and stack-size dependent strategy adjustments. Hand evaluation combines actual engine results with aggressive bias toward drawing hands (+0.3 multiplier), suited connectors (+0.15 bonus), and any ace (+0.1 bonus). **Decision Matrix:** Uses lowered thresholds for action, strong hands (>0.65) always raise for value, medium hands (>0.45) frequently pressure with raises, and even weak hands (>0.25) call cheaply or bluff against tight opponents. Incorporates 25% base bluff frequency that increases with tilt and opponent fold rate. **Opponent Modeling:** Tracks opponent fold rates and action frequencies to adjust bluffing strategy, with automatic pressure increases against passive players and late-game aggression to exploit short stacks.

2.3.3 Chaos Poker AI (`semi_random_player.py`)

Approach and Strategy: deliberately incorporates randomness, superstitions, and behavioral patterns to create an unpredictable opponent. This experimental agent explores whether semi-random play can be effective against pattern-matching opponents. **Core Design:** Base chaos level of 0.7 with dynamic decision patterns that change every 3-8 rounds, mood swing system affecting hand strength evaluation, and superstition-based modifiers. The agent cycles through six behavioral patterns: `aggressive_spree` (low fold threshold 0.3), `tight_phase` (premium hands only), `bluff_heavy` (60% bluff rate), `call_station`, `fold_happy`, and `balanced`. **Superstition and Mood Systems:** Random superstitions modify hand evaluation - red cards provide +0.2 bonus, suited hands receive -0.15 penalty if "cursed", and lucky numbers add +0.15. Five psychological states (Euphoric +0.2 to Paranoid -0.2) change every 2-6 rounds, affecting decision thresholds. **Chaos Override:** After pattern-based decisions, 20-30% of actions randomly ignore the underlying strategy with modified action types and sizing, creating highly unpredictable play while maintaining baseline poker logic.

3 Performance Analysis and Results

With the project codebase updated close to the deadline, I have limited time to retrain my machine learning agents and rerun evaluations. As the models aren't fully trained, their statistics aren't very meaningful, so I won't include exact figures below. However, if the TAs wish to see prove that I've implemented the machine learning agents, you may check out this GitHub repository: https://github.com/HowardHsuuu/texas-holdem-agents

3.1 Baseline Competition Results

Table 1 summarizes the performance of each agent against baseline opponents 0-7, showing win rates and average profit over 20 games. (The experimental agents' performance reveals more about the baseline opponents than the agents themselves. Their statistics suggest the baselines tend to favor aggressive play over conservative strategies. This is backed up by a more conservative rule-based variant that performed poorly against the first few baselines. Note: such variants were just a by-product of my exploration and are therefore excluded from the report.)

Table 1: Agent Performance Against Baseline Opponents (20 games each, win rate and avg profit)

Agent	B0	B1	B2	B3	B4	B5	B6	B7
Rule-Based	**85%** (+690.4)	80% (+536.2)	65% (+469.5)	80% (+321.7)	20% (-483.5)	30% (-113.1)	35% (+169.9)	30% (-90.7)
Probability(V2)	**85%** (+253.7)	**100%** (+247.8)	**95%** (+122.0)	**100%** (+126.0)	**70%** (+264.1)	**50%** (-187.2)	35% (+41.5)	**45%** (+12.6)
Monte Rule	65% (+300.0)	85% (+700.0)	35% (+183.7)	90% (+594.9)	25% (-160.5)	30% (+7.2)	35% (+152.2)	25% (-185.1)
All-In	50% (+0.0)	50% (+0.0)	50% (+0.0)	35% (-163.3)	35% (-300.0)	40% (-102.0)	**40%** (-111.5)	30% (-265.2)
Aggressive	60% (+234.0)	90% (+781.4)	80% (+747.0)	**100%** (+667.4)	50% (+199.0)	30% (+168.2)	25% (-35.7)	40% (-91.2)
Chaos	60% (+219.5)	75% (+344.8)	65% (+328.3)	70% (+249.7)	20% (-58.4)	20% (-37.1)	15% (-151.2)	35% (+58.4)

3.2 Performance Analysis and Key Insights

The probability-based agent achieved superior performance (and thus **selected**), demonstrating the effectiveness of Monte Carlo simulation combined with mathematical expected value calculations. The Monte Carlo Rule player also excelled, confirming that hybrid approaches combining rule-based logic with probabilistic evaluation are highly effective. Baselines 4-7 proved significantly more challenging than baselines 0-3 across all agents, suggesting the higher-numbered baselines employ sophisticated counter-strategies exploiting common poker AI patterns. Results clearly favor mathematical approaches over behavioral strategies, with probability-based and Monte Carlo methods consistently outperforming aggression-based and random strategies. Notably, statistical robustness proved more effective than complex human-derived heuristics. Despite extensive rule-based systems with detailed opponent modeling, mathematical approaches using Monte Carlo simulation achieved superior consistency, indicating probabilistic modeling captures game complexities more effectively than crafted decision trees. Machine learning agents faced significant challenges from limited training data, sparse reward signals, and computational constraints. Even after retraining the MCTS agent for 200 games against baseline 5 (the only slightly meaningful statistic I was able to obtain after the update), it achieved only about 56% win rate with unstable performance against other baselines, especially 4, 6, and 7. This reinforces that under limited time and training data, like the 3-day window we have after the update, statistical approaches demonstrate superior generalizability and robustness compared to ML ones.

3.3 Challenges and Future Directions

The recent codebase update significantly slowed evaluation runtime, limiting effective iteration on both non-ML and ML approaches. Most methods likely haven't reached maximum potential due to insufficient tuning time. The 5-second decision limit severely restricts Monte Carlo simulation effectiveness, allowing only 40-80 iterations when thousands would be beneficial. Future work could explore Counterfactual Regret Minimization (CFR) for game theory optimal play, deep reinforcement learning with extended training periods, multi-agent environments, and hybrid architectures combining neural networks with classical game theory. Offline training against diverse opponents and self-play environments could address training data scarcity. This project demonstrates that machine learning isn't always optimal for every problem. Well-designed non-ML methods with strong domain knowledge achieve excellent performance in constrained environments. Mathematical approaches highlight that computational efficiency often trumps theoretical optimality—simple, fast algorithms making good decisions quickly outperform complex algorithms struggling under time constraints. Different strategies excel against different opponent types, reinforcing that adaptability and opponent modeling are crucial for competitive performance.

4 Conclusion and Discussion

This evaluation reveals that algorithmic effectiveness depends heavily on environmental constraints and characteristics. The clear superiority of mathematical approaches demonstrates that computational efficiency sometimes matters more than theoretical sophistication in real-world applications. The performance gap between baselines 0-3 and 4-7 highlights the importance of adaptive algorithmic design for different opponent strengths. While machine learning approaches struggled in this constrained environment, this emphasizes matching algorithmic complexity to available resources rather than invalidating ML methods entirely. In an era where practitioners increasingly default to machine learning for every problem, this work underscores the importance of exploring domain-specific solutions that may be more resource-efficient while achieving equal or even superior performance. Future development should focus on hybrid architectures that combine mathematical efficiency with machine learning adaptability, while accounting for real-world constraints from the design phase rather than treating them as afterthoughts.

5 Project Feedback

This was an interesting project and I genuinely learned a lot while working on it. However, I have to be honest that my enthusiasm was worn out by the chaotic changes near the deadline. The updated codebase, released just 3 days before the original due date, drastically increased training time and runtime, making it infeasible for me to continue with my ML-based agents, which I had spent most of my effort developing. I had to fall back to an less optimized probability-based agent, and didn't even have time to rerun evaluation scripts or properly tune parameters for this sudden switch. While I appreciate the update clarifying that reports don't need to be revised, the report only accounts for part of the grade. Other parts still rely on agent performance, which was heavily affected by the late-breaking changes. I understand that being a TA is tough, but I do believe this situation was mishandled. Ignoring student concerns, then pushing major changes and rule updates right before and even on the deadline, created a deeply unfair environment, especially for students who started early and had planned their time. Frankly, despite the initial excitement, this project ended up feeling like a major disappointment and a waste of time.