

Programming Project 1

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.

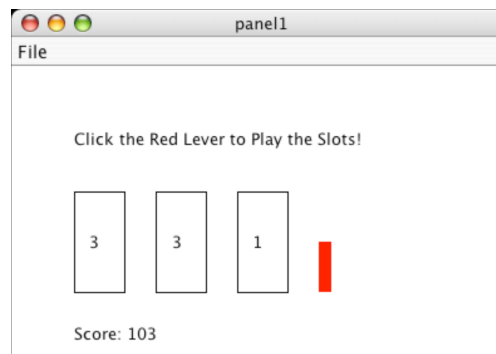
You may not discuss these problems with anyone aside from the course instructors. You may only ask the TA's for help with hardware problems or difficulties in retrieving your program from a disk or network. The use of any other outside help or sources is a violation of the Honor Code.

This project is due by 4pm on Friday, October 22. 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:

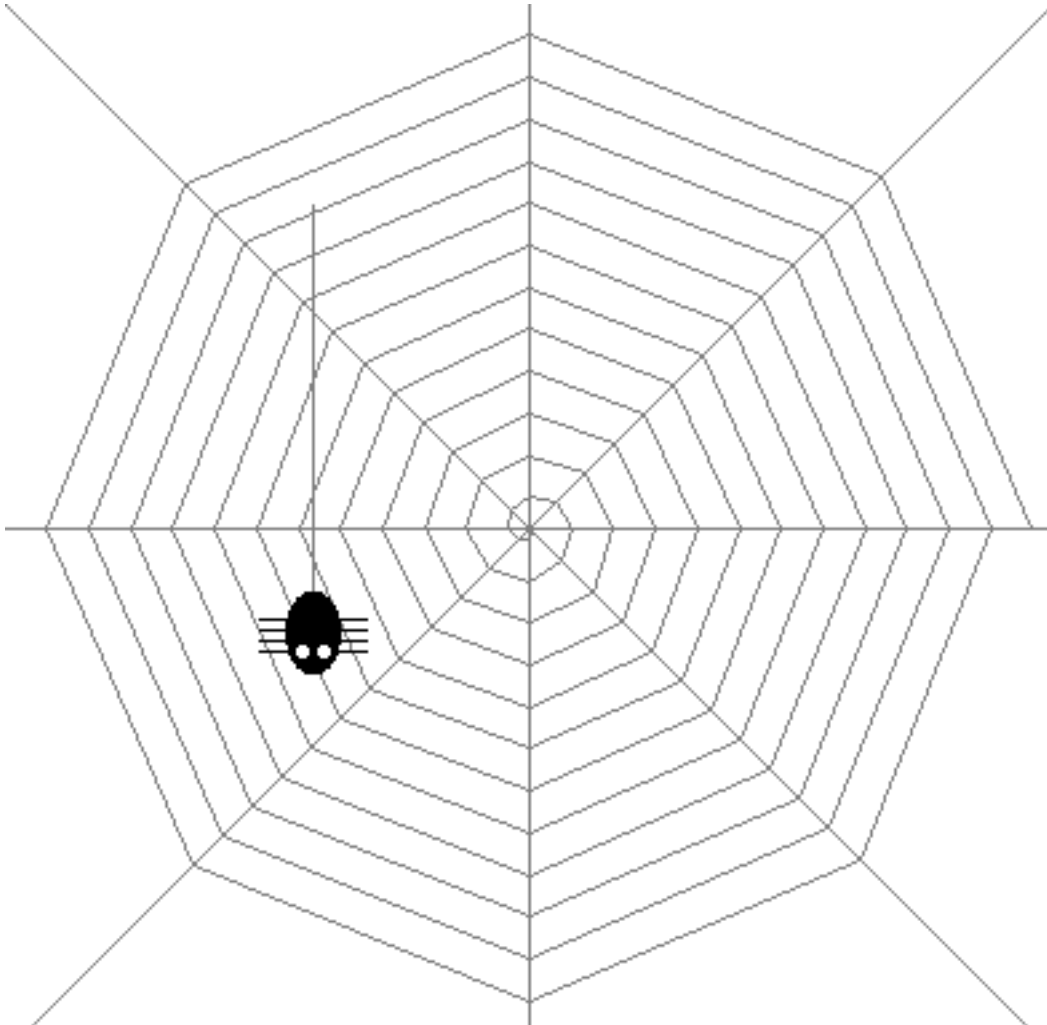
1. If all three slots have matching value x , the player receives $2 \times x$ points.
2. If the first two slots match but the third does not or if the final two slots match and the first does not, then the player receives x points where x is the matching slot value.
3. In all other cases, the player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

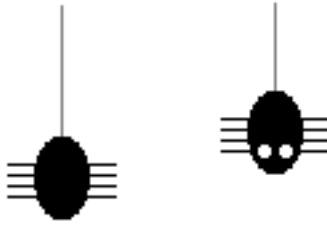
Program 2: Hanging By a Thread

This question asks you to write a program to animate a simple spider.



The canvas should initially be empty, except for the spider web background. (The starter code reuses our spiral program to draw the web for you.) When you press the mouse in the window, the spider appears centered at the location where you clicked. As you drag the mouse, the spider should descend on a gray thread that stretches as the spider moves down the screen. After descending for a while, the spider turns around and climbs back up the thread to where it began. On the climb up, the spiders eyes should be open and its thread should disappear as it climbs. When the spider again reaches where it started, it turns around and start its track down and up again.

Here are close-ups of what our spider looks like with its eyes open and closed. Feel free to enhance the spiders appearance in any way you like (provided that it behaves according the rules we have described).



When the mouse is released, the spider and its thread disappears. Pressing the mouse at a different location will cause the spider to appear at that position.

Your program should be divided into two classes: a window controller called `SpiderController` and a `Spider` class. *You will only need to modify the `Spider` class.* We have written the `SpiderController` class for you. For the `Spider` class, the `Spider` constructor should expect three parameters: the `Location` where the spider should appear, a `double` specifying the how far the spider will descend, and the `canvas`. The `Spider` class should define the following methods used by the controller to implement the functionality described above:

- `public void move(double distance)` moves the spider by the specified distance. The spider should keep track of whether it should move up or down by that amount.
- `public void removeFromCanvas()` makes the spider disappear from the canvas.

The spider will only move when the mouse moves, so `Spider` should not extend `ActiveObject`.

Program 3: Three-Cow Monty

The goal of your final program is to write 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:



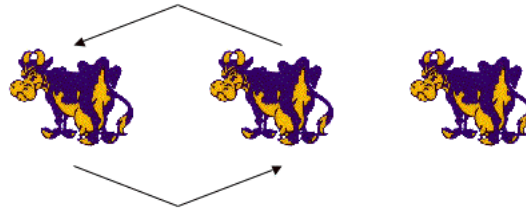
Correct = 0 Wrong = 0

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 `Moover` active object class that performs the shuffling. The `ThreeCowMonty` class creates the three cows and handles the mouse interaction. The `Moover` 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 `Moover` is shuffling the cows. In other words, the window controller should not test whether you click on the correct cow or create a second `Moover` if a `Moover` 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 several cow pictures to choose from and the mooing noise in the starter folder, but feel free to shuffle your choice of livestock, basketballs, or anything else. (Suggestion: you may find it useful to make the correct cow look different while testing your program.)

If the code for your `Moover`'s `run` method begins to get complicated, remember that you can define additional additional methods in your class 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 “Connect 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

- Hanging By a Thread

- Spider and thread drawn correctly
- Spider climbs down
- Eyes open and spider climbs up
- Spider disappears

- Three-Cow Monty

- Cows placed correctly initially
- One cow moos
- Scores awarded 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 6 pts) While not required, feel free to experiment and add embellishments to your programs once you have the basic features working. You may receive a small amount of extra credit for any extensions.