CS 4053/5053
Homework 05 – Blobster
Due Tuesday 2020.04.07 at 11:00pm.

All homework assignments are individual efforts, and must be completed entirely on your own.

In this assignment you will combine interactive editing and 2D convex hulls to build a basic app for visualizing social networks. You will combine an incremental convex hull algorithm and multi-segment path generation to draw an envelope around the entire network of nodes. You will also support interactions to let the user interactively adjust the network's structure and appearance.

Start by reviewing the slides on polygons, circles, and arcs; parametric and Bezier curves; and 2D coordinate transformations. I also recommend that you review chapter 3.4–3.6 and 5.1–5.2 before starting on the code. Follow the order below, getting each part to work before moving on.

The Implementation Process

Go to the `edu.ou.cs.cg.assignment.homework05` package in `ou-cs-cg`. It's a reduced version of the `reflect` package, with a grid and an example object added in to help you learn affine transformations. You'll use a new class (`Network`) included in the `utilities` package. Complete the sections commented `TODO` in the `Model`, `View`, and `KeyHandler` classes.

You shouldn't need to modify the other two files in the `homework05` package. You may add Java files/subpackages to the package, but you shouldn't need to. Regardless, all new code must be yours. Structure your code clearly and document it thoroughly, particularly the code that implements the iterative convex hull algorithm. The corresponding program to run is `hw05`.

Managing and Drawing Nodes

Let the user create, select, and delete nodes that represent people in the social network. Get a list of all names using the `Network.getAllNames()` method, and use appropriate collections to manage names and nodes. (You'll also need to store how each node has been transformed.) Starting with zero nodes, support the following interactions and corresponding graphical effects:

- The `'<'` and `'>'` keys cycle through names *that don't yet appear as nodes*. Show the currently selected name as yellow text in the lower left corner of the view; edit `drawMode()` to do that. Don't forget to handle the case in which there are no unused names left!
- The `<return>` key adds a node for the currently selected name. Represent each visible node as a regular n-sided polygon. Call `Network.getSides()` to get the number of sides and `Network.getColor()` to get the fill color for each name. Initially center each node at the origin and set its initial radius to 0.05.
- The `','` and `'.'` keys cycle through the visible nodes. Allow only one node to be selected at a time. Draw the edge of the selected node in white. Don't draw the edges of the other nodes. When a node is added, set it to be the selected one.
- The `<delete>` and `<d>` keys remove the selected node (if any) and put it back on the list of available names to add. Upon removal, automatically select the "next" node, if there is one.
- The four arrow keys *translate* the XY coordinates of the currently selected node. See the `KeyHandler.keyPressed()` method in the `interaction` package for suggestive code.

- The `'['` and `']'` keys *scale* the radius of the selected node by x0.8 and x1.2.
- The `'{'` and `'}'` keys *rotate* the selected node around its center by -15º and +15º.
- Clicking inside a polygon selects its node. (If there are multiple hits, select any one of them.)

Calculating the Convex Hull

Calculate the convex hull around the centers of all visible nodes. For a set of 2D points, the hull is the smallest convex polygon that contains them all of them. Use the following algorithm. It's not optimal, but is easy to implement, and works well when points move a lot. To insert a point:

- If the hull is empty, it becomes just the insert point.
- If the hull is one point, it becomes the line segment connecting it to the insert point. (Unless the points are the same, in which case the new point is "on" the hull and you're done.)
- If the hull is a line segment, it becomes the triangle connecting the points counterclockwise. (Unless the three points are co-linear, in which case the hull becomes the minimal line segment containing all three.)
- If the hull is a polygon (three or more sides): (1) Starting with any point on the polygon, loop around the edges of the existing hull clockwise, until you find one that the insert point is strictly "to the left" of. (2) From there, loop around the edges counterclockwise until you find one that the insert point is strictly "to the right" of. If there isn't such an edge, the point is on or in the hull and you're done. (3) Loop counterclockwise starting from the *next* edge, removing the start point of each edge from the hull until the insert point is strictly "to the left" of an edge. Add the insert point to the polygon just before the start point of that edge.

To remove a point (brute force, but easy and effective):

- If it's not one of the hull points, you're done.
- If it is one of the hull points: (1) Remove it from the hull. (2) Loop through *all the other non-hull points*, trying to insert each one following the point insertion steps above.

Use node centers as the points for calculating the hull. Use a deque to store the hull points. Use any reasonable collection to store all of the inserted points that are not on the hull at any given time. Push and pop the ends of the deque to loop around the hull in each direction. Update the hull when adding, deleting, or moving nodes. *To move a point, remove it then (re)insert it.*

Drawing the Convex Hull

Draw the convex hull using the balloon effect to visually contain the nodes in the social network:

- Add code to draw the hull as a simple polygon of its points. Fill and edge it with colors of your choice. Use the edge color to draw the hull when it is only a line segment or a point.
- Add a parameter to the model: hull radius. Set its initial value to 0.0.
- Add code to "balloon" the hull. To do that, move each edge of the hull polygon outward by the hull radius, in the direction of the clockwise perp vector. Connect each pair of edges with an arc to form another, longer polygon. *As a fallback, a simple but less appealing option is to connect the edge ends directly with line segments.* Handle the special cases of non-polygon hulls appropriately (a circle when there is one hull point, a rounded tube when there are two hull points). When the hull radius is zero, it should look like a non-ballooned hull.

- Add code to make the `'q'` and `'w'` keys increase and decrease the hull radius by -0.1 and +0.1, respectively. Restrict the hull radius to non-negative values.

Turning It In

Turn in a complete, cleaned, renamed, zipped **COPY** of your `homework05` directory:

- Take a screenshot of your application window when it's in an interesting graphical state.
- Put the screenshot in the `Results` directory as `snapshot.png` or `snapshot.jpg`.
- Go into the `ou-cs-cg` directory.
  - Make sure it contains all of the code modifications and additions that you wish to submit.
  - Run `gradlew clean` to reduce the size of your build.
  - If you're using Eclipse, run `gradlew cleanEclipse` and delete the `bin` directory.
- Append your 4x4 to the `homework05` directory; mine would be `homework05-weav8417`.
- Zip your entire renamed `homework05` directory.
- Submit your zip file to the Homework05 assignment in Canvas.

You will be scored on: (1) how completely and correctly you realize the graphical appearance and dynamical behavior of the listed implementation parts; (2) how completely and correctly you support the specified interactive adjustments; (3) the clarity and appropriateness of your code; and (4) the clarity and completeness of your comments, especially for the convex hull algorithm.