Computer games

Report for the project:

Poker bot

<table>
<thead>
<tr>
<th>Program and year</th>
<th>Computer Science, IV rok</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecturer</td>
<td>Ph. D. Bartosz Ziółko</td>
</tr>
<tr>
<td>Student</td>
<td>Paweł Kwiecień</td>
</tr>
<tr>
<td>Course and lab time</td>
<td>Computer Games, Thursday 13.30</td>
</tr>
<tr>
<td>Source code</td>
<td><a href="http://149.156.196.114/mansion-svn/PokerBot_Kwiecien/">http://149.156.196.114/mansion-svn/PokerBot_Kwiecien/</a></td>
</tr>
</tbody>
</table>
The aim of this project was to create an efficient, smart Poker bot. Poker bot is a computer program designed to play the game of poker against human player or other computer players.

Poker is a game of incomplete information, which makes predicting the final result and value of the hand impossible. Poker bots are based on many techniques such as Bayes theorem, neural networks, poker strategies, but all of them are imperfect.

I have implemented several poker bots. Again, all of them were imperfect due to the rules of the game.

**Timeline:**

First of all, through the semester, few groups decided to implement new poker platform in Java. In order not to wait until the end of November, I decided firstly to delve into already ready poker platform implemented in C++. All in all, I have implemented 3 poker bots for that platform, in the following paragraphs I will describe all of them in detail. Later on, as the Java platform was ready, I wrote poker bot for the new platform.

**First bot – C++**

MyBot – the first of the bots. It uses some of the ideas of other, already available bots, and adds new features, in order to make it potentially more effective. This bot tries to win by playing with good cards and folding with bad cards. Techniques used and algorithm description:

- Using Pot equity and pot odds - the ratio of the current size of the pot to the cost of a contemplated call
- Using Harrington zones – described below
- Using Sklansky-Malmuth grouping card algorithm - grouping the starting hands based on strength. Starting with the strongest set of hands the player can be dealt in group 1 (to which belong AA, AKs, KK, QQ, JJ), the hands get progressively weaker working down the table until the almost unplayable hands in group 9.
- Taking into consideration the position of the dealer – the closer to the dealer the position of the player, then he has more information about previous players’ game
- Taking the amount of players into consideration – used in the algorithm of evaluating the hand
- Detecting when the players is weak and can be out bluffed – if the player is weak (for example is on a short stack) then we can try stealing the pot
- Using the evaluator – evaluating the chance of winning on each round given the hand cards and the cards on the table
Especially interesting was the concept of Harrington zones. It proved to be an easy idea but providing good results. Harrington zone’s strategy is based on defining Harrington’s M. M in its simplest form is the ratio of blind to the current chip stack. The system categorizes every situation in a game into five separate sections. So, for example, if the player has 900 chips and the blinds are 15/30, then M would be 20 (15+30=45, 900/45=20). The sections:

<table>
<thead>
<tr>
<th>Level</th>
<th>Zone</th>
<th>Correct play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Green zone</td>
<td>When M is above 20 there are many poker strategies available. The player can raise, re-raise and still have left chips in a hand</td>
</tr>
<tr>
<td>2</td>
<td>Yellow zone</td>
<td>M is between 10 and 20. The stack is decent, but some aggressive plays have to be taken. Small pairs and suited connectors are less desirable.</td>
</tr>
<tr>
<td>3</td>
<td>Orange zone</td>
<td>M is between 6 and 10. The stack is diminishing and the player is no longer able to ‘re-steal’. The player must play aggressively in order to stay alive.</td>
</tr>
<tr>
<td>4</td>
<td>Red zone</td>
<td>M is 5 or less. The player lost all the flexibility and if he plays the hand it should be all-in.</td>
</tr>
<tr>
<td>5</td>
<td>Dead zone</td>
<td>M is less than one – critical situation – any high cards are good enough for all-in move.</td>
</tr>
</tbody>
</table>

The poker platform in C++ provides already some bots. I have tested my bot with the bots from the table below in the heads-up game.

The sample output:

```
> Player Random leaves with 0 chips
Game Finished after 30 deals.
Winner: MyBOT (AI: AIMyBot)
> Player MyBOT finishes at place 1. Score: 1000
> The AI of player MyBOT was: AIMyBot
> The AI of player Random was: Random
```

Statistics - the number of wins and losses specifies the separate, new games:

<table>
<thead>
<tr>
<th>Bot</th>
<th>Opponent bot</th>
<th>Wins</th>
<th>Losses</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyBot</td>
<td>CallBot</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>MyBot</td>
<td>RandomBot</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>MyBot</td>
<td>SmartBot</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>MyBot</td>
<td>RaiseBot</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

As it is visible, MyBot is not an always-winning bot. What is interesting is the fact that the random strategy in poker is not an efficient one. On the other hand the RaiseBot was equally good as MyBot. This stems from the fact that RaiseBot uses a lot of bluffing. Even if it doesn’t have good cards, it raises, so that the opponent is mistaken all the time, unless has a good or monster hand.

Second Bot – C++
The second bot that I implemented I called the riverBot. In this implementation I wanted to experiment with an idea that was observed in the previous battles of MyBot vs RaiseBot. RiverBot plays all the cards until the river. On preflop, flop and turn it calls the opponent, and finally on the river it uses an complex algorithm of evaluating the hand.

I used here the hand evaluator that was provided in the platform. I have researched it on the Internet – it is well known as being an advanced and reliable one. Of course, it is not an ideal one, but quite smart one.

Statistics - the number of wins and losses specifies the separate, new games:

<table>
<thead>
<tr>
<th>Bot</th>
<th>Opponent bot</th>
<th>Wins</th>
<th>Losses</th>
</tr>
</thead>
<tbody>
<tr>
<td>riverBot</td>
<td>RandomBot</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>riverBot</td>
<td>SmartBot</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>riverBot</td>
<td>RaiseBot</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>riverBot</td>
<td>MyBot</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

In the result table it is seen, that the riverBot technique is an efficient one against the random players or the players that only raise as in RaiseBot (interestingly the riverBot was also winning one with the first bot – MyBot. This is probably due to the reason that the game continues to the river, and in that moment uses the same hand evaluator but MyBot takes into consideration also other sophisticated factors as Harrington Zones and maybe be suspecting the opponent of bluffing – thus sometimes surrendering without reason). Finally, riverBot was not winning against the SmartBot which took into consideration much more factors and was making ‘smart’ decision on each round.

Third bot – C++

Finally I implemented guessingBot – it was an attempt to solve the problem of the players which make the moves that are very difficult to guess what they have (too aggressive or too loose players). GuessingBot waits for a good or monster hand, and then makes a move.

Algorithm description:
First of all the boldness and bravery is chosen randomly, which means either a my bot will be a bit more aggressive or more loose. It helps to randomize the game and fool
the opponent. Then for each round the probability for winning is evaluated. Moreover in the method safeRaise we check if we have enough chips – if yes then we can safely raise, if we are short stack then we better call.

I also introduced the ‘fear factor’ – the latter the stage, the more aggressive the bot plays. It is also more ready to bluff. The algorithm also uses Sklansky-Malmuth hand ranking and finally based on all the gather information it evaluates the hand and it’s winning potential which lead to the final decision on each round.

Sample console output after the game:
```
> Player checkFold leaves with 0 chips
Game Finished after 370 deals.
> Winner: riverBot (AI: guessingBot)
> Player riverBot finishes at place 1. Score: 1000
> The AI of player riverBot was: guessingBot
> Player checkFold finishes at place 2. Score: -1000
> The AI of player checkFold was: CheckFold
```

Statistics - the number of wins and losses specifies the separate, new games:

<table>
<thead>
<tr>
<th>Bot</th>
<th>Opponent bot</th>
<th>Wins</th>
<th>Losses</th>
</tr>
</thead>
<tbody>
<tr>
<td>guessingBot</td>
<td>RandomBot</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>guessingBot</td>
<td>SmartBot</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>guessingBot</td>
<td>MyBot</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>guessingBot</td>
<td>checkFoldBot</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>guessingBot</td>
<td>riverBot</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

From the results it is seen that the goal was achieved. The guessingBot plays well against these players who are making random or difficult to evaluate actions such as RandomBot, CheckFoldBot or riverBot. It encountered problems with the bots that are more smart as MyBot or SmartBot.

---

**Java platform**

In December the Java platform was finally in operation so I was able to test my solution on the new platform. As the implementation of the guessingBot was pretty difficult and time consuming but promising, I decided to implement it on the Java platform.

First problem that I encountered was the lack of helping functions on the platform such as evaluator. As written above, the C++ provided already implemented, well-known hand evaluator. I did find two similar open-source, ones implemented in Java, that are used for most of the Poker bots. These evaluators are:

- PartialStageFastEval

After some reading I decided to use the PartialStageFastEval being more efficient one.

Each evaluation method takes 5, 6, or 7 cards represented as integers in the range [0..51] and encoded by the given method. The return value of each evaluation method is an integer representing the equivalence class of the hand, ordered such that better hands have higher equivalence class numbers.

The code that I implemented in Java is equivalent to the one that I implemented in C++. For the tests I compared my bot with the sample bot already implemented. I tested my bot in a multiple players environment as heads up game on that platform was always returning the same values. I compared in the table the results of the

Sample output on Java platform:

```
Winners size: 1
Winner is 3; and he gets 262276
User: 1; 788928
User: 2; 99970
User: 3; 262276
User: 4; 51356
User: 5; 97470

Raise bot 62276
Random bot -248644
KwiecienBot 97470
Call bot 488928
Fold bot 99970
OK
```

```
Winners size: 1
Winner is 3; and he gets 804074
User: 1; 95986
User: 2; 99970
User: 3; 804074
User: 4; 100000
User: 5; 99970

Raise bot 704074
Random bot -200000
KwiecienBot 99970
Call bot -204014
Fold bot 99970
OK
```
Statistics - the number of wins and losses specifies the separate, new games:

<table>
<thead>
<tr>
<th>Bot</th>
<th>Opponent bot</th>
<th>Wins</th>
<th>Losses</th>
</tr>
</thead>
<tbody>
<tr>
<td>KwiecienBot</td>
<td>RaiseBot</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>KwiecienBot</td>
<td>FoldBot</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>KwiecienBot</td>
<td>CallBot</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>KwiecienBot</td>
<td>RandomBot</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

The results of the bots were very random, at some time for example CallBot was winning, on the other time it was the worst. So, all in all, the results on Java platform were also quite random. KwiecienBot was usually achieving the result of making for the second place. My bot was noticeably better for both too aggressive and to loose players.

All in all, the results from C++ platform are more reliable, as the C++ platform provided more flexibility.

My source code for the C++ platform with the bots, and the Bot for Java is located on SVN on: [http://149.156.196.114/mansion-svn/PokerBot_Kwiecien/](http://149.156.196.114/mansion-svn/PokerBot_Kwiecien/)