CIS 201 – Computer Science I
Laboratory Assignment 4

Introduction

In this lab you will create a game that will incorporate:

- selection (`if` and `if...else...` statements)
- boolean expressions
- the `Color` class and customized colors
- timed animations

The `Racing` game

Create a `Lab04` directory in your `CS1` directory. In that directory, create a Java file with a `Game` class named `Racing`.

Be sure to create an empty class template in the new file with a header comment, an import statement, a class declaration, and our standard two methods, as follows:

```
/**
 * Describe class Racing here.
 *
 *
 * Created: Aug. 3, 1978
 *
 * @author <a href="mailto:student1@potsdam.edu">Indiana Jones</a>
 * @author <a href="mailto:student2@potsdam.edu">Luke Skywalker</a>
 * @version 1.0
 */
import fang.core.Game;

public class Racing extends Game {
  /**
   * Describe setup method here
   */
  public void setup() {
```

```
  }

  /**
   * Describe advance method here
   *
   * @param dT time (in fractional seconds) since the last call to advance
   */
  public void advance(double dT) {

  }
} // class Racing
```

Add a car

For this game, a car is simply a pie slice drawn on the screen. You should add a `PieSprite` field to your game and, during setup, construct it with the following parameters: width=0.35, height=0.35, start=185.0, and end=195.0. Locate the car at the point (0.1, 0.9). If you want more details about how to construct a `PieSprite`, look it up in the FANG documentation. Study the `NewtonsApple` code to see how to get things started. You will need to give a variable name to the car you are creating. What would be a good name for this variable?? The code to add a car should be in your `setup` method. Be sure to use `addSprite` to add the car to the game canvas.

The car should be some shade of red. You can use the `getColor` method to create a custom color if you call it with three integers that represent the amount of the three primary colors red, green, and blue. All three parameters are on the range 0..255; all at 255 is white, all at 0 is black. Feel free to experiment a little bit. Use the value returned by `getColor` to set the color of the car.

Checkpoint 1

Show us your working code at this point.

Add walls

Add two long, thin rectangles that extend eighty percent of the way across the screen, one from the right edge of the screen and one from the left edge of the screen. The right-edge wall should be about 1/3 from the top of the screen, and the other wall should be about 1/3 from the bottom. These will be the walls that your car must avoid as you “drive” the car from the bottom of the screen to

the top. Be sure that the walls give sufficient room for the car to turn around and to get from the bottom of the game canvas to the top.

You should select your own custom colors using the `getColor` method that expects three integers. Make the top wall lightly colored, make the bottom one darker, and make sure both are visible on the black background of the game.

Checkpoint 2

Show us your working code at this point.

Make the car turn

Part of what makes racing games fun is the need to steer. Your racing car moves one direction: straight ahead. In its initial position, with no rotation, it is pointing to the right. You will need to write code to have the car turn a small amount to the left or right when the left or right arrow keys are pressed.

Looking at the `Sprite` documentation, you will see a method called `rotateRevolutions` (along with `rotateDegrees`, `rotateRadians`, and `rotateRevolutions`). These methods turn the sprite (in your case, the car) by some amount. If we knew how many revolutions the car should turn per second, then we could scale that value (just like we scale the speed of falling apples) inside of the `advance` method.

In particular, if we have a variable `carTurningSpeed` in revolutions per second, then we would be able to use something similar to what we did with the apple and pass the scaled value to `rotateRevolutions`. For this game, you should set the car turning speed to 1.5 revolutions per second.

Where will you declare the `carTurningSpeed` variable, and where will you initialize it to 1.5?

Next, add a *multi-way* `if` statement to `advance` that will turn the car. If the you press the left-arrow key, turn the car to the left; if you press the right-arrow key, turn the car to the right. What methods can you use to detect if the left- or right-arrow keys have been pressed?

Checkpoint 3

Show us your working code at this point.

Move the car

Now it is time to move the car. You will do exactly what you did to move the apple but instead of `translateY` or `translateX`, you will use the `forward` method of `Sprite`. What does `forward` do? What parameter does it expect? Read the documentation and see if you have questions.

If you press the up-arrow key, you should move the car forward with its speed scaled by the time since the last `advance`. You will need a variable to hold the car's speed; it should be about 4 screens/second. The check for the up-arrow key should be part of the same multi-way `if` you did above.

Timing

Let's just keep track of elapsed time. We will start by adding a field variable `elapsedTime` that is initialized to `0.0`. Again, where should it be declared and where should it be initialized?

The elapsed time also needs to be updated. Since `advance` gets called with the parameter `dT` equal to the number of seconds since the last call, you can update the elapsed time in `advance` by adding the time since last call to your elapsed time variable. However, you should update the elapsed time only if the game has not finished. How can you tell if a game has finished? Look at the documentation.

You also need a `StringSpirte` to display the elapsed time. Initialize it properly and position it somewhere inside the upper wall. Make an `updateTimer` method which formats the elapsed time as "MM:SS" where "MM" is the number of minutes and "SS" is the number of seconds elapsed. The `setText` method of a `StringSprite` allows you to determine what gets displayed in the sprite.

Problem: We want the number of seconds in the current minute (as well as number of minutes). The `elapsedTime` variable is a double that gives you the number of elapsed seconds. You will need to define a local integer variable to hold the current number of seconds and another local integer variable to hold the current number of minutes.

Assign the elapsed time to the seconds variable first – you will need to use a cast to do this. Then find the minutes by dividing the seconds by 60, and finally modify the seconds to be the remainder upon division by 60. The integer division and modulus operator (\%) should be used here:

```
int seconds = (int) elapsedTime;
int minutes = seconds / 60;
seconds = seconds % 60;
```

Once you have the number of minutes and seconds, you can set the text of the `StringSprite` to the expression `minutes + ":" + seconds`.

Checkpoint 4

Show us your code at this point.

Crash

Add crashing to the game. If the car hits either of the walls, it should have its position and rotation set back to their initial values. You may use only one `if` statement to handle collision with either wall. Suppose `car` is the race car and `wall1` is one of the walls. The boolean expression

```
car.intersects(wall1)
```

will return `true` if the car and the wall intersect, and will return `false` otherwise. What boolean expression should you use to determine if the car intersects *either* `wall1` or `wall2`?

If the car collides with either wall, use `setLocation` and `setRotation` to reset the car to the initial settings, putting the car back at start.

Question: Should the collision detection be part of the multi-way selection statement for the three arrow keys?

Checkpoint 5

Show us your work at this point.

The finish line

You should check if the car has driven into the "finish zone" at the upper right of the screen. You should do this in a single `if` statement. Check for winning only after you check for wall collision!

The finish zone is above your topmost wall to the top of the screen. When the driver drives into the finish zone, turn the color of the score to red and stop the game.

There are a couple ways to see if the car is in the finish zone. The first would be to check the location of the car (the `getX()` and `getY()` methods will do the trick) and check to see if it's within the appropriate rectangle. The other way is to create a "finish zone" rectangle sprite that is colored black (so you don't see it) and that occupies the finish zone. Then the game is won if the car has intersected that rectangle.

When the user enters the finish zone, set the game to be over and turn off the advancing of the timer. Setting the game to over is easy: just call the `stop()` method.

Checkpoint 6

Show us your finished game.