PROJECT III—POKER AGENT

--SELECTED EXCERPTS--

ARTIFICIAL INTELLIGENCE

RICHARD BANISTER, REETI KUMAR, KARL SHOULER

MAY 8, 2008

UTILITY AGENT—BILLY

The second approach uses a basic utility measurement to generate actions. In this paradigm, the agent chooses whichever action is expected to yield the highest utility, or agent “happiness.” An agent nicknamed “Billy” was created to play poker on these principles.

WHY UTILITY?

In a sense the utility approach is a modified generalized search algorithm. We did not opt for a game tree search algorithm, as the tree would have had a branch factor of 5 and so one ply in a four player game would have required 5^4 nodes. This would have quickly exhausted available memory resources when attempting to generate more plys. Another option, planning, would have required us to specify an initial state and a goal state. Although the initial state could be easily identified, defining the goal state would have been a non-trivial task. Furthermore, the planning methods available to us would not account the uncertainty of the poker environment (namely its partially observable and strategic nature). In the end, we pursued a utility approach, as it takes into account the benefits of given states reached by performing a particular action and can accommodate the probabilistic nature of poker; furthermore we felt we could implement a utility approach with the knowledge available to the agent and the tools we had.

METHOD

For each action available to an agent, a numerical value representing the expected utility of taking that action is estimated. In general, the possible actions for each round are `:fold`, `:check`, `:call`, `:raise` *`c`* `amount`, and `:allin`. At some points in the game some actions are unavailable. For example, one of the constraints of the game engine is that only three bets per round are allowed; if a player tried to raise after this limit has been reached they are coerced into a `:fold`. `Billy` takes this situation into account by equating the utility of raising at that point with the utility of folding. Other specific situations are handled similarly.

`Billy` determines the estimated utility of an action by the following method:

```
EU(a) = Avg [ P( Result(a) | Do(a)) * EU (Result(a))]
```

That is, the estimated utility of an action `a` is the average of the weighted outcomes of doing `a`. Each weighted outcome is given by the probability of getting the result multiplied by the utility of the result. This is inherently recursive, as the utility of a state may be taken as the highest expected utility of possible actions from that state. `Billy` limits this recursion to a certain depth. Results are the expected outcomes of performing an action. `Billy` takes outcomes as being possible positions it will be in at its next turn. For example, if `Billy` decides to `:allin`, there are a few possible results: all other players will `:allin`, some players will `:allin` and some will `:fold`, and all other players will `:fold`. In the first case, the next state `Billy` encounters will see the pot grown by `Billy`'s raise for each player in the game (this, is of course, a simplification as other players may not be able to cover the bet. Further logic could be added to add accuracy here). `Billy` will also have a much higher stake in the game, which should negatively affect the "happiness" by adding some "anxiety."

Probability values were estimated and not based on any data. These values were manually revised after watching `Billy` play. Some alterations in probability significantly affected performance, as might be expected. For example, if we believe the probability that every player will fold if we `:allin` is very high, the expected utility of `:allin` will increase accordingly. However, it is not likely the case that this actually will happen in play, especially against opponents that may not be explicitly aware of other agents' actions. This expected probability must then be reduced. This would be an excellent place to incorporate machine learning techniques; adjusting probabilities of actions on the fly according to observations of current opponents could be a very valuable technique.

UTILITY FUNCTION

The base utility of a state is taken when `Billy` comes to an outcome where there are no more possible moves, or when the search depth limit is reached. For example, there are no possible moves after a `:fold` so utility is measured directly in that case. The `Billy` software allows utility functions to be plugged in to the basic expected utility algorithm. A number of different functions were tested, with the one used in competition given as:

```
U(s) = COW * ln [ pot / bank ]
```

Where `COW` is the chance of winning, pot is the amount of money that stands to be won, and bank is the amount of money left in the player's bank. `COW` is generally given by the hand strength measurement as discussed previously, but in some cases also takes folding players into account. `ln [ pot / bank ]` is a simple representation of what the player stands to gain in this round, henceforth simply called `gain`.

Appropriate utility functions must meet the following requirements:

```
U( high COW ^ high gain ) > U( high COW ^ low gain ) >
U( low COW ^ low loss) > U( low COW ^ high loss)
```

The reasoning behind this is that the agent should be happiest when more likely to win a lot of money. If the agent is likely to win but there is low gain, there is a path to increase possible gains. If the agent is likely to lose, its best bet is to minimize losses. And if the agent has a low chance of winning, there may be a path to increasing those odds by bluffing.

The utility function given attempts to adhere to the given requirements by multiplying `COW` with a simple representation of gain, given by the natural log of `pot/bank`. When the player has a lot of money in the bank, `gain` is taken to be smaller. However, when the player has little money, every dollar is important so `gain` should be higher. A natural log is taken to simulate the nature of monetary desire, in which small differences in large numbers mean less than small differences in small numbers.

PERFORMANCE AND EVALUATION

In matches against `Freddy` and two dummy agents, `Billy` won fairly consistently. The specific circumstances of many such wins are not known, although matches ranged from very short to very long. It is possible that actions taken by `Freddy`, who is somewhat reckless, had significant effects on `Billy`'s chances of winning. In one-on-one matches against `Freddy`, both agents won about half the time. In the tournament, `Billy` pulled a daring `:allin` early in the first game but was defeated. `Billy` proved to be quite aggressive in the second game, pulling ahead before the engine crashed.

`Billy` tends to raise a lot, with large amounts, and has a strong aversion to folding. `Billy` also exhibits a self-destructive tendency to go all-in at ill-advised times. `Billy` certainly does not play perfect poker. There are a few likely reasons for this. The most glaring problem is the deficiency of the utility function. It is not guaranteed to meet the stated requirements. It does not accurately reflect the concept of "losses," and for this reason does not allow for the minimization of losses through folding in the case of a weak hand. This is obviously critical functionality for successful play. Testing revealed that the expected utility of folding a weak hand did not tend to exceed the expected utility of calling, or even raising. This leads to an agent that plays every hand. It is possible that `Billy`'s aggressive nature in raising is a form of bluffing on weak hands, in the hopes that some players will fold. This was not an explicitly programmed goal; if it is true, then it may come from `Billy`'s own analysis.

A second deficiency is in accuracy of numbers used in the algorithm. In particular, there was no basis in actual data for the probabilities of expected outcomes, which leads to skewed ideas of what the future will hold – likely a significantly worse outcome with a statistically higher probability. More accurate values would be needed here to improve play. Hand strength is also just a rough estimation; getting true numerical accuracy would require a much more precise estimate. In the case of `Billy` this was not required due to the low precision of other factors, such as the outcome probabilities. Some alterations of `COW` are done in the utility estimation when players are expected to `:fold`, with the idea that chances of winning increase with fewer players. However, this is also a very rough estimate, which is done in favor of re-running the expensive hand strength function. This estimation likely overvalues the utility of a player leaving and should be replaced with a more accurate strength lookup.

The amount of a raise is also not a well-reasoned process. A raise is determined by randomly adding some multiplier to the blind – up to five times as much. Ideally a raise amount would be reasoned according to how strong the agent's position is and what the agent wants other players to believe about their position. This was not taken into account, however, and thus `Billy` tends to raise quite aggressively.