PS5 Walkthrough

CPSC 112: Introduction to Programming

Neil Zheng and Lincoln Swaine-Moore
Part 1: Monte Carlo Simulation

Key programming skills:

- Complicated conditional
- Accumulative loop
What is Monte Carlo simulation?

Basic idea:

Do an experiment a bunch of times, and count how frequently each result occurs. Can use the counts of results to estimate the probability it occurs.
Simple Monte Carlo Example

**Goal:** Want to figure out probability that you roll a 1 on a six-sided die.

**Method:** Roll the die a bunch of times (N). Count how many times you roll a 1 (M). To find probability that you roll a 1, divide the number of times you rolled a 1 (M) by the number of times you rolled total (N). Then M/N is (approximately) your answer!

**Remark:** The more times you roll (e.g. the higher your value of N), the better your estimate will be.
Okay...so?

**Ultimate Goal**: Estimate $\pi$

**Intermediate Goal**: Estimate probability that a random point dropped in this square falls in this circle

**How does this help?**: Probability from intermediate goal is equal to ratio of area of circle to area of square

- We know the area of the square
- We know the area of the circle in terms of $\pi$
- So we can solve for an estimation of $\pi$
Big Picture

- Need to make for loop that runs N times (each time is one Monte Carlo trial)
- Each time through, drop a random point, check if it’s in the circle, and if it is, add to a variable that you’re using to count the number of points that have landed in the circle (H)
- “Regularly” print a line saying where you are in the loop (iteration number), and what your current estimation of pi is (more on what I mean by “regularly” on “Output” slide)
- Need to use the three methods provided in skeleton code:
  - public static int getN()
  - public static boolean isInCircle(double x, double y)
  - public static double estimatePi(int pointsInCircle, int pointsDroppedSoFar)
  - Rest can happen in main
- Turn in code, and also an image of graph showing result where N = 5000
Input

- `public static int getN()`
- `import java.util.Scanner;` (should be familiar by now!)
- N is the number of Monte Carlo trials we are running (in other words, the total number of points we are dropping in the square)
- Use provided “PROMPT” in the skeleton code
- What if N < 5?
  - System.out.println an error message
  - System.exit(1);
Dropping Points

- Math.random() -- takes no arguments, returns a double between 0.0 and 1.0
  - Run it twice each iteration (once for x, once for y)
- This gives us random points in the square!
  - Are they in the circle? Maybe! We need to check
Checking if points are in the circle

- public static boolean 
isInCircle(double x, double y)
- We know (x,y) is in the square, but is it in the circle?
- Equation of a circle:
  - \((x - a)^2 + (y - b)^2 = r^2\), where \(r\) is radius, (a, b) is center
  - In skeleton code, a = CIRCLE_CENTER_X, b = CIRCLE_CENTER_Y, r = RADIUS (should use these global variable in your code)
- Equation above describes points on circle, what about points inside circle?
  - Hint: make it an inequality

http://www.mathematics-monster.com/images5/circle_equation_diagram.jpg
Estimating pi

- public static double estimatePi(int pointsInCircle, int pointsDroppedSoFar)
- pointsInCircle / pointsDroppedSoFar approximates the probability that a random point dropped in the square will be within the circle -- this is the basic Monte Carlo principle!
- The probability that a random point dropped in the square is within the circle is the same as the ratio of the area of the circle to the area of the sphere
  - In other words, pointsInCircle / pointsDroppedSoFar ≈ probability(dropped point in square is in circle) = \( \frac{A_{circle}}{A_{square}} \)
- We know the area of the square \( A_{square} = \text{length}_{square} \times \text{width}_{square} \)
- We know the area of the circle \( A_{circle} = \pi \cdot \text{radius}_{circle}^2 \)
- So we can plug these values into the formula pointsInCircle / pointsDroppedSoFar ≈ \( \frac{A_{circle}}{A_{square}} \) and solve for an estimated value of \( \pi \)
- Return this estimated value of \( \pi \)
Earlier, said that we want to regularly print a line saying where you are in the loop, and what your current estimation of pi is
  ○ “Regularly” here = Every 5 times through loop before you’ve gone through loop 50 times, every 50 times through loop before you’ve gone through loop 1000 times, and every 1000 times through the loop after that
  ○ Can calculate above (that is, whether an iteration is a printing iteration) with a single (complicated) if statement that takes into account what number iteration you’re currently on
    ■ Hint: You’ll need some and’s (&&) and some or’s (||)
• Put a “\t” between iteration number and current estimation of pi

Note: your output will not look exactly like this...it’s supposed to be random!
Output (graphs)

Note: your graphs will not look exactly like this...it’s supposed to be random!
Bonus

- Use StdDraw to make a graph as you go through the loop, connecting lines between estimations
- Useful functions:
  - StdDraw.setCanvasSize
  - StdDraw.setXscale
  - StdDraw.setYscale
  - StdDraw.line
- Hint: when drawing a line, you need to know both the starting and endpoints (previous and current estimate)

Example with N = 500 (so you can see the early volatility better)
Questions on Part 1?
Part 2: TaskMan

We want to take a file like below and produce our pretty task manager to the right.

4
PS4 | 2/28/2017 11:55:00 PM
Exam 1 | 3/10/2017 11:35:00 AM
PS5 | 3/6/2017 11:55:00 PM
Summer | 5/1/2017 8:00:00 AM

Notes:
- 4 is the number of tasks
- Date format: Month/Day/Year
- Time Format: Hour:Min:Sec AM/PM
We find the following variables:

1. A constant to convert milliseconds to seconds
2. The file name and number of tasks
3. Two arrays to fill later
   a. String[] taskNames
   b. Long[] taskDeadlines
4. Parameters for the graphic (the alarm clock), which we won’t need to worry about
5. The string format for when we print our deadlines
A quick look at main

```java
public static void main(String[] args) {
    loadTasks();

    initStdDraw(TOTAL_WIDTH, PER_TASK_HEIGHT * N);

    for (;;) {
        // Get current time in EPOCH time (MS)
        long currentEPOCHtime = System.currentTimeMillis();

        for (int i = 0; i < N; i++) {
            // First draw the image
            StdDraw.picture(PICTURE_X,
                PER_TASK_HEIGHT * N / 2,
                IMAGE);

            // Next compute the time difference to deadline of task i
            long timeDiffMS = taskDeadlines[i] - currentEPOCHtime;

            int timeDiffSec = (int) Math.round(timeDiffMS / 1000.);

            // Begin of TODO ******************************************
            // Set pen color
            StdDraw.setPenColor(Color.BLACK);
            // End of TODO ******************************************

            String timeDiffStr = fromSec2TimerString(Math.abs(timeDiffSec));

            StdDraw.textRight(TASK_X,
                PER_TASK_HEIGHT / 2 + i * PER_TASK_HEIGHT,
                taskNames[i] + " " + taskDeadlines[i] / 1000 + " sec");

            StdDraw.textRight(TIMER_X,
                PER_TASK_HEIGHT / 2 + i * PER_TASK_HEIGHT,
                timeDiffStr);
        // end of for each task
        StdDraw.show(MS_PER_SEC);
        StdDraw.clear();
    } // end of for (;;)
} // end of main
```

main() will do the following:
1. loadTasks();
   a. Opens file
   b. Parses each line of file and fills taskNames[] and taskDeadlines[] with information

2. initStdDraw();
   a. Sets up the canvas for us to draw our task manager
   b. This method is complete in the skeleton code

3. Our animation loop
   a. Gets current time
   b. For each of the N tasks
      i. Draws alarm clock
      ii. Finds time until deadline
      iii. Change the pen color to green or red depending on if we before or after the deadline respectively
      iv. Converts from time to string
      v. Prints task name and string
   c. Show and clears
**EPOCH Time?**

Epoch time is the number of milliseconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970.

1. We have the current epoch time
2. Then we convert our deadline to epoch time
3. We can take the difference between our deadline epoch time and current epoch time to find the difference in milliseconds
4. Then with the difference, we convert it back to a more readable days/hours/minutes/seconds format
Loading the tasks

public static void loadTasks() {
    Scanner input = null;
    // File IO with exception handling
    try {
        input = new Scanner(new File(TASK_FILE));
    }
    catch (FileNotFoundException e) {
        System.out.println("Could not open " + TASK_FILE);
        System.exit(1);
    }
    
    N = input.nextInt(); // number of tasks
    input.nextLine(); // skip the remaining 
    taskNames = new String[N];
    taskDeadlines = new long[N];
    
    // read N tasks
    for (int i = 0; i < N; i++) {
        String line = input.nextLine();
        // Parse line to obtain taskNames[i] and taskDeadlines[i]
        // First, split into 2 pieces: first piece is task name
        // second piece is deadline
        
        // Begin of TODO: fix below ***********************
        taskNames[i] = "TODO"; // TODO
        System.out.print("Task name: "+taskNames[i]+" /");
        String taskDeadlineStr = "2/24/2016 11:55:00 PM";
        // End of TODO ***********************
        
        taskDeadlines[i] = timeString2EPOCHTime( taskDeadlineStr );
    }
    
    input.close();
} // end of loadTasks

loadTasks() will do the following:
1. Create a scanner with the file input -- try-catch will check if the file we give exists
2. Gets the number of tasks, N, which tells us the number of tasks we need to read
3. For loop to read N tasks
   a. Gets next line
   b. Splits line into task name and deadline
   c. Stores task name into taskNames[i] array
   d. Converts our deadline string to epoch time, stores in taskDeadlines[i] array
String parsing

We are given our String line = input.nextLine(); We expect line to look something like PS4 | 2/28/2017 11:55:00 PM. Let’s look at how we might parse (split) this line using str.split():

1. Note that in str.split(), str will be the name of our String variable. In this case, we would use line.split().
2. The argument we pass to line.split() will be the string or character we would like to split by, in our case " \| ". Note: we use \ to “escape” |, which would otherwise be treated as OR. Also we add a space before and after to remove unneeded spaces.
3. line.split() will return an array of two strings,, so we will need to store the returned strings in a string array like String[] parts.
4. Then we can store parts[0] in taskNames[] and store parts[1] in our temporary String taskDeadlineStr before converting and storing to taskDeadlines[].
Converting to EPOCH time

```java
// A method to convert time string to EPOCH time
public static long timeString2EPOCHTime(String timeStr) {
    // split method with regular expression.
    String[] pieces = timeStr.split("/ ");
    int month = Integer.parseInt(pieces[0]) - 1;
    int day = Integer.parseInt(pieces[1]);
    int year = Integer.parseInt(pieces[2]);
    int hour = Integer.parseInt(pieces[3]);
    int min = Integer.parseInt(pieces[4]);
    int sec = Integer.parseInt(pieces[5]);

    // Begin of TODO ********************************************
    // GregorianCalendar allows us to convert to EPOCH time,
    // but it has its own requirement:
    // Month is 0 (Jan) to 11 (Dec)
    // Hour is that
    // 12 am -> 0, 1 am -> 1, ..., 11 am -> 11,
    // 12 pm -> 12, 1 pm -> 13, ..., 11 pm -> 23
    // End of TODO ********************************************

    if (!isValidInput(month, day, year, hour, min, sec)) {
        System.out.println(timeStr + " is not valid");
        System.exit(1);
    }

    GregorianCalendar task = new GregorianCalendar(year, month, day,
            hour, min, sec);

    System.out.println("Due Date: " + task.getTime());
    // convert to milliseconds from EPOCH
    return task.getTimeInMillis();
}
```

The skeleton code already splits our deadline string into the component parts for us. Then we need to prepare the various inputs for `GregorianCalendar(year, month, day, hour, min, sec)` to work properly.

1. We note that pieces should have a seventh component, `pieces[6]`, that holds either AM or PM.
2. We can write an if-else statement that adds 12 hours to hour `int hour` variable dependent on `pieces[6]`.
3. Formatting the months for `GregorianCalendar` is already taken care of in the skeleton code

We need to check if our inputs for `GregorianCalendar()` are valid. If they are, we can create a new `GregorianCalendar` object called `task`, which we can get the epoch time from using `task.getTimeInMillis();`
Checking input is valid

```java
public static boolean isValidInput(int mon, int day, int year,
                                   int hour, int min, int sec) {
    // Begin of TODO ****************
    // year should be greater than or equal to 0
    // month should be between 0 (Jan.) and 11 (Dec.).
    // day should be bigger than 0
    // day should not be larger than number of days in the given month
    // hour should be between 0 (12 AM) and 23 (11 PM)
    // min should be between 0 and 59
    // sec should be between 0 and 59
    return true;
    // End of TODO ****************
} // end of isValidInput
```

This method checks to make sure all the date and time components are valid so that we can pass them to GregorianCalendar();

One could write 7 if statements for each of date-time components, but the method can be written in a single if-else statement using &&’s.

Note: Be aware that the upper limit for days depends on the given month (i.e. deadlines in March should have days <= 31). We could write another method to return the number of days in a given month.
Converting from seconds to timer format

// A utility method to convert from sec to the timer string format
public static String fromSec2TimerString(int sec) {
    // Begin of TODO ******************************************
    int day = 0;
    int hour = 0;
    int min = 0;
    sec = 0;
    // End of TODO ******************************************
    return String.format(TIMER_FMT, day, hour, min, sec);
}

Exam 1: 8d 11h 08m 42s

PS4: 5d 00h 31m 18s

// Timer format
final static String TIMER_FMT = "%d %dh %dm %ds";

At this point in main, we have taken the difference between our deadline epoch time and current epoch time. We have also set the pen color depending on if we are before or after the deadline.

This method should look familiar from Cashier.java from PS4. We want to take the quotient of our total number of milliseconds and the milliseconds in one day, one hour, etc.

Note that the number of days, hours, minutes, seconds are always positive integers. Consider Math.abs();

We return a string with the given global string format variable.
Questions on Part 2?
Have a wonderful break!