INF100 H17 Semester Assignment 1

Due on Friday, September 29, 2017

Albin Severinson, Dag Haugland

Abstract

For the first INF100 semester exercise, we will explore one of the key ideas that went into the development of nuclear weapons: the Monte Carlo Method! It is a method that uses randomness to solve (potentially very difficult) problems. It was formally proposed by Stanislaw Ulam while working on a nuclear weapons project in the US and it has since been used extensively within finance and almost every other engineering domain. We, on the other hand, are going to use it to simulate the life expectancy of Jack the pirate. As for the previous assignment we will start off with the basics and gradually add more things. We will only grade the program for the final problem of the assignment. The program is graded on a 0 to 100 points scale. If you have questions specific to this assignment you should send them to me at `albin.severinson@uib.no`.

Submission

1. Add your program file to a `.zip` archive named `sx1_firstname_lastname.zip`. You do this by right clicking on the file ending with `.java` that contains your program code and choosing compress or add to archive (exactly what it is called varies by operating system and version). You should only add your final program, i.e., the one for problem 5. Ask your group leader if you are having issues with this. Note that handing in the wrong file may cause you to fail the assignment!
2. If you complete the voluntary getting ahead part of the assignment, it should be handed in separately. If you complete this part you thus need to hand in two files (the `.java` file for problem 5 and the `.java` file for the getting ahead part). Put both files into the same `.zip` archive.
3. Log in to `https://mitt.uib.no` and press the box marked with "INF100" to enter the page for this course.
4. Press "Oppgåver" on the left. Next, select "Semester Assignment 1".
5. Press "Lever oppgåve" and upload the `.zip` file you created in step 1.

Grading

- This assignment is worth a total of 100 points. Each problem is worth 20 points.
- We will deduct points for poor code. For example, lack of comments, poorly chosen variable names (please do not use non-ASCII characters such as Ø in variable names :)), incorrect indentation, and code repetition may be reason to deduct points.
- Have a look at the Google Java style guide if you are unsure on how to structure some part of your code: `https://google.github.io/styleguide/javaguide.html`.
- Make sure your solution actually does what the problem asks for. We may deduct points if you implement something other than what the problem asks for.

General Guidelines

These are general guidelines that will (hopefully) help you maintain your sanity through your programming endeavours.

- Have your computer indent your code for you. You do this in DrJava by selecting all of your code, right clicking, and selecting `indent lines`.

- You should make a copy of all your code after completing each problem so that you can roll back to a previous version when you inadvertently break something. Even the developers at Google sometimes have to roll back.
- Set aside time to clean up your code. The number that gets bandied around on the internet is that the best programmers spend roughly half their time cleaning up and rewriting their code, i.e., you are halfway done when you get it working.
- You will save time by making sure you properly understand your current program before moving on to the next problem.
- You should probably be spending a significant part of your time on stackoverflow when working on these assignments.
- If you want to figure out how a standard class or method works (`Scanner`, say), you can look it up on the Java docs page. Instead of trying to find what you are looking for at the Java docs webpage you should probably just Google it (searching for "java Scanner" on Google gives you its docs page as the first result).

Pirate Survival Simulator

Jack the pirate is marooned on the tropical island of Mojang. We are interested in simulating how long (measured in number of steps) Jack will survive before falling into the shark-infested waters surrounding the island. It is complicated to compute this number exactly. Instead, we will run computer simulations and take the average number of steps from the simulations as our answer. This idea is the so-called Monte Carlo method.

Mojang is a square island consisting of a 10 by 10 grid bordered on all sides by water. We denote each grid of the square by a pair of coordinates (x, y). Any coordinate for which both $10 \geq x > 0$ and $10 \geq y > 0$ is on the island. Any other coordinate is not. For example, $(1, 1)$, $(1, 10)$, and $(3, 7)$ are all on the island and $(0, 3)$, $(11, 2)$, and $(8, 11)$ are all off the island. See fig. 1.

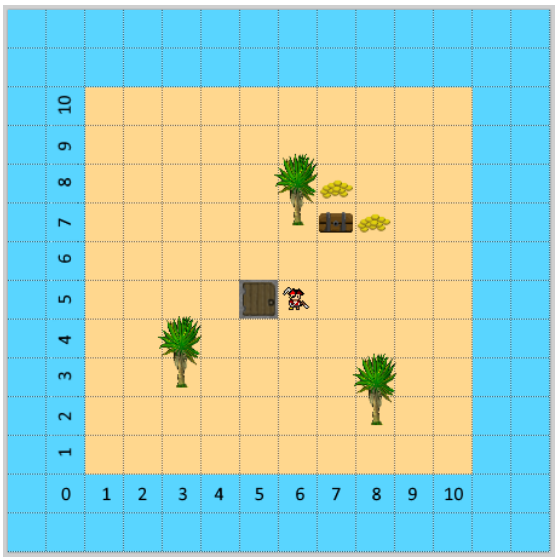

Figure 1: The island of Mojang. Jack has taken a single step from his starting position $(5, 5)$. Picture by Oskar Leirvåg.

Now, Jack has gotten himself drunk from the secret rum stash he found at the center of the island (coordinates $(5, 5)$). He proceeds to go for a walk around the island. Every step he takes brings Jack to one of the four adjacent squares. Due to his drunkedness, he always moves to a randomly selected adjacent square whenever he takes a step. For example, he might move like this in five steps: $(5, 5)$, $(5, 4)$, $(4, 4)$, $(3, 4)$, $(3, 5)$. The palm trees and treasure does not affect Jack's movement.

Problem 1 (20 points)

Consider first the simplified case that the island is one-dimensional, i,e, the island has dimension 10 by 1 and we use only the x coordinate, ignoring the y coordinate. Write a program that starts Jack off at the coordinate $x = 5$, randomly takes one step either west ($x = 4$) or east ($x = 6$) and prints Jack's new location. You can use this trick to generate a random integer in $\{0, 1\}$ that you can then use in a `switch` (remember to put in `break` after each case!). You can of course use `if` and `else if` instead if you prefer.

```
int direction = (int) (Math.random() * 2); // random integer in {0, 1}
```

Example

Example output for three times running the program, i.e., pressing run thrice.

```
Jack took one step and ended up at x = 4
```

```
Jack took one step and ended up at x = 6
```

```
Jack took one step and ended up at x = 4
```

Problem 2 (20 points)

Use a `while` loop to have Jack take random steps until he falls into the ocean. Add a variable that counts the total number of steps Jack takes. We are still considering the simplified case that the island is one-dimensional. You can use && to check two conditions in a `while`, i.e.,

```
while (x > 0 && x <= 10) {

}
```

Example

Example output for three times running the program, i.e., pressing run thrice.

```
Jack was eaten by sharks after 17 steps
```

```
Jack was eaten by sharks after 13 steps
```

```
Jack was eaten by sharks after 31 steps
```

Problem 3 (20 points)

Make the island two-dimensional. This means that you need to add a y coordinate to Jack's location and to the `while` loop. Furthermore, you need to update the means by which you randomly select which direction Jack moves in to include west, east, north, and south.

Example

Example output for three times running the program, i.e., pressing run thrice.

```
Jack was eaten by sharks after 29 steps
```

```
Jack was eaten by sharks after 47 steps
```

```
Jack was eaten by sharks after 18 steps
```

Problem 4 (20 points)

So far the program has exited once Jack has fallen off the island. To get an accurate result we would need to compute the average number of steps over many runs. Add an outer `for` loop counting the total number of times Jack has fallen into the ocean, i.e.,

```
final int ITERATIONS = 100000;
for (int i = 0; i < ITERATIONS; i++) {

}
```

You need to be careful with where you define your variables now. Remember that a variable defined in the body of a loop gets reset every time it runs (for example, you want Jack's location to be reset each time he falls off, but the steps counter should not be reset). You can then compute the average number of steps by dividing the total number of steps by the total number of times Jack has been eaten by sharks.

Example

Your program output should look like this.

```
Jack was eaten by shark after 35.039 steps (averaged over 100000 runs)
```

Problem 5 (20 points)

So far we have been placing all of the code within `main(String[] args) { ... }`. This is fine for small programs, but as your program grows you will need to learn more powerful ways of organizing your code. This is where methods come in. Here is a short refresher on methods: `https://www.youtube.com/watch?v=-IJ5izjbWIA`.

Move the code used to simulate one of Jack's life cycles into a separate method

```
public static int simulate() {

}
```

It is especially important to make a copy of your code before starting this problem. Even the best programmers make mistakes when re-organizing code and you need a way to go back to a working version if you do. After making this change the only code that should be within your `for` loop should be something like `steps += simulate();`.

Getting Ahead

If you have completed the assignment (well done first of all!), you have a pretty good idea on how long Jack will survive on the desolate island of Mojang. However, there is still lots of things we can learn. For example, what is the probability of Jack surviving until a ship comes to rescue him if the ship arrives after Jack has taken exactly 32 steps? To find out we need a way to store the results of our simulations for analysis. Use an `Array` to store the number of steps Jack takes in each simulation. Here is a quick introduction to arrays: `https://www.youtube.com/watch?v=L06uGnF4IpY`.

This part is meant as preparation for the coming course material and will not be graded. However, we will still look at it and comment it! Furthermore, you never have to catch up if you are always getting ahead. This part should be handed in separately. If you complete this part you thus need to hand in two files (the `.java` file for problem 5 and the `.java` file for this part). Put both files into the same `.zip` archive.

Once you have stored the number of steps taken in each iteration in an array, use it to compute the probability of Jack falling off when taking his 32-th step. More generally, compute the probability of Jack falling off the island after taking exactly n steps for all n in the range $n = 1, 2, \ldots, 200$. Print the probability of Jack falling off for each n to the terminal.

With this knowledge you can answer a much wider array of questions regarding Jack's fate. This is essentially how machine learning works. You collect or generate data about some phenomena (in this case the number of steps taken by Jack) and use your computer to infer some pattern in the data (such as the probability of surviving exactly 32 steps).