IT3105 Project I: Texas Hold ‘Em

Student: *André Christoffer Andersen*
Username: *andrecan*
Date: *2011-09-28*

1 Introduction

In this assignment we will explore three different autonomous poker bot which have increasing degree of complexity. The basic poker bot uses simple hand ranking as a basis for decisions, while the medium and advanced uses simulations rollouts to calculate the probability of success. The advance poker bot also implements a player model that aims to capture the opponents hand strength basted on the context and betting behavior. We will see that the complexity and lack of tuning makes the advanced poker bot unable to beat the simpler medium poker bot.

2 Code Structure

The code for this project was written in Java 1.6. It is divided in to multiple packages, most noteworthy, card, game, log, logic, player and opponent.

2.1 The Card Package

The card package houses typical playing card data structures which are used through the project. The individual playing cards are represented by the class Card which contains a unique integer id for each possible card. In increasing value, the clovers range from 0 to 12, diamonds 13 to 25, hearts 26 to 38 and spades 39 to 51. The main card collection is class Cards which is a superclass of multiple other card collections like Board (common cards), Deck and Pocket. Each card collection has added functionality as needed, e.g., the class Deck has an initial shuffling function.

2.2 The Game and Log Package

The game package houses the central Game class. It implements the main functionality of a fully functioning poker simulator. When this class is initialized as an object it receives all necessary data to carry out a simulation. It contains the players, common resources and constants. However, it does not directly contain the game state. Instead, it delegates the current state to the Hand class which is the basis for a disposable Hand object that contains the current state of the playing hand of poker, e.g., Deck, Pot, Board, nonFoldedPlayers, etc. Hand objects are convenient for passing game states around. However, the Hand class should not be confused with the GameState class in the log package. The GameState and PlayerState classes are pure information containers used for exporting data. It is not used at this time. The GameCounter class is used to keep track of wins, losses and ties, and can compute ratios as well – all for logging purposes.

2.3 The Logic Package

The logic package contains the most important computationally heavy code. The HandEvaluator class takes care of hand ranking and tie breaking. The HandStrengthSim class simulates multiple post-flop hands in order to fix a numerical value to its strength. This is done as follows

$$\left(\frac{Wins + \frac{Ties}{2}}{Wins + Ties + Losses} \right)^k$$

Next, the logic package houses the RolloutSim class which performs the desired pre-flop simulations for calculating winning probability of pocket cards. This class also takes care of saving and loading of these results to disk.

2.4 The Player Package

The player package contains multiple player AIs that all extends the Player class. The most important common denominator is the makeAction(…) method. This method is called by a Game object for every betting round. Both a Hand and Game object is passed to this method as input with an enum Action as output with the following possible values FOLD, CALL, RAISE and NONE. The final poker bots for each project phase are implemented, respectively, by the BasicPlayer, MediumPlayer and AdvancedPlayer classes. In addition there is a HumanPlayer for testing purposes, and various other experimental bots and dummies.

2.5 The Opponent Package

Because of the larger scope of the poker bot form Phase III, I chose to put its unique logic in a separate opponent package. It handles the context-action pair data structure and collects them in the multiple OpponentModel classes. The OpponentModel objects contains a HashMap container of context-action pairs that point to a list of Double hand strengths. The OpponentModel can take a context-action object as input and output the mean hand strength of the mentioned list.

2.6 The Tests Package

Finally, there is a test package with tests for various hand rankings, tie breaking and nitpicking code.

3 Player Logic

The basic, medium and advanced players build on each other respectively. To simplify matters we will only allow fixed rising – the minimum raising amount allowable – the big blind.

3.1 Basic Player – Project Phase I

The basic player is a very rudimentary player, indeed. For the preflop phase it randomly (uniformly) picks an action. For the postflop phases it simply chooses to RAISE if it actually has a better than three-of-a-kind hand, and it CALLS if it has one or two pairs, thus FOLDing when only having a high card. There are no risk profiles for the basic player.

3.2 Medium Player – Project Phase II

The medium player uses pre-flop and post-flop simulations to calculate probability measures. The pre-flop rollout gives a measurement for pocket strength and the online post-flop calculations yield the hand strength. To make decisions based on them we need two limits that separate the actions RAISE, CALL and FOLD. These limits were defined for three degrees of risk aversion.

	RISK AVERSION		
	HIGH	MEDIUM	LOW
CALL/RAISE	.90	.65	.40
FOLD/CALL	.60	.35	.10

The pre-flop rollout probability calculations are done over 10.000 rollouts, once for each possible number of players. They were then stored for later use, and automatically regenerated if missing.

3.3 Advanced Player – Project Phase III

The advanced player uses an opponent models in order to predict what hand strength the opponents have based on their previous games. Additionally the advanced player uses tricks and rules of thumb in order to improve winnings.

3.4 Opponent Model

There is one opponent model per player. These models are shared by all advanced players, since they are based on common knowledge from showdowns. The context features which are used are as follows.

- *Last action*: The action a player used is very indicative of what sort of hand they have.
- *Game phase*: The phase, or sometimes called turn, should provide information that might help in determining the hand strength of a player. Late games are different than early games.
- *Pot odds level*: The pot odds is a measure of how much you have to invest in order to stay in the game compared to the total pot. Here we represent this by making it discrete: HUGE, HIGH, MEDIUM and LOW.
- *Players left in game*: The number of players left seems like a something that would make a player change their betting behavior. Someone might be inclined to be more risky at the end stages of a hand.

The context collection procedure starts with continuously filling up a collection of Context-Action-HandStrength triplets. When a hand is completed and a showdown occurs the collection is added to the respective opponent models for collective use. Contexts from player who have folded are discarded.

When a model is needed for hand strength prediction it is done as context-action query to the correct model. The model fetches the context and retrieves a list of hand strengths which are averaged for use.

This hand strength is used to compare to the model user in question's own estimated hand strength. There is no clear cut way of determining what action to take based on this. None the less you might make rules such that you RAISE if you have a better estimated hand than 80% of the other players or just HOLD when you only a better hand than 50% of the other players. Based on the chosen risk aversion the following were rule was used.

	RISK AVERSION		
	HIGH	MEDIUM	LOW
CALL/RAISE	.8	.65	.50
FOLD/CALL	.5	.35	.20

Note that this is not hand strength thresholds as with the medium player. It is a measure of how many you need to beat in order to stay in the game. Of course, if you have too few data points for a context we revert to using the medium players risk aversion profile applied to hand strengths directly.

3.5 Tricks and Rules of Thumb

- If the advanced player is the big blind it makes some sense to not FOLD even with a bad hand, since this betting round is sunk cost. This is taken in to consideration.
- A random error is added to the risk aversion thresholds. This is a kind of bluff factor that can put of other opponent models.
- If there are few hand strengths for a given context it is ignored and we fall back to the medium player play style.

4 Results

4.1 Basic Player Results – Project Phase I

In the table below bp0 to bp3 stands for Basic Player 0 to 3. All five simulation runs had identical setup, and ran for 1000 hands.

	bp0	bp1	bp2	bp3
Wins	247	247	236	236
Ties	18	21	13	18
Losses	735	732	751	746
Shows	609	621	609	603
Win rate	0,25	0,25	0,24	0,24
Chips	2595	4006	-4809	2208

	bp0	bp1	bp2	bp3
Wins	224	242	248	256
Ties	18	16	15	13
Losses	758	742	737	731
Shows	589	614	623	601
Win rate	0,23	0,25	0,25	0,26
Chips	-54	-3113	-818	7985

	bp0	bp1	bp2	bp3
Wins	226	276	235	220
Ties	25	21	24	19
Losses	749	703	741	761
Shows	604	642	589	598
Win rate	0,23	0,28	0,24	0,22
Chips	-251	5848	519	-2116

	bp0	bp1	bp2	bp3
Wins	243	255	242	225
Ties	17	17	24	17
Losses	740	728	734	758
Shows	618	626	632	603
Win rate	0,25	0,26	0,25	0,23
Chips	1312	3854	-1240	74

	bp0	bp1	bp2	bp3
Wins	239	243	234	250
Ties	17	20	19	17
Losses	744	737	747	733
Shows	623	632	652	622
Win rate	0,24	0,25	0,24	0,25
Chips	2988	6	-4860	5866

4.2 Medium Player Results – Project Phase II

The following four simulations show how the mentioned risk aversion profiles bode against each other. There are two of each risk profiles, totaling 6 players. “mp0h” stands for Medium Player 0 - Highly Risk, etc.

	mp0-h	mp1-h	mp0-m	mp1-m	mp0-l	mp1-l
Wins	40	34	172	172	231	316
Ties	0	0	15	17	19	19
Losses	960	966	813	811	750	665
Shows	58	42	283	317	556	650
Win rate	0,04	0,03	0,17	0,17	0,24	0,32
Chips	742	2015	2389	-575	-1911	3340

	mp0-h	mp1-h	mp0-m	mp1-m	mp0-l	mp1-l
Wins	38	36	178	208	251	255
Ties	1	0	19	9	17	22
Losses	961	964	803	783	732	723
Shows	53	44	326	325	552	612
Win rate	0,04	0,04	0,18	0,21	0,26	0,26
Chips	1277	1759	170	3861	423	-1490

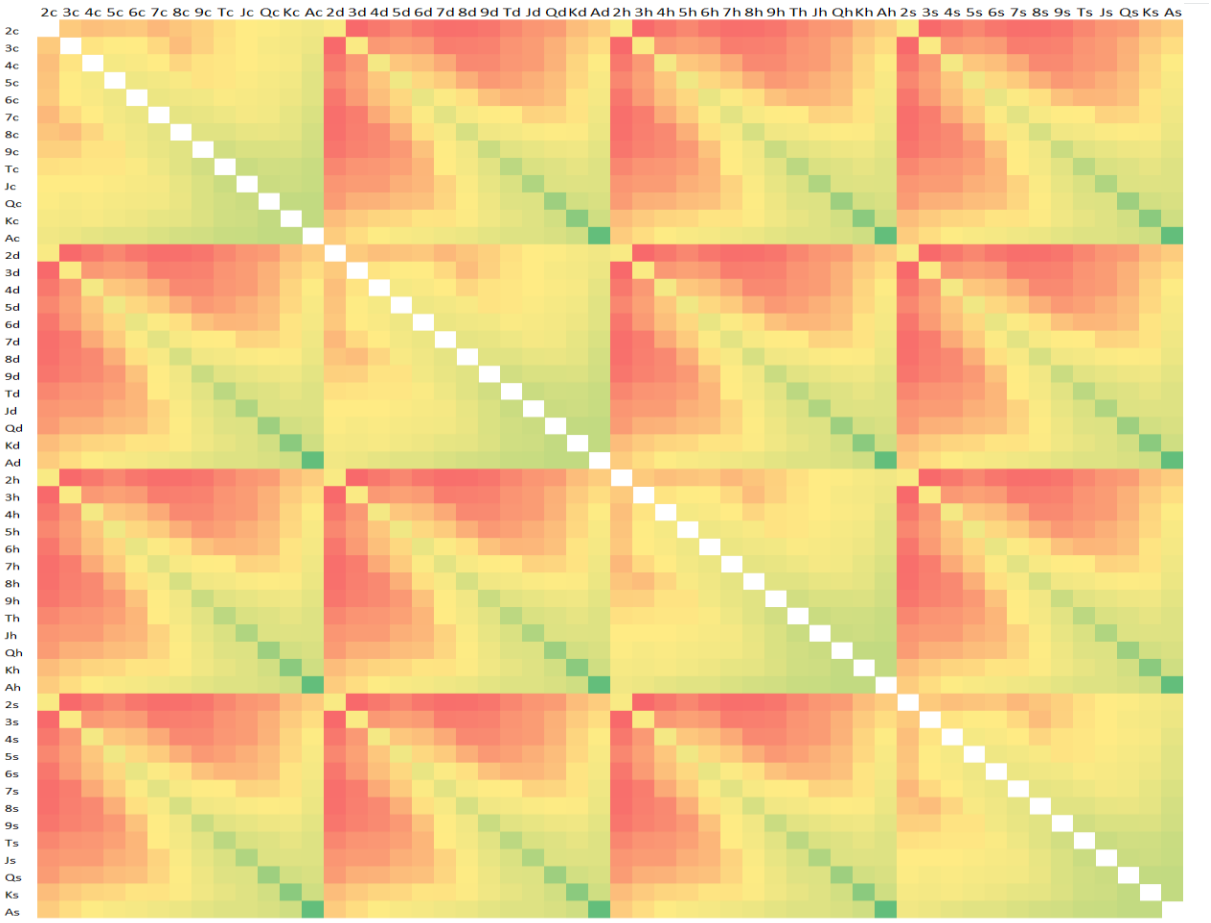
	mp0-h	mp1-h	mp0-m	mp1-m	mp0-l	mp1-l
Wins	33	45	204	181	273	239
Ties	0	1	12	6	19	13
Losses	967	954	784	813	708	748
Shows	45	62	343	298	605	557
Win rate	0,03	0,05	0,21	0,18	0,28	0,24
Chips	1574	1112	2795	2059	203	-1743

	mp0-h	mp1-h	mp0-m	mp1-m	mp0-l	mp1-l
Wins	40	35	183	209	272	236
Ties	1	0	8	7	14	20
Losses	959	965	809	784	714	744
Shows	52	45	317	334	588	570
Win rate	0,04	0,04	0,18	0,21	0,28	0,24
Chips	1416	1032	1425	1601	778	-252

Next, we have a simulation where we play three medium players of against three basic players.

	mp0-h	mp0-m	mp0-l	bp0	bp1	bp2
Wins	21	172	307	150	161	159
Ties	0	7	18	15	10	10
Losses	979	821	675	835	829	831
Shows	27	238	553	555	553	575
Win rate	0,02	0,17	0,31	0,15	0,16	0,16
Chips	1199	15434	21628	-10593	-10071	-11597

The following heat map illustrates how the preflop rollout simulations for pocket strengths, i.e., winning probability, are distributed.



4.3 Advanced Player Results – Project Phase III

The last tables show various advanced player versus the medium players. The advanced payers were given a 2000-hand game to build contexts from, and then applied on the follwing five 1000-hand games.

	mp-h	mp-m	mp-l	ap-h	ap-m	ap-l	ap-hn	ap-mn	ap-ln
Wins	14	213	309	19	92	132	34	83	97
Ties	0	2	4	0	2	3	0	1	2
Losses	986	785	687	981	906	865	966	916	901
Shows	17	338	542	22	98	155	35	94	128
Win rate	0,01	0,21	0,31	0,02	0,09	0,13	0,03	0,08	0,1
Chips	1265	3143	6555	-1552	-893	1305	-522	-469	168

	mp-h	mp-m	mp-l	ap-h	ap-m	ap-l	ap-hn	ap-mn	ap-ln
Wins	6	218	295	11	58	133	18	110	139
Ties	0	6	8	0	2	2	1	1	5
Losses	994	776	697	989	940	865	981	889	856
Shows	8	296	527	13	72	162	21	119	168
Win rate	0,01	0,22	0,3	0,01	0,06	0,13	0,02	0,11	0,14
Chips	1253	6704	9341	-4703	-3173	-55	-3266	1691	1208

	mp-h	mp-m	mp-l	ap-h	ap-m	ap-l	ap-hn	ap-mn	ap-ln
Wins	16	193	320	7	84	132	34	76	127
Ties	0	8	8	2	0	1	0	2	1
Losses	984	799	672	991	916	867	966	922	872
Shows	20	291	557	9	93	158	36	88	154
Win rate	0,02	0,19	0,32	0,01	0,08	0,13	0,03	0,08	0,13
Chips	1752	4039	6992	-2493	1	1263	-2323	-517	286

	mp-h	mp-m	mp-l	ap-h	ap-m	ap-l	ap-hn	ap-mn	ap-ln
Wins	7	195	343	6	82	103	19	85	146
Ties	0	5	13	0	1	2	0	5	2
Losses	993	800	644	994	917	895	981	910	852
Shows	8	298	575	7	91	127	20	106	167
Win rate	0,01	0,2	0,35	0,01	0,08	0,1	0,02	0,09	0,15
Chips	1187	3276	8617	-2261	-1285	662	-1124	-2921	2849

	mp-h	mp-m	mp-l	ap-h	ap-m	ap-l	ap-hn	ap-mn	ap-ln
Wins	10	204	351	9	67	99	35	65	146
Ties	0	5	10	0	1	4	0	1	7
Losses	990	791	639	991	932	897	965	934	847
Shows	12	299	587	11	75	124	38	72	173
Win rate	0,01	0,21	0,35	0,01	0,07	0,1	0,04	0,07	0,15
Chips	1297	3051	9260	-2529	-18	-694	-1740	-1054	1427

5 Discussion and Conclusion

Even though you'd think that the advanced player might have an edge over the basic and medium player it is clear from the results that the medium player is better than both the basic and the advance player. It also seems to be the less risk averse medium player who does best, that is, the more aggressive player. This is true against both for the basic and advance player. There also seems to be a looser coupling between win rate and chips acquired than expected. The highly risk adverse medium player won only 1% of the games, yet, was often in the middle of the pack when it comes to chips. The advanced player has a very low win rate, however, this might not be such a bad thing since you can often do well by playing safe and waiting for the big pay outs. This of course works on one type of opponent. Playing manually against the basic player quite easy knows that it is purely and simply procedural. An interesting ploy to get around the risk aversion problem might be to vary it randomly. Furthermore a more systematic approach towards bluffing might be reasonable.

The main problem with the advance player seems to be, just that, it is too advance. The underlying complexity requires much more calibration and tuning. If I were to redo the project I might have scrapped the context approach all together in favor of nearest neighbor or logistic regression, or even a neural network. Another thing I might have done differently is to use a better suited programming language. Even though Java is something we all know, it might have been better to just use Python or even Matlab.