Alexander Scher
CSCI 4931
12/2/25

PLO8 Project Report

This project has been both challenging and foundational, and it is still very much in its early stages. The original goal was to build an artificial neural network (ANN) capable of playing poker. However, in order to do this properly, I first needed to design and implement a complete poker engine from scratch. Approximately 80% of the effort invested in this project so far has gone into building that engine, with the remaining 20% focused on developing and experimenting with the ANN itself—primarily struggling with optimization strategies and training speed.

A significant early mistake was attempting to implement every component from the ground up, particularly the poker hand evaluator. I initially believed I could outperform existing solutions by encoding cards as prime numbers, multiplying them to represent hands, and then deriving hand rankings through logic that factored these values to detect straights, flushes, pairs, and so on. This assumption proved incorrect. State-of-the-art poker hand evaluators use perfect hash functions that allow constant-time lookup of hand rankings for all possible card combinations. This approach completely outclassed my custom implementation in both speed and simplicity. As a result, I ultimately adopted existing modules from the pypokerengine Python library and refactored my engine to integrate with them.

To support gameplay and interaction, I implemented game rendering using the pygame library, resulting in a fully Python-based stack from top to bottom. The renderer displays the current game state and collects player actions. To reduce the action space and make ANN training more tractable, the game restricts players to five possible actions: check/fold, minimum bet/call, half-pot bet, three-quarter-pot bet, and full-pot bet. This design choice significantly reduces complexity while encouraging betting-frequency convergence that loosely aligns with optimal strategies observed in more extensively studied games such as Texas Hold’em.

In retrospect, I spent far too much time polishing the user interface. During self-play training, the game progresses far too quickly for the UI to provide meaningful insight, making it ineffective for observing learning behavior. Consequently, the current training process runs entirely without rendering. Additionally, when attempting to analyze completed hands, the renderer offers little value; what is truly needed is a dedicated hand-replay system. Developing such a replayer is now a high-priority task. That said, the renderer does serve a useful purpose by allowing a human player to play directly against the ANN.

The game engine itself supports most core mechanics of Pot-Limit Omaha with up to nine players, with several important constraints: there are no antes, only fixed ½ big blind and 1 big blind structures are supported, and full betting logic is implemented only for two-player games. Once more than two players are involved, side pots become necessary when players go all-in with differing stack sizes. In the worst case, this can result in up to eight separate pots (one main pot and seven side pots), each with different eligible players and each requiring separate high-hand and low-hand evaluations. The resulting payout logic becomes extremely complex. To keep the project scope manageable, gameplay is therefore limited to two players.

Turning to the ANN itself, the network architecture was largely a best-guess prototype. The input layer encodes a substantial amount of game-state information and consists of 114 input nodes. This is followed by hidden layers of 256, 128, and 64 neurons, and a 5-node output layer corresponding to the available player actions. Ideally, the training process would incorporate expected-value calculations, positional opening ranges, betting ranges, and more sophisticated reinforcement-learning techniques. Due to time constraints, however, I settled on a very simple reward-based approach in which the network is rewarded or penalized proportionally based on the number of chips won or lost in a hand.

This learning strategy converges extremely slowly and is insufficient for producing a competitive agent. It is already clear that this approach will be discarded in the next iteration of the project in favor of a more principled reinforcement-learning framework.There is considerably more that could be discussed, but this report captures the core challenges, lessons learned, and current state of the project. Thank you for taking the time to read it.