Lab 3 Critters README

By Niranjan Telkikar EID: nnt479

The data structure used to hold the Critters was a simple population list provided.

I created two new classes which were the Critter1 and Critter2 classes.

Classes

Critter1:
The Critter1 class contains a critter who reproduces 4 times in its doTimeStep method and walks right. In the fight method it returned true. For the toString method it returns 1.

Critter2:
The Critter2 class contains a critter which moves right during the fight and wants to fight (returns true). It does nothing in its timestep. The toString representation is 2.

Critter Class:

Methods inside Critter Class:
walk(): This method makes the critter walk one step in the specified direction
run(): This method makes the critter move 2 steps which is run.
move(): * This method first checks if the energy is still>energyCost and if it is it continues and subtracts the energy cost.

* After that based on the direction specified, the x-coordinate is updated. For example, if the direction is specified as 0, which means to move right dx[0] is dx is 1 and y is 0 as y is

* not supposed to change. Note that it is multiplied by the number of steps and added to x-coord and divided by Params.world_width to account for the torus grid.
Reproduce: * This method reproduces a critter and creates its babies! First, if this.energy is not greater than or equal to the minimum reproduce energy, then the method returns.

* Otherwise, the energy of the offspring is cut in half and the energy of the parent is halved (but rounded up).

* The offspring's coordinates are set in a similar way to the move method. The ycoord is added to the direction dx and world_width. After that the remainder of that sum and Params.world_width is used for the coordinates

* Lastly, the offspring are added to the list of babies.

makeCritter: * Firstly, the class of the critter is extracted using the forName method by combining myPackage with the critter_class_name.
* A new instance of Critter is created.
* A random x coord is assigned to critter
* A random y coord is assigned to critter
* The critter's energy is set as Params.start_energy
* The critter is added to the population of Critters.
* If the critter class is not found, InvalidCritterException is invoked.

getInstances: * 1. First the critterClass is extracted from myPackage+"."+critter_class_name"
* 2. In the population of critters, if it is part of critterClass it is added to the results list (instances of that specific Critter class)
* 3. The list is returned
* 4. An exception is returned if the class does not exist.

runStats: * Prints out how many Critters of each type there are on the board.

clearWorld: * Clear the world of all critters, dead and alive
* The list of population and babies are cleared using the clear method

worldTimeStep: The worldTimeStep does all of the necessary things as per the document.
* 1. First for the population of critters, each individual critter does their own timestep
* 2. Then encounters are resolved, if there are two critters at the same position.
* 3. First the fight method is invoked for both critters to see which one wants to fight.
* 4. If true is returned, the Critter wants to fight
* 5. If the critters wants to fight a random number is rolled for a critter called rolla.
* 6. The same thing is done for the other critter.
* 7. If rolla>rollb then 1/2 of critter2's energy is given to critter1 and critter1's energy is set to zero
* 8. If rollb>rolla then 1/2 of critter1's energy is given to critter1 and critter2's energy is set to zero
* 9. If rolla=rollb the winner is selected randomly.
* 10. Params.restenergycost is subtracted from the energy of the critters
* 11. New algae are created, with random x and y coords given, and energy set to Params.start_energy. The critters are then added to the population list
* 12. All of the babies are added to the population
* 13. The list of babies is cleared
* 14. The dead critters are removed from the population.

displayWorld: * A two-dimension grid array is created to display the world with the Params.world_height and Params.world_width
* For each row in the grid, teh array is filled with space
* Then, the grid is filled with the critters using the toString method (so characters) representing one critter with many different types
* The top boundary is printed.
* The minus sides are printed at the side
* The bottom boundary is printed.
*/

Main Class:

The main class reads the command which is split based on whitespace. Based on the commands, either the setSeed, makeCritter, or runStats method is invoked.

This is what the main class invokes:
quit – [STAGE 1] terminates the program
• show – [STAGE1] invoke the Critter.displayWorld() method
• step [<count>] – [STAGE1] The <count> is optional (count is [STAGE2]). If <count> is included, then <count> will be an integer. There are no square brackets in this command, this notation is used simply to indicate that the <count> is optional. For example, “step 10000” is a legal command, as is “step”. In response to this command, the program must perform the specified number of world time steps. If no count is provided, then only one world time step is performed.
10/1/2024 10
• seed <number> -- [STAGE2] invoke the Critter.setSeed method using the number provided as the new random number seed. This method is provided so that you can force your simulation to repeat the same sequence of random numbers during testing.
• make <class_name> [<count>] – [STAGE3, for stages 1 and 2, edit your main function so that 100 Algae and 25 Craig critters are always placed into the world when it starts, for STAGE3, the world should start empty] as before, the <count> argument is optional. The command “make” must be provided verbatim. The <class_name> argument will be a string and must be the name of a concrete subclass of Critter. When this command is executed, the controller will invoke the Critter.makeCritter static method. The <class_name> string will be provided as an argument to makeCritter. If no count is provided, then makeCritter will be called exactly once. If a count is provided, then makeCritter will be called inside a loop the specified number

of times. For
example “make Craig 25” will cause
Critter.makeCritter(“Craig”); to be invoked 25 times