Simulation, and Designing more complex programs

CMSC 104 Section 2
October 30, 2025

Administrative Notes

Classwork 7 today

Quiz 3 is in two weeks - November 13

Longer programs

Everything we've covered so far has involved writing a short program

- To solve a simple problem
- To illustrate how something works

In the "real world" programs are usually longer

- "Software development" - the process of putting together programs and sets of programs in terms of units

Why do we do it that way?

Because the complexity of software can rapidly increase faster than the ability of one individual human to easily understand it*

- There is no human who understands how every line of Windows 11 works and why it works that way
  - Replace “Windows 11” with “iOS” or “Excel” or any similar massive program

- Which is part of the allure of software development using AI - it eliminates the need for a human to understand anything

Programs are designed by:

- Breaking the problem up into “units” which can be built, tested and understood
- And then combining these units into things that work in ways we mostly understand

Simulation

“Simulation” means building a computer system that “models” a real-world event or system so that we can learn something about it

- All models are wrong; some are useful - George Box

We’re going to simulate part of the game of racquetball so that we can predict the winner of a match BEFORE the match is played

- It will a very simple simulation, but the ideas are exactly what more complex systems use
  - “74.2% BPI chance of winning…”

Racquetball

See e.g. https://www.youtube.com/watch?v=EMrHBYIMT6c

(No endorsement of the site or the coaches is implied)

What we care about:

- Games are played to 15 points
  - The instant one player has 15, the game ends
- You only get points when you're serving
  - If the server loses the point, it's a "side out" and the other player becomes the server
    - We really don't care how many "side outs" there are; that doesn't impact the result
- Each player has an established probability of winning a point on their own serve
  - How did we get that? By collecting a massive amount of data about every game the player has ever played, and then applying proprietary algorithms
  - Or, alternatively, by just making up a number

How are we going to develop the program?

We'll start by figuring out the probability that each player wins each point

Then we'll keep repeating that over and over until one player has 15 points

Start with Player A serving.

- Generate a random number between 0 and 1.
- If the number is less than or equal to Player A's probability of winning the point, Player A wins that point and serves again
- If the random number is greater than Player A's probability of winning the point, it's a side out and Player B serves next

How to simulate one game - a function

```
def simOneGame(pA, pB):
    """

    :param pA: probability of A winning a point on
serve
    :param pB: probability of B winning a point on
serve
    :return: the score of the game for A, and for B
    """
    serving = "A"
    scoreA = 0
    scoreB = 0
    while not gameOver(scoreA, scoreB):
        if serving == "A":
            if random() < pA:
                scoreA += 1
            else:
                serving = "B"
        else:
            if random() < pB:
                scoreB += 1
            else:
                serving = "A"
    return scoreA, scoreB
```

gameOver() is a “helper function” that just determines if either player now has 15 points and thus the game is now over.

gameOver() returns a Boolean.

The random() calls are from a system library we’ve imported

But that’s not sufficient

If you’ve taken Statistics/Math classes, you know that in general the larger a sample is, the more accurate its results are.

- Because “random” numbers sometimes give you weird results

So we don’t just want to simulate one game between our two players. We want to simulate many games - a thousand, or more

- This is the same idea the prediction services use: they run their simulations thousands to millions of times, thinking that the overall averages will be more accurate

Simulating many games

```
def simNgames(n, pA, pB):
    """

    :param n: number of games to simulate
    :param pA: probability of A winning a point on
serve
    :param pB: probability of B winning a point on
serve
    :return: winA: the number of games won by A;
winB: the number of games won by B
    """
    winA = 0
    winB = 0
    for i in range(n):
        scoreA, scoreB = simOneGame(pA, pB)
        if scoreA > scoreB:
            winA += 1
        else:
            winB += 1
    return winA, winB
```

This is a pretty straight-forward function. You call simOneGame repeatedly.

Each call gives you a winner, A or B.

You add one win to the appropriate total.

When you're done, you return the number of wins for A and the number of wins for B. Both of these are integers.

That’s the core of our program, but it’s not all

We need a way to get the inputs we need from the user:

- Player A’s probability of winning a point on their own serve
- Player B’s probability of winning a point of their own serve
- The number of games we need to simulate in order for the prediction to be sufficiently accurate

We need a way to present the results to the user

We really should have some introductory text that is presented to the user, so the user knows what’s going on

And we need a “main program” to tie it all together

Fortunately, you already know how to write each of those things

```
def printIntro ():
    """
    This function prints out introductory text for
the user
    :param: none
    :return: none
    """
    print("This program simulates a game of
racquetball between two")
    print("players called 'A' and 'B' The skill of
each player is")
    print("represented by the probability of that
player winning a point")
    print("when that player serves. This probability
is expressed as a")
    print("floating point number between 0 and 1")
    print("PLayer A will always serve first")
```

```
def getInputs():
    """
    This function prompts the user for three input
values: the probability of
    A winning a point on serve; the probability of
B winning a point on
    serve; and the number of games to simulate.
(More games gives a truer
    probability of winning)
    :return: a: A's probability of winning; b: B's
probability of winning;
    n: Number of games to simulate
    """
    a = float(input("Please enter the probability of
A winning a point when serving: "))
    b = float(input("Please enter the probability of
B winning a point when serving: "))
    n = int(input("Please enter the number of games
to simulate: "))
    return a, b, n
```

More of the units

```
def printSummary(winA, winB):
    """
    This function prints out the results of your
simulation, in whatever
    format is desired.

    :param winA: the number of games won by A
    :param winB: the number of games won by B
    :return: None
    """
    n = winA + winB
    print("Games played: " + str(n))
    print("Wins for A: ", winA, "or :", 100*winA/n,
" %")
    print("Wins for B: ", winB, "or :", 100*winB/n,
" %")
```

```
if __name__ == "__main__":
    printIntro()
    a, b, n = getInputs()
    winA, winB = simNgames(n, a, b)
    printSummary(winA, winB)
```