Object-Oriented Design
COMP 2000
Spring 2024

Lab 8 Supplement 2

Once you have a ship displayed on the screen you will want to get it to move and turn. In order to accomplish this you will need to utilize the fact that the `Asteroids` class implements the `KeyListener` interface, providing implementations of `keyTyped()`, `keyPressed()`, and `keyReleased()`. (see https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/event/KeyListener.html).

- The three methods of `KeyListener` that are required in order to implement that interface each respond to keys on the keyboard in different ways. Even though we will leave `keyTyped()` empty, it must be added to satisfy the conditions of the `KeyListener` interface.

- You will have to decide which keys on the keyboard you want the user to press to control your ship (if you select something other than the arrow keys, please make sure you document that clearly so that I don't have trouble testing your code). You will need a key to move forward, one to turn right, and one to turn left. To keep track of which key is currently being pressed, add `boolean` instance variables to your `Asteroids` class for the forward, and left and right turn keys. Each variable will be set to `true` if the corresponding key is being pressed, and to `false` if it's not.

- All of the `KeyListener` methods take a `KeyEvent` object as their parameter. This object contains information about which key was pressed or released. You can get the key code, which is a number representing the key pressed or released by calling the non-static method `getKeyCode()` for the `KeyEvent` instance that was passed in. The list of the constants corresponding to the key codes is on the `KeyEvent` API page. Fill in the `keyPressed()` and `keyReleased()` methods so that when the forward key is pressed or released, its corresponding boolean variable changes appropriately.

- Note: the `Asteroids` game is registered as a `KeyListener` in the `Asteroids` constructor method with the following: `addKeyListener(this);`

Now let's get moving:

- Because we coded the `KeyListener` methods to update a `boolean` instance variable of `Asteroids`, we need to create a mechanism that allows this information to get passed along to your ship. In order to do this we need to add a new method to `Ship` that can be called from `Asteroids` that will store the information in a `boolean` instance variable of `Ship`. The signature of this method should be `public void thrust(boolean value);`

- Now in the `paintComponent()` method of `Asteroids` add a call to `ship.thrust()` passing it the value of `forward`. Also add a call to `ship.update()`, for reasons that will be clear in the next step.

- Edit the method `public void update()` in your `Ship` class. This method will change the position of your ship if the forward key is being held (which should now be locally stored information, thanks to the `thrust` method). To do this, think about what variable holds your ship's current position? How do you access the x coordinate and y coordinate of its current position?

- To change the rotation of your ship if either of the turn keys are being held you should add logic to the `paintComponent()` method of `Asteroids`. Depending on the values of the boolean variables for right and left you should make calls to `ship.rotate()`. See the source code for Polygon where this method is defined.

- When you run the program you should see the ship move when you press the forward key and then stop when you release the key. If you hold forward long enough, the ship will disappear off the edge of the screen.

What remains is to tweak the code to make sure that the ship moves the way we want.

- First, when the ship goes past the edge of the screen it should appear to come back into the screen from the opposite edge (left/right and top/bottom). In order to do this you need two things: (1) you need to know the dimensions of the window (how will ship know this?); and (2) you need to know how to detect when the ship goes off screen and how to adjust the coordinates to perform the wrap-around behavior.

- Next, it is likely that your ship continues to move in the same direction regardless of the direction it is facing. In order to have it move in the direction it is facing you need to use some trigonometry make sure you are incrementing the x and y coordinates the correct amount to go in the desired direction. Specifically, if you are currently incrementing the x and y coordinates by the same amount, you need to instead multiply the increase in the x coordinate by `Math.cos(Math.toRadians(rotation))` and multiply the increase in the y coordinate by `Math.sin(Math.toRadians(rotation))`.

- Finally, rather than having the ship move linearly (move by a constant distance for each time interval in which the forward button is pressed), we would like to simulate zero-gravity acceleration. This will allow the forward key to act as an accelerator rather than a go/stop key. The code for this uses the `accelerate()` method in the original Lab 8 handout. You will modify the `update()` method to increment the x and y using the instance variable `velocity` rather than a hard-coded constant.

Now that you have a fully operational ship, you can use what you learned so far to create another subclass of `Polygon` called `Asteroid`.

- Just like you instantiated a `Ship` object in the constructor for `Asteroids`, you can also instantiate a `List` of `Asteroid`. It should be declared like this:

  ```
  private List<Asteroid> asteroids = new ArrayList<Asteroid>();
  ```

- Then in the `paint()` method of `Asteroids` call each of the asteroids' `paint()` methods just like you did with `ship`.

- Make sure that you place each asteroid in a different location on the screen. You can also give each one a fixed rotation and velocity.

All work must be done individually. Never look at someone else's code. Please refer to the course policies if you have any questions about academic integrity. If you have trouble with the assignment, I am always available for assistance.