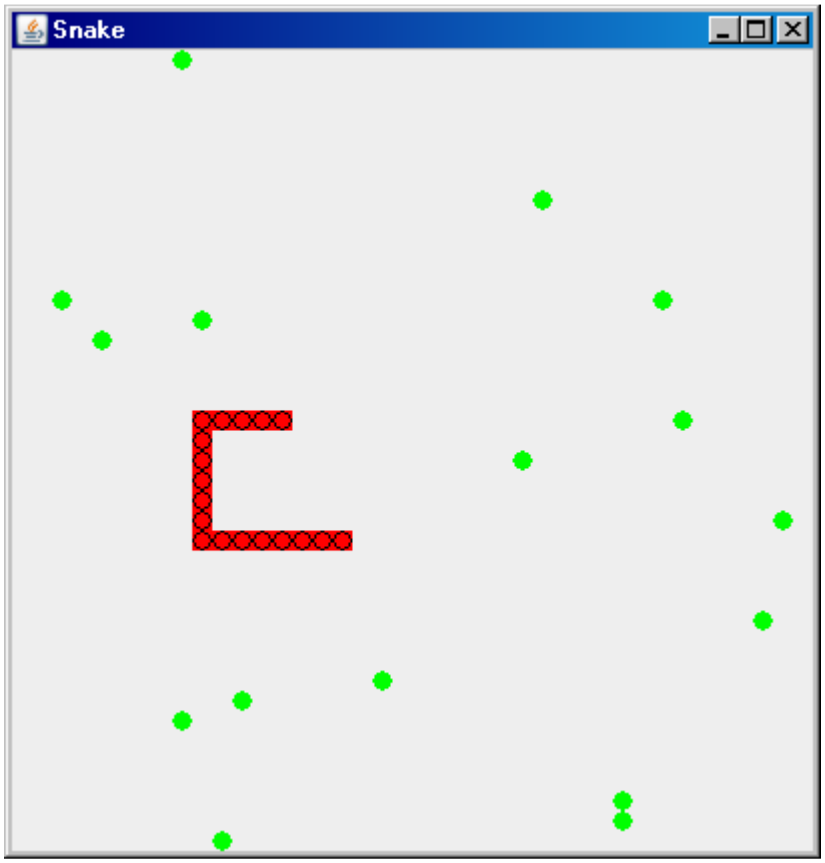
Assignment 8: Snake: ArrayList, KeyListener, Timer, Animation

Due Wednesday November 4, 2009 11:55PM

For this assignment, you'll be making the popular game Snake: http://www.snakegame.net/phonesnake.htm. Our version will have multiple apples to eat.
Please be sure to read all of the instructions before beginning.

Requirements

1. The game should have a snake that the user controls with the arrow keys (the snake keeps moving forward in the direction it's facing). The snake shouldn't be allow to back into itself (ie, if it's going left, it can't immediately turn around and go right).
2. If the snake runs into the walls of the game area or it runs into itself, the user loses.
3. Apples (as green circles in the picture below) should regularly appear randomly on the game area.
4. When the snake eats an apple, the snake should grow longer (and hence harder to keep from running into itself).
5. The game should use a Timer to drive the animation (the javax.swing.Timer one).



Possible Design

This is a possible design to use for your program (this is the one I used to make the screenshot above).

Snake Class:

1. An enum to represent the direction it's facing: up, down, left, or right.
2. The snake holds the following information:
    a. An ArrayList of Point objects to keep track of each segment of the snake. In this implementation, each point is at a multiple of the grid size to keep everything aligned.
    b. It's direction (enum from 1)
    c. The x and y location of the head of the snake
    d. Integer variables to hold the width, height, and grid size of the game area.
3. The constructor takes in its initial length, location, and the width and height of the game area (for collision detection), and grid size of the game area. The constructor creates the initial number of segments for the snake, and sets its initial direction.
4. It has the following methods:
    a. `draw`: Takes in a graphics object and draws all the segments of the snake (using the ArrayList of points and the grid size)
    b. `eat`: Increases the length of the snake (hint: just add another segment to the end that's at the same x, y location as the last segment)
    c. `move`: Move the snake one segment in the direction it's facing (hint: remove the last segment of the snake, and add a new one to the front)
    d. `checkCollision`: Checks to see if the snake has run into the walls of the game area, or if it has run into itself (ie, its first segment occupies the same space as another segment).
    e. `setDirection`: Sets the direction of the snake (hint: the snake shouldn't be allowed to instantaneously reverse its direction, i.e., if it's going left, it can't turn 180 and go right, and similarly, if it's going up, it can't immediately start going down. If it could, then it would immediately run into itself)
    f. `getLocation`: Returns the location of the snake

SnakePanel Class:

1. Holds:
    a. A Snake object
    b. An ArrayList of Point objects to keep track of where the apples are
    c. The Timer object to drive the animation
    d. A Random object for placing apples
    e. A variable to keep track of the number of timer ticks (as a delay for creating apples)
    f. A constant for the grid size to divide up the game area
2. The panel draws the snake and all the apples
3. It has an addApple method to add an apple at a random location (but still aligned to the grid!)
4. It implements KeyListener (or has a private inner class) for controlling the snake
5. It implements ActionListener (or has a private inner class) for the Timer
    a. This is where the animation is driven: Move the snake, check for collisions, check to see if it ate an apple, add an apple if enough timer ticks have gone by, and repaint the panel. If there was a collision, the game tells the use they've lost.

Turn-in Procedure

Turn in the following files on T-Square. Double-check that you have *submitted* and not just saved as draft (if the submission is successful, you will receive an e-mail from T-Square within a few minutes). Also, be sure all of your files compile and run.

- SnakeMain.java (the main entry point into your program)
- Any other files needed to run your submission

All .java files should have a brief descriptive javadoc comment.

Don't forget your collaboration statement. You should include a statement with every homework you submit, even if you worked alone.