P2: Curious Hungry Robot **Due 11 / 3 / 2018** Barnes Comp 182/L Fall 2018

Using the ADTs in the table below write a simulation experiment for a robot that is looking for energy in a virtual environment. While you must have these ADTs you can also have others. Conceptually the environment is an infinite plane with horizontal dimensions in X and vertical dimensions in Y. Use the class Point to represent locations on the plane. The robot is initially placed at the origin, location (0, 0), on the plane and is curious. The robot becomes inactive when it runs out of energy. There are energy locations (15) randomly placed on the plane within a planeEnergyRadius radius (200) from the origin and they are at least energyIntervalDistance units (20) apart. Each energy source initially has an energyInitialCapacity (200) of energy units. A simulation trial ends when the robot becomes inactive. A simulation trial adds the distance the robot travelled to the Sample object used to gather simulation statistics. The simulation has 1000 trials of each experimental memory condition. Use your Sample class to gather and report statistics.

Type	**Name**	**Source**
interface	Deque	java.util
class	ArrayDeque<Energy>	java.util.ArrayDeque<E>
class	Point	java.awt.Point
class	Robot	you
class	Energy	you
class	CuriousHungryRobot	you
class	Sample	you (from P1)

Robot States. The robot has two active states: curious and hungry. The robot starts in the curious state and becomes inactive when it no longer has any energy. The robot has an energy store with a fixed capacity of energyInitialCapacity (200 energy units). The robot is initialized to full capacity. The robot is curious when it has more than half its energy units (not hungry, not inactive). The robot is hungry when it has half or less energy units and is not inactive. The robot's three mutually exclusive states are:

Curious: robot energy > half its energy capacity. While curious the robot explores randomly (moves towards its curiousGoal) and can detects and remember energy sources. While curious the robot does not consume any energy, it only uses its energy.

Hungry: robot energy <= half its energy capacity. While hungry the robot attempts to use its memory to move towards a remembered energy location (memoryGoal). While the robot moves towards its memoryGoal it can detect and remember new energy stores, but it doesn't change its memoryGoal. If the robot does not have a memoryGoal it switches back to its curious state and continues towards its curiousGoal for that move. Note it will be become hungry again on its next move.

Inactive: robot energy <= 0 its energy capacity. The current simulation trial is over when the robot becomes inactive.

Robot Memory type. The robot has memory for energy sources that it has detected during its movement. The robot's memory stores the Energy instance it detects. The robot's memory is

maintained by a JCF ArrayDeque<Energy> that operates in one of two different "experiment conditions", or, types of memories: a stack or a queue. The robot's memory should be a Deque interface that filters an ArrayDeque<Energy> instance. When the robot becomes hungry it attempts to retrieve information about energy sources from its memory. It attempts retrieval in the order determined by its memory structure (stack / LIFO or queue / FIFO). If the robot can retrieve an energy from memory, that energy becomes its "memory goal". The "memory goal" does not change until the robot reaches its "memory goal". The robot also knows how far it has travelled and how much energy it has. Note: use the interface Deque for both your stack / LIFO and queue / FIFO robot memory types. For example, with the stack / LIFO (last-in, first-out) memory structure you would use Deque's stack equivalent methods: addFirst() and removeFirst(). With the queue / FIFO (first-in, first-out) memory structure you would use Deque's queue equivalent methods: addLast() and removeFirst().

Robot Moves. The robot always moves to a location that is -13, 0, or 13 in the horizontal (X) and vertical (Y) dimension. For example, if the robot is at location (X = 65, Y = 39) it has 9 possible moves:

(52 ,26)	(65, 26)	(78, 26)
(52, 39)	(65, 39)	(78, 39)
(52, 52)	(65, 52)	(78, 52)

The robot's uses the distance of its move as the amount of energy to make the move. The longest diagonal move the robot can make is approximately 18.38 (use the actual double distance value) and the shortest move has a distance of 13. Distances should be doubles. After each move the robot's energy level is decremented by the distance it just moved. A simulation trial continues until the robot becomes inactive.

When curious, the robot moves from its current location towards its curiousGoal. When a robot becomes curious (or initially) it randomly selects its curiousGoal; a location within and including the points within a 200 radius from the origin. The curious robot's next move is towards its curiousGoal. While curious the robot will make the move closest to its curiousGoal until it reaches that goal, becomes hungry, or becomes inactive. If the robot is within its snapDistance (9 units = ceiling of 2/3 its moveDistance) from its goal location it "snaps" to that goal location. If it reaches its curiousGoal and is still curious the robot selects a new curiousGoal. In this way the robot randomly explores the space.

When the robot is hungry and it has a "memoryGoal" it moves towards that location. The robot still moves ± moveDistance in X and Y from its current location. If the robot's move ends at an energy location the robot consumes energy at the energy location, even if its energy is negative. While the robot is hungry and moving towards its "memory goal" it can detect and remember energy locations. This can change the robot's memory contents, but not its "memoryGoal".

Consuming energy. When hungry and at an energy source the robot (1) forgets its "memory goal" and (2) consumes energy until it is either full or there is no more energy left at the source. Energy sources that are empty are not detected on future moves and are forgotten (removed) from the robot's memory: they are no longer detectable. Of course, the robot can still be hungry after consuming all the source's energy. If the robot is still hungry it attempts to retrieve a new "memory goal" from its

memory. If the robot has no energy after consuming all the energy at the source than it becomes inactive (simulation run over).

Energy Source Detection. After each move, the robot detects and can learn all energy sources that are within its robotDetectionRadius (I may change this value). The dectectionRadius is the moveDistance (13) unit radius of its current location. A robot only learns (stores) one memory for each energy location. When the robot learns (detects) an energy that it has in its memory, the known ("in-memory") energy item is removed from the store and the "just detected" energy item is inserted. (This "updates" the position of the detected energy.) Again, detection does not change the hungry robot's "memory goal".

Use the following "named" or "symbolic" constants in your simulation. I reserve the option to change these values in updates to this assignment.

Constant	value
simulationTrials	1000
planeEnergyRadius	200
moveDistance in X and Y	± 13
snapDistance	9
robotEnergyCapacity	200
robotRobotDetectionRadius	13
energyLocations	15
energyIntervalDistance	20
energyInitialCapacity	200

ImageWindow Framework. You can use the ImageWindow package for testing/debugging of your solution. This is not required, it is optional. If you modify the ImageWindow package you must submit your version with your assignment and you must document your modifications.

Simulation Experiment. Before you begin designing your solution to this assignment write down what experimental condition you think will allow the robot to move the largest distance and why. In your report provide your actual results in a table. In addition write a brief description explaining your results. Which condition did you initially believe would perform better and why did you think that. Why did you get the results you obtained? What is the best overall memory structure for robots?

Design before coding. A good design will facilitate the implementation of a correct solution for this assignment. I STRONGLY recommend that you spend time thinking about and testing your design. I believe you will find that UML helps. You could "play" the simulation on paper before trying to program. You could have a set of limited "test versions" that verify single events: becoming inactive, detecting an energy, moving to an energy. Try solving a simpler versions of the assignment and then combining the simpler solutions that work together.

Submission. You must submit all source files (*.java), a UML class diagram of your solution, and a brief report of your results. You can submit any other documentation or design material you think is

necessary to describe your submission for grading. All submission must be in a single zipped archive. Submit your assignment using Canvas.

ImageWindowDemo example image. This images is not what your program's robot movement or energy placement should look like. It just shows what ImageWindow can do. This is the 4th image generated. It shows markers colored green (> half energy) or black (< half energy) with energy values and a spiral path of the bot.