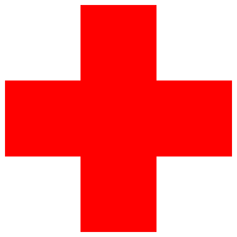
This week's questions will all center and revolve around working with PGL graphics in preparation for the midterm.

1. Write a graphical program that using the PGL library to accomplish the following:

    1. Creates the below cross as a `GCompound` containing two filled rectangles:

       The color of the cross should be red, as in the emblem of the International Red Cross. The horizontal rectangle should be 60 × 20 pixels in size and the vertical one should be 20 × 60. They cross at their centers.

    2. Add the cross to the graphics window so that it appears in the exact center of the window.

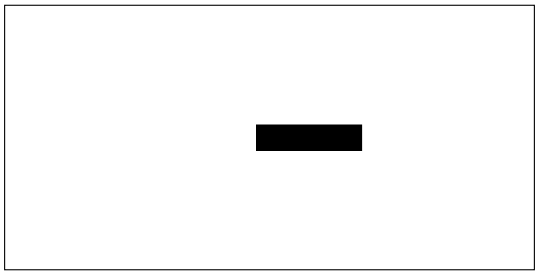
    3. Moves the cross at a speed of 2 pixels every 20 milliseconds in a constant random direction, which is specified as a random real number between 0 and 360 degrees.

    4. Every time you click *inside the cross*, its direction changes to some new constant random direction–again chosen as a random real number between 0 and 360 degrees–but its speed remains the same. Clicks outside the cross have no effect.

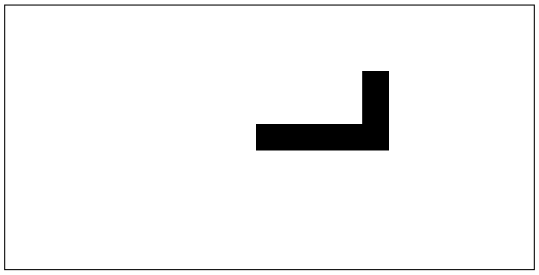
    If you were actually working on a program with this setup, you would presumably supply some means of making it stop, such as when the cross moves off the screen. For this problem, just have the program run continuously without worrying about objects moving off-screen or how to stop the timer.

2. In this problem, your mission is to write a program that plays a very simplified version of the graphical game called "Snake", which was popular on the first generation of Nokia phones. When the program begins, there is nothing on the graphics window, but there is an invisible snake head at the center of the window. In each time step of the animation, the snake head moves 15 pixels in some direction, leaving behind a 15 × 15 filled square at the position it just left.

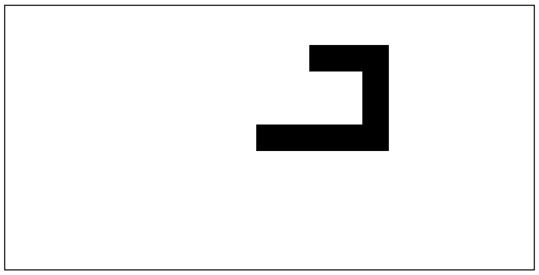
    At the beginning of the game, the snake head is moving eastward, so after four time steps, it will have generated a trail of four squares (which run together on the screen), like this.



In this implementation of the game, you turn the snake by clicking the mouse. In this initial situation, with the snake moving horizontally, clicking the mouse above the current y position of the snake head sends it northward, and clicking below the current y position would send it south. In this particular example, let's assume that the player clicked above the snake head, so that its direction changes to the north. After three more time steps, the window would look something like this:



When the snake is moving vertically, clicking to the left of the snake head sends it westward, and clicking to the right sends it eastward. Clicking to the left would therefore send the snake off to the west, as follows:



In the actual game, the player loses if the path of the snake moves outside the window or closses its own path. For this problem, you can ignore the problem of stopping and just get the motion working.

While the program may seem complicated, there are not that many pieces which you need to get it working. Here are a few general hints or things to keep in mind as you get started:

- You need to keep track of three things: the x and y positions of the snake head, and some indication of what direction the snake is currently moving (North, South, East or West).
- The function that executes each time step must create a new black square whose center is at the current position, and *then* move the head on to the next square by updating the coordinates as appropriate to the current direction.
- The listener method that responds to mouse clicks has to look at the current position and direction and then use those together with the mouse click location to determine how to update the direction.
- Remember that you don't need to check whether the snake remains on the window or crosses its own path. You can implement those features on your own time after the exam if you really want!