Simulation – A bigger programming project

Read: Chapter 9 from textbook

Lab 8: The blackjack project you'll build below is needed for your lab problems. Therefore, it is essential that you implement it following the process described below and with lots of comments. Keep it in your directory, in order to have it available in your lab.

In this unit, we will deal with the development of a bigger programming project that will incorporate what you have learn so far. The project falls in the scope of a (maybe the most) basic application of computers, namely simulation. Using simulation, we try to solve real-world problems by modelling real-world processes to provide otherwise unobtainable information. A basic component of simulation is randomness. Therefore we will begin by producing (pseudo-)random numbers in Python, and then we will move to building our project. Along the way, there is a parallel (similar) project that you should be building, following the example of the book.

Randomness

• Python has the library random with functions used to produce pseudorandom numbers\(^1\). We are going to use two functions, randrange() and random(). Let's import them:

```python
>>> from random import random, randrange
```

The randrange(a,b,c) will produce an integer multiple of c that is between a and b, not including b (a,b,c are integers). Try a few times the same call to randrange, say randrange(3,95,11), and note that the result “looks” random.

The random() function produces a random floating point number between 0 and 1. Note that the probability of random() returning a number between 0 and 0.65 is exactly 65%. We will take advantage of this in our projects.

Simulation project

For this project, we will follow what has been called the “top-down” approach.

• Informal statement: Racquetball is played between two players using a racquet to hit a ball in a four-walled court. One player starts the game by putting the ball in motion (serving). Players try to alternate hitting the ball to keep it in play, in what is referred to as a rally. The rally ends when one player fails to hit a legal shot. The player who misses the shot loses the rally. If the loser is the player who served, service passes to the other player. If the server wins the rally, a point is awarded. Players can only score points during their own service. The first player to reach 15 points wins the game. In our simulation, the ability-level of the players will be represented by the probability that the player wins the rally when he or she serves. Write a

\(^1\) The numbers are pseudorandom and not truly random, because they are produced by a deterministic (instead of a random) process, starting from a seed. Every time someone provides the same seed to this process, it will produce exactly the same sequence of pseudorandom numbers.
program that asks the user for the service probability for both players, simulates multiple games with these probabilities, and prints out the results.

**Practice project:** The Battleship game we are going to create is the following: There are two players, you and the computer. Each player has two 7x7 matrices. In the “My fleet” matrix, the player places his/her ships (an Aircarrier (of length 5), a Cruiser (of length 3), and a Destroyer (of length 2)) either horizontally or vertically each. In the “My hits” matrix, the player records his/her shooting record: if there was a shot at the (X,Y) cell, then this cell will contain a “M” if the shot was a miss, a “A” if it was an Aircarrier hit, a “C” if it was a Cruiser hit, or a “D” if it was a Destroyer hit. The game is played in turns (starting with the computer): in each turn the shooting player announces their shot with a pair of coordinates (e.g., 3,4 means cell (3,4)), and the other player announces the result ("M", “A”, “C”, “D” respectively). To help the user, when the computer announces the result of the user's shot, it also prints out the updated “My hits” matrix for the user. There should also be a request for a repeat if the user shoots a cell that has already been shot. Whoever sinks all the opponent's ships first, is the winner. For now, the computer places its fleet randomly (make sure that its ships don't overlap!), and shoots randomly (but not at the same cell twice!).

An example of a “My fleet”/“My hits” pair of matrices looks like the following:

```
**AAAAA    *M**MD*
D*****    M*A****
D***C**    *****M*
****C**    **AM***
****C**    D*****M
*******    *MAM***
```

• Detailed specification: You can find this in the bottom half of p. 269.

• **Practice problem:** Write a detailed specification for the practice project.

• **Top-Level design:** Here is a first attempt in breaking down the solution in distinct tasks:

  Print an introduction
  Get the inputs: probA, probB, n
  Simulate n games of racquetball using probA and probB
  Print a report on the wins for playerA and playerB

**Practice problem:** Do the same for your practice project.

After developing this basic break-down (p. 273-274), a picture of the algorithm starts emerging:

```python
def main():
    printIntro()
    probA, probB, n = getInputs()
    winsA, winsB = simNGames(n, probA, probB)
    printSummary(winsA, winsB)
```

**Practice problem:** Do the same for the practice project.
• **Second-Level design:** The different tasks can be further elaborated by using a *structure (or module hierarchy)* chart.

![Diagram of module hierarchy](image)

**Practice problem:** Do the same for your practice project.

At this point we can start mapping subtasks to functions. The `printIntro` and `getInputs` tasks can be implemented as follows:

```python
def printIntro():
    # Prints an introduction to the program
    print("This program simulates a game of racquetball between two")
    print('players called "A" and "B". The abilities of each player is')
    print("indicated by a probability (a number between 0 and 1) that")
    print("the player wins the point when serving. Player A always")
    print("has the first serve.\n")

def getInputs():
    # RETURNS probA, probB, number of games to simulate
    a = eval(input("What is the prob. player A wins a serve? "))
    b = eval(input("What is the prob. player B wins a serve? "))
    n = eval(input("How many games to simulate? "))
    return a, b, n
```

**Practice problem:** Check your practice project design so far, to see whether any particular module can be mapped to a function.

For the `simNGames` module, the process of developing ever more detailed versions is described in Section 9.3.4. We already have the signature (or interface) of the function from the chart:

```python
def simNGames(n, probA, probB):
    # Simulates n games of racquetball between players A and B
    # RETURNS number of wins for A, number of wins for B
```
The final result is the following:

```python
def simNGames(n, probA, probB):
    # Simulates n games of racquetball between players A and B
    # RETURNS number of wins for A, number of wins for B
    winsA = winsB = 0
    for i in range(n):
        scoreA, scoreB = simOneGame(probA, probB)
        if scoreA > scoreB:
            winsA = winsA + 1
        else:
            winsB = winsB + 1
    return winsA, winsB
```

The updated chart is the following:

---

**Practice problem:** Do the same for the second-level design of your practice project.

*Third-level design:* Note that now your chart has a third level. We can proceed in designing it. The design process is described in section 9.3.5. A crude description of what SimOneGame module should do is the following:

Initialize scores to 0
Set serving to “A”
Loop while game is not over:
  Simulate one serve of whichever player is serving
  update the status of the game
Return scores
After many rounds of refinement, the final function is described in p. 281, while the updated chart is the following:

Practice problem: Design the third level of the practice project.

- The process continues in the same vein with the fourth-level design etc. until the final product is `rball.py` in p. 282-283. Notice the last line of the program: this is how a typical Python program is run (instead of just calling `main()`); for now we are not interested in explaining this technical detail.

Practice problem: Finish the implementation of the practice project.

- Testing: After any implementation, the most important stage is the testing stage. One way to do your testing, is to test each of your modules (functions) separately, usually in the reverse order of designing them (so, we would first test `gameOver`). In fact, you can perform this testing once a function implementation is complete, i.e., when the function and all its sub-levels have been implemented, without waiting for the whole implementation to finish; this gives you the ability to test smaller chunks of code, and, therefore, your testing can be thorough. See Section 9.4.

Practice problem: Test the correctness of your practice project implementation. *(Hint: Since you are going to modify your practice project in the lab, I hope you did remember to write plenty of comments...)*