Programming Project 1

Handout 7
CSCI 134: Spring, 2008
5 March

Guidelines

A programming project is a laboratory that you complete on your own, without the help of others. It is a form of take-home exam. You may consult your text, your notes, your lab work, our on-line examples, and the web pages associated with the course web page, but use of any other source (human or otherwise) for code is forbidden. You are encouraged to reuse the code from your labs or our class examples.

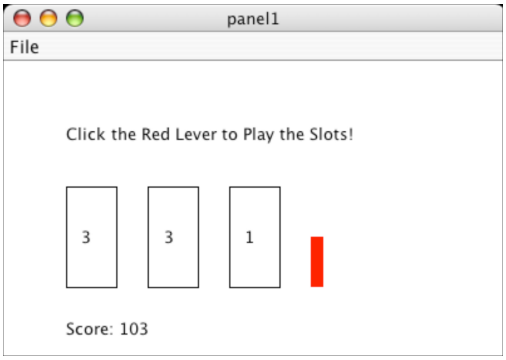
You may not discuss these problems with anyone aside from the course instructors. You may only ask the TA's for help with hardware problems or difficulties in retrieving your program from a disk or network. The use of any other outside help or sources is a violation of the Honor Code.

This project is due by 4pm on Friday, March 14. No late work will be accepted.

The projects will be graded out of 100 points, with roughly half the points given for correctness and half the points given for style. A more detailed breakdown appears at the end of this document. Thus, even imperfect programs can receive close to full credit, but poor design and style can lead to quite low final scores.

Program 1: Pull Your Luck!

The Commonwealth of Massachusetts is considering the economic and cultural impact of casinos on the state. You've been asked to write a simple slot-machine simulator as part of their study. The slot machine should have three slots, initially empty, and a red lever. Additionally, there should be a score below the slots, initially 100, and some text providing direction on how to play. Here is an example of the simulator after one pull where the player wins according to Rule 2 below.



When the player clicks on the lever, a random integer value in the range 1–5 appears in each slot. Based on the outcome, the player will either gain or lose points. The Commonwealth is considering several scoring systems. The system they want you to test and implement is as follows:

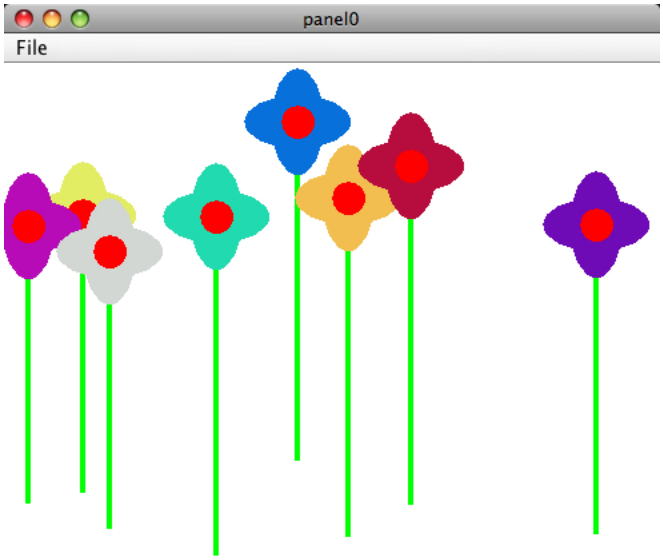
1. If all three slots have matching value x, the player receives $2 \times x$ points.
2. If the first two slots match but the third does not or if the final two slots match and the first does not, then the player receives x points where x is the matching slot value.
3. In all other cases, the player loses 2 point.

Here are some example slot pulls and their respective scores:

- $5 - 5 - 5$ [10 points — rule 1]
- $3 - 3 - 4$ [3 points — rule 2]
- $1 - 2 - 1$ [-2 points — rule 3]
- $2 - 4 - 1$ [-2 points — rule 3]

Program 2: Popping Up All Over

We are tired of snow. But it's almost spring, and the flowers will be popping up through the snow at any moment. This program will help you prepare for that day by rendering that moment in true objectdraw-fashion:



The canvas initially starts empty. When you click on the canvas, a flower starts growing at that point. Initially it will be just a sprout, but as you drag the mouse around, it grows. When it reaches its full height, the stem stops growing and petals appear. The color for the petals should be chosen randomly. The flower won't grown any more, but if you click on the petals after it has bloomed, the flower will change color.

When you are happy with the flower's color, you can grow another one by clicking somewhere else in the window. When the mouse moves out of and then back into the window, the scene resets itself to be empty.

Program Design. Your program should be divided into two classes: a window controller called `Spring` and a `Flower` class. We have actually provided a complete `Spring` class in the starter folder. You should not modify our `Spring` class in any way. Instead, you have to implement the `Flower` class so that it works with our `Spring`. In particular, the `Flower` constructor should expect three parameters: the Location where the `Flower`'s stem should be planted, a double specifying the maximum height of the `Flower`, and the canvas. The `Flower` class should define the following methods used by the controller to implement the functionality described above:

- `public void changeColor()`: sets the color of the flower petals to a random color.
- `public boolean flowerContains(Location point)`: returns true if the petals or center of the flower contains the point.
- `public void grow()`: make the flower grow a bit, or make petals appear when it reaches its full height.

The flower will only grow when the mouse moves, so `Flower` should not extend `ActiveObject`. Feel free to add other graphic items to the `Flower` class to make the flowers look more attractive.

Program 3: Three-Cow Monty

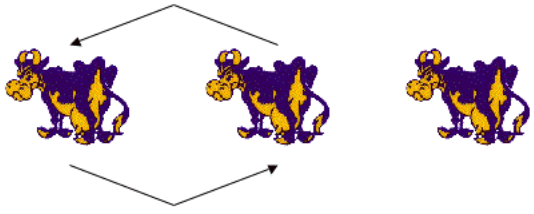
The goal of your final program is to play a simple game called "Three-Card Monty" (sometimes also known as the Shell Game). Of course, this is Williams, so our version actually plays "Three-Cow Monty". The program begins with three seemingly identical purple cows lined up in a row as shown below:

Correct = 0 Wrong = 0

However, one of the cows is different. If you click on the middle cow, it moos and you score a point for picking the correct cow. If you click on either other cow, they do not moo and your incorrect score increases by one. In itself, this is not much of a game, but if you click outside any cow the program shuffles their order in front of you. The goal is to follow the moving cows and click on the one that moos after they have been shuffled.

Your program should be divided into two classes- the `ThreeCowMonty` window controller class and the `Shuffler` (or Moover!) active object class. The `ThreeCowMonty` class creates the three cows and handles the mouse interaction. The `Shuffler` moves the three cows, as illustrated in our demo. It selects two cows and gradually moves each of them from its original location to the opposite cow's location. Once those two cows are repositioned, it randomly selects two new cows and moves them again, repeating this process perhaps 10 or 20 times before stopping. While it is more natural to describe the shuffler as randomly picking two cows to move, you may find it is easier to write the required code if you instead think of the shuffler as randomly picking one cow that it won't move and then moving the two that were not picked. The `ThreeCowMonty` class should prevent anything from happening if the user clicks while a `Shuffler` is shuffling the cows. In other words, the window controller should not test whether you click on the correct cow or create a second `Shuffler` if a `Shuffler` is currently running.

You may wish to start by just moving the cows horizontally until the shuffling works. The up-and-down motion makes the game more entertaining, but it can be added after the rest of the program is working. The motions of the cows in the demo look like smooth curves, but they actually move in straight lines, as in the following picture:



We provide you with the cow picture and the mooing noise in the starter folder, but feel free to shuffle your choice of livestock, frogs, basketballs, or anything else.

If the code for your `Shuffler's run` method begins to get complicated, remember that you can define private methods to perform common operations and then invoke these methods from `run`.

Submitting Your Work

Once you have saved your work in BlueJ, please perform the following steps to submit your assignment. *Please do not submit three separate folders. Instead, place the folders for all three of your complete programs into one folder, make sure that your name appears in the title of the main folder and each of the subfolders, and then place the main folder in our dropoff folder.*

- First, return to the Finder. You can do this by clicking on the smiling Macintosh icon in your dock.
- Click "Go" and select "Conect to Server."
- For the server address, type in "Cortland" and click "Connect."
- Select the button next to "Guest" and click "Connect."
- A selection box should appear. Select "Courses" and click "Ok."
- You should now see a Finder window with a "cs134" folder. Open this folder.
- You should now see the drop-off folders for the three labs sections. Drag your project folder into the appropriate lab section folder. When you do this, the Mac will warn you that you will not be able to look at this folder. That is fine. Just click "OK."
- Log off of the computer before you leave.

Grading Guidelines

Points will be assigned roughly as follows:

Style (16 pts per program)

- Proper use of boolean conditions
- Proper use of ifs/whiles
- Proper use of variables (instance/local, public/private)
- Descriptive comments
- Good names
- Good use of constants
- Appropriate formatting (indenting, white space, etc.)
- Parameters used appropriately

Correctness (16 pts per program)

- Pull Your Luck

  Initial screen layout
  Reponds to mouse click
  Picks and displays 3 random numbers
  Implements scoring rules
  Updates score on screen

- Popping Up All Over

  Initial stem and bulb created correctly
  Stem grows and bulb rises
  Flower opens when stem reaches height
  Flower color changes correctly

- Three-Cow Monty

  Cows placed correctly initially
  One cow moos
  Scores awared and displayed correctly
  Pairs to move chosen randomly
  Motion of cows animated correctly
  Motion of cows animated simultaneously

Miscellaneous (4 pts)

Extra Credit (up to 4 pts)