**Document of Opponent Modeling**

**Name:** Ho Phi Long

1. **Constraints, problem properties and some assumptions of necessary conditions**
   1. *Constraints*
      1. Very limited computation power (0.2s)
      2. Play large number of games (500) with large number of rounds (1000)
   2. Problem properties
      1. Since game type is Limit Poker (can only raise a fix number of time and amount) 🡪 Player will have tendency to play loose since the amount of money lost will not be as valuable as information you gain about opponents. This is also demonstrated in real life recorded data of human players as well 🡪 We can have higher number of opportunities to observe opponents, especially in the showdown
      2. As the game purpose is to win in the long run 🡪 sacrifice some starting rounds to gain data is more valuable than saving the money by folding
      3. With limited computation power in the problem, opponent also less like to switch strategy in short amount of time (unless his hard code many different strategies).
      4. Need to adapt to different strategy for different opponent
      5. With limited computation power, optimistic e*stimation gains better result than pessimistic estimation*
   3. *Some assumptions*
      1. This modeling assume that opponent will play consistently in a required amount of time (currently set it to 500)
      2. Maximum 4 time raises in the whole game and max 2 time raises in and street 🡪 The number of all possible combinations of action sequence is small enough to construct a tree contain all possible action sequences (around 6000 nodes)
2. **Modeling scheme**
   1. Abstract:
      1. To adapt to different strategies, our agent
      2. Search using minimax idea on action sequence tree which
         1. Have 3 kind of nodes
            1. Opponent node: EV(O) = sum PiEVi
            2. Player node: EV(P) = max/mix EV(f), EV(c), EV(r)
            3. Leaf node: EV(f) = Pwin\*Pot – Cost of using reinforcement learning with more factors to be considered
   2. Basic implementation
      1. At the start of the game, initialize an all possible action sequence tree start at flop round
         1. Noted: since after the flop, action can be considered symmetric 🡪 we can abstraction that opponent sequence and our sequence is the same, they used the same data
      2. Store a history cell divided into 10 range to show hand strength opponent used when reach this cell
      3. Record opponent action and player action throughout the game
      4. If showdown is reached
         1. Record opponent exact hands
         2. Calculate opponent hand strength (HS) in each round
         3. Update history cell in the correct action sequence at correct hand strength cell at each street/river by tracing back the recorded history
         4. In each round, with the HS computed, record the number of calls, raises if raise < 2, or just the raise if raise = 2 or raise = 4 before the end of that round
         5. Add the number of calls/raises of that HS to calls/raises HS array
         6. Compute medium, variance and standard deviation of the 2 arrays
         7. Using Chebyshev Inequality, we can find the range of HS that opponent has at least more than 90% of performing that action (90% confidence)
         8. Update our prediction model of range opponent perform certain action with 90% confidence (mix strategy)
      5. When there are enough data points (around 5 - 10 data points), switch from base player to using opponent modeling
      6. When we reach a history cell in the search of action sequence tree
         1. Look at history to calculate the percentage of opponent in certain continuous HS range from opponent model we currently have that has:
            1. Boundary by 2 cells that just over the range
            2. Ex: we have call: 0.4 - 0.7, then we calculate the probability of having 0.3 – 0.8 range
         2. Calculate Probability opponent perform that action: Pr(action) = Pr(range) \* confidence level
         3. Feed to evaluation function, push up the action tree and using max/mix (EV(f), EV(c), EV(r)) to convert to action
      7. Evaluation is trained using reinforcement learning that require these parameter
         1. Current opponent model
         2. Current board information/Board potential
         3. Probability of opponent having certain HS range at that point
         4. Sequences of action for both player
         5. Pot value and Cost
         6. Noted: In the worst case, we can use MonteCarlo simulation at that point and make EV = P(win)\*Pot – Cost
   3. Experimental result:
      1. Testing methods

The base line is a pure strategy player which is finely tuned to consistently beat others simple strategies and some pure/mix strategies player, including always raise players, always call players,

* 1. Limitations and optimizations
     1. Not enough data points
        1. The tree is design so that the higher level of the tree, the more data we have 🡪 The most efficient phase to use is turn and flop, until we have more data when the number of rounds reaches 200+, and it better to search less level (due to action time constraint of 0.2s)
        2. Since we have limited computation, that also applied to other agents and they will less likely to change strategy quickly 🡪 there will be more round that we could collect data from
        3. As we play 500 game with 1000 rounds each 🡪 If data can be store to a file, we are likely to have a complete view about opponent after 50 – 70 games
        4. Since this is Limit Poker, player will have tendency to play loosely 🡪 more data
        5. If storing value is not possible, 2nd abstraction layer is the solution to deal with not enough data points
           1. Abstraction 2: every opponent sequence has the same number of raise and call, and every player sequence has the same number of raise and call are considered the same sequence

We trade off accuracy for much more data, and use a linear summation of (1 – mNumber of data points) \* Ai to determine the value (Ex: m = 0.95)

All abstractions contribute to the data value

* + 1. Data inconsistency due to opponent switching strategy
       1. Solution 1: We can introduce a forget factor h that (1 – hNumber of new data from that point) to the record value, make it less important in the computation over time (if storing is not allowed)
       2. Solution 2: We can try sampling for period of 50 rounds, and store back up before continuing to sample next 50 rounds.
          1. If old data and new data have the same pattern, add them up
          2. Elsestore data until we seeing the pattern come back

This strategy try to model opponent routing a number of strategy, which can benefit next game since agent is unchanged