Navigation

Due: Monday, 17 September 2018

Navigation

Introduction

The Puzzle

Program Design

Instrumentation and Testing

Discussion

Submission and Version Control

Project 0: Galaxies

Due: Monday, 17 September 2018

Navigation

- Introduction
- The Puzzle
- Program Design
- Instrumentation and Testing
- Discussion
- Submission and Version Control

Introduction

This initial programming assignment is intended as an extended finger exercise: a mini-project rather than a full-scale programming project. The intent is to give you a chance to get familiar with Java and the various tools used in the course.

We will be grading *solely* on whether you manage to get your program to work (according to our tests) and to hand in the assigned pieces. There is a slight stylistic component: the submission and grading machinery require that your program pass a mechanized style check (`style61b`), which mainly checks for formatting and the

style61b guide for a description of the style it enforces and how to run it yourself.

First, make sure that everything in your repository is properly updated and checked in. Before you start, the command

```
cd  ~/repo
git status
```

should report that the directory is clean and that there are no untracked files that should be added and committed. *Never* start a new project without doing this.

To obtain the skeleton files (and set up an initial entry for your project in the repository), you can use the command sequence

```
git fetch shared
git merge shared/proj0 -m "Get proj0 skeleton"
git push
```

from your Git working directory. Should we update the skeleton, you can use the same sequence (with an appropriate change in the `-m` parameter) to update your project with the same changes.

On the instructional machines (only), you can run the staff version of this program with the command

```
staff-galaxy
```

It takes the same options as your project (see Instrumentation and Testing).

Navigation

Due: Monday, 17 September 2018

Navigation

Introduction

The Puzzle

Program Design

Instrumentation and Testing

Discussion

Submission and Version Control

The Puzzle

collection of GUI games (available on a variety of operating-system configurations, including Ubuntu.) He attributes the game to the puzzle-game publisher Nikoli, which published it as Tentai Show (天体ショー *tentai shō*). In this mini-project, you are given an incomplete Java program that creates these puzzles and allows its user to solve them, and you must supply the missing parts to complete the implementation.

The puzzle itself is quite simple. It is played on a rectangular grid of square cells. A number of "galactic ***centers***" (also known as simply "centers"), displayed as small circles, are placed about the board. Each galactic center is either in the center of a ***cell***, at the ***intersection*** of two grid lines and surrounded by four cells, or in the center of an ***edge*** between two cells. No centers appear on the outside edges of the puzzle board. The user who is solving the puzzle must place barriers along the edges of the board in order to divide the cells of the board into contiguous "***galaxies***" (also known as simply "regions"), where each galaxy must enclose exactly one galactic center. To successfully solve this puzzle, the user must make galaxies that cover all the cells without overlapping that also satisfy the following criteria:

1. It must be possible to reach any cell in the galaxy from any other cell via a sequence of cells in the same galaxy that are only horizontally or vertically adjacent (not diagonally adjacent).
2. The galactic center must be entirely enclosed in the galaxy.
3. The galaxy must be symmetric around the galactic center: if the galactic center is at coordinates (x, y) and there is a cell in the galaxy at coordinates

Navigation

Due: Monday, 17 September 2018

Navigation

Introduction

The Puzzle

Program Design

Instrumentation and Testing

Discussion

Submission and Version Control

$(x - ax, y - ay)$ must also be in the galaxy.

Navigation

Due: Monday, 17 September 2018

Navigation

Introduction

The Puzzle

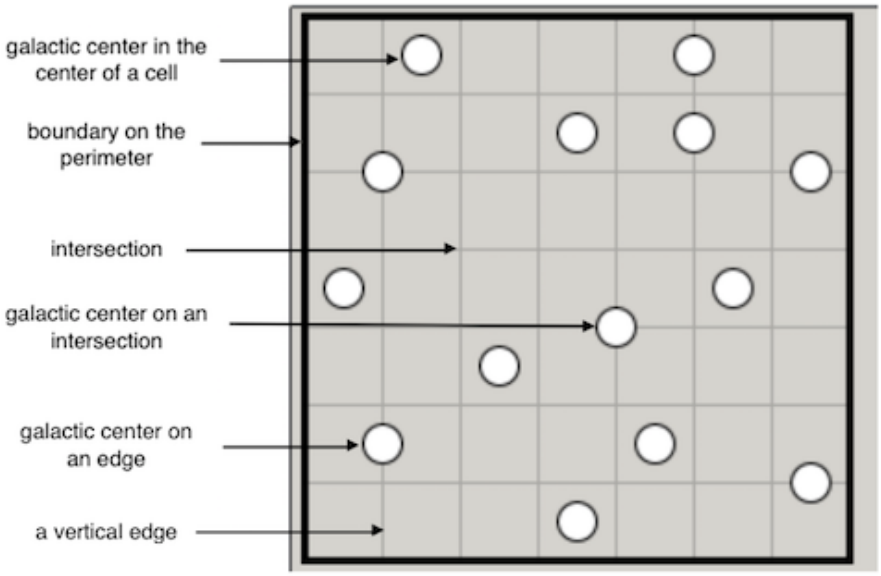
Program Design

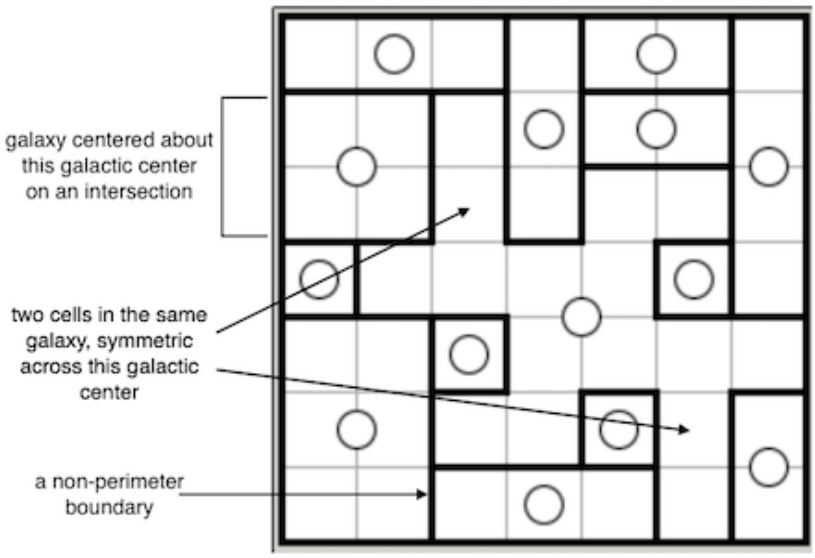
Instrumentation and Testing

Discussion

Submission and Version Control

The diagrams below show a sample puzzle on the left and its solution on the right. The bottom-left corner of the board has the coordinates $(0, 0)$, with the positive x-axis running horizontally to the right, and the positive y-axis running vertically upwards. The large block comment above the class declaration in `Model.java` explains the coordinate system and the significance of even and odd numbers in the coordinates.





Navigation

Due: Monday, 17 September 2018

Navigation

Introduction

The Puzzle

Program Design

Instrumentation and Testing

Discussion

Submission and Version Control

boundaries along the edges between cells. Clicking on an edge between two cells toggles a boundary on that edge. The program detects when a boundary completely encloses a properly symmetric galaxy and when the puzzle is completely solved. There must not be ***stray*** boundary edges in the middle of a galaxy that are not part of its outer boundary. The perimeter of the game board is a permanent boundary enclosing all the cells as well.

Throughout the project, you will see mention of "marks". These are integer values that will be assigned to cells. If cells are marked with the same value, then they are part of the same set of cells. These sets of cells are used in a few ways, one of which is to denote which cells to display white in the GUI.

Program Design

The skeleton exhibits a *design pattern* in common use: the Model-View-Controller Pattern (MVC).

The MVC pattern divides our problem into three parts:

- The *model* represents the subject matter being represented and acted upon—in this case incorporating the state of a board game and the rules by which it may be modified. Our model resides in the `Model` and `Place` classes.
- A *view* of the model, which displays the game state to the user. Our view resides in the `GUI` and `BoardWidget` classes.
- A *controller* for the game, which translates user actions into operations on the model. In our case, it also notifies the view when it may need to check with

Navigation

Due: Monday, 17 September 2018

Navigation

Introduction

The Puzzle

Program Design

Instrumentation and Testing

Discussion

Submission and Version Control

controller resides mainly in the `Controller` class, although it also uses the GUI class to read mouse clicks.

Your job for this project is just to modify and complete the `Model` class. Don't let that stop you from looking at all the other code in the project. That's actually part of the point of giving you the skeleton. You can learn a great deal about programming by reading other people's programs.

In fact, you can change other source files if you want, just so long as the program continues to behave the same when tested.

Instrumentation and Testing

To facilitate automated testing of your work, there are a few features that you can use to record sessions and to play back moves for testing or debugging purposes. The skeleton is set up so that when you start your program with

```
java -ea galaxy.Main --log
```

you'll get a record on the standard output of all of the edges clicked, all the randomly created puzzles, and all the menu commands entered. You can capture this log using redirection, like this:

```
java -ea galaxy.Main --log > testing/myscript1.in
```

The `--seed` option will allow you to prime the random number generation so that you can get the same set of random numbers each time:

```
java -ea galaxy.Main --seed=42
```

Navigation

Due: Monday, 17 September 2018

Navigation

Introduction

The Puzzle

Program Design

Instrumentation and Testing

Discussion

Submission and Version Control

The `--testing` option reads in a script produced by `--log` and uses it (in place of user clicks and random numbers) to supply the results of `getPuzzle` and `getCommand`. It also prints out a textual display of the board before any edges are added and after each edge, which `Tester.py` can use to test the program. For example, to read back the file `myscript1`, use

```
java -ea galaxy.Main --testing < testing/myscript1
```

This will print out what tester.py sees, which it compares to `testing/myscript1.out`. Using `--log` and `--testing`, you can thus create your own `.in` and `.out` testing files to augment the ones we've provided in `testing`.

We have set up the Makefiles in the `proj0` directory and in the `galaxy` subdirectory to have targets `check` and `unit`. In order to use make, you'll need to install it. Follow the instructions here to do so.

Once installed, the command

```
make unit
```

will run *unit tests* on `Model.java`. A unit test checks a particular unit of a program—typically a method or small group of methods. If you try this on the skeleton, you'll see that, unsurprisingly, it fails all these tests, giving you an indication of what needs to be changed.

In addition, `make check` in the `proj0` directory will run *integration tests*, which run the program and check its outputs against expected output using the scripts `tester.py` and `testing.py` in the `testing` subdirectory.

Navigation

Due: Monday, 17 September 2018

Navigation

Introduction

The Puzzle

Program Design

Instrumentation and Testing

Discussion

Submission and Version Control

Discussion

This project is largely an exercise in reading someone else's program and understanding its intent, which is actually a common activity in "real-world" programming as well. While you can, in fact, scrap everything and start from scratch (you are free to change files other than `Model.java`), you certainly should *not* do so simply because you don't understand the skeleton provided.

You'll see numerous uses of parts of the Java library we haven't talked about. However, you *have* seen all these data structures in Python. Where Python has lists, Java has arrays, `ArrayList`s, `ArrayDeque`s, and `LinkedLists` (among others). Where Python has dictionaries, Java has `HashMap`s and `TreeMap`s. Where Python has sets, Java has `HashSet`s and `TreeSet`s. Where Python uses "duck typing"—as when it calls something a sequence because we can apply generic operations like `len` and `map` to it and iterate over it—Java explicitly identifies supertypes of its various library types, such as `List`, `Set`, and `Collection`. These and many more library classes are all defined in the online Java library documentation, which should constitute much of your bedtime reading for this semester.

So your first step is to understand what `Model` is supposed to do and what the comments on each of its methods are supposed to mean. You might ask, "but how can I understand it unless I know how it's being used?" It certainly helps to understand its use, but you should start developing the ability to understand and work on modules in isolation, one aspect of the "separation of concerns" that makes the construction of large systems possible.

Navigation

Due: Monday, 17 September 2018

Navigation

Introduction

The Puzzle

Program Design

Instrumentation and Testing

Discussion

Submission and Version Control

basically) that you think will make it possible, in principle, to implement all the methods. The question here is, "what information does my program need to do X?". At this point, you should be in a position to implement the simple stuff, such as `xlim`, `ylim`, `isCenter`, `isBoundary`, `centers`, `placeCenter`, `toggleBoundary`, `mark`, and `markAll`.

This leaves the more complex methods, such as `unmarkedSymAdjacent`, which tells how a galaxy might grow. The method `maxUnmarkedGalaxy` requires that you add cells to a region so as to keep it a valid galaxy until the region is as large as possible. This is a kind of *fixpoint process*, in which you repeat an operation until no such repetitions are possible. In this case, we add to the region two as-yet unmarked cells that are symmetric about a center and adjacent to other cells already in the region, ***mark*** these cells now that they have been added, and then repeat, expanding to more unmarked cells until no more additions are possible. Feel free to discuss with other students how this might be organized (as usual, share ideas, not code).

This project tries to make good use of abstraction; the simpler functions you write will be instrumental in helping you accomplish more complex goals. Be sure to respect the abstraction barriers to minimize the impact of bugs or logical errors in your code. Further, if you find yourself bogged down by the details of a particular method, consider pausing to break it down into "helper" functions, each performing some coherent piece of the original function (only, as a stylistic matter, do give these new functions descriptive names, generally avoiding the word "helper" in the name.)

Navigation

Due: Monday, 17 September 2018

Navigation

Introduction

The Puzzle

Program Design

Instrumentation and Testing

Discussion

Submission and Version Control

Submission and Version Control

It is important that you commit work to your repository *at frequent intervals*. Version control is a powerful tool for saving yourself when you mess something up or your dog eats your project, but you must use it regularly if it is to be of any use. Feel free to commit every 15 minutes; Git only saves what has changed, even though it acts as if it takes a snapshot of your entire project.

The command `git status` will tell you what you have modified, removed, or added since the last commit. It will also tell you what you have not yet sent to your central repository. You needn't just assume that things are as you expect; `git status` will tell you whether you've committed and pushed everything.

If you are switching between using a clone of your central repository on the instructional computers and another at home (or on your laptop), be careful to synchronize your work. When you are done working on one system, be sure to push your work to the central repository:

```
git status                          # To see what
git add <filepath>                  # To add, or s
git commit -a -m "Commit message"   # To commit ch
git push
```

If you start working on the other system, you then do

```
git status          # To make sure you didn't acci
                    # stuff uncommitted or untrack
git pull --rebase   # Get changes from your centra
```

Submit your project by committing and tagging it:

```
git tag proj0-0     # Or proj0-1, etc.
git push
```

Navigation

Due: Monday, 17 September 2018

Navigation

Introduction

The Puzzle

Program Design

Instrumentation and Testing

Discussion

Submission and Version Control

Be sure to respond to all prompts and to make sure the messages you get indicate that the submission was successful. Don't just "say the magic words" and assume that everything's OK.