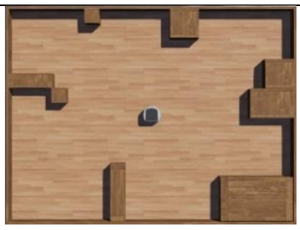
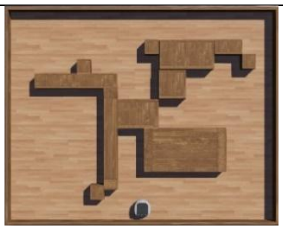
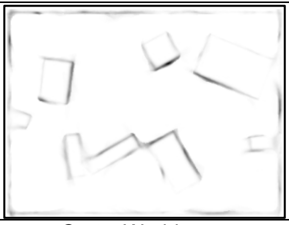
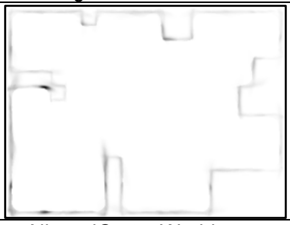
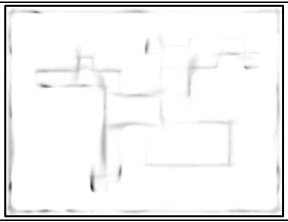
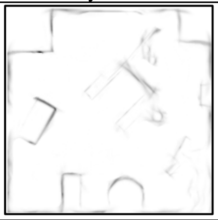
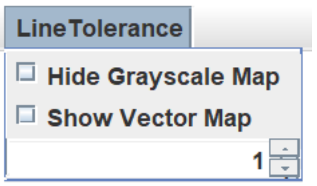
LAB 14 – Converting to Vector Maps

(1) The goal of this lab is to apply the incremental line-fitting algorithm to our border-traced grid maps to form vector maps. Again, in this lab, we will not use the Webots environment. Instead, you will compile the **MapperApp** program separately, using any Java IDE that you like. However, you must not arrange your code in packages (some IDEs do this by default). Download the **Lab14_ConvertingToVectorMaps.zip** file and unzip it. We will be using pre-computed maps as shown here:

			
CratesWorld	AlignedCratesWorld	CombinedCratesWorld	StillMessyRoomWorld
			
CratesWorld.map	AlignedCratesWorld.map	CombinedCratesWorld.map	StillMessyRoom.map

The **.map** files have been provided for you in the same directory as the source code. The main application that you will run is the **MapperApp** which has an additional **LineTolerance** menu added to it. This new menu has a **Hide Grayscale Map** option (to hide the occupancy grid) as well as a **Show Vector Map** option. The spinner box lets you set the line tolerance for the line-fitting algorithm. A value of **0** indicates to include every point on the obstacle boundary as a vertex of the obstacle. Increasing it will reduce the number of vertices that make up each obstacle.



Of course, the **LineTolerance** menu functions do not work yet. You have some work to do to make that happen. You will be writing ALL your code for this lab within the **Map.java** and **VectorMap.java** classes (which both make use of **Obstacle.java**).

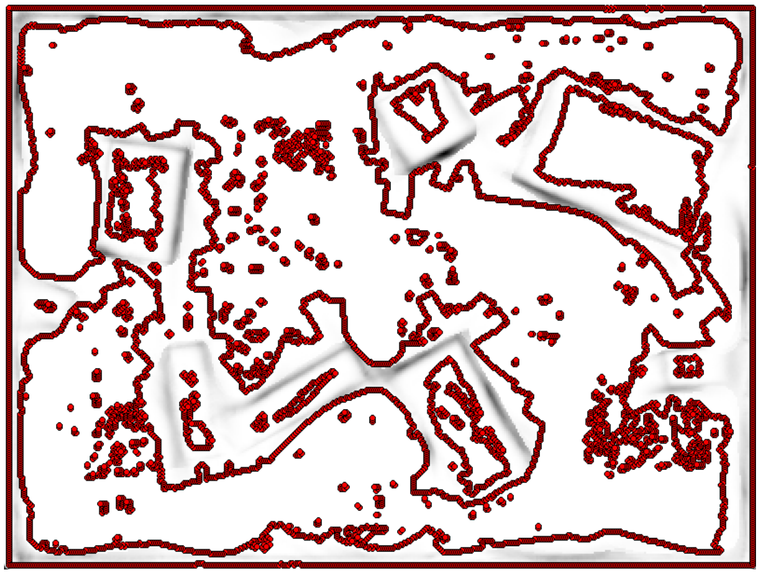
The **Obstacle** class is used to represent a polygonal obstacle in the environment. An obstacle is just an **ArrayList<Point>** of vertices. There are useful functions such as **addVertex(x, y)**, **getVertices()**, **getVertex(i)** and **size()** which you will make use of.

The **VectorMap** is essentially just an **ArrayList** of **Obstacle** objects. There are functions you will use such as **addObstacle(obstacle)** and **clear()** to empty out the list of obstacles. The **Map** class has been adjusted to include a **vectorMap** attribute … which will contain the **VectorMap** that you will create.

(2) The code for computing the obstacle borders (from the previous lab) has been provided. If you recall, we traced the boundary of the obstacles in counter-clockwise order. You will begin by creating a polygon **Obstacle** for every complete border that has been traced. There will be one polygon vertex for each grid cell on the border of the traced obstacle (see first image in slide 4 of the notes). The vertices will be joined together in counter-clockwise order to make a polygon with a LOT of vertices! In the next part of the lab, you will be reducing the number of vertices.

The **computeBorders()** method in the **Map** class currently finds the start of each border in the map and traces it out (solution from the last lab). When it finds a new border point that begins the tracing of a new map obstacle, you will need to create a new **Obstacle** and add that border point to it. To add the border point, just add `(x, y)` as the first vertex of the obstacle object. Once the border is traced, you should then add that newly-created **Obstacle** object to the **vectorMap** if and only if <u>the obstacle has at least 3 vertices in it</u>.

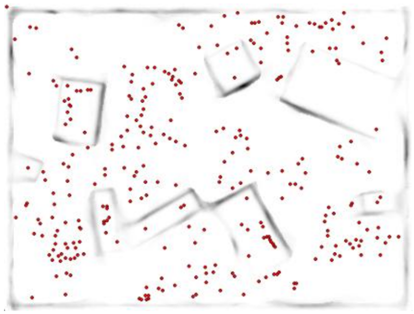
You will also need to make changes to the **traceWholeBorder()** method so that it adds all the border points to the obstacle that is currently being traced. You will have to add the **Obstacle** object as a parameter to the method and then when a new border point is found (i.e., (**xc**, **yc**)), you should add that point as a vertex to the obstacle.



Once you get it all working, run your code, load up the **CratesWorld.map** grid map. You should see a vector map that looks very similar to what you see here on the right. Take a screen snapshot and save it as **Snapshot1.png**. You can play around with the spinner in the **BinaryThreshold** menu. As you increase the threshold, the number of vertices will be reduced.

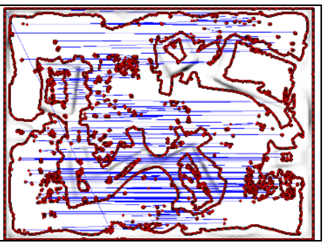
Debugging Tips:

- If your code runs, but does not display any vertices, first check to make sure that the **Show Vector Map** checkbox has been checked on in the **LineTolerance** menu. If you still do not see any vertices, check to make sure that you are actually adding your newly-created obstacle to the **vectorMap** after you complete the **traceWholeBorder()** call.
- If that doesn't fix things … you need to check the two places in your code that you are supposed to be adding the vertices. If you want to know if your starting vertex is ok, add the newly-created obstacle to the **vectorMap** even if it has less than three vertices. You should see something like what is shown here on the right, if you added the starting vertex properly. If you do not see this, then you forgot to add the starting vertex.



- If you only see those starting vertices, then you forgot to add all the other vertices from within the **traceWholeBorder()** method each time you find a new border point.

- If your code seems to hang for a bit and then you see many blue horizontal lines as shown here, then you are likely not creating a new Obstacle each time you are starting a new trace. You probably created one new Obstacle outside the FOR loops … which means that there is only one obstacle altogether.



(3) Now you need to reduce the number of vertices in the map. At the end of the **computeBorders()** method, you will notice that the **applyLineTolerance()** method in the **VectorMap** class is being called. You must complete the method so that it applies the incremental line-fitting algorithm (slides 6 to 8 in the notes) to each of your obstacles. This method should go through all the obstacles in the map (that you created from part 2 above) and reduce the number of vertices according to the **LINE_TOLERANCE** variable, which is set by the spinner box in the **LineTolerance** menu.

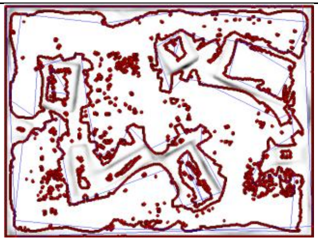
The method begins by creating a new **ArrayList** of obstacles. You need to go through all the obstacles and apply the incremental line-fitting algorithm by traversing all vertices of each obstacle. For each obstacle that is being line-fitted, you need to create another new obstacle that will represent the *reduced* version (i.e., less vertices due to line-fitting) of it. If the *reduced* obstacle ends up with 3 or more vertices it should be added to the **newObstacle** ArrayList of obstacles, otherwise it should not be added.

At the end of the method, the vector map's **obstacles** attribute is set to this new ArrayList of reduced obstacles and so you should only see the reduced obstacles on the screen from now on.

Run your code again to make sure it works. Adjust the **Line Tolerance** spinner in the menu and observe the effect.

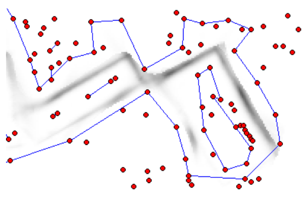
Debugging Tips:

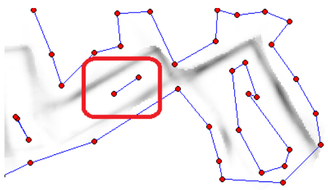
- If you run your code and adjust the line tolerance, but it continues to show the same vertices, then perhaps you forgot to remove `newObstacles.addAll(obstacles);` at the top of the method.



- If you do not see any vertices or edges, then perhaps you did not add your *reduced* obstacle to the **newObstacles** array.

- You might find that you are getting some polygons with less than three vertices. (You can check for this error by setting the **LineTolerance** menu value to **6**).
- If you see a lot of single dots (see top picture here), then you likely forgot to check to ensure that there were **3** vertices before you added the obstacle.
- If you see a line segment to the southwest of the map's center (see bottom picture here), then you likely do not have a proper IF statement condition for line 7 of the pseudocode to not add duplicate vertices.





Save the following snapshots for the 4 worlds according to the settings defined in the middle column.

CratesWorld	Show Grayscale Map Binary Threshold = **0** Line Tolerance = **1**	**Snapshot2.png**
	Show Grayscale Map Binary Threshold = **1** Line Tolerance = 1	**Snapshot3.png**
	Show Grayscale Map Binary Threshold = **1** Line Tolerance = **3**	**Snapshot4.png**
AlignedCratesWorld	Show Grayscale Map Binary Threshold = **1** Line Tolerance = **3**	**Snapshot5.png**
	Show Grayscale Map Binary Threshold = **4** Line Tolerance = **4**	**Snapshot6.png**
CombinedCratesWorld	**Hide** Grayscale Map Binary Threshold = **1** Line Tolerance = **2**	**Snapshot7.png**
	Hide Grayscale Map Binary Threshold = **3** Line Tolerance = **4**	**Snapshot8.png**
StillMessyRoomWorld	**Hide** Grayscale Map Binary Threshold = **0** Line Tolerance = **5**	**Snapshot9.png**
	Hide Grayscale Map Binary Threshold = **1** Line Tolerance = **1**	**Snapshot10.png**

Submit your **java** code. It is important that there are no package statements at the top of your source files (some IDEs require you to put everything into a project or package). Just submit the source files without the package lines at the top (if any from a different IDE). If you are using IntelliJ and are unsure how to do this, please see step **(9)** of the "Using the IntelliJ IDE" file provided.

Submit the **.map files** and the **10** snapshot **.png** files.

Make sure that your name and student number is in the first comment line of your code.