15-110: Principles of Computing, Summer 2011
Lab Monte Carlo

Released: Saturday, July 9th 2011
Due: Wednesday 13th 2011

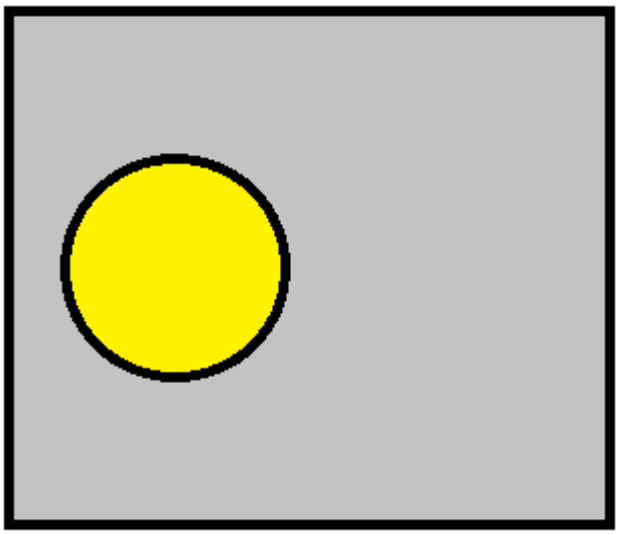
Lab Background

Monte Carlo algorithms are a method for finding an approximation for the solution to an equation when conventional methods of solving the equation are either insufficient or inefficient. Monte Carlo methods are widely used in various fields of computational science, such as computational physics.

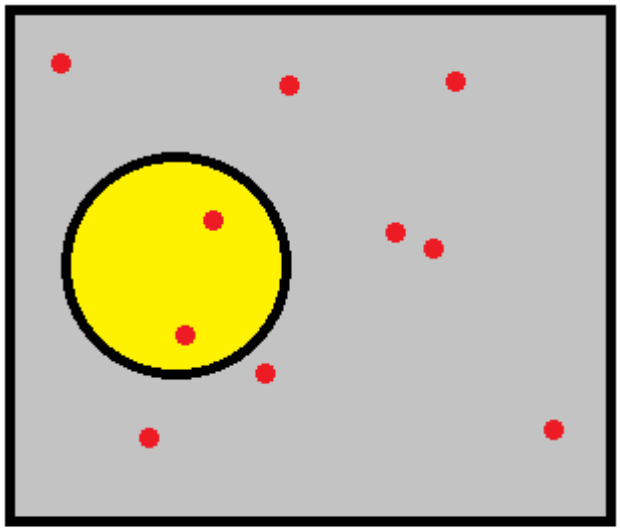
Introduction to Monte Carlo

The basic form of a Monte Carlo algorithm follows as such:
1. Guess a random value.
2. Check if that that value falls within the bounds of the problem.
3. Store the results.
4. Repeat.
...
5. Once you have a suitably large sample size, analyze your hits and misses to calculate the answer.

> **Example**:
> Say you wanted to calculate the area of a circle, but had no idea what the constant pi was equal to. If you had sufficient computing power, one viable option might be to attempt a Monte Carlo solution.
>
> The first step would be to place the circle inside a shape that you knew the area of, like a square:



Then you would place a dot at a random place inside the square and test whether or not it was also within the circle (in this case, it can be done either visually or with the Pythagorean theorem). Each time you do this we say that you have completed a trial.



Trials: 10

Hits: 2

Area of Circle: ~ 0.2

Let's say that after a million trials, only a hundred thousand have fallen inside the circle. We can then conclude that the circle has about 100,000/1,000,000th (or 1/10th) of the area of square. If you know the area of the square, you can find the area of the circle.

Task 1: Card Hands

You will write a Monte Carlo algorithm that will calculate the chances of drawing at least one pair in a five card poker hand. You will need to use the functions that you make in earlier tasks to help you with the later tasks.

Provided Code:

Included in the lab2.py file, we have written several functions which will help you complete the assignment. They are as follows:

Test functions: With the exception of Task 1.2.1, all functions are given test functions. As before, these test cases are by no means exhaustive, and you are encouraged to add your own. The last test function has some special considerations which will be discussed in that section.

`drawCard()`: This function takes no parameters and returns a two letter string which represents a playing card. The first letter is the card's suit and the second is its face value.

Examples:
"CA" – The ace of clubs
"H2" – The two of hearts

"D0" – The ten of diamonds
"SJ" – The Jack of spades

You can think of this card as being drawn from the top of a virtual deck. You are not responsible for knowing how this function works.

`resetDeck()`: This function reshuffles the virtual deck. When you call `drawCard`, cards are permanently removed from the deck. If you start out with a full deck and then run a trial (which calls `drawCard` five times), you will have 47 cards left at the end, instead of 52. In order to put the five cards back, you need to call `resetDeck` at the beginning of every trial.

We encourage you to play around with drawCard() a bit before jumping into the second half of this assignment, so you can be familiar with the type of information it returns.

Task 1.1: Detecting Pairs

First, you will write a helper function that will detect whether or not there is at least one pair of cards in a poker hand with the same face value. This task has two parts:

Task 1.1.1: isPair

Write the function `isPair(card1, card2)`, which will take two two-letter strings. It returns True if both `card1` and `card2` have the same face value and `False` if they have different face values or are not formatted correctly. You may assume that both `card1` and `card2` are strings.

Task 1.1.2: pairInHand

Write the function `pairInHand(hand)` which takes a list, where every element is a string two characters long. It returns `True` if there is at least one pair in the list and `False` if there is not. The function also returns `False` if the `hand` is not formatted correctly. At this stage, the list can be any length. You should use `isPair` to help you write this function. You may assume that `hand` is a list that contains only strings.

Task 1.2: Running Trials

Now we write the main function, which is also split into two tasks.

Task 1.2.1: Running a Trial

Write the function `runTrial()` which takes no input. `runTrial` draws five cards and returns `True` if there is a pair among them and `False` otherwise. Use `pairInHand` to help you write this function.

Remember to call resetDeck() at the beginning of `runTrial()`, so `drawCard` will work properly.

Task 1.2.2 Monte Carlo

Write the function `monteCarlo(trials)` which takes an integer representing the number of trials you wish to run. It should return a floating-point (decimal) between 0 and 1 representing the percentage of trials which were a success compared to the total number of trials. `monteCarlo` should return -1 if it receives malformed input.

Although you should use `runTrial` to help you complete this task, you should not include anything specific to `isPair` or `pairInHand`. This is because when we test your submissions, we will supply an alternate version of `runTrial` which checks another poker hand.

Testing Monte Carlo

Even though you will not be graded on efficiency, your code should be able to run our tests in a second or two. If it takes you more than ten or fifteen seconds to run our `monteCarlo` test, you may have an infinite loop in your code. If your computer is particularly old, you may want to reduce the number of trials required in the test case from 10,000 to a lower number.

Given the random nature of this test, there is a small (less than 1%) chance that your code can be right and still fail the `monteCarlo` test. If you believe that this is the case, run the case a few more times. After three trials, the chance of an error drops to one in a million.

If you find an error in either this document or in the lab2.py, please email **staff-110@cs.cmu.edu**