# CS 2112 Fall 2021
## Assignment 6
### Graphical User Interface Design

Due: Tuesday, December 7, 11:59PM
Design document due: November 22, 11:59 PM

In this assignment you will use the JavaFX API to build a graphical visualization of the critter world described in the Project Specification. The visualization will have a graphical user interface (GUI) that will display the positions of critters, rocks, and food, permit the user to load and inspect critters, start and stop the simulation, adjust the rate, or advance the world one step at a time.

The majority of the work for this assignment will be on new functionality. However, you will also be expected to fix bugs in your code for Assignments 4 and 5 as necessary, and since this is the final submission of your project, it will be weighted more heavily than the previous assignments.

## 1 Changes

- No changes so far!

## 2 Instructions

### 2.1 Grading

As usual, solutions will be graded on design, correctness, and style. A good design makes the implementation easy to understand and maximizes code sharing while maintaining modularity. Your program should compile without errors or warnings and behave according the requirements given here. Your code should be clear, concise, and easy to read.

We will evaluate your user interface on visual appearance, layout, and design of the controls. We are looking for an attractive and functional interface that offers an enjoyable experience for the user. We have not specified precisely what this means, as we would like you to think it through and come up with your own design.

A good idea is to storyboard your design, and also to experiment with different layouts to see what works best. Do not get locked into design decisions too early in the process.

### 2.2 Final project

This assignment is the third and final part of the three-part final project for the course. Consult the Project Specification to find out more about the overall structure of the project.

### 2.3 Partners

You will work in groups of two or three for this assignment. This should be the same group as in Assignment 5. Remember that the course staff are available to help with problems you run into. For help, read all Ed posts and ask questions that have not been addressed, attend office hours, or set up meetings with any course staff member.

### 2.4 Restrictions

For the first time, you will be starting from scratch. We are not releasing any starter code for this assignment, other than the Gradle file used to allow your code to be properly compatible with JavaFX.

You may use any classes from the standard Java system library. If there is a third-party library you would like to use, please post to Ed to receive a confirmation before using it. You may code the GUI in JavaFX's XML, use a GUI builder such as the JavaFX Scene Builder (older version from Oracle, or a newer open-source version from Gluon). You may also hand-code your GUI. There are no restrictions on design tools.

## 3 Design overview document

We require that you submit an early draft of your design overview document before the assignment due date. The Overview Document Specification outlines our expectations. Your design and testing strategy might not be complete at that point, but we would like to see your progress. This is also a good time to submit design sketches for the GUI. You can go over your design document with the course staff and receive feedback in lab.

## 4 Version control

As in the last assignment, you must submit a file `log.txt` containing the commit history of your group.

## 5 Requirements

Your program should be able to display all aspects of the current state of the world. It should graphically render hexes and their contents, including food, rocks, and critters. It should be possible to distinguish critters of different species and to see the size and direction of each critter.

The GUI should allow the selection of world files, preferably using a FileChooser. After the user has selected a world file, the program should load the file and initialize the world as done in Assignment 5. The critters will be controlled by critter programs using the interpreter and simulation engine you built in Assignment 5.

The GUI should allow the user to step the simulation one step at a time or let the world run continuously. It should be possible to pause and resume a continuously running simulation. The graphical display will be updated continuously as the simulation progresses to reflect the current state of the world.

The total number of time steps taken during the simulation and the total number of critters alive in the world should be displayed. As in Assignment 5, the user should be able to create a new random world, load a world, or load a specified number of critters.

When loading a critter program file, the user should be able to either specify a number of critters to be randomly placed throughout the world or select a particular hex to place a critter.

The user should be able to set the maximum rate at which the simulation advances (including 0). Regardless of how quickly the simulation progresses, the graphical display should not be updated more often than 30 times per second. If the simulation is progressing faster than this, some intermediate world states should not be displayed.

Another part of the user interface will allow the user to inspect a single critter somewhere in the world. The user can click on the hex containing a critter to make it the currently displayed critter. The user interface will indicate which critter is currently displayed and will also display the state of the selected critter, corresponding to the 7 initial memory locations, along with the critter program and information about the most recently executed rule. As the simulation progresses, this information will be updated accordingly.

The particulars of the design are up to you. You should strive for an interface that is intuitive, user-friendly, and visually appealing.

### 5.1 A5 Compatibility

Do not delete any of the interfaces that were implemented in A5; otherwise, we will not be able to grade your solution effectively. The code to launch your simulation in the terminal should still be there.

### 5.2 Improved food sensing

In A5, you implemented a simple version of smell that doesn't take into account obstacles or turning. Critters are now given a sense of smell so that they can find food more easily. For A6, the sensor expression `smell` should take into account conditions ignored in A5: obstacles and the need for the critter to turn, including any initial turn(s) to start heading in the right direction. For instance, Figure 1 illustrates an environment in which the critter is heading northeast. Without obstacles, the closest food would be at distance 3 to the northwest (direction 4 relative to the critter's current orientation). Because of the rock wall, however, at least 18 turns and moves are required to reach the food. Food C is closer at 6 turns and moves with relative direction 0. This route is faster than an alternative route of length 7 with relative direction 5. Meanwhile, no obstacles stand in the way of Food F at distance 4 with relative direction 0. Figure 2 shows the distance from the critter

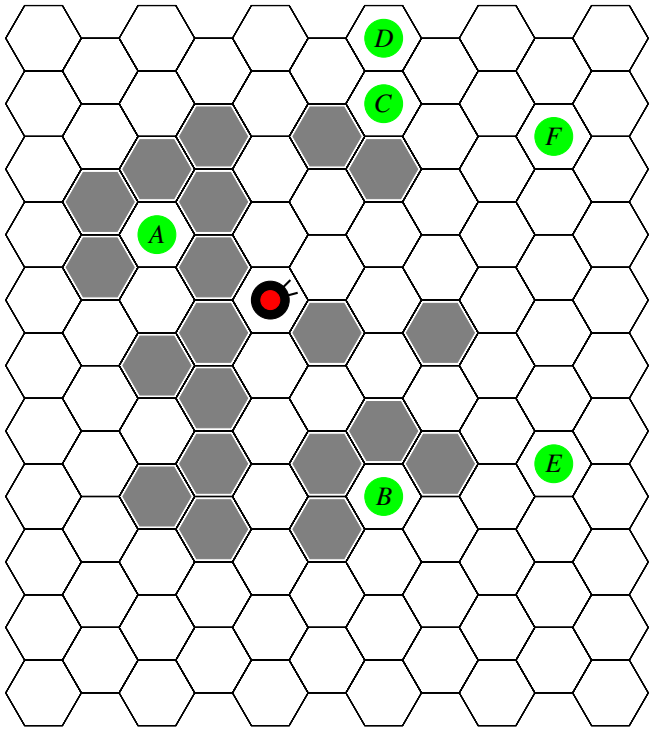


**Figure 1:** Finding food in a challenging environment

to various hexes, taking obstacles into account.

The result of the `smell` expression is based on two parameters: *distance* and *direction*. The distance is the smallest number of forward moves and turns to a hex adjacent to the food with the critter facing toward the food, provided it is no more than `MAX_SMELL_DIST` = 10. This is the smallest number of forward moves and turns for the critter to be in a position to eat food. The direction is relative to the critter's current orientation and is toward a hex that decreases the minimum number of turns or moves to food.

In Figure 1, Foods C and D are at distance 6. If Food F did not exist, either of these could be chosen. To reduce the distance to these food items, the critter could either turn left and move north to get closer to Food D, or move northeast to get closer to Food C. The corresponding relative directions are 5 and 0.

Given distance and direction as defined, the result of the `smell` expression is:

$$\mathit{distance} * 1,000 + \mathit{direction}$$

Since Food F is the closest in the example, the sensor should have value 4000. If Food F were not present, then the directions to either Food C or Food D could be returned, which are 6000 and 6005 respectively. If there is no food within a 10-hex walk, `smell` evaluates to 1,000,000.

Describe your implementation approach in the overview document.

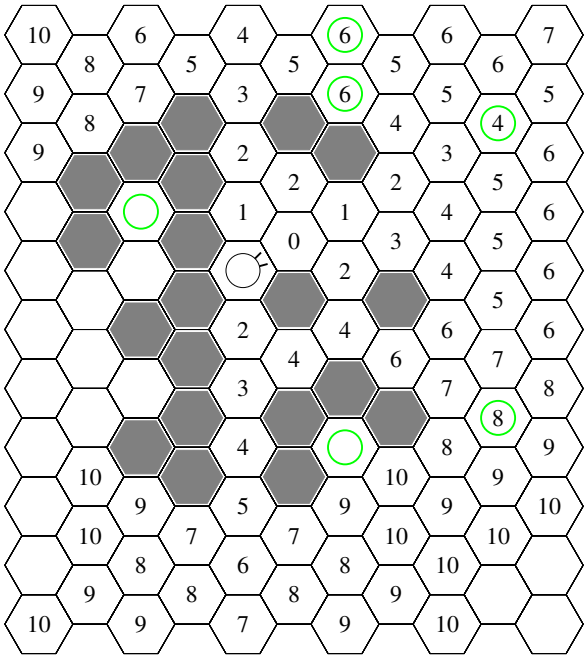

**Figure 2:** The least number of turns and moves from the critter to various hexes

## 6 Running your program

It should be possible to run your program with the following command:

```
java -jar <your jar>
```

Your program should start up with a default world populated by randomly placed rocks, initialize the GUI, and wait for user input. The simulation should not be running initially.

Note that there are no command line options. All further user interaction should be done through the graphical user interface.

## 7 GUI Design Choices

Good GUI design can be difficult. It is largely subjective, and there are no hard and fast rules for what makes a good design. That said, there are a few simple strategies you can follow that will enhance the experience for your users.

### 7.1 Buttons and Control

Users should not need to play a guessing game to find out what buttons do. They should be clearly and succinctly labeled to describe their function. In some cases, an icon can be better than words.

Placement of buttons depends on function and frequency of use. A small button way off on the side of the screen is difficult to access compared to a large button near the focus of the window, and can be annoying if the user needs to use it repeatedly. On the other hand, the close/resize buttons at the top right of windows in Windows and the top left on a Mac are typically used only infrequently. Placing them far away from the central area of the screen makes it unlikely to click them accidentally.

A GUI can provide *keyboard shortcuts*, so that the user doesn't need to move the cursor to initiate an action, or *context menus*, where a user can right-click to display a menu wherever the cursor happens to be, enabling actions that make sense at that particular location. *Tool tips* can be displayed when hovering with the mouse over a component to describe its function.

Consider using some of these features to provide an intuitive and manageable interface.

### 7.2 Color Schemes

As a general rule, use highly saturated colors sparingly. Saturated colors make sense in the (few) places where you want to draw the user's attention. Avoid having too many colors at once; monochromatic, adjacent, triad, or tetrad schemes work well. A useful site for picking color schemes is paletton.com, or for a more random approach coolors.co

### 7.3 Navigation

Scrolling and zooming must be implemented to make it easy to view and navigate large worlds. You should also be able to resize the window of your program so that it can be effectively used on screens different than your own.

## 8 Concurrency

You have been given a tree structure with some methods implemented for you. You will implement `give()` and `query()` to be thread-safe versions of `insert()` and `get()`. We have given you implementations of these methods for reference, as well as test cases that will measure not only the correctness of your implementation, but also whether it supports parallelism well.

Note that the `give()` function uses an interesting rule for breaking ties: it calls a customizable comparator and keeps the greatest value. (`insert()` does the same thing) This makes more sense here than keeping the most recently inserted value, as we don't have an idea of which values were inserted first with a parallel structure.

The file `TreeTest.java` has a method `testAll()` which measures the run time of your `give()` and `query()` methods when the work is split up over multiple threads. You should see the performance increasing as you add more threads, especially for the first few. We also encourage opening your Task Manager or System Monitor (or whatever your OS calls

it) to see the load on your CPU; you should expect to see work happening on all cores when you have as many threads as cores. Do bear in mind that modern schedulers will try to balance threads and move them between cores in ways that may not be obvious, so if you have 8 cores, run 4 threads, and see modest load on all 8 cpus, your OS is probably just switching those 4 threads around.

Consider this as the written problem for this assignment. You may use the concurrent tree you implement in your design for the GUI, but it is not required. (There is probably no use for it in this project.) Concurrency is a difficult topic, and we advise all team members to help on this one!

### 8.1 Synchronization

We have provided the beginnings of an implementation in `ConcurrentTreeImpl.java`, but you are free to change this implementation as much as you like as long as it still implements the `ConcurrentTree` interface.

To make your tree work correctly, you need to add some kind of synchronization to prevent threads from interfering with each other and breaking invariants. A simple approach that you might start with is to lock the whole tree around every call to `give()` and `query()`. However, this approach will drastically reduce parallelism and prevent your performance from scaling with the number of threads. To get more parallelism, finer-grained locks are needed. Think about how to lock only parts of the tree so that two threads can be working in parallel as long as they are not touching the same part of the tree.

### 8.2 Helper functions

We have tried to make this as painless as possible by providing you with some helper functions. The `inOrder()` and `preOrder()` methods are easy ways to print out your tree traversal, and can help you check the Order invariant as well as whether all the elements you expect are there for smaller trees. The `testAll()` test function is also going to be something you should check often. It prints output to a CSV file, so you can graph the data in whatever spreadsheet you prefer. As the thread count goes up, you should see the first number going up and the second going down. Turn in a graph of the first value (the inverse of run time) vs. the number of threads in a file `TreePerformance.pdf`.

## 9 Overview of tasks

Determine with your partner how to break up the work involved in this assignment. Here is a list of the major tasks involved:

- Implement a GUI to display the state of the critter world and respond to user input.
- Connect the simulation engine from Assignment 5 to the display. This means allowing the display to update as the simulation progresses and to start, stop, and step the

simulation.
- Implement the loading of critter and world files and the placement of critters into an existing world.
- Implement smell specified above so that critters can find food.
- Finish implementing the `ConcurrentTree` interface.

## 10 Tips and Tricks

This is your final submission of the project, you want to make the project as a whole work well. We suggest aiming to get the GUI functionality working in advance of the due date so you have a few days to polish the project.

Take care not to entangle your world model with this new user interface. Proper separation of the simulation and GUI is important and something we will be looking for. The model should not depend on the user interface in any way. As usual, we will be looking for good documentation of your classes and their methods.

GUI code can become quite long, and you will likely have to make a conscious effort to keep it clean and readable, more so than with previous assignments. In addition to organizing your classes in packages, think about how to organize your resources (fxml files, images, icons). Never access resources using absolute pathnames; they should load properly even if the project's location changes. Instead, use one of the methods in ClassLoader or Class that look for resources in your program's classpath.

If you are having issues with FXML files, make sure they are located within `src/main/resources` under the same package as your main class. If, for example, your main class resides within a separate package `view`, then you must put your resources within `src/main/resources/view`. You should then be able to access the FXML files using `getClass().getResource()`.

## 11 Submission

You should submit these items on CMS:

- `overview.txt/pdf`: Your final design overview for the assignment. It should also include descriptions of any extensions you implemented. Additionally, you should document the different aspects of your GUI. Do not assume that all observable features of your GUI are noticeable to an unfamiliar user. You should also indicate the operating system and the version of Java you use.
- Zip and submit your `src` directory. This directory contains:
  - **Source code**: You should include all source code required to compile and run the project. All source code should reside in `src/main/java` with an appropriate package structure.
  - **Resources**: All resources for your GUI should be under `src/main/resources`. These will be included by simply zipping up `src`.

  - **Tests**: You should include code for all your test cases in `src/test/java` and resources for your tests in `src/test/resources`. You are welcome to create subpackages to keep your tests organized.

  Do not include any files ending in `.class`. Git users can save space by excluding the hidden `.git` folder when zipping.
- `build.gradle`: As mentioned, you are allowed to include external dependencies for this project. (If you do, please clear it with the course staff in advance). You should submit your `build.gradle` file containing the correct main class name and any dependencies you require. Make sure your `build.gradle` includes everything the original released file contained along with any additional dependencies.
- `screenshots.pdf`: A `.pdf` file containing 3–4 pages of screenshots showcasing your GUI.
- `log.txt`: A dump of your commit log from your version control system.
- `TreePerformance.pdf`: a `.pdf` file containing your performance graph of the Concurrent Tree.
- `ConcurrentTreeImpl.java`: Your implementation of the `ConcurrentTree` interface. Don't change anything in any of the other files for the tree, as we won't have them. We will not be evaluating your test cases for this part.