Comp160
Lab 7

Fall 2018

In Section 3.7 we're introduced to the virtual pet world which we finally bring together starting in Section 5.11. In class and in this lab we'll be working on bringing this program together along with a few cool features.

Lab 7

Let's collect a few features of the virtual cat that are presented in section 3.7 and 5.11 and add one more feature.

- Draw different cats on even and odd valued locations for a better animation
- Cat happiness is on a 0 to 100 scale and goes down by 0.1 every tick. Pressing the down arrow (petting the cat) will increase happiness by 1/5 point. Press the up arrow (feeding the cat) will increase happiness by 1/3 point. Happiness should go no higher than 100 and no lower than 0. When the happiness is 100 the bar goes away. When happiness is 0, the game is over.
- The cat moves back and forth across the scene, changing directions when it reaches the edge of screen.
- Draw a red border around the scene when the cat's happiness is below 20.

You've been given a starter for this program. This starter includes complete data definitions with templates for all defined data. This lab is meant to be an exercise in programming by following the template. The new element to this methodology is that we'll aggressively design new functions to manage the fields of the structure. Lab is mostly about setting up this design. We'll finish the implementation (possibly in class) once that's done.

Tock

Design by template thinking suggests that we write three new functions, one that manages the change per tick of each field of the structure. Let's call these functions *tock-loc*, *tock-hap*, and *tock-dir*.

1. Write signatures, purpose statements, and headers for each of these functions. They each must take their respective data type

(location, happiness, or direction), but might also need more information so think carefully about the signatures. We also need to be careful when turning the cat around. If the cat is on the right side of the scene and should turn around, then we should be certain not to actually advance the cat to the right but instead take a step to the left. This makes turning around not just changing direction, but taking a step in the opposite direction that you're currently moving.

2. Use the data definitions for location, happiness, and direction as a guide to write a complete set of tests for each of these functions.

3. Utilize your new helper functions to write the complete definition for *tock-vcat*.

4. Now go back and finish the definition for your tock helper functions and debug any issues you encounter. If you lack full test coverage, then write more tests.

Drawing

Now turn your attention to *draw-vcat*. Once again, we'll develop helpers to manage the vcat field values. For this task you get a bit more freedom with the helper design. Write any set of helpers you deem appropriate under the following restrictions: they can only take as inputs some combination of location/happiness/direction data and they must return an image. Try to think about how each individual field gets "drawn" such that your design has a similar feel to tock where each helper worked out the next location, happiness, and direction respectively.

5. Write signatures, statements of purpose, and headers for your helper function design.

6. Complete the definition for *draw-vcat* using your helpers. The definition of *draw-vcat* must not contain a conditional. Here we're trying to imagine how the functions all come together before we get too invested in implementing our design. Keep tweaking your helper design as needed until you can clearly state, in Racket, how they all come together in the definition *draw-vcat*.

7. Develop tests for your helper functions functions. Run and debug them until all tests pass and the helpers have full test coverage.

Stopping

The starter is not setup to stop the program when happiness reaches zero. You program should, at this point run and animate the moving cat and the steady decrease of the cat's happiness.

8. Add a *stop-when?* event to the big-bang and design the function for detecting the program termination event. Try to stick to the "follow the template" design you've been doing so far.

Key Events

The key event handler poses an interesting challenge. It takes two inputs and both have a template. Which should we follow?

9. Provide signature, purpose statements, and headers for the helpers you would design if *keys-vcat* first follows the CatKey template. Hint: If *keys-vcat* takes care of the CatKey structure, then your helpers are left to manage the VCat structure. Remember to follow the VCat template!

10. Provide signature, purpose statements, and headers for the helpers you would design if *keys-vcat* first follows the VCat template. Hint: If *keys-vcat* takes care of the VCat structure, then your helpers are left to manage VCat field structure and CatKey structure.

11. In a comment, briefly discuss which design you prefer and why.

12. Finish your preferred design.