CS 1102

A term. 2013.

Assignment #2

Due Tuesday September 17, 2013 @ 9:00 a.m. via turnin

For all assignments in this course, you must create your functions according to HtDF templates. Therefore, you are expected to supply test cases and commenting guidelines for function headers.

You will need to use the 2htdp/image and 2htdp/universe packages.

Part 1: extending our cat example

You will expand our example of moving a cat image across the screen. You should start with the "changing cat starter.rkt" file on the course myWPI page. Your program should have the following behaviors:

1. The cat moves `SPEED` pixels upwards on each clock tick
2. On each clock tick, the cat should rotate at a rate defined by `ROTATION-SPEED`
3. When the cat's movement takes it to the top of the screen, it should wrap to the bottom of the screen. The wrapping does not need to be smooth, as in the cat appearing half at the top and half at the bottom of the screen. Having the entire image move to the bottom of the screen, as we did in class, is fine.
4. Pressing the "1" key should cause the first cat image to be displayed. Pressing the "2" key should switch the cat displayed to the second one in the list (see provided starter file).
5. When the displayed cat changes, it is ok if your program forgets the current amount of rotation and starts the new cat off with no rotation.

Representation: the state information is a bit more complicated than the example we did in class as you will need to maintain both the location of the cat, and which cat is currently being displayed. For this question on this assignment, represent these pieces of information as a list of length 2 such that the first element is the cat's position, and the second element is that image of the cat to be displayed. There are better approaches, but one goal is for you to see why we have better solutions to this problem than lists of mixed types.

Part 2: fireworks

For Part 2, you will create a fireworks display that is initiated via the mouse. Whenever the user clicks on the screen, a firework is created that travels upwards until its timer reaches 0, at which point it explodes. This program should have the following interactivity:

1. The background scene should be black (since we see fireworks at night)

2. A mouse click action creates a new firework
3. A firework is displayed as a circle of radius FIREWORK-RADIUS
4. There is no limit on the number of fireworks created by the user

The representation of fireworks is more complicated than the cat example in Part 1. Your program will have to be able to keep track of an arbitrary number of independent fireworks.

1. The x and y coordinates of the firework are identical to the mouse coordinates when the firework is created
2. A firework has a vertical speed based off of the constant `DY`. The rate at which a firework goes up on the screen is initialized with a random value over the range [0, `DY`]+1.
3. A firework has a horizontal velocity based off of the constant `DX`. Horizontal velocity is initialized with a random value over the range [-DX/2, +DX/2]. For example, if DX is 4, then a firework is created with a horizontal velocity between -2 and 2. In other words, fireworks can travel left or right.
4. A firework has a color, which is randomly generated with an alpha channel of 255.
5. Fireworks will explode when its fuse burns up. This delay is a random value in the range [0, FUSE]+1 (the "+1" is to avoid ground bursts :-).

On each clock tick, the fireworks have the following behavior:

1. Each firework moves upwards and horizontally according to its individual velocity components.
2. Its fuse decreases by 1
3. When the fuse reaches 0, the firework explodes

When the firework explodes, it should have the following behavior:

1. The explosion is of the same color as the firework.
2. The explosion is represented as a circle of radius EXPLODE-RADIUS
3. The explosion should remain on the screen for EXPLOSION-DURATION clock ticks
4. Explosions should retain the same horizontal and vertical velocity as the originating firework
5. After the firework explodes, all traces of it are gone both from the screen and from your program. In other words, your representation should not store "old" fireworks.

Tips:

1. Getting functions to work correctly before you start working with the worlds interface could save you some grief.
2. Start with testing functions individually, and thinking of some useful test cases.
3. Implementing a world with 1 firework that follows the above guidelines is a good next step.
4. Then think about how to generalize your design to include an arbitrary number of fireworks.

Assignment requirements:

1. You must include all data definitions and function templates. The data definitions for Part 2 of the assignment will be more complicated.
2. Functions must specify what they consume and produce, and what their functionality is.
3. You must create test cases. For assignment #1 some groups skimped out here. Show us that you've thought of a reasonable set of things that can go wrong. At a minimum, your test cases must execute all lines of your program. You should also consider edge conditions.
4. If the assignment refers to a constant in capital letters, you should use the same constant name in your program.
5. The starter files provide the name of functions you should be creating. This approach will simplify our testing of your code later.
6. Submit two racket files, "cat.rkt" and "fireworks.rkt"