Lecture 19

Topics: *Chapter 9. Simulation and Design*

- Moving to graphics library

9.4.1 Unit Testing

9.5 Other Design Techniques

9.4.1 Unit Testing

When we finish writing a function (a component of a program) it is good to give it a good testing. This process is called ***unit testing***.

Testing each function independently makes it easier to spot an error. By the time we get around to testing the entire program, chances are that everything will work smoothly.

9.4.1 Unit Testing

When we finish writing a function (a component of a program) it is good to give it a good testing. This process is called ***unit testing***.

Testing each function independently makes it easier to spot an error. By the time we get around to testing the entire program, chances are that everything will work smoothly.

Separating concerns through a modular design makes it possible to *design sophisticated programs*.

9.4.1 Unit Testing

When we finish writing a function (a component of a program) it is good to give it a good testing. This process is called ***unit testing***.

Testing each function independently makes it easier to spot an error. By the time we get around to testing the entire program, chances are that everything will work smoothly.

Separating concerns through a modular design makes it possible to *design sophisticated programs*.

Separating concerns through unit testing makes it possible to *implement and debug sophisticated programs*.

9.4.1 Unit Testing

When we finish writing a function (a component of a program) it is good to give it a good testing. This process is called ***unit testing***.

Testing each function independently makes it easier to spot an error. By the time we get around to testing the entire program, chances are that everything will work smoothly.

Separating concerns through a modular design makes it possible to *design sophisticated programs*.

Separating concerns through unit testing makes it possible to *implement and debug sophisticated programs*.

These programs work will less overall effort and far less frustration.

9.4.1 Unit Testing

racquetball game

we should test the following methods/functions:

1. `input()`
2. `simOneGame(Pa,Pb)`
3. `simNgames(n,Pa,Pb)`
4. report(n,WinsA,WinsB)

9.5 Other Design Techniques

Sometimes we can get stuck with the *top-down design*
(i.e. cannot refine one of the steps; or the original specification is so complicated that refining it level-by-level is way too hard)

Another approach to design is to start with a simple version of a program (or a program's component) and then try gradually add features until it meets the full specification.

9.5 Other Design Techniques

The initial stripped-down version is called a ***prototype***.

Prototyping often leads to a sort of ***spiral development process***:

- we design, implement and test a prototype, then
- new features are designed, implemented and tested, and
- we make many mini-cycles through the development process as the prototype is incrementally expanded into final program.

9.5 Other Design Techniques

The initial stripped-down version is called a ***prototype***.

Prototyping often leads to a sort of ***spiral development process***:

- we design, implement and test a prototype, then
- new features are designed, implemented and tested, and
- we make many mini-cycles through the development process as the prototype is incrementally expanded into final program.

Spiral development is particularly useful when dealing with new or unfamiliar features or technologies.

It is helpful to try it out - just see what can we do.

Sometimes for novice programmer everything may seem new, so prototyping might prove useful.

9.5 Other Design Techniques

It is important to note though that spiral development is not an alternative to top-down design.

They are complimentary approaches.

When designing the prototype, you will still use the top-down techniques.

Later you'll see another design techniques.

9.5 Other Design Techniques

I used the code of *racquetball.py* and converted it to graphics:

See *racquetballGraphics.py*

Lecture 18 In-class assignment 3

Craps game

Craps game is a dice game played at many casinos. A player rolls a pair of normal six-sided dice. If the initial roll is 2, 3, or 12, the player loses. If the roll is 7 or 11, the player wins. Any other initial roll causes the player to "roll for point". That is, the player keeps rolling the dice until either rolling a 7 or re-rolling the value of the initial roll. If the player re-rolls the initial value before rolling a 7, it's a win. Rolling a 7 first is a loss.

We want to write a program to simulate multiple games of craps and estimate the probability that the player wins. For example, if the player wins 249 out of 500 games, then the estimated probability is 249/500 = 0.498 or 49.8%.

User will be prompt for the number of games.

Lecture 18 In-class assignment 3

Sketch of the main function:

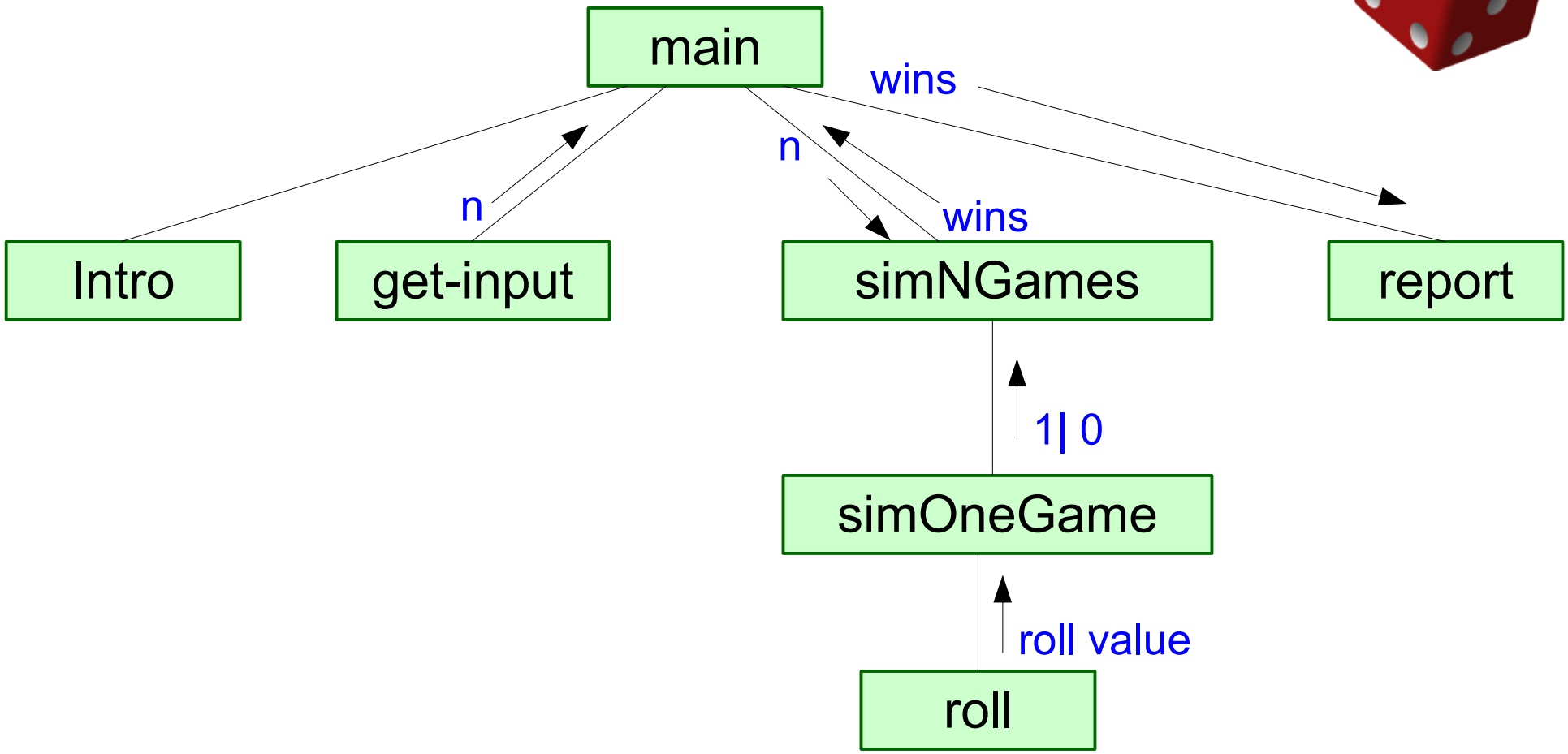
```
def main():
    Intro() # print an introduction

    n = getInput() # get input from the user

   # simulate n games,
   # return the result of simulation
    wins = simNgames(n)

    report(n,Wins) # report back to the user
```

Lecture 18 In-class assignment 3



Lecture 18 In-class assignment 3

Craps game

In-class and homework assignment:

implement the text version of the craps game

make it fool proof (take care of “bad” input)

you may work in groups

Optional: make graphics version (after you finish working on the text version)