CS 101: Computer Programming and Utilization

05-Testing-Debugging

Instructor: Sridhar Iyer
IIT Bombay
Predict what will happen

When the Run flag is clicked repeatedly:

1. Output will be the same as the first Run
2. Sprite will continue to grow more and more
3. Sprite will grow but will stop growing after some number of Runs

Run: demo04-growing-cat.sb
Predict what will happen + Peer discussion

What will the sprite do?
First, write your answer

Then, check your neighbour's answer
Activity: Debugging

Your friend wrote the code of the previous slide, to make the cat jump 3 times and then move across the stage; But the cat is hopping across (Jump, Move, Jump, Move, Jump, Move)

Help your friend to debug the code:

• Think (Individually): What changes will you make in the code, to get the desired result? Multiple solutions are possible

• Pair: See if your neighbour's solution is the same as yours. If not, whose solution is “better”? Why?

• Share: Discuss the solution categories that emerge

Run: demo05-hopping-cat.sb, Do live modifications for debugging
Program Correctness, Testing and Debugging

Correctness:
- For every input, the program should produce the expected output; Related to “Functional Specification”
- **Proving** correctness is the area of Formal Verification
  - Usually attempted only for safety-critical systems

Testing:
- Validating whether the program works as expected for a given set of test inputs

Debugging: Determining causes of failure for a test case, and fixing the errors so that the program passes the testing
How to do testing

0. DO NOT wait to finish writing the entire program!

1. Test each block of code independently, as you write it
   ➔ Start from the smallest block for which you can write test cases; For example, a section of one thread of one sprite
   ➔ This is called Unit Testing

2. Then, test whether two blocks of code work as expected, when run “simultaneously” or when they “interact”
   ➔ For example, two threads in a sprite or in different sprites
   ➔ This is called Integration Testing

3. Finally, test whether the entire program works as expected
   ➔ This is called System Testing
Activity: Testing

Revisit the program on Slide 3

• Unit Testing:
  • What are the units in which your friend could have done Unit Testing? And in which order?
  • First write down a list in your notebook and then order it
  • Again, write down your answer; Don't just sit and stare!
  • Class discussion

• Integration Testing:
  • How will you do integration testing for this program?
  • Class discussion
How to write test cases

0. DO NOT stop as soon as your code “works” for one input case; This is just the beginning of testing!

1. Write down and test the favourable cases first
   - Write down what output you expect to get for each input that you consider 'valid' or 'allowed'
   - For multiple threads, write down the execution sequences for which you expect to get the desired results/output

2. Then, write down and test the boundary conditions
   - For each variable, what if it takes some extreme values, either smaller or larger than what you consider 'valid'
Activity: Writing test cases

Revisit the Arrays program from class 03, slide 5

• Favourable test cases
  • Write down as many favourable test cases as you can think of
    – Done in class discussion mode

• Boundary conditions
  • Write down as many boundary conditions as you can think of
    – Done in class discussion mode
How to write test cases (contd.)

3. After you are done with testing for expected inputs and boundary cases, list down various 'what-if' scenarios:
   ➔ 'what-if' the user does 'this' when I am expecting 'that'
   ➔ 'what-if' this thread reaches 'here' when that one is 'there'

4. Write test cases to check each 'what-if' scenario above
   ➔ generate test cases, or make minor modifications to the code, to ensure that the scenario occurs during your run!

5. Ideally, testing should 'cover' each 'path' in your program
   ➔ For each condition, do a positive test and a negative test

6. Alpha-testing is done and you are ready for a beta-release!
(Optional) Activity: Writing test scenarios

Revisit the program on Slide 3 – test scenario?

Revisit the cat-and-bird demo from class 4, slide 3

- First, write as many 'what-if' scenarios as you can think of; Then, write what input or what change in the program can be used to test each scenario

- Class discussion: activity was deferred to a later class

<table>
<thead>
<tr>
<th>What if Scenario</th>
<th>What will I do to test it</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How to do debugging

You find that your program is not producing the desired result for a given test case, so ...

0. Simplify the test case and check the program output
   ➔ Repeat till you get a test case for which the program works as expected

1. Take each failed test case and do a hand-execution
   ➔ Assume that you are Mr. Dumbo, from lecture 1, and attempt to find the error that is causing the test to fail

2. If hand-execution is not successful, do a print-debugging
   ➔ Introduce 'say' statements with appropriate messages, and 'wait' statements wherever required, to help you to follow the program execution
Activity: Debugging

- The information on generating test scenarios and debugging are included here for completeness, in case you want to read further on your own.
- These activities are deferred to later classes/labs:
  - Simplifying the test case
  - Hand execution
  - Print debugging

- Using a debugger ← What is this?
  - Deferred to a much later lab!
Program Testing

This so important that it is worth repeating the following:

- Test each block of code as you write it
- Never defer testing till you have 'finished' programming
- Debugging small blocks of code is easier than attempting to debug an entire program all at once

- Test for various inputs, conditions and scenarios
  - First, check if your code works as expected for favourable inputs and execution sequences
  - Then, test boundary conditions and 'what if' scenarios

We will keep revisiting these ideas periodically in this course
Why should you care?

- You will start on your Scratch Project in this week's lab
- You will come up with ideas for applications that will require:
  - multiple sprites, multiple threads, lots of programming
  - multiple backgrounds ← what is this? Learn on your own!
- You need to be systematic, both during development and testing phases, in order to successfully demo your project
- Top few projects will get to showcase their demo to all
- This is how you will do software development in any field, not just in the software industry
- This training in systematic thinking will help you in any field!
Muddy Points from last week

Discussion on questions posted on Moodle:

- Executing both parts of 'If-else block':
  - Why is it not possible?
  - Is it 'smart' to write statements like if (!(printf("hello ")))?
- Time of execution: ← Good question
  - Explicitly determine the time of execution of a sprite?
  - Does 'wait 10000 sec' mean the cpu rests for this time?
  - Is it an atomic statement?
- Program for one sprite to follow another:
  - See first answer posted on Moodle ← Good answer
  - Come up with another way to do the same ← Quiz!
What all have we learnt in Scratch?

• Quickly revisit slides of all previous lectures
  • Recall the programming concepts, scratch skills, and software design practices discussed so far

• Take two minutes to fill out Muddy Points chits for drop box
  • You need not wait to post on Moodle
  • You can ask questions anonymously, if you wish

• Today: You should submit one or two muddy points from topics done in the class/lab so far
Next class - from Scratch to C++

- Slides 29-36 from cs50.pdf

- What functionality are identical, requiring only syntax translation?

- What functionality are similar but not identical?

- What functionality are missing?