THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 23
Simulation and Design – II

Objectives

- To understand and be able to apply bottom-up and spiral design techniques in writing complex programs.
- To understand unit-testing and be able to apply this technique in the implementation and debugging of complex programming.

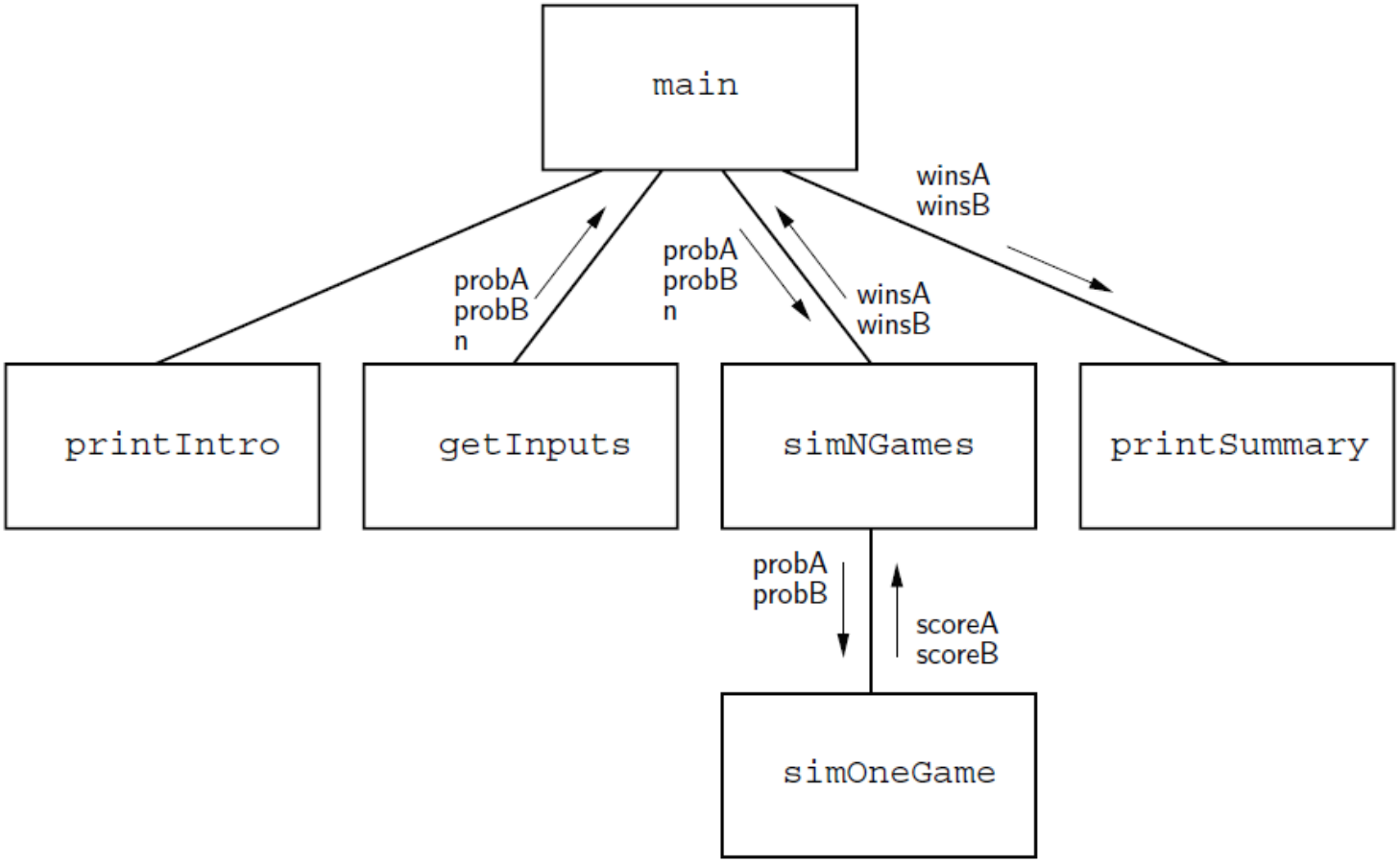
Revision: Simulation

- *Simulation* can solve real-world problems by modeling real-world processes to provide otherwise unobtainable information.
  - *Physical techniques unavailable or very expensive*
- Computer simulation is used to predict the weather, design aircraft, create special effects for movies, etc.

Revision: Top-Down Design

- In top-down design, a complex problem is expressed as a solution in terms of smaller, simpler problems.
- These smaller problems are then solved by expressing them in terms of smaller, simpler problems.
- This continues until the problems are trivial to solve. The smaller pieces are then put back together as a solution to the original problem!

Revision: Racquetball Problem Decomposition



Revision: Summary of the Design Process

1. Express the algorithm as a series of smaller problems.
2. Develop an interface for each of the small problems.
3. Detail the algorithm by expressing it in terms of its interfaces with the smaller problems.
4. Repeat the process for each smaller problem.

Bottom-Up Implementation

- Even though we've been careful with the design, there's no guarantee we haven't introduced some silly errors.
- Implementation is best done in small pieces.
  - *Start with the functions you know you need to put together*

Unit Testing

- A good way to systematically test the implementation of a modestly sized program is to start at the lowest levels of the structure, testing each component as it's completed.
- For example, we can import our program and execute various routines/functions to ensure they work properly.

Unit Testing

- When testing, need to have reproducible behaviour. That is, the program behaves the same way each time it is executed.
  - *For programs involving pseudo-random numbers, this means using seed functions to fix the starting point*

```
import random as rnd
TESTING = True
def main():
    if TESTING :
        rnd.seed(7) # For reproducible behaviour during testing
    printIntro()
    probA, probB, n = getInputs()
    winsA, winsB = simNGames(n, probA, probB)
    printSummary(winsA, winsB)
```

Unit Testing

- We could start with the `gameOver` function.

```
>>> import rball
>>> rball.gameOver(0,0)
False
>>> rball.gameOver(5,10)
False
>>> rball.gameOver(15,3)
True
>>> rball.gameOver(3,15)
True
```

Unit Testing

- Notice that we've tested `gameOver` for all the important cases.
  - *We gave it 0, 0 as inputs to simulate the first time the function will be called.*
  - *The second test is in the middle of the game, and the function correctly reports that the game is not yet over.*
  - *The last two cases test to see what is reported when either player has won.*

Unit Testing

Now that we see that `gameOver` is working, we can go on to `simOneGame`.

```
>>> simOneGame(0.5, 0.5)
(11, 15)
>>> simOneGame(0.5, 0.5)
(13, 15)
>>> simOneGame(0.3, 0.3)
(11, 15)
>>> simOneGame(0.3, 0.3)
(15, 4)
>>> simOneGame(0.4, 0.9)
(2, 15)
>>> simOneGame(0.4, 0.9)
(1, 15)
>>> simOneGame(0.9, 0.4)
(15, 0)
>>> simOneGame(0.9, 0.4)
(15, 0)
>>> simOneGame(0.4, 0.6)
(10, 15)
>>> simOneGame(0.4, 0.6)
(9, 15)
```

Unit Testing

- Testing each component in this manner is called unit testing.
- Testing each function independently makes it easier to spot errors, and should make testing the entire program go more smoothly.
- Then need end-to-end, or integration, testing.

Simulation Results

- Is it the nature of racquetball that small differences in ability lead to large differences in final score?
- Suppose Denny wins about 60% of his serves and his opponent is 5% better. How often should Denny win?
- Let's do a sample run where Denny's opponent serves first.

Simulation Results

```
This program simulates a game of racquetball between two
players called "A" and "B".  The abilities of each player is
indicated by a probability (a number between 0 and 1) that
the player wins the point when serving. Player A always
has the first serve.
What is the prob. player A wins a serve? .65
What is the prob. player B wins a serve? .6
How many games to simulate? 5000

Games simulated: 5000
Wins for A: 3329 (66.6%)
Wins for B: 1671 (33.4%)
```

- With this small difference in ability , Denny will win only 1 in 3 games!

Other Design Techniques

- Top-down design is not the only way to create a program!

Prototyping and Spiral Development

- Another approach to program development is to start with a simple version of a program, and then gradually add features until it meets the full specification.
- This initial stripped-down version is called a prototype. (The method is sometimes called rapid prototyping.)

Prototyping and Spiral Development

- Prototyping often leads to a spiral development process.
- Rather than taking the entire problem and proceeding through specification, design, implementation, and testing, we first design, implement, and test a prototype.
  - *Basis of the Agile design methodologies*
- We take many mini-cycles through the development process as the prototype is incrementally expanded into the final program.
  - *At each step, consult with the client*

Prototyping and Spiral Development

- How could the racquetball simulation been done using spiral development?
  - *Write a prototype where you assume there's a 50-50 chance of winning any given point, playing 30 rallies.*
  - *Add on to the prototype in stages, including awarding of points, change of service, differing probabilities, etc.*

Prototyping and Spiral Development

```
from random import random

def simOneGame():
    scoreA = 0
    scoreB = 0
    serving = "A"
    for i in range(30):
        if serving == "A":
            if random() < 0.5:
                scoreA += 1
            else:
                serving = "B"
        else:
            if random() < 0.5:
                scoreB += 1
            else:
                serving = "A"
        print(scoreA, scoreB)
```

Note: No arguments

Ask yourself: is the function/program doing sensible things?

```
>>> simOneGame()
0 0
0 1
0 1
…
2 7
2 8
2 8
3 8
3 8
3 8
3 8
3 8
3 8
3 9
3 9
4 9
5 9
```

Prototyping and Spiral Development

- The program could be enhanced in phases:
  - ***Phase 1:*** *Initial prototype. Play 30 rallies where the server always has a 50% chance of winning. Print out the scores after each server.*
  - ***Phase 2:*** *Add two parameters to represent different probabilities for the two players.*
  - ***Phase 3:*** *Play the game until one of the players reaches 15 points. At this point, we have a working simulation of a single game.*
  - ***Phase 4:*** *Expand to play multiple games. The output is the count of games won by each player.*
  - ***Phase 5:*** *Build the complete program. Add interactive inputs and a nicely formatted report of the results.*

Prototyping and Spiral Development

- Spiral development is useful when dealing with new or unfamiliar features or technology.
- If top-down design isn't working for you, try some spiral development!

Summary: The Art of Design

- Spiral development is not an alternative to top-down design as much as a complement to it – when designing the prototype you'll still be using top-down techniques.
- Good design is as much creative process as science, and as such, there are no hard and fast rules.
- The best advice?
  - *Three words*
    1. Practice
    2. Practice
    3. Practice