Chapter 3

Computer Poker Basics

> *Do not expect to arrive at certainty in every subject which you pursue. There are a hundred things wherein we mortals. . . must be content with probability, where our best light and reasoning will reach no farther.*
>
> – Isaac Watts (1674-1748)

The fundamentals of poker strategy are determined by the probabilistic nature of the game. The way a hand is played depends on the probability of both playing the hand profitably, and how it affects the profitability of future hands (by changing your table image). Winning an individual hand is not as important as winning in the long run. As mentioned earlier, poker strategy requires both mathematical understanding of the game and psychological understanding of the opponent. To play well, the concrete mathematical foundations must be in place. The psychological or intuitive aspect of poker can be used to enhance the accuracy of this foundation.

There are several skills required to play strong poker. These will all be required for a world class poker agent:

Hand Evaluation:

Accurate assessment of the current strength of our hand, and the potential strength of our hand when the future board cards are dealt, is crucial. We need the probability that our hand is the best, given the context of the game and the opponents we are up against.

Unpredictability:

Our actions must not give away the strength of our hand. We must play deceptively by *bluffing*, *slow-playing*, and *check-raising*. It is important that our betting strategy be non-deterministic, and mixes strategies in order to conceal information about our hand.

Opponent Modeling:

> We must understand how our opponent plays in order to both exploit their weaknesses and defend against their strengths. We must be able to deduce the strength of hands our opponents hold by interpreting their actions, and we must be able to anticipate how they will play.

This chapter discusses the pre-flop play, post-flop hand evaluation, and betting strategies used by *Poki*. Many of the methods in this chapter are described more thoroughly in [34].

3.1 Poki's High-Level Architecture

A poker game consists of a *dealer* together with multiple *players* that represent either human players or computer players. In our Java implementation, these players are defined as objects. The dealer handles the addition and removal of players from the game, deals the cards to each player at the start of a new hand, prompts each player for an appropriate action when it is their turn, broadcasts player actions to other players, and updates a public game context as the game progresses. The game context contains all of the public information about the game, including the names and relative locations of the players, and the board cards.

Each player extends an abstract *Player* class, that has access to all the public game information, as well as the player's private information (their cards). This class is extended to add the player's personal strategy. A player can be a *bot* (software program), or an interface for a human to play in the game, just by extending this class appropriately.

Likewise, the *Dealer* is a generalization for game controllers. We have implemented several different dealer interfaces: an IRC-Dealer for playing against other players on the IRC poker server, a Tournament-Dealer for self-play experiments, and a TCP/IP-Dealer that allows *Poki* to play against humans using a web browser, and against other programs using a published protocol (see Appendix B).

Each player is responsible for responding to a prompt by choosing an action: fold, call, or raise. In addition to the public game context, each player knows its private hand and the previous actions of all of the players. Each player is defined by an interface, and each version of *Poki* is a different implementation of that player interface.

Figure 3.1 shows the high level architecture of our Java-based poker system. The arrows depict the flow of information. The public game state and history is comprised of a *GameInfo* object, also containing several *PlayerInfo* objects which track public information for each player. The *Player* objects are each given cards by the dealer, and are then prompted for each action in the game. The players may

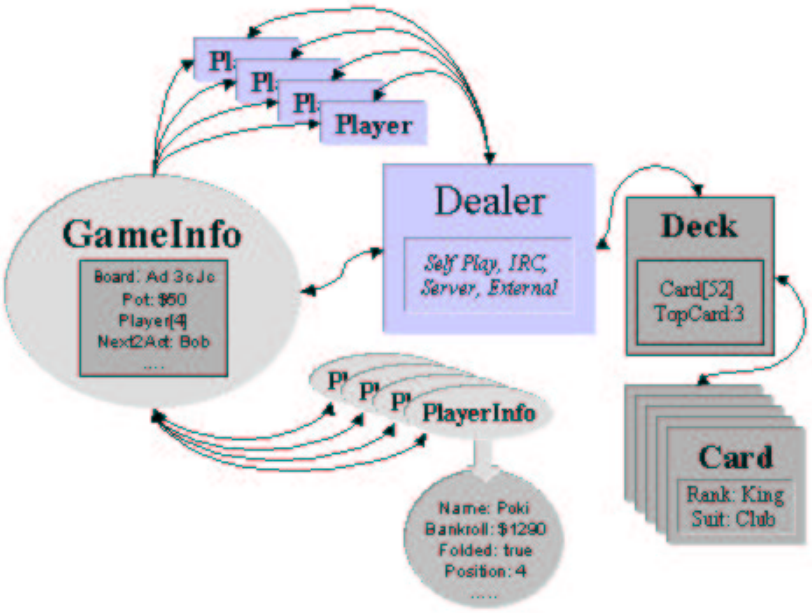

Figure 3.1: High level architecture.

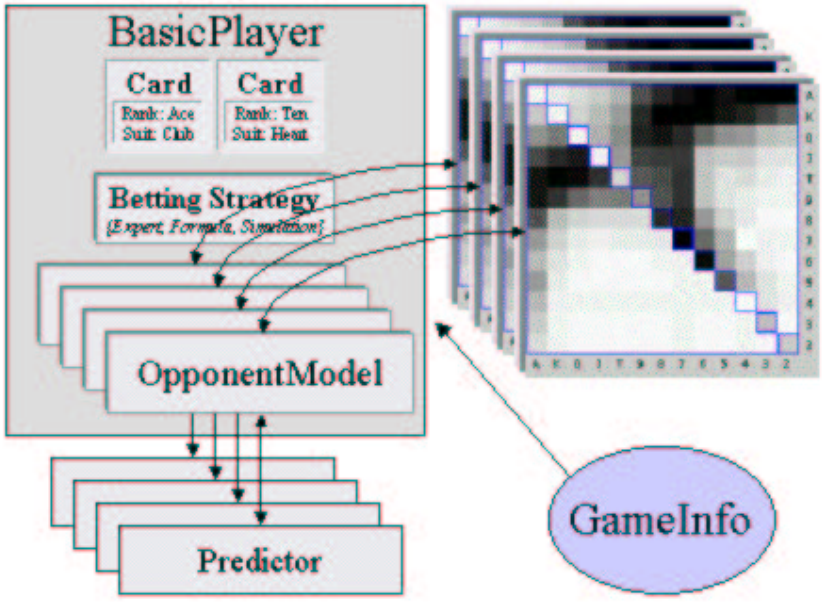

Figure 3.2: High level player architecture.

access the GameInfo and PlayerInfo objects to make their decisions. The dealer then modifes the GameInfo and PlayerInfo objects according to the rules of the game.

Although each version of *Poki* represents and uses the available information in a different way, all versions share a common high-level architecture. In addition to the public game context, *Poki* has two private components: its hand and a collection of persistent opponent models. *Poki* maintains statistical models describing each player participating in the game, including itself. When the dealer broadcasts a player action, the corresponding opponent model is responsible for processing that information.

3.2 Pre-flop Betting Strategy

The strategy of Texas Hold'em in the pre-flop stage (before any board cards are dealt) is different than in the post-flop stages. Because there are no public cards available, the state in the pre-flop is far simpler than later in the game. In total, there are $\binom{52}{2} = 1326$ different hands, but many of these hands are equivalent before the flop. Without board cards, the particular suit of the cards is irrelevant. Both Q♢-3♢ and Q♣-3♣ have the same probability of making a flush, and in all other respects they are equal. If the flop contains any diamonds or clubs, then the two hands can no longer be treated identically. Using this knowledge, there are only 169 distinct hand types in pre-flop Hold'em.

Poki's pre-flop strategy is a simple expert system built on top of a set of machine-learned tables containing an expected income rate for each hand. These tables were calculated off-line using a simple technique called *roll-out simulations*. Several million hands were played where each player calls the blind and then checks to the showdown. The cumulative wins and losses for each hand type are kept in the table. Although based on unrealistic play, these values give a good baseline for the relative profitability of each of the 169 hand types.

A refinement to roll-out simulation is to use repeated iterations of the technique, where the previous results govern the betting decision for each player. For instance, in a ten player simulation, a net negative value in the previous iteration would dictate that the hand be folded, rather than calling the big blind. This introduces some hand selection into the roll-outs, which reduces the number of active players in each hand. This generates a more realistic distribution of opponents and probable hands. Many mediocre hands that were profitable to play against players with no hand selection become negative in this more realistic simulation.

After each round of simulations has reached a reasonable degree of stability, another iteration is performed. This process eventually reaches an equilibrium, defining a set of hands that can be played profitably against the blinds and the other unknown hands. The results are most applicable to the "play or don't play"

Hand	IR-10	Iterated	Hand	IR-10	Iterated	Hand	IR-10	Iterated
AA*	+2112	+2920	ATs	+736	+640	KQo	+515	+310
KK*	+1615	+2180	99*	+553	+630	QTs	+655	+280
QQ*	+1224	+1700	KQs	+858	+620	QJs	+720	+270
JJ*	+935	+1270	AQo	+555	+560	A9s	+538	+220
TT*	+714	+920	KJs	+767	+480	ATo	+335	+200
AKs	+1071	+860	88*	+450	+450	KTs	+694	+190
AKo	+718	+850	77*	+364	+390	KJo	+391	+160
AQs	+915	+780	AJo	+430	+380	A8s	+469	+110
AJs	+813	+680	JTs	+657	+360	66*	+304	+40

Table 3.1: Iterated income rate (profitable hands).

decision for each player. Although much better than a simple roll-out simulation, this technique is still far from perfect, because other important considerations such as betting position and known opponent actions have not been accounted for.

In our experiments, each iteration lasted for 50,000 trials. A diminishing noise factor was added to each income rate, analogous to the cooling factor used in simulated annealing. This gives negative expectation hands a chance to recover as the prevailing context changes. After ten generations, the remaining positive expectation hands were played for another 500,000 trials, to ensure stability. The resulting set of profitable hands, shown in Table 3.1, is in strong agreement with expert opinion on this matter [45]. The table shows a comparison of the income rates for 10-player roll-out simulations (IR-10) and the results refined by iterating (Iterated). The values shown are in *milli-bets* (*eg.* a hand with an income rate of +1000 should win an average of one small bet each time it is played). The iterated values are reasonable estimates of actual income rates, unlike the simple roll-out values, which are only used as relative measures.

We can generate *near-optimal* solutions for the two-player game of pre-flop Hold'em with iterated roll-out simulations. The results closely match Selby's results [41] for the optimal two-player pre-flop Hold'em game.

3.3 Hand Evaluation

The post-flop betting strategy is far more difficult than the pre-flop strategy. There are several different strategies for the post-flop that will be described later. Before we do that, we must first describe how hands are evaluated. Later, the betting strategies will show how all of the evaluation information can be tied together into a rational betting decision.

```
HandStrength(ourcards,boardcards) {
   ahead = tied = behind = 0
   ourrank = Rank(ourcards,boardcards)
   /* Consider all two card combinations of the remaining cards.*/
   for each case(oppcards) {
      opprank = Rank(oppcards,boardcards)
      if(ourrank>opprank)        ahead += 1
      else if(ourrank==opprank)   tied += 1
      else /* < */              behind += 1
   }
   handstrength = (ahead+tied/2) / (ahead+tied+behind)
   return(handstrength)
}
```

Figure 3.3: Hand strength calculation.

3.3.1 Hand Strength

A crucial piece of information needed to make good decisions is the strength of our hand. A player must have a good estimate of the probability that they currently have the winning hand. A simple calculation of hand strength (HS) is to enumerate all possible other hands and compare them to ours. For instance, after the flop there are only 47 unknown cards remaining (52 cards minus our hole cards and the three board cards). Thus there are $\binom{47}{2} = 1081$ possible two card combinations we could be up against.

By counting the number of these other hands that win, tie, and lose to our hand, we can compute the probability of having the best hand, against one random hand. Figure 3.3 gives the algorithm for a simple hand strength calculation [34]. Essentially, this is just a calculation of the proportion of possible scenarios in which we have the best hand.

For example, suppose our hand is A♢-Q♣ and the flop is J♡-4♣-3♡. There are 1081 possible hands an opponent could be holding. Of these possible hands there are 444 that are currently better than our hand, 9 that would tie, and 628 that are worse. Counting ties as one half, this corresponds to a percentile ranking, or hand strength, of 0.585. In other words, there is a 58.5% chance that A♢-Q♣ is better than a random hand.[1]

This value, called the *hand rank* (HR) or *raw hand strength* (RHS), is a good first-order approximation of the strength of our hand. However, for a real game of poker, this value is too crude. If our opponent has called one or more bets, then we are most certainly not up against a random hand anymore. Rational opponents will not play most weak hands they are dealt. It is typical for Hold'em players to fold over 50% of their hands before the flop. In most situations, if an opponent

[1]Hand Strength calculations for more than one opponent will be discussed below in Section 3.3.5.

has not folded and has voluntarily put money into the pot, there is a much higher probability they are holding a hand of higher quality than average. Hands that are clear losers would normally be folded. It is too simplistic to count unlikely hands as having the same probability as likely hands.

To make the measurement more realistic, the enumeration can be weighted by the probability of each hand being held. Instead of adding one to each counter for *ahead*, *tied*, and *behind*, a weight representing the relative probability that an opponent has a specific hand can be used instead. Weighting the enumeration will be described in more detail in Section 3.3.3.

3.3.2 Hand Potential

Hand strength is a measure of the current strength of our hand. However, after the flop, the turn and river cards are yet to be dealt. These future community cards can significantly affect our hand strength. For instance, suppose Alice has K♣-K♠, and Bob has 7♡-9♡, with a flop of 8♡-6♣-4♡. At this point, Alice has a high hand strength (RHS = 0.945), while Bob has a low hand strength (RHS = 0.104). However, Bob's hand has several *outs* (cards that improve the hand) that would give him a winning hand (any of 9 remaining hearts to make a flush, plus the T♠, T♢, T♣, 5♠, 5♢, and 5♣ for 6 ways to make a straight). The probability that any one of these cards will come down on the turn or river is high (15/47 unknown cards gives a 32% chance the next card will improve Bob's situation). Despite the weak hand strength, Bob should stay in the hand because of the high potential.

The *Positive Potential*, or *PPot*, is the probability that our hand will *improve* after a new card is dealt. In general, improvement is measured by counting each situation where we are currently *behind* and will end up *ahead*, or *tied*. Likewise, *Negative Potential*, or *NPot*, is the count of hands where we are currently ahead, but will end up behind.

The actual potentials of a hand can be computed by enumeration of all the possibilities. On the flop, there are still two cards to be dealt, so a full calculation requires a two-card look ahead. There are 1,070,190 combinations in a two card look ahead (1081 possible opponent hands, multiplied by the 990 possible turn and river cards). On the turn, there is only one card to come, making for only 43,560 combinations. The algorithm for hand potential is given in Figure 3.4 [34]. A breakdown of cases from the example hand above is shown in Table 3.2.

$$PPot = \mathcal{P}(ahead \mid behind) + \frac{\mathcal{P}(tied \mid behind)}{2} + \frac{\mathcal{P}(ahead \mid tied)}{2} \tag{3.1}$$

Using this equation we can compute the one and two card look ahead potentials for this example from Table 3.2. PPot1, the one-card potential is 0.406, while the two card potential, PPot2, is 0.620.

```
HandPotential(ourcards,boardcards) {
  /* Hand potential array, each index represents ahead, tied, and behind.  */
  integer array HP[3][3]  /* initialize to 0 */
  integer array HPTotal[3] /* initialize to 0 */

  ourrank = Rank(ourcards,boardcards)
  /* Consider all two card combinations of the remaining cards for the opponent.*/
  for each case(oppcards) {
    opprank = Rank(oppcards,boardcards)
    if(ourrank>opprank)      index = ahead
    else if(ourrank=opprank) index = tied
    else /* < */             index = behind
    HPTotal[index] += 1

    /* All possible board cards to come.  */
    for each case(turn) {
      for each case(river) {
        /* Final 5-card board */
        board = [boardcards,turn,river]
        ourbest = Rank(ourcards,board)
        oppbest = Rank(oppcards,board)
        if(ourbest>oppbest)      HP[index][ahead ]+=1
        else if(ourbest==oppbest)HP[index][tied  ]+=1
        else /* < */             HP[index][behind]+=1
      }
    }
  }

  /* PPot:  were behind but moved ahead.  */
  PPot = (HP[behind][ahead] + HP[behind][tied]/2 + HP[tied][ahead]/2)
       / (HPTotal[behind]+HPTotal[tied]/2)
  /* NPot:  were ahead but fell behind.  */
  NPot = (HP[ahead][behind] + HP[tied][behind]/2 + HP[ahead][tied]/2)
       / (HPTotal[ahead]+HPTotal[tied]/2)
  return(PPot,NPot)
}
```

Figure 3.4: Hand potential calculation.

Before	After	Turn Cases	%	River Cases
Ahead	Ahead	4380	9%	89836
Ahead	Tied	0	0%	4449
Ahead	Behind	480	0.9%	12635
Tied	Ahead	81	0.17%	3240
Tied	Tied	324	0.67%	5625
Tied	Behind	0	0%	45
Behind	Ahead	17663	36%	590955
Behind	Tied	0	0%	3010
Behind	Behind	25717	53%	360395

Table 3.2: Breakdown of cases in a hand potential for the hand 7♡-9♡ with a flop of 8♡-6♣-4♡.

Just as hand strength can be computed by enumeration, so can hand potential. Figure 3.4 shows the enumeration algorithm for computing PPot and NPot. One card look ahead is computable on today's hardware (see Appendix A) in roughly 280 milliseconds. Two card look ahead can be done in 2000 milliseconds, which may limit its usage in practical algorithms for real-time play.

3.3.3 Weighting the Enumerations

The calculations of hand strength and hand potential in Figures 3.3 and 3.4 assume that all two card combinations are equally likely. However, the probability of each hand being played to a particular point in the game will vary. For example, the probability that the opponent holds Ace-King is much higher than 7-2 after the flop, because most players will fold 7-2 before the flop.

During the course of each hand, a *weight table* is maintained for each opponent. This is simply an array of values that represent the probability of the opponent holding each possible combination of hole cards. For each of the 1326 entries (one for each two-card combination)[2], we store a weight in the range 0.0 to 1.0. This weighted distribution is updated after each opponent action to reflect the hands that are consistent with the betting decisions observed throughout the current hand. The precise details of this *re-weighting* process depends on our method of modeling each opponent (described in Chapter 4).

The strength of each possible hand is assessed, and a probable distribution of actions is determined by a formula-based betting strategy. These values are then used to update the weight table after each opponent action. The algorithm is shown in Figure 3.5. For each possible opponent hand, the opponent model returns a probability distribution over the likely actions with that hand, given the current

[2]We use a 52 by 52 table for indexing convenience. Weights for all card combinations containing known cards (in our hand and on the board) are set to zero.

```
UpdateWeightTable(Action A, WeightTable WT, GameContext GC, OpponentModel OM) {
  foreach (entry E in WT){
    ProbabilityDistribution PT[FOLD, CALL, RAISE]
    PT = PredictOpponentAction(OM, E, GC)
    WT[E] = WT[E] * PT[A]
  }
}
```

Figure 3.5: The algorithm for updating a Weight Table. The probability of the observed action (A) is taken from an opponent model's (OM) prediction of the opponent (PT) based on the hand (E) and the current game context (GC). This probability (PT[A]) is used to weight the table entry (WT[E]).

game context. From this distribution, the probability of the actual observed action is used to multiply the weight table entry.

For example, suppose an opponent calls before the flop. The updated weight for the hand 7-2 might be 0.01, since it would normally be folded. The probability of Ace-King might be 0.40, since it would seldom be folded before the flop, but is often raised. The relative value for each hand is increased or decreased to be consistent with every opponent action.

Figure 3.6 shows the reweightings of an opponent's weight table. The images are constructed to show the 1326 different hands in a 13 by 13 table. The axes represent the rank of each of the cards. Suited hands are in the upper-right half of the table, unsuited hands are in the lower-left half, and pairs are along the diagonal. The shading in the cell represents the maximum value for the hands grouped in that cell. The darker the cell, the higher its weight. For instance, for the *Ace-King Suited* cell (in the upper-right half of the grid), its value is determined by the maximum weight of the four different Ace-King suited hands (Ace-King of Spades, Ace-King of Hearts, etc...).

From Figure 3.6, we can see the difference in the weights for an opponent that folds, calls or raises in the pre-flop betting round. In the case of folding, the strong and mediocre hands are all shaded lightly or not at all. The weak hands (low cards, unsuited, unconnected) are more likely to be folded and thus have high (dark) weights. For the calling scenario the high weights are in the middle – hands that are consistent with being called. Weak hands would likely have been folded and strong hands would likely have been raised, so their weights are low. Likewise, the raising example shows strong weights only for the very good hands, and low weights for everything else.

To better understand how the weights are updated, consider the weight tables in Figure 3.7, generated from the sample hand in Chapter 1.2.1. The weight tables for Alice on the flop (J♠-8♣-4♠), after she bets $10, might look something like in

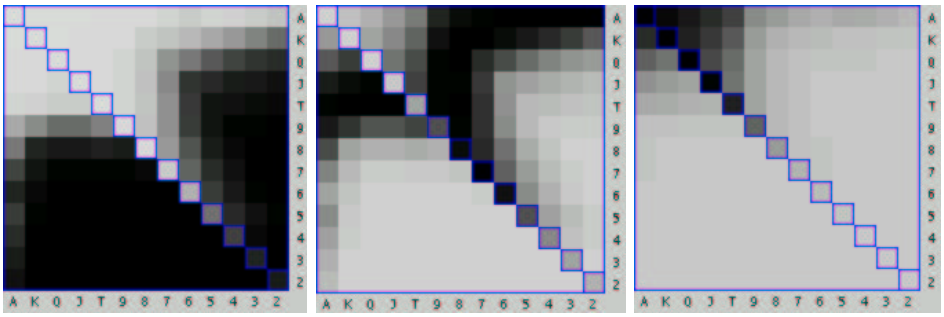

Figure 3.6: An example weight table after a pre-flop fold (left), call (middle), or raise (right).

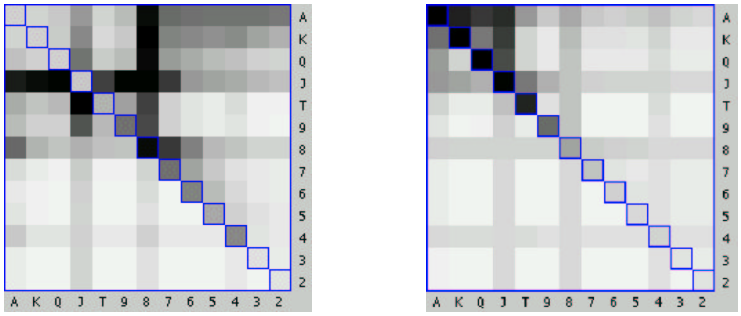

Figure 3.7: Two scenarios for calling pre-flop (left) and raising pre-flop (right), followed by betting after a flop of J♠-8♣-4♠ (See Chapter 1.2.1).

Figure 3.7. On the left is a table for the scenario where she calls pre-flop, and on the right is how the table would appear if she had raised pre-flop (as she did in the example).

The table for the pre-flop call scenario (left) represents hands such as any hand with a Jack (except for a pocket pair of Jacks, because that hand would have been raised pre-flop). Hands that make a pair of Eights are likely, if they have a large *kicker* (for example, King-Eight has a high weight, but Eight-Three does not). Hands that make two-pair (like Jack-Eight) also have relatively high weights. The upper-right triangle of the table represents suited hands, and because there are two spades on the board, the suited half of table is darker than the unsuited half – Alice could be betting a strong flush draw.

In the pre-flop raise scenario (right), There are several differences. Jack-Jack now has a high weight, since it would have been raised pre-flop, then bet on the flop. Other hands like Ace-Ace, and Ace-King are much higher than they were in the other scenario, since they too are hands that are more likely to be raised pre-flop rather than called. Hands like Four-Four are weaker than in the calling scenario, since Four-Four is less likely to be raised in the pre-flop round.

Noise-Factors

Each action a player makes conveys some form of information, however little, about the hand they are holding. The reweighting process attempts to modify the table based on this information. In order to do this accurately, we need to know how much information the observed action gives us, and how much this information should alter the contents of the weight table.

A *noise-factor* is used to dampen the amount of reweighting done to a table with each action. The noise-factor is a value between zero and one, that is extracted from the probability distribution $\{Pr(fold), Pr(call), Pr(raise)\}$ used in reweighting, and then smeared into the distribution equally. For each action a in the probability distribution D, a is updated by $a \Leftarrow a - a * \beta + \frac{\beta}{|D|}$, where β is the given noise factor.

A noise-factor of 0.0 leaves the probability triple untouched, whereas a noise-factor of 1.0 would flatten the triple into a uniform distribution. For instance, suppose we have a triple of {0.0, 0.2, 0.8} and adjust it with a noise-factor of 0.5. The resulting triple would become roughly {0.167, 0.267, 0.567}. That is, half of the distribution has been proportionally smeared toward uniformity (half of one-third is 0.167).

The noise-factor represents the amount of uncertainty we have about the opponent's actions reflecting information about the cards they hold. If an opponent always plays without deception, using a noise-factor would be detrimental. However, if an opponent plays deceptively, some uncertainty must be kept in the weight table.

3.3.4 Pot Odds

> *'A new car built by my company leaves somewhere traveling at 60 mph. The rear differential locks up. The car crashes and burns with everyone trapped inside. Now: should we initiate a recall?*
>
> *You take the number of vehicles in the field (A) and multiply it by the probable rate of failure (B), multiply the result by the average out-of-court settlement (C). A times B times C equals X. If X is less than the cost of a recall, we don't do one.'*
>
> *'Are there a lot of these kinds of accidents?'*
>
> *'You wouldn't believe.'*
>
> *'... Which car company do you work for?'*
>
> *'A major one.'*
>
> – Fight Club (1999)

Returning to our earlier example where Player A has K♣-K♠, and player B has 7♡-9♡, with a flop of 8♡-6♣-4♡, recall that B's hand has high potential, but weak strength. Should B call A's bet? Player B has a 32% chance of his draw coming in on the turn. If the pot size is currently $50, with $10 bet by player B, then it is correct for B to call the $10. B is getting a 32% chance to win with a 5-to-1 bet.

For a one bet investment, we will win 5 bets 32% of the time. The remaining 68% of the time we miss our draw – at worst we can fold and have lost only 1 bet each of these times. The return on our investment is $(\$50 \times 0.32) - (\$10 \times 0.68) = +\$9.20$.

There are several different ways to estimate pot odds. For instance, the *immediate pot-odds* (the current odds for calling a bet), is formulated as:

$$immediate\_pot\_odds = \frac{amount\_to\_call}{pot\_size + amount\_to\_call} \tag{3.2}$$

Using this equation, our pot odds are $(\$10/(\$50 + \$10)) \approx 0.167$. If we call in situations where we have more than a 16.7% chance of making a winning draw, then the expected payoff is positive. For instance, if we only had a 15% chance of winning, we would not have sufficient odds to call. We would win 5 bets only 15% of the time, and lose 1 bet 85% of the time for a net loss of $(\$50 \times 0.15) - (\$10 \times 0.85) = -\$1.00$.

Pot odds give the return on investment for calling a bet. We must have a chance of winning greater than the pot odds in order to win money.

Pot odds calculations should also take future betting rounds into account. If we make our hand on the next round, there is a good chance we will be able to win a pot much larger than the current size. The *implied pot odds* account for the payoff from winning future bets as well as the current pot. There is no simple formula to measure implied pot odds – players rely on judgment to calculate these odds.

Conversely, the *reverse implied odds* are used in situations where we are playing with a made-hand, but against an opponent who may be on a draw or may already have us beat. In the next betting round the opponent may miss their draw and fold, or they may make the draw or already have you beat in which case it will cost you more to see the showdown – increasing the cost of your investment to win the pot.

3.3.5 Multi-player Considerations

The *Hand Strength* computation gives the strength against one random hand. If the enumeration is weighted, it gives the strength against a player with the hand distribution reflected by the weight table. If we are up against more than one opponent then we must, of course, beat all of them. If a weight table is kept for each opponent, then a good approximation is to multiply the hand strengths for each weight table together. This is the probability that we have a hand better than player one, *and* player two, through to player n.

$$HS_n = HS_{p1} * HS_{p2} * \ldots * HS_{pn} \tag{3.3}$$

This is not an exact value, because the hand strengths of the opponents are not independent of each other. For instance, the weight tables do not reflect the fact that two players cannot hold the same cards. However, the full computation is too complicated for a real-time system. Tests comparing the two methods in [34] showed that the error from Equation 3.3 was small (less than 3%).

Another data structure that is used for multi-player reasoning is the *field array*. This is another weight table that represents the average of all opponent weight tables. This allows us to reason about the whole field of opponents at once, and is a further approximation that reduces accuracy in exchange for speed. For instance, a hand strength against the field array, raised to the power of the number of opponents, can be a reasonable estimate of strength.

$$HS_n = (HS_{field})^n \tag{3.4}$$

The field array has been found to be a large source of error in *Loki*. Recent versions of *Poki* have eliminated most of the dependence on the field array, using it only when the regular calculations are too expensive.

A fundamental problem with the field array in *Loki* is that it is an average of the opponent weight tables. This seems to be the wrong abstraction. Suppose we are up against four opponents, and one has a very high probability of holding a strong hand such as a pair of Aces. Further, suppose the three other opponents have very low probabilities of having such a hand, but instead, they all have mediocre hands represented in their weight tables. If the field array is constructed from the average of the four tables, hands like a pair of Aces will have a low value (say, 0.25), while the mediocre hands held by the majority of opponents will have relatively high weights (say, 0.75). This biases the field array in favor of having the weaker hands. What matters most, of course, is whether we can beat the *best* hand in the field. If we cannot beat the pair of aces held by one opponent, it does not matter that we can beat the three other opponents.

Poki has been enhanced by constructing the field array from the maximum of each entry in the table, instead of the average, which is a more realistic approximation. Also, in many places where the field array was used in the past, it has been replaced with the more expensive, but more accurate Hand Strength calculation (3.3).

3.3.6 Effective Hand Strength

For purposes of simplification it can be useful to have a single value for the overall strength of a hand that includes potential. What we want is an estimation of the

probability that we will hold the best hand after all the cards have been dealt. We call this the *effective hand strength* (EHS). Equation 3.5 combines the hand strength, negative potential and positive potential, into a single value.

$$EHS = HS \times (1 - NPot) \quad + \quad (1 - HS) \times PPot \tag{3.5}$$

However, in practice, we remove the negative potential from the measure and use Equation 3.6. At first this may seem counter-intuitive. Negative potential is the chance that we are currently ahead, but further board cards will improve the opponent's hand enough to beat ours. When the negative potential is high it is usually necessary to bet anyways since the *free card* danger is high. We must protect our hand and force drawing hands to pay for their draw.

$$EHS = HS \quad + \quad (1 - HS) \times PPot \tag{3.6}$$

3.4 Basic Betting Strategy

The basic betting strategy is a fairly simple formula-based system that uses the hand evaluation described above as input and outputs an appropriate betting action.

First we evaluate our Hand Strength, Potential, and Pot Odds. The formula generates a *probability triple* over the available actions, $\{Pr(fold), Pr(call), Pr(raise)\}$. For instance, a very strong hand might generate a triple such as {0.0, 0.05, 0.95}. This means that in this situation we should call five percent of the time, raise ninety-five percent of the time, and we should never fold. To make a betting decision, a uniform random number is generated between 0 and 1. This value is used to pick an action from the probability distribution.

Randomization is important to make the strategy less predictable [35]. If we always raised with strong hands, folded weak hands, and called drawing hands, we would quickly get taken advantage of by any perceptive opponent. When we raised, opponents would learn to fold, giving us only small pots. When we don't bet, opponents would know to raise, forcing us to fold our weak or drawing hand. By using a properly mixed strategy, we can hide the strength of our hands.

The formula-based strategy makes an attempt to do this, but it is far from perfect. It is also sensitive to the accuracy of the input. If our opponent modeling is poor, the values generated for strength and potential will be bad estimates.

For many hands in Hold'em, the obvious strategy is sufficient. If we have a extremely strong hand ($EHS > 0.85$), we should almost always raise, with some percentage of check-raises. If we have a very weak hand ($EHS < 0.25$), we should almost always fold (unless we have a high potential that gives us correct pot-odds to call, or we think a bluff is in order). Somewhere in the middle, we should mix

our strategy to add unpredictability. Some moderate hands should occasionally be raised – this way, the opponent cannot be sure that we have a strong hand when we raise.

The formula is based around two thresholds for calling and raising. For hands that are well above the raise threshold, we raise nearly all of the time, and for hands well below the calling threshold, we almost always fold. Likewise, hands well in the mid-range are almost always called. The hands that are near the thresholds are treated differently. These borderline hands are mixed between calling and raising, or folding and calling, in proportion to their strengths.

This simple betting strategy plays a fairly solid and safe game, but it is too predictable. Any fixed formula will eventually show exploitable weaknesses (unless it is optimal). The formula could be developed further and further, patching weakness and adding more intelligence. This, however, is of minor scientific interest, as it would be little more than an expert system. Not much work has been done to improve the formula based system since *Loki-2* [35]. The formula was only ever intended as a first-cut approximation. It should have been revisited and improved to incorporate more knowledge and information hiding, but it never was. A fully adaptive knowledge-free system is of much more interest as the techniques could be applied more broadly to imperfect information domains.

3.5 Selective Sampling and Simulation Betting Strategy

In perfect information games like chess, minimax game-tree search algorithms such as alpha-beta [38] provide a dynamic method for discovering good moves based on minimal knowledge. The formula based betting strategy described in the previous section can be thought of as the equivalent of a static evaluation function in deterministic perfect information games. Since game-tree search works so well in these domains, the natural step is to test something analogous to game-tree search in the poker domain.

Unfortunately standard game-tree search algorithms like alpha-beta do not work for poker. The nature of the imperfect information game tree makes traditional search of little utility.

In most two-player perfect information games it is effective to search the game-tree several moves ahead, and then call an evaluation function. In these games it is possible to recursively search to find the opponent's best response to our moves. In poker the best response for the opponent is unknown because we do not know what cards they are holding. The minimax principle does not apply because nodes of the game tree are not independent. It is insufficient to call an evaluation function part-way through the tree, and the game-tree is generally too large to search completely

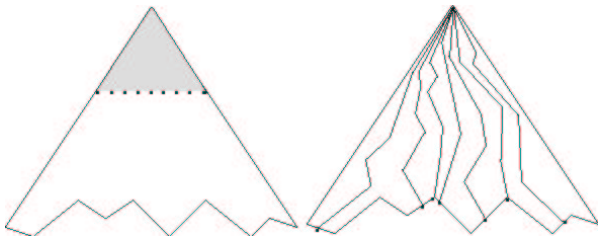

Figure 3.8: The difference between classic game-tree search and simulations.

to the leaves.[3] The leaf nodes of a poker game tree can be evaluated exactly, since the game-state is fully known. However, little can be inferred about the value of internal nodes without searching first to the leaves.

A technique known as *Selective Sampling* can be used to assign probable values to the unknown variables (in poker, the unknown cards). Instead of doing a comprehensive but shallow search, a deep but sparse simulation can probe numerous times to the leaves. Each trial of the simulation selectively samples the search space. With enough trials the cumulative selective sample converges to a statistically confident result.

Selective sampling has been used successfully in other games such as Scrabble [43], backgammon [48], and bridge [20]. *Likelihood weighting*, a similar technique used for Bayesian inferencing algorithms, is another way to bias stochastic simulations [19, 42].

Figure 3.8 shows the difference between standard game-tree search algorithms like alpha-beta (left), compared to the selective sampling simulation method. Alpha-beta will search only an upper region of the tree as deeply as possible, but there is essentially full coverage of the game-tree in the upper region. An evaluation is then applied to the nodes on the frontier. Simulations, however, search a selection of lines to the leaf nodes of the tree. Each line searched samples the game-tree in order to get a statistically relevant expected value for our moves [7].

When *Poki* is faced with a betting decision, it can simulate each of its possible actions, and then simulate the game to the end of the hand. Since we don't know the complete game-state (the cards of our opponents, and the future board cards remain unknown), we must assign probable cards to each player, in each *trial* of the simulation. The weight table of each opponent can be used to selectively bias the chosen cards to match the hands they are likely to hold [35, 8].

After simulating several hundred trials, the average amount won or lost by folding, calling, and raising will typically converge to a stable estimate of the *Expected Values* (EVs) of each action. The action with the largest EV can be chosen as the

[3]For example, Darse Billings has estimated the two-player Texas Hold'em game-tree to be 3×10^{17} in size (counting the number of terminating nodes).

```
simulate() {
   trials = callEV = raiseEV = 0;
   while (trials < MAX_TRIALS) {
      assignCards();
      callEV += simulateHand(call);
      raiseEV += simulateHand(raise);
      trials++;
   }
   callEV = callEV/trials;
   raiseEV = raiseEV/trials;

   return (callEV,raiseEV)
}
```

Figure 3.9: Pseudo-code for poker simulations.

maximal action to take. This, of course, is not a proper mixed-strategy (unless the opponent modeling used in the simulations is sophisticated enough to take the effect of mixed-strategies into account). In practice, the problem of converting the EVs to produce an appropriate mixed strategy remains.

A simulation-based betting strategy for poker proceeds as follows. For each *trial*, probable hands must be assigned to each opponent to hold. These hands can be picked from the distribution of hands in each opponent's weight table. We also choose hands for players that have already folded. This effectively changes the composition of the remaining deck by removing cards that are consistent with folded hands. Once each simulated player has a hand, the unknown board cards are chosen randomly from the remaining cards in the deck.

For each trial, the hand is simulated twice. In the first simulation, *Poki's* first action will be a call (or check if there is no bet). In the second simulation, *Poki* will raise (or bet) for its first action. Once the first action is made, the rest of the hand is simulated to completion. When a simulated opponent must act, their opponent model is asked to predict what the opponent would likely do in the current context, with the hand we have assigned to them. Once the simulated hand is finished, the net amount won or lost is recorded.

After many of these trials the average EV for calling, and raising becomes fairly stable. This must be accomplished within the time constraints of a poker game (usually one or two seconds), limiting the number of trials (typically 200-300 trials for flop decisions).

This technique has several benefits over the formula-based system. It is not a fixed strategy – depending on the opponent models, it can shift dramatically, and adapt to any mix of different opponents. Computing properties like implied pot-odds and draw-odds can be difficult and expensive. Simulations uncover this information naturally, and produce a simple EV for each action. Furthermore, simulations can uncover complex strategies without any specific expert knowledge. For example, tactics such as *check-raising*, *slowplaying*, and *bluffing*, are discovered

as an emergent property of the simulations.

The mix of possible cards throughout the trials helps provide realistic estimates of the value of our hand and actions. This process can work effectively, but it is also sensitive to errors and biases both in the simulation, and in the opponent models. Consider the situation where an opponent model incorrectly predicts that the opponent will fold 30% of the time in a particular context, but in actuality the opponent folds only 20% of the time. 10% of the simulations will erronously predict that by raising, *Poki* will immediately win the money in the pot uncontested. This bias will greatly inflate the EV for raising and *Poki's* resulting behaviour will be far too aggressive. Or consider a situation where the opponent model's weight table is representing mostly mediocre hands, when in fact the opponent has *slowplayed* a very good hand. The simulations will consistently pick the wrong set of cards for that opponent, making the EVs unreliable and resulting in a greater loss.

However, given a correct opponent model, the simulation-based betting strategy works well. In general, improvements in the opponent models yield improvements in the accuracy of the simulations. Simulations are sound in principle, but they have not been robust in practice. Chapter 6 will revisit the problems with simulations and present a more robust formulation based on our new *Miximax* algorithm.

3.6 Summary

Poki is a complete reimplementation of the *Loki* AI. The fundamental algorithms for determining hand strength and hand potential involve weighted enumerations of the unknown cards. Weight tables are maintained for each opponent. These tables reflect the hands consistent with their actions throughout the hand. Two different betting strategies have been used to choose actions based on our hand evaluation. The formula based strategy is simple and well tested. The simulation based strategy is a much more adaptive strategy, but it is dependant on quality opponent modeling to get good results. The opponent modeling components of *Poki* will be discussed in the next chapter.