

## APCS :: Lab 23 – Grid Implementations

### Briefly:

- Write TWO complete implementations of a Grid.
- BOTH (for now) will be unbounded Grids.
- Your implementations should extend AbstractGrid.
- Write one Runner Class
- To use your new Grid you'll have to add it to the list of possible grid choices in the menu in the GUI. Page 44 in the GridWorld manual shows how to add a new Grid implementation the menu of the GUI.

### Assignment:

0. If you want some more background on the Grid you can read section 5 of the Gridworld manual (it's actually pretty short).
1. As a warm-up exercise: write one unbounded Grid using a TreeMap - this should be a very easy exercise since a HashMap implementation already exists - it is provided with the GridWorld code. In fact, you could just copy-paste the code and switch the initialization of the map to a new TreeMap<...>() instead of a HashMap(). Of course, this wouldn't help you to learn about the Grid at all. Try making it from scratch before referring to the other code.
2. For the other implementation choose one or a combination of: LinkedList and/or ArrayList. Either should contain a list of all the actors in the Grid. You should also feel free to modify the way things are added/removed/looked up in your lists. For example, you may want to keep the ArrayList of Actors in order by location. This would allow for BinarySearch lookups on the ArrayList.
3. You should be able to VERY easily implement bounded versions of your two implementations without having to write very much code. Think about what the real difference is between a bounded and unbounded Grid that uses, say, a TreeMap. Think about it. Do it. DO NOT COPY/PASTE any of your code.
4. You should write ONE Runner class that adds all of your Grid implementations to the World (see p. 44 of the GridWorld manual) and fills the grid with A LOT of flowers, rocks and bees. You should then test the various grid implementations to see how they perform with a lot of actors. We're going for a stress test here, so if a simulation is running quickly, add more actors until it starts to grind. In other words, find the limits of the implementations.
5. Be prepared to answer a lot of Big-Oh, and design questions about various ways to implement the grid. For example: If the grid has a lot of actors but not a lot of critters, how might this affect your grid implementation choice, and how might it affect the

running time? What about the reverse situation: a lot of critters?

## **6. EXTRA CREDIT**

6.1 Write a Critter called Hummingbird that eats Bees. A humming bird can only move “forward” to one of the 5 locations in front, left, or right of it. If one of those 5 locations contains a Bee it will move there and "eat" the Bee. Otherwise it should pick a new location randomly. If a humming bird acts 7 times without eating a Bee it should **die**. (I've chosen 7 arbitrarily. Choose your own number to get more satisfying results).

6.2 Modify your Bee to avoid Hummingbirds. If a Bee sees a Hummingbird in a neighboring location, it should move to the location in the opposite direction of the hummingbird if possible - otherwise it should choose a different location to avoid the hummingbird. If there is not a Hummingbird in a neighboring location, the Bee should visit flowers as before. Hummingbird avoidance should take precedence over flower pollination.