CS 458/558 - Problem Set 1

Assigned: Wednesday September 9

Deadline: Wednesday September 23, 11:59pm

One of the earliest domains for automated decision systems was game playing. We have discussed chess and checkers, including Art Samuel's seminal checkers program.

For this problem set, we look at a common casino card game: blackjack. We have implemented a simple version in Python, using a Python development environment, codeskulptor.

- Blackjack implementation in codeskulptor. Click the run button in the top left.
- Codeskulptor site. Check out the docs and demos. Codeskuptor has very good python reference materials, as well as tutorials for learning python.
- Raw python code for blackjack. Note: this will likely not run properly outside the CodeSkulptor environmant. However, you may wish to modify this code for your own purposes.

This program implements a pretty vanilla form of blackjack. You cannot double down, buy insurance, or count cards. There is one deck and it is reshuffled before every hand. The dealer has no discretion. She has to stay on 17 or higher.

For this code, the human is in the loop. The player has to click the hit or stand buttons. Your job is to automate this process.

Create a function `hitme(playerhand, dealerfacecard)` which returns a boolean value, true or false, specifying whether the player should ask for another card or not. That is, after the cards are initially dealt, the program will call `hitme` and either ask for another card or stand - untouched by human hands.

Even though the example code uses graphics, your code should not. It should be built for speed, not looks.

The `hitme` function will simply perform a table look-up. You will have a 2 dimensional matrix comprising the optimal strategy for all possible combinations of player hands and dealer face cards. The values in the matrix will simply be true or false. For example,

```
hitme(12, 1) ==> true
```

If your hand value is 12 and the dealer has an ace, you should ask for another card.

```
hitme(18, 4) ==> false
```

If your hand value is 18 and the dealer shows a four, you should stay put.

Actually, it is not up to you to populate the values of the table through dint of insight. Rather, you will write another function `sim(trials)`, which performs Monte Carlo simulation.

The `sim` function will perform a boatload of trials and produce as output a matrix of probabilities. For example, in the cell corresonding to a player hand of 18 and a dealer face card of 4, the value may be `235 / 978` meaning that this combination occurred 978 times in the simulation and in only 235 times did the player win if she asked for another card. You would then convert that matrix to boolean values, based on some threshold ratio value. The normal approach would be to use 50%, but you may decide to use a different value.

Once you have your `hitme` table installed, run your own simulated games, keeping track of the win/loss ratio. Report your results for simulations of at least 100,000 hands.

For this assignment, you may use either R or python or both. R is well suited to running the simulations for calculating the table. Python may be easier for playing the game - given that you already have most of the code. However, writing the game playing code in R is not that difficult either.

Finally, write a function `play(trials)` which will play the given number of hands and return your overall winning percentage.

Depending on your choice of languages, you may have one or two program files to submit. In any event, they should be named `hw1.R` and `hw1.py` In addition, you should hand in a `README` file which explains what function does what, as well as a transcript including the output of Monte Carlo siulation, and the summary output of 100,000 game playing trials - how many games the player won. In the past when giving assignments involving automated game playing, I have award a prize to the most successful program. Keep that in mind.

The process for creating this program is similar to most large scale decision systems: caching the results of a massive simulation that can enable rapid, real time decisions.