Computer Science 202
Introduction to Programming
The College of Saint Rose
Fall 2013

Topic Notes: Random Numbers

We have seen many cases where it is useful have computer programs choose random numbers. Programs that implement games might need to make decisions randomly. This could involve choosing a random direction for a character to move, an order for shuffling a deck of cards, or to simulate the roll of a die.

We have (mostly) used two Objectdraw-provided utilities to generate random numbers this semester: the `RandomIntGenerator` and the `RandomDoubleGenerator`. Since those are specific to Objectdraw, we want to use the more standard mechanisms for selecting random numbers provided in standard Java.

We will see how this works by looking at an example:

See Example: RandomDemo

If we want to use random numbers in our program, we need to construct a `Random` object that we can ask to generate our random numbers. This first requires that we add an appropriate `import` statement:

```
import java.util.Random;
```

Then, we construct an instance:

```
  Random randomGenerator = new Random();
```

We can then get random values from our `Random` object by calling its methods including `nextInt` and `nextDouble`. See the `RandomDemo` example code for specifics.

A Random Number in a Game

We can use this capability in many ways. We will first implement a simple guessing game. We will have the computer pick a random number between 1 and 100, and the user gets to make repeated guesses until the guess is correct. The program helps out by giving a “higher” or “lower” response.

See Example: GuessingGame

Here, we just need to choose our random number for the answer, then have a loop that reads guesses until the correct number is entered.

A Monte Carlo Method to Compute π

Not only games make use of random numbers. There is a class of algorithms knows as *Monte Carlo methods* that use random numbers to help compute some result.

We will write programs that use a Monte Carlo method to estimate the value of π.

The algorithm is fairly straightforward. We repeatedly choose (x, y) coordinate pairs, where the x and y values are in the range 0-1 (*i.e.*the square with corners at $(0, 0)$ and $(1, 1)$. For each pair, we determine if its distance from $(0, 0)$ is less than or equal to 1. If it is, it means that point lies within the first quardant of a unit circle. Otherwise, it lies outside. If we have a truly random sample of points, there should be an equal probability that they have been chosen at any location in our square domain. The space within the circle occupies $\frac{\pi}{4}$ of the square of area 1.

So we can approximate π by taking the number of random points found to be within the unit circle, dividing that by the total number of points and multiplying it by 4!

We can see a simulation of this at:

On the web: http://en.wikipedia.org/wiki/File:Pi_30K.gif at

Our Java program for this:

See Example: MonteCarloPi

And a version that makes use of Objectdraw:

See Example: MonteCarloPiVisual