Pokerbots Strategy Report

Team Name: EctomynX
Members: Brian Xie, Tiancheng Qin, Allen Cheng

Abstract: We first reduced the complexity of the game tree by encapsulating certain aspects of the game state into parameters. We used Java to implement this modified game tree, along with a multidimensional array of weights of taking each set of actions. We also constructed an engine that would initialize two players and play them against each other. This training would refine our weights and optimize each player's game play. We used the fixed-strategy iteration CFR algorithm to update weights. In this algorithm, we play some number of iterations/hands with a fixed strategy. At the end of the cycle, we update weights according to the accumulated profits of taking the corresponding line. After running many iterations, our weights converge closer to optimal play.

Table of Contents:

0. <u>Initial Impressions</u>

All three of us are proficient HUNL players, with hundreds of hours of experience among us. As such, we first played several hours of HUNL with discards to get a general idea of optimal strategy. We primarily tried to understand how the discard mechanism would impact certain aspects of gameplay and adjusted the regular HUNL strategy accordingly. Below are some of our insights:

a. General Strategy

Equities in this game are far closer together with the discard mechanism. Each discard opportunity functions somewhat like an additional street, with more opportunity to improve one's hand. As such, a player's initial holdings are not as relevant. Average hand strength will also be significantly higher than in HUNL. Discarding also causes several other major changes:

- **Pocket pairs drastically decrease in value.** Pocket pairs are strong hands in HUNL. However, they are unable to take advantage of the discard mechanism because doing so would break up the pair. As a result, pocket pairs have very static hand strength, whereas unpaired holdings have more opportunities to improve.
  - Playing low to middle pocket pairs becomes almost completely a set-mining proposition because without a set, these pairs typically make weak pairs that cannot stand up to pressure. In fact, we will see that 22 has a measly ~34% equity against any two random cards!
  - Big pocket pairs should be played very aggressively preflop. These pairs lose a lot of their value postflop because of the higher and more dynamic hand strengths, so it's crucial to put in as many chips into the pot as possible with a guaranteed equity edge, especially with such a static hand.

- **Slowplay less often**. HUNL with discards is a bit like Omaha. The discard mechanism drastically increases the chance that your opponent's hand strength improves. As such, it's crucial to extract as much value as possible early on and prevent your opponent from freely actualizing his/her equity.

- **Suitedness and connectivity is not as important preflop**. Flopping a flush/straight draw is great. But in order to take advantage of it, neither card associated with the draw can be discarded without breaking up the draw. While suited/connected cards are still slightly more valuable than their unsuited or disconnected counterparts, the difference is not as great. Having high cards is more important than connectivity; for example, A3 is significantly stronger than 89 in this variant although their values both sum to 17, because A3 can discard its 3 and on the average improve far more.

- **Bet sizing should be larger than in HUNL**. Equities are closer, so in order to force opponents to pay an appropriate price to draw out bet sizing must be higher.

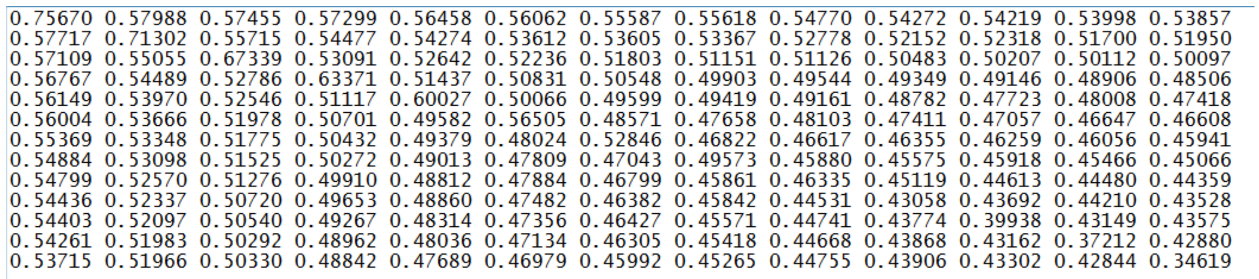
Due to discards, the two hand strengths that become significantly more likely are one pair and two pair. This means that hands that can make hands that can beat two pair have very good implied odds (e.g. higher two pairs, three of a kind, straights/flushes, etc). This complicates strategy a lot because while hands like small pairs and suited connectors decrease in relative value, they now have even better implied odds. As such, these hands can still be played, but with the goal of mining for a strong hand only.

b. Equity

In order to gain a better understanding of how individual hands perform, we developed a program that would compute:

- The equity of two given hands against one another, and
- The equity of a given hand against a random hand.

We did this by using our equity.java program and our discard algorithm, which we will discuss later. We simulated the results of 2 billion random hands, and determined each hand's winning probability overall. Here are our results:



```
0.75670 0.57988 0.57455 0.57299 0.56458 0.56062 0.55587 0.55618 0.54770 0.54272 0.54219 0.53998 0.53857
0.57717 0.71302 0.55715 0.54477 0.54274 0.53612 0.53605 0.53367 0.52778 0.52152 0.52318 0.51700 0.51950
0.57109 0.55055 0.67339 0.53091 0.52642 0.52236 0.51803 0.51151 0.51126 0.50483 0.50207 0.50112 0.50097
0.56767 0.54489 0.52786 0.63371 0.51437 0.50831 0.50548 0.49903 0.49544 0.49349 0.49146 0.48906 0.48506
0.56149 0.53970 0.52546 0.51117 0.60027 0.50066 0.49599 0.49419 0.49161 0.48782 0.47723 0.48008 0.47418
0.56004 0.53666 0.51978 0.50701 0.49582 0.56505 0.48571 0.47658 0.48103 0.47411 0.47057 0.46647 0.46608
0.55369 0.53348 0.51775 0.50432 0.49379 0.48024 0.52846 0.46822 0.46617 0.46355 0.46259 0.46056 0.45941
0.54884 0.53098 0.51525 0.50272 0.49013 0.47809 0.47043 0.49573 0.45880 0.45575 0.45918 0.45466 0.45066
0.54799 0.52570 0.51276 0.49910 0.48812 0.47884 0.46799 0.45861 0.46335 0.45119 0.44613 0.44480 0.44359
0.54436 0.52337 0.50720 0.49653 0.48860 0.47482 0.46382 0.45842 0.44531 0.43058 0.43692 0.44210 0.43528
0.54403 0.52097 0.50540 0.49267 0.48314 0.47356 0.46427 0.45571 0.44741 0.43774 0.39938 0.43149 0.43575
0.54261 0.51983 0.50292 0.48962 0.48036 0.47134 0.46305 0.45418 0.44668 0.43868 0.43162 0.37212 0.42880
0.53715 0.51966 0.50330 0.48842 0.47689 0.46979 0.45992 0.45265 0.44755 0.43906 0.43302 0.42844 0.34619
```

The table is read as follows:

- The rows correspond to the first card. The first row corresponds to A, second K, third Q, all the way down to 2.
- The columns correspond to the second card, in the same way.
- Entries in the upper right correspond to suited cards, entries in the lower left correspond to unsuited cards.

For example, row 2 column 3 (0.55715) corresponds to KQs, and row 13 column 12 (0.42844) corresponds to 32o. As expected, most unpaired cards have very close equity, and the equity of pairs drops off steeply as their values decrease.

We also simulated some generalized situations:

1. Overpair vs Underpair (overpair has ~81% equity)
2. Overpair vs Undercards:
    a. AA vs KQ – AA wins 77%
    b. TT vs 86 – TT wins 68%
    c. 66 vs 42 – 66 wins 60%
3. Underpair vs Overcards:
    a. QQ vs AK – QQ wins 53%
    b. 88 vs QT – 88 wins 48%
    c. 44 vs 86 – 44 wins 40%
4. Pair vs Top/Bottom:
    a. KK vs AQ – KK wins 61%

   b. 99 vs T8 – 99 wins 53%
   c. 44 vs 53 – 44 wins 45%

5. Top/Top vs Bottom/Bottom:
   a. AK vs 23 – AK wins 61%
   b. A9 vs 84 – A9 wins 58%
   c. 98 vs 76 – 98 wins 54%
6. Top/Bottom vs Middle/Middle
   a. T7 vs 98 – T7 wins 51%

As we can see, small pocket pairs have low relative value, and the equity gap between overcards/undercards is small.

I. <u>Gameplay Overview</u>

HUNL is far too complex to play if we consider every possible game state separately. As such, we first simplified the game by encapsulating certain aspects of a hand into parameters, and making decisions based on the values of our parameters. These are the parameters that we constructed:

a. Game State Parameters

1. Betting History

Instead of considering every bet size separately, we developed a system of *bet levels*. These bet levels are integers ranging from 0 to 5. Our bet levels are defined in a way such the thresholds approximately distinguish *n*-bets and *n+1*-bets, where *n* is some nonnegative integer. For example, the threshold for bet level 1 approximately distinguishes a bet from a raise, according to typical bet sizes. We expect many bots to use strange/unconventional bet sizing. This model allows us to react appropriately to unusually large bet sizes (rather than just viewing it as whether or not a bet was made) while keeping the number of parameters very low.

For example, suppose the threshold for bet level 1 is 10 chips. Then, a bet of 8 would be interpreted as a level 1 bet, but a bet of 15 would be interpreted as a level 2 bet. Raises are interpreted by the amount raised to.

When we obtain our strategy, we keep track of all previous bet history. We will discuss the specifics behind preflop and postflop and their corresponding thresholds later.

2. Absolute Hand Strength

We categorize each hand into 14 different bins based on its absolute hand strength postflop:
- Nothing
- Weak Draws (gutshot straight draw, or 2 overcards)
- Strong Draws (flush draws, 2-way straight draws, or 2 overcards + gutshot)
- Weak pair (below 2nd pair)
- Middle pair (2nd pair or between 2nd and top pair)
- Top pair
- Overpair
- 2 pair
- Three of a kind
- Straight
- Weak flush (below 3rd nut flush)
- Strong flush (3rd nut flush or higher)
- Full house
- Quads+

Note that we do not consider pairs that exist on the board as part of our hand. For example, if we have A9 on a board of KK23, we still have “nothing” although our best 5 card hand technically contains a pair.

3. *Board Texture*

We develop two metrics in classifying the community cards: *wetness* and *volatility*. Wetness measures the likelihood that a hand connects with the board in some way, and volatility measures the likelihood that future community cards will change relative hand strengths. For example, a flop of QsJsTs is very wet, while 962 rainbow is not. A board of 6s5s3c is very volatile while A92 is not. We refer to boards with high and low degrees of wetness *wet* and *dry* respectively, and we refer to boards with high and low degrees of volatility *volatile* and *static* respectively.

The combination of absolute hand strength and board texture approximately determines relative strength. In general, relative hand strengths are lower on wet boards and higher on dry boards. Additionally, it is more imperative to play aggressively with strong hands for value and protection early on when the board is volatile and not so much when the board is static.

We will discuss how we implement this in the postflop section.

b. Preflop Play

Preflop play is fairly simple. By using the hand strength parameter to model postflop, we can consider preflop and postflop play separately. The game tree is very small, so we distinguish between all 169 possible hands. Each hand contains a strategy (which is different for BU and BB) using the aforementioned bet level model.

Each hand’s strategy contains six decimals corresponding to the probabilities that the player is willing to “play to” a particular bet level – 0, 1, 2, 3, 4, or 5. For example, if the BU strategy is 0.50000 0.20000 0.30000 0.00000 0.00000 0.00000 for a particular hand, he/she will fold 50% of the time, limp 20% of the time (and fold to a raise) and raise 30% of the time (and fold to a reraise). We ignore the possibility of limping and then re-raising for the sake of convergence. This makes our limps highly exploitable, but we don’t expect most bots to have exploitative strategies.

For typical bet sizes, this mechanism uniquely determines preflop strategy. For example, if BU uses level 3, then he/she will raise and call a 3-bet, but he/she will not 4-bet. We define two arrays – “ideal” preflop raise sizing and level thresholds.

Bet Level	Ideal Preflop Raise Sizing	Level Threshold
0	0	0
1	2	2
2	6	8
3	18	24
4	46	72
5	200	200

The level threshold is slightly higher than the ideal preflop sizing progression to account for potentially larger than normal raise sizing. For example, although a raise to 8 is larger than what we would prefer to make, if opponents make it, it should be treated as a single raise and not a 3-bet. However, an abnormally large raise such as a raise to 15 is treated as a 3-bet, which is fine.

Generally we will follow our ideal preflop raise sizing. However, if opponents use abnormally large sizing that throws off the bet levels, we will simply use 3x as a default.

Define the *chip level* as the amount of chips we would be willing to play at a given situation (or the threshold).

At a junction in which we can fold, call or raise, we use the following algorithm to determine which action to take:

- Raise if: raise amount <= chip level
- Call if: previous bet <= chip level < raise amount
- Fold if: chip level < previous bet

At a junction in which we can bet or check: (note that betting only applies to postflop)

- Bet if: bet amount <= chip level
- Check if: chip level < bet amount

c. Postflop Play

Recall that a game state contains information about previous betting history, board texture, and absolute hand strength. Each game state contains varying numbers of decimals, corresponding to the probabilities that the player takes some betting line. We consider discarding as a static algorithm instead of a probabilistic event, which we will discuss later.

Our bot still bets according to the bet level model. However, the number of strategic options is doubled. The BU needs to consider two possible scenarios – if the BB bets or checks, and the BB needs to consider two options – whether he wants to lead out or check with the intention of calling/raising potential bets.

As such, each game state has up to 13 corresponding decimal weights. The first corresponds to nonexistence, if the betting round has not yet been reached (e.g. if we're on the flop, the turn betting state does not exist). For the BB, the first *n* bet levels corresponds to leading out and the second set of bet levels corresponds to check/call or check/raising. For the BU, the first *n* bet levels correspond to if BB checks and the second set corresponds to if BB bets.

For example, if the BB selects bet level 2 and BU selects bet level 9 (corresponds to bet level 3, but in the case BB bets), then the action proceeds as follows:

- BB leads out.
- BU raises, with the intention of calling a potential reraise.
- BB calls.

The postflop bet thresholds are defined as follows, where x is the size of the pot.

- 0 == 0 < bet <= 0.2x
- 1 == 0.2x < bet <= x
- 2 == x < bet <= 3x
- 3 == 3x < bet <= 9x

  …etc

For example, a bet of 8 into a pot of 10 is interpreted as a single bet, but a bet of 15 into a pot of 10 is interpreted as a 2-bet. A bet of 2 into a pot of 10 will actually be interpreted as a check and will be called if we intend to play to level 0, to prevent our bot from overfolding to extremely small bets.

Our bet sizing is always in relation to the size of the pot. If we choose to bet, we will always bet 2/3*pot. Our raises are generally 3 times the size of the previous bet/raise. If the amount we would bet/raise to puts more than 60% our chips behind into the pot, we shove instead. This prevents our bot from folding when already pot-committed.

In determining the game state, preflop action is encapsulated as the level which was actually played to, not the intended level. For example, if BU intended to play to level 4, but BB only called the preflop raise, the preflop bet level would be recorded as 2, not 4. Similarly for postflop – actual play levels are kept track of in determining strategy. However, when we accumulate profits, we update all bet levels between what was actually played and the maximum level to varying degrees. Also note that the large majority of betting histories are not possible since our stacks are only 100BB. This decreases the number of bet history parameters down significantly. In total, there are approximately 400 betting histories we account for.

In determining board texture, we introduce several factors.

- *Connectivity*: a metric of how connected the community cards are
- *Suitedness*: a metric of how community cards of the same suit affect board texture
- *Highness*: how high the community cards are (higher cards more likely hit hand ranges)
- *Highcard*: a metric that approximates how the highest card on the board affects volatility

Then, we define:

- *Wetness = connectivity + suitedness + highness*
- *Volatility = connectivity + suitedness + highcard + pair adjustment*

These metrics are computed differently for the flop/turn/river to account for the number of cards. They are as follows (note that volatility is only defined on the flop):

1. *Flop:*
   a. *Suitedness*: +0 for rainbow, +1 for 2 of a suit, +3 for 3 of a suit
   b. *Connectivity*: +3 * sum(1/1.5^(pairwise "distances" between distinct cards))
      i. "Distance" is just the absolute value of the difference of their values. Note that if multiple cards of the same value exist on the board, it is treated as if only one of that card exists. This decreases wetness significantly when the board is paired, as desired.
   c. *Highness*: + ¼ * sum(card values) / 3

   d. *Highcard*: + 7 – 1.15 ^ (highest card value)
   e. *Pair adjustment*: -1 for a pair, -2 for trips
   f. VOLATILITY (v) BINS:
      i. Low: v < 3.5
      ii. Medium: 3.5 <= v < 7.0
      iii. High: 7.0 <= v
   g. WETNESS (w) BINS:
      i. Very low: w < 4.0
      ii. Low: 4.0 <= w < 6.0
      iii. High: 6.0 <= w < 8.0
      iv. Very high: 8.0 <= w
2. *Turn*:
   a. *Suitedness*: +0 for rainbow, +1 for each 2 of a suit, +3 for 3 of a suit, +6 for 4 of a suit
   b. *Connectivity*: +2 * sum(1/1.5^(pairwise "distances" between distinct cards))
   c. *Highness*: + ¼ * sum(card values) / 4
   d. WETNESS (w) BINS:
      i. Very low: w < 5.0
      ii. Low: 5.0 <= w < 7.5
      iii. High: 7.5 <= w < 10.0
      iv. Very high: 10.0 <= w
3. *River*:
   a. *Suitedness*: +2 for 3 of a suit, + 5 for 4 or more of a suit
   b. *Connectivity*: +1.5 * sum(1/1.5^(pairwise "distances" between distinct cards))
   c. *Highness*: + ¼ * sum(card values) / 5
   d. WETNESS (w) BINS:
      i. Very low: w < 5.5
      ii. Low: 5.5 <= w < 7.0
      iii. Medium: 7.0 <= w < 8.5
      iv. High: 8.5 <= w < 10.0
      v. Very high: 10.0 <= w

d. Swap Algorithm (Swap.java)

The swap algorithm we devised is approximately optimal in the large majority of situations. There are some rare situations which it isn't (e.g. if we hold 84 on a board of 99884, we won't discard the 4 as we should).

We did not apply CFR to the discard mechanism to ensure more accurate convergence. This means that our discards won't take into account previous actions/opponent ranges, and we also don't consider whether our opponent discarded in the game state. These factors are definitely not negligible (e.g. not discarding might indicate a pocket pair or strong 2 pair+ hands), but we felt that it wouldn't be worth the extra computation time and complexity.

A key aspect of our algorithm is whether a hole card is *associated* with a draw or not. A card is *associated* with a draw if without it, the draw no longer exists. For example, if our hand is AcQh and the board is 2c3c5c, the Ac is associated with the flush draw, but the Qh is not. This boolean flag allows us to accurately discard with draws. Our swap algorithm is as follows:

1. If a hole card has the same value as another hole or community card, don't swap it.
2. Then, check whether a hole card is associated with a flush or flush draw. If it is, don't swap it.
3. Then, check whether a hole card is associated with a straight. If it is, don't swap it.
4. Then, check whether there are 3 or more of a suit on board. If there is, ignore our straight draws. Otherwise:
    a. Define the *power* of a card as the number of straight draws it is associated with.
    b. If a card is associated with a 2-way straight draw (typically this means its power is 2, but there are some unusual circumstances which have been accounted for in our algorithm), don't discard it.
5. Finally, determine which card to swap (if any):
    a. If both cards are flagged as "don't discard", don't discard anything.
    b. If exactly one card is flagged, discard the one that's not flagged.
    c. If neither card is flagged, discard the one with lower power. If they have the same power, discard the one with the lower value.

In the appendix, we include the mathematical reasoning behind our swap algorithm.

II. Program Outline

We decided to use Java because we were all fairly experienced in it, and for the ease of object oriented programming. We constructed the following classes:

- *PokerTrainer.java*: This is the engine and main program that trains our bot. It contains the BU and BB as Player instances and includes several primary methods that play them against each other, such as playHand. It also includes other helper methods such as drawCard.

- *Player.java*: This class encapsulates the information known to a player, such as his/her hole cards, strategy, and accumulated profit. It contains methods that correspond to player actions, such as determining bet sizes, updating weights, check, raise, etc.

- *PerformedAction.java*: This is a mostly static class that allows us to easily keep track of previous actions in a hand.

- *GameTree.java*: All the weights/strategies are stored in a GameTree instance. Each instance contains a 4-dimensional array of weights (containing information about previous bet history) corresponding probabilities of taking actions. There is a separate GameTree instance for each board and hand strength. This class also includes the initialization mechanism (which we will discuss later), normalization algorithm, and implementation for IO.

- *IntTree.java*: Similar to GameTree, but contains a 4-dimensional array of integers to keep track of the number of times a game state has been reached, to easily normalize accumulated profits.

- *Swap.java*: A mostly static class that implements our swap algorithm.

- *Card.java*: this class encapsulates a card (with a suit and value), and includes many static methods that determines card-related properties, such as wetness and volatility. It also provides methods to help with indexing and such.

III. <u>GameTree Structure</u>

As mentioned, the strategies are stored in GameTree instances. Each Player instance has a 4-dimensional array of GameTrees, and each GameTree contains a 4-dimensional array of decimal weights. The general structure is as follows:

Each Player contains an instance:
- GameTree[index1][index2][index3][index4], each of which contains:
    - Weights[index5][index6][index7][index8]

The indices represent the following about the game state:

1. The first index is an integer from 0 to 13, depending on the player's hand strength.
2. The second index is an integer from 0 to 12 that determines the flop texture. 0 indicates nonexistence (i.e. flop has not been dealt). Recall that we binned wetness into four categories; let these be 1, 2, 3, 4, from lowest to highest. Similarly, let volatility be 1, 2, or 3. Then, the flop index is determined by (wetness – 1) * 3 + volatility.
3. The third index is an integer from 0 to 4 that determines turn wetness. 0 indicates nonexistence, and wetness is binned into 1, 2, 3, 4.
4. The fourth index is an integer from 0 to 4 that determines river wetness.

5. The fifth index is an integer from 0 to 5 that determines preflop bet history. Recall that we keep track of this using the level actually played to, not the intended bet level.
6. The sixth index is an integer from 0 to 13 that represents flop betting.
7. The seventh index is an integer from 0 to 13 that represents turn betting.
8. The eighth index is an integer from 0 to 13 that represents river betting.

When we construct each of the weights[][][][] arrays, we actually do so in a specific way to eliminate nonsense histories. For example, weights[0][1][1][1] is nonsense for the BU because the game would have terminated preflop after he/she folded. By eliminating these nonsense histories, we significantly reduce the amount of memory we need.

IV. CFR Update Mechanism

We use the fixed-strategy iteration CFR (FSICFR) algorithm to update weights and optimize our strategy.

Initially, when we first construct our player instances, we set the probabilities at each decision node as a random, uniform distribution. We then initialize some of the weights using poker intuition to ensure more accurate and faster convergence. The weights are initialized as follows:

a. Preflop Initialization

See the appendix; which contains the entire initialized strategy for BU and BB preflop.

b. Postflop Initialization

The tables for our postflop initialization are shown below. A probability in 3+ indicates that all bet levels greater than or equal to 3 will be uniformly distributed, summing to that probability. If some bet level is not achievable because the stack-to-pot ratio is low, include all probabilities of higher bet levels in the highest achievable bet level.

For the BB, we also significantly decreased the probability of check/raising in relation to check/calling and betting, since we believe that check/raising is a powerful tool that should be used sparingly.

FLOP

Hand Strength	3+	2	1	0
Nothing	0	0	0.2	0.8
Draws	0.2	0.2	0.5	0.1
Weak/Middle Pair	0	0.05	0.75	0.2
Top pair/Overpair	0.2	0.6	0.1	0.1
2pair or better	0.75	0.1	0.1	0.05

TURN

Hand Strength	3+	2	1	0
Nothing	0	0	0	1.0
Draw	0	0	0.5	0.5

Weak/Middle Pair	0	0	0.5	0.5
Top pair/Overpair	0.1	0.5	0.3	0.1
2pair or better	0.6	0.25	0.1	0.05

RIVER

Hand Strength	3+	2	1	0
Nothing	0	0	0	1
Weak/Middle pair	0	0	0.5	0.5
Top pair/Overpair	0	0.3	0.6	0.1
2pair	0.2	0.4	0.35	0.05
3 of a kind or better	0.4	0.4	0.15	0.05

Once we have initialized our weights, we are ready to begin training. We set the number of iterations per update cycle at 1e9. We have approximately 1e8 distinct game states. Using 1e9 iterations balances convergence rate (we don't want it to converge too slowly) and convergence accuracy (too few iterations will result in choppy, inaccurate convergence). Each one of these iterations took approximately an hour to run on an external GPU. We looked into multi-threading to optimize runtime, but this was unfeasible due to the way our strategies were stored in a single object.

For every iteration, we play one hand. Once the hand terminates, either by a player folding or by showdown, profits are accumulated. The terminal node accumulates a profit equal to the net change in chips. We then propagate the profit backwards. Each time we reach a decision node, we multiply the profit by a predefined factor (we used 0.8) to account for the fact that prior actions do not completely determine the result, and accumulated the adjusted profit. For example, consider the following hand:

- Preflop: BU raises to 6, BB calls
- Flop: BB checks, BU bets 8, BB folds.

Then BU will accumulate +6 for betting on the flop and +4.8 for raising preflop for this game state. BB will accumulate -6 for check/folding and -4.8 for calling preflop for this game state.

After 1e9 iterations of playing, using the same fixed strategies, we update strategies. We first normalize the profit matrix by dividing by the number of hands played at each game state to get a profit/hand matrix. This ensures that the weight update mechanism will not be biased towards game states that are played more often. Then, we multiply every weight by an exponential factor depending on the profit/hand, which is exp(profit per hand * normalizing constant). Finally, we re-normalize all the weights, dividing each weight by the sum of all the weights in its game state, to ensure that all the weights corresponding to a game state sum to 1.

We picked an exponential model to update weights for several reasons:

- Multiplying by some exponent ensures that all the weights are still positive.
- Earning twice as many chips from a hand on average is far more than twice as profitable. Using an exponential model models this well.

We run the above algorithm indefinitely. The strategies slowly converge.

V. Reflections

Our bot definitely did not do as well as we would have hoped; in fact looking at some of the logs of our scrimmage games, we are reasonably confident that we could have done very well just by hard-coding our bot. There were several major problems that arose in our bot, mostly due to coding errors, but several due to design issues.

Problems with Design:

- The bet level model is useful as a way to categorize bets into different bins based on size. However, it is problematic as a tool to select actions, and it also makes updating weights challenging.
  - A lot of the times, our bot is far too aggressive, particularly from the BB. There are a lot of situations in which you may want to call a bet as a bluff-catcher, but not necessarily bet yourself, such as with bottom pair on the flop. Another situation that comes up often is that our bot will raise far too often. This likely occurs when our bot has a hand that he's willing to bet-raise with. This model often treats calling a bet and betting yourself as essentially the same.
  - We had a lot of trouble devising a good way to update weights when the intended bet level and the actual bet level were different. Consider the preflop decision node for AA, for example. Obviously, you are willing to play to max bet level 5. However, most of the time you will not achieve this, and the other bet levels will accumulate most of the profit. If we update only the intended bet level (which we did at first), the bot becomes far too aggressive. But if we only update the played bet level, a different problem occurs – the lower bet level is overly filled with samples in which we didn't get reraised and forced off the hand, creating a different bias. We decided to update all the weights between the played and intended bet level, but the fact that we had to do this shows the deficiencies in our design.

- We definitely didn't need so many decision nodes for postflop betting – 3-bet and higher ranges are so narrow that they can mostly be binned together.

Instead of using the bet level model to determine actions, we should have used the traditional check/bet and fold/call/raise model, with probabilities corresponding to each of the actions at a given decision node. While this significantly increases the number of parameters (by approximately a factor of 2-3), the benefits far outweigh the potential drawbacks.

Problems with Implementation:

- There were a lot of bugs that arose. At some point in time, each of the following aspects of our programs did not function properly:
  - Hand strength didn't take into account pairs on board, which caused our bot to play far too loose on paired boards.
  - A small typo recognized any bet under 2/3pot as a check, which caused our bot to be overly aggressive.

  - A small sign error caused our initialization to backfire, making check-raising 2.5x more likely than bet-calling, the opposite of what we wanted to occur.
  - Etc.

We wholly believe that if we had implemented the traditional check/bet and fold/call/raise model instead of our bet level model to determine actions and had some additional time to fix bugs and retrain our bot, our system could have done very well. Unfortunately, we weren't able to recognize the bugs until very late on because they surfaced after training completed. As each training iteration took approximately an hour, computational time is very significant for us.

Future Improvements

One way we could improve convergence rate significantly is to implement "strategy cleaning". Once the weight of some strategic option in a given game state dips below a predefined threshold (such as 0.005), we "clean" the option from our strategy by setting it to zero. Since we use the exponential model, we can never fully reach 0, so using strategy cleaning helps prevent our bot from randomly and unexpectedly taking less profitable lines.

As mentioned, our bet level model was definitely suboptimal, and the convergence didn't work out properly. Some hands seem to be played fairly optimally, but many strange things happened in other hands, including:

- 3-bet shoving the river with 10 high (the probability of this should have been initialized to 0, but somehow it still happened?)
- Flopping middle pair, betting out, then swapping to make a set, and checking the rest of the way down (why are we checking once we make a monster hand?)
- Folding to a bet of 8 when the pot is 384 (we coded the player to recognize this as a check, so we're not sure why this happened).

In any case, many bugs and suboptimal bet level design prevented our weights from converging properly.

Another appropriate adjustment is instead of fixing bet/raise sizes beforehand, we can make larger bets when the board is more wet/volatile and smaller bets when the board is dry/static. The wetter a board, the more likely hole cards connect with the board, allowing them to generally call larger bets when we have value hands. The more volatile a board, the more imperative it is for strong hands to extract value immediately and price out draws/overcards. For example, we could make our bet sizes on the flop as follows, where x is the sum of the wetness (1-4) and volatility (1-3):

- 2 == bet 0.5 * pot
- 3 == bet 0.6 * pot
- 4 == bet 0.7 * pot
- 5 == bet 0.8 * pot
- 6 == bet 0.9 * pot
- 7 == bet 1.0 * pot

This would not increase the size of the game tree, since each game state contains information about the board. We would need to take more care in initializing the weights but it is relatively easy to implement. Also, instead of defining raises to be a multiplicative factor of the previous raise, we could make them a function of pot size. This would be a better defense against very small bets. For example, we could make 0.7 * pot size raise or a pot sized raise. Again, we would make larger raises on wet/volatile boards and smaller or dry/static boards.

VI. <u>Appendix</u>

a. Mathematical Analysis for Swapping

This mostly concerns flush/straight draws, since if you make a pair with a given card, it's fairly obvious that card should not be swapped.

- Benefits of Swapping
    - Potentially make a pair, which could lead to a showdown-able hand, or make two pair, leading to a strong hand.
    - Swapping allows you to improve your hand *before flop/turn betting occurs*. This is crucial if your opponent might bet. (This is obviously not important for considering pure equity)
    - Depending on the board, it might be easier to represent strong hands due to a wider possibility of ranges from swapping.

- Benefits of Not Swapping
    - Flush/straight draws generally have good implied odds.
    - Not swapping allows for good semi-bluffing opportunities (such as check-raising), since not swapping often indicates strength (two pair or greater).
    - Not swapping balances out an otherwise very strong range of two pair or greater

- Some Strategic Ideas
    - Keep in mind that you want to compare your hand strength to your opponent's range. For example, if you expect your opponent to have a strong range (two pair or better), drawing to one pair is absolutely worthless, and you more than often should chase the draw instead of going for pair opportunities.
    - While it's unclear how the frequency of flushes changes with the new discard rule, it's clear that other value hands (especially two pair and full houses) increase in frequency significantly. Thus, when you have a smallish flush, you can be somewhat less worried about being against a higher flush.
    - Be wary of paired boards. The potential of making a full house increases drastically with the discard rule. Flushes/straights are most likely not strong enough to make value raises in these spots, especially if the board has many high cards.
    - Straight draws are significantly devalued when there are flush draws, or three of a suit. Lean more towards swapping out offsuit/low cards in this case.
    - Keep in mind that the quality of the pairs you make matters.
        - The higher your cards, the more you should lean towards not swapping (especially with overcards) because making top pair is relatively strong and showdownable. On the other hand, you might be more willing to swap out low cards because the pairs that they might make by not swapping are relatively worthless.
        - With pair + straight draws, the lower the pair the more you should lean towards swapping the paired card out.

We did some computations to mathematically proof our swapping algorithm. For reference, here are some relevant probabilities we used:

- 20% ~ chance of hitting a flush draw on given street
- 18% ~ chance of hitting a straight draw on a given street
- 13% ~ chance of hitting a pair on a street (with two unpaired cards)
- 9% ~ chance of hitting a gutshot on a given street
- 20% ~ chance of hitting a pair by discarding on the flop
- 25% ~ chance of hitting a pair by discarding on the turn
- 20% ~ chance of re-creating a flush draw by discarding

These probabilities are approximate and don't account for card removal effects, but they are mostly negligible. Consider the following situations *on the flop*:

- **Flush Draw (no pair):**

Suppose we have a flush draw without a pair, such as Jh9h on a board of Ah8h4d. By not swapping, we have a 20% chance of improving to a flush and a 13% chance of improving to a pair by the turn. By swapping, we have a 30% chance of improving to a pair by the turn, and a 20% chance of re-obtaining a flush draw, which makes a flush 4% of the time. We can also make two pair or three of a kind approximately 2-3% of the time. Thus we have the following outcomes, with respective probabilities:

	Nothing	Draw, no pair	Pair, no draw	Pair + draw	Two pair/trips	Flush
Swap	45%	16%	28%	5%	2%	4%
No swap	0%	67%	0%	13%	0%	20%

If we treat the above hand strengths as strictly increasing in value from left to right, not swapping nearly completely dominates swapping. As such, we *don't swap flush draws*, unless it's a one-card flush draw, in which we obviously discard the offsuit card.

- **2-way Straight Draw (no pair):**

A 2-way straight draw functions similarly as a flush draw. The probability of making a straight is slightly lower than that of making a flush (8 outs vs 9 outs), but it is not significant enough to impact strategy. However, straight draws are notably not as valuable on boards with many cards of the same suit. In that case it is much more valuable to swap and chase a flush/pair, since your opponent will likely do the same, so making a straight is not as valuable. As such, we *don't swap open-ended straight draws*, unless there's three or more of a suit on board.

- **Monster Draw (no pair)**

We do not swap a monster draw, which is a combination of a flush draw and a straight draw (open-ended or gutshot). Since we strictly do not swap flush draws, and monster draws have even more equity of making strong hands, it makes sense not to swap it.

- **Gutshot Straight Draw (no pair)**

We swap out gutshot straight draws. The probability of hitting a gutshot is only 9%, which is not significant enough to justify the 30%+ chance of making a pair or better and obtaining a showdown-able hand. While our winrate when making a pair is obviously nowhere near 100%, it is definitely high enough to justify the swap. Also note that swapping out a card in a gutshot has a significant probability of making a stronger open-ended straight draw. Here's the corresponding table:

	Nothing	Draw, no pair	Pair, no draw	Pair + draw	Two pair/trips	Straight
Swap	~50%	~14%	28%	~4%	2%	~2%
No swap	0%	67%	0%	13%	0%	9%

Note the probabilities are approximate because it depends highly on the nature of the gutshot. But it is easy to see that the probability of making a pair is very relevant. Also, keep in mind that the draws that swapping creates will on average be much stronger because some of them will be open-ended straight draws.

- **Flush Draw + pair:**

Having both a pair and a flush draw complicates things. Of course, swapping out the unpaired card is strictly better than swapping out the paired card. Suppose we have Jh9h on a board of AhJd4h. Then the probabilities are as follows, for swapping the unpaired card:

	Pair, no draw	Pair + draw	2pair +	Flush
Swap	53%	16%	27%	4%
No swap	0%	69%	11%	20%

Again, we can see that not swapping dominates swapping because the hand strengths are increasing from left to right. 20% + 11% = 27% + 4% and 69% = 53% + 16%. Therefore we *do not swap with a flush combo draw.*

- **Straight Draw + pair:**

If the draw is a gutshot straight draw, we can ignore it and discard the unpaired card. However, if it's an open-ender it's not as clear, since we might optimally want to discard either the paired card (to increase our probability of making a straight) or unpaired card (to increase our probability of making 2pairs and the like). Since swapping the unpaired card is functionally very similar to swapping out the unpaired card of a flush draw (except the draws re-obtained can either be gutshots or open-enders), we will ignore the possibility of swapping out the unpaired card. The probabilities for swapping out the paired card are as follows:

	Pair, no draw	Draw	Pair + draw	2pair+	Straight
Swap	4.5%	40%	20%	2.5%	33%
No swap	0%	0%	69%	11%	18%

It is not immediately obvious what to do. Against a very strong range that contains a lot of 2pair type hands, it is good to swap to chase the straight. However, in most cases, the pair holds a lot of valuable showdown value, and swapping it out is a very negative EV play. As such, we will *not swap straight combo draws*.

b. Preflop Initialization

The hands are indexed according to the table mentioned in the section about equity. Starting from the bottom right corner (22), proceed left across a row. Follow rows left first, then columns up. For example, 0 == 22, 1 == 32o, 13 == 32s, 167 == AKs, 168 == AA. The strategies listed below start from index 168 (AA) and work its way down. The pocket pairs have been marked in bold for reference.

(Both players are initialized to the same values)

0.00000 0.00000 0.00000 0.00000 0.00000 1.00000 AA
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000
0.00000 0.00000 0.00000 0.00000 0.60000 0.40000
0.00000 0.00000 0.00000 0.15000 0.60000 0.25000
0.00000 0.00000 0.00000 0.30000 0.60000 0.10000
0.00000 0.00000 0.00000 0.40000 0.50000 0.10000
0.00000 0.00000 0.00000 0.45000 0.45000 0.10000
0.00000 0.00000 0.00000 0.50000 0.40000 0.10000
0.00000 0.00000 0.00000 0.50000 0.45000 0.05000
0.00000 0.00000 0.00000 0.55000 0.40000 0.05000
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000 KK
0.00000 0.00000 0.00000 0.00000 0.20000 0.80000
0.00000 0.00000 0.00000 0.10000 0.30000 0.60000
0.00000 0.00000 0.00000 0.15000 0.45000 0.40000
0.00000 0.00000 0.00000 0.20000 0.50000 0.30000
0.00000 0.00000 0.00000 0.30000 0.50000 0.20000
0.00000 0.00000 0.00000 0.40000 0.50000 0.10000
0.00000 0.00000 0.00000 0.50000 0.50000 0.00000
0.00000 0.00000 0.05000 0.60000 0.35000 0.00000
0.00000 0.00000 0.25000 0.50000 0.25000 0.00000
0.00000 0.00000 0.45000 0.40000 0.15000 0.00000
0.00000 0.00000 0.65000 0.25000 0.10000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000 QQ
0.00000 0.00000 0.00000 0.40000 0.20000 0.40000
0.00000 0.00000 0.00000 0.50000 0.20000 0.30000

0.00000 0.00000 0.00000 0.65000 0.20000 0.15000
0.00000 0.00000 0.00000 0.80000 0.20000 0.00000
0.00000 0.00000 0.05000 0.80000 0.15000 0.00000
0.00000 0.00000 0.10000 0.80000 0.10000 0.00000
0.00000 0.00000 0.20000 0.80000 0.00000 0.00000
0.00000 0.00000 0.40000 0.60000 0.00000 0.00000
0.00000 0.00000 0.60000 0.40000 0.00000 0.00000
0.00000 0.00000 0.80000 0.20000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000
0.00000 0.00000 0.00000 0.10000 0.30000 0.60000
0.00000 0.00000 0.00000 0.40000 0.20000 0.40000
0.00000 0.00000 0.00000 0.25000 0.50000 0.25000 JJ
0.00000 0.00000 0.00000 0.50000 0.40000 0.10000
0.00000 0.00000 0.00000 0.55000 0.45000 0.00000
0.00000 0.00000 0.10000 0.60000 0.30000 0.00000
0.00000 0.00000 0.20000 0.65000 0.15000 0.00000
0.00000 0.00000 0.30000 0.70000 0.00000 0.00000
0.00000 0.00000 0.40000 0.60000 0.00000 0.00000
0.00000 0.00000 0.60000 0.40000 0.00000 0.00000
0.00000 0.20000 0.60000 0.20000 0.00000 0.00000
0.00000 0.40000 0.60000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000
0.00000 0.00000 0.00000 0.15000 0.45000 0.40000
0.00000 0.00000 0.00000 0.50000 0.20000 0.30000
0.00000 0.00000 0.00000 0.50000 0.40000 0.10000
0.00000 0.00000 0.00000 0.25000 0.50000 0.25000 TT
0.00000 0.00000 0.00000 0.80000 0.20000 0.00000
0.00000 0.00000 0.20000 0.70000 0.10000 0.00000
0.00000 0.00000 0.30000 0.70000 0.00000 0.00000
0.00000 0.00000 0.45000 0.55000 0.00000 0.00000
0.00000 0.20000 0.50000 0.30000 0.00000 0.00000
0.00000 0.30000 0.60000 0.10000 0.00000 0.00000
0.00000 0.40000 0.60000 0.00000 0.00000 0.00000
0.00000 0.60000 0.40000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.60000 0.40000
0.00000 0.00000 0.00000 0.20000 0.50000 0.30000
0.00000 0.00000 0.00000 0.65000 0.20000 0.15000
0.00000 0.00000 0.00000 0.55000 0.45000 0.00000
0.00000 0.00000 0.00000 0.80000 0.20000 0.00000
0.00000 0.00000 0.00000 0.25000 0.50000 0.25000 99
0.00000 0.00000 0.20000 0.80000 0.00000 0.00000
0.00000 0.00000 0.40000 0.60000 0.00000 0.00000
0.00000 0.10000 0.60000 0.30000 0.00000 0.00000
0.00000 0.30000 0.60000 0.10000 0.00000 0.00000
0.00000 0.50000 0.50000 0.00000 0.00000 0.00000
0.50000 0.30000 0.20000 0.00000 0.00000 0.00000

1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.15000 0.60000 0.25000
0.00000 0.00000 0.00000 0.30000 0.50000 0.20000
0.00000 0.00000 0.00000 0.80000 0.20000 0.00000
0.00000 0.00000 0.10000 0.60000 0.30000 0.00000
0.00000 0.00000 0.20000 0.70000 0.10000 0.00000
0.00000 0.00000 0.20000 0.80000 0.00000 0.00000
0.00000 0.20000 0.50000 0.30000 0.00000 0.00000 88
0.00000 0.00000 0.40000 0.60000 0.00000 0.00000
0.00000 0.00000 0.60000 0.40000 0.00000 0.00000
0.00000 0.20000 0.70000 0.10000 0.00000 0.00000
0.20000 0.30000 0.50000 0.00000 0.00000 0.00000
0.40000 0.30000 0.30000 0.00000 0.00000 0.00000
0.70000 0.30000 0.00000 0.80000 0.00000 0.00000
0.00000 0.00000 0.00000 0.40000 0.50000 0.10000
0.00000 0.00000 0.00000 0.50000 0.50000 0.00000
0.00000 0.00000 0.05000 0.80000 0.15000 0.00000
0.00000 0.00000 0.20000 0.65000 0.15000 0.00000
0.00000 0.00000 0.30000 0.70000 0.00000 0.00000
0.00000 0.00000 0.40000 0.60000 0.00000 0.00000
0.00000 0.00000 0.50000 0.50000 0.00000 0.00000
0.00000 0.30000 0.50000 0.20000 0.00000 0.00000 77
0.00000 0.40000 0.40000 0.20000 0.00000 0.00000
0.20000 0.50000 0.30000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.40000 0.50000 0.10000
0.00000 0.00000 0.00000 0.60000 0.40000 0.00000
0.00000 0.00000 0.10000 0.70000 0.20000 0.00000
0.00000 0.00000 0.50000 0.50000 0.00000 0.00000
0.00000 0.00000 0.70000 0.30000 0.00000 0.00000
0.00000 0.15000 0.70000 0.15000 0.00000 0.00000
0.00000 0.30000 0.60000 0.10000 0.00000 0.00000
0.00000 0.50000 0.50000 0.00000 0.00000 0.00000
0.00000 0.50000 0.50000 0.00000 0.00000 0.00000 66
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.45000 0.45000 0.10000
0.00000 0.00000 0.00000 0.70000 0.30000 0.00000
0.00000 0.00000 0.30000 0.60000 0.10000 0.00000
0.00000 0.00000 0.50000 0.50000 0.00000 0.00000
0.00000 0.30000 0.40000 0.30000 0.00000 0.00000
0.00000 0.60000 0.30000 0.10000 0.00000 0.00000

0.00000 0.80000 0.20000 0.00000 0.00000 0.00000
0.00000 0.90000 0.10000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.15000 0.50000 0.35000 0.00000 0.00000 0.00000 55
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.50000 0.40000 0.10000
0.00000 0.00000 0.10000 0.70000 0.20000 0.00000
0.00000 0.00000 0.40000 0.60000 0.00000 0.00000
0.00000 0.00000 0.70000 0.30000 0.00000 0.00000
0.00000 0.40000 0.60000 0.00000 0.00000 0.00000
0.30000 0.40000 0.30000 0.00000 0.00000 0.00000
0.60000 0.40000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.30000 0.50000 0.20000 0.00000 0.00000 0.00000 44
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.50000 0.45000 0.05000
0.00000 0.00000 0.20000 0.60000 0.20000 0.00000
0.00000 0.00000 0.50000 0.50000 0.00000 0.00000
0.00000 0.40000 0.40000 0.20000 0.00000 0.00000
0.40000 0.40000 0.20000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.40000 0.50000 0.10000 0.00000 0.00000 0.00000 33
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.50000 0.50000 0.00000
0.00000 0.00000 0.20000 0.60000 0.20000 0.00000
0.00000 0.00000 0.50000 0.50000 0.00000 0.00000
0.00000 0.40000 0.50000 0.10000 0.00000 0.00000
0.40000 0.40000 0.20000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.50000 0.50000 0.00000 0.00000 0.00000 0.00000 22