Victoria University of Wellington
School of Engineering and Computer Science

SWEN221: Software Development

Assignment 5

Due: Friday 7th June @ Mid-night

1 Settlers of Catan

Catan is a popular board game where up to 4 players collect resorces, build settlements and play for points on a hexagonal board.

3:1
3:1
3:1
3:1
8
4
6
11
2
12
10
5
9
11
9
3
8
3
6
10
4
5
CATAN MAPS .COM

On each turn, players may place settlements on the corners of an available hex, or a road on an available edge. At the start of each turn 2 dice are rolled and resources are produced from the hexes labelled with the rolled number. These resources are then used by players to build settlements to hopefully collect more resources until some player collects enough points to win. Points are earned by building settlements. The full rules also include a set of cards called “development cards” and a set of trading posts, but we are modelling a variation that **does not feature development cards or trading posts**, so you do not need to worry about them in this assignment. To better understand the rules, you can find the official rule set at `https://www.catan.com/en/download/?SoC_rv_Rules_091907.pdf`

1.1 Setup

Before the game takes place, setup must be performed.

1. A hexagonal board is constructed as in the above Figure by placing 3 concentric layers of hexagonal tiles. That is, a single tile, surrounded by six tiles, which are then also surrounded by 12 additional tiles.
2. Each tile is assigned a resource from a randomly sorted pool of 4 **Wood**, 3 **Stone**, 3 **Brick**, 4 **Wheat**, 4 **Sheep** and 1 **Desert**. This is done row by row, from left to right and top to bottom. For example in the image on the first page, the leftmost tile on the top row is assigned stone, followed by wood for the middle top tile and brick to the rightmost top tile. Then wheat is assigned to the leftmost tile on the second row, and so on.
3. Each tile is assigned a resource counter with a number (from 2 to 12). This is done in a clockwise fashion starting from the top leftmost tile and spiralling inwards. In the above figure we start with 8, then 4, 6, 10, 9, 6, 5, 4, 10, 3, 5, 11, 2, 12, 11, 3, 8, 9. The final tile in this example is a Desert, but in most games the desert will not necessarily be in the centre, and will be skipped when placing resource counters.
4. Each player take turns to place two settlements and two roads on the board. Settlements are placed at the corners of tiles and roads along the edges. Settlements may not be adjacent to another player’s settlement. Roads have to be placed next to a player’s settlement.

1.2 Order of Play

A player’s turn consists of the following actions:

1. Distribute Resources: Two six-sided dice are rolled giving a number between 2 and 12. All tiles marked with that number distribute resources to all players who own settlements on that tile according to the following rules:
   - Each town gets 1 resource
   - Each city gets 2 resources
2. Build: The player may build new towns, upgrade existing towns to cities or build roads.

There are several rules that govern how the above actions take place. Firstly, a player may not place a settlement adjacent to another player’s town. Each possible site for a settlement or a road may only contain one settlement or road. Except in the setup phase, players may only place towns in open spots next to roads they own.

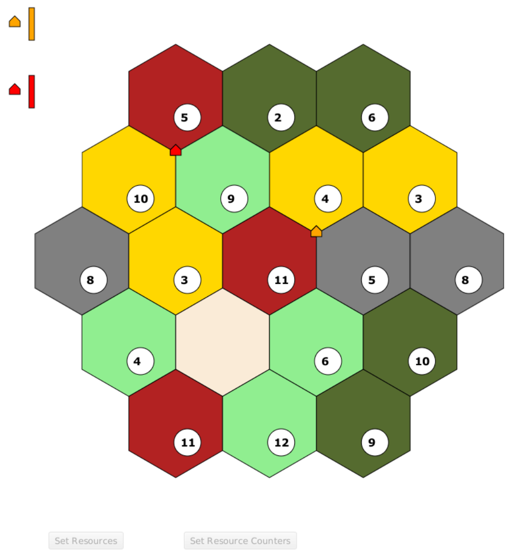
Points are awarded in the following way: 1 point for each town and 2 points for each city.

2 Getting Started

In this assignment you will implement certain parts of the Catan game. Using *Generic types* you will implement a hexagonal graph where each node is a tile on the board. Using *Java Streams* and *Java Lambdas* you will implement parts of the logic of the setup.

You can complete this assignment by working on the code in `catan.jar`. However, there is a GUI available in `catanGUI.jar`; you can run the GUI with the command line `java -jar catanGUI.jar`. It expects a jar `catan.jar` in the same folder. At the start, the GUI will not work properly since you need to implement more behaviour in `catan.jar`. For the submission, you only need to submit your improved version of `catan.jar` (with source code, as usual).

When you have completed `catan.jar`, you should see a GUI come up similar to the following:



When your code is completed, pressing the buttons “Set Resources” and “Set Resource Counters” will allocate resource types (e.g. stone, brick, etc) to the tiles and amounts to those tiles. You can then build settlements and roads by dragging from the respective icons.

2.1 Understanding the Code

To get started, import the `catan.jar` file from the lecture schedule on the course website. For this assignment, there are three key classes you will be working with:

- (`HexNode`). The class `game.HexNode` represents an hexagonal tile on the board and implements the interface `graph.Node`. This class gives structure to the underlying board allowing, for example, one to determine adjacent tiles, etc.
- (`CatanTile`). The class `catan.game.CatanTile` provides the contents for a `HexNode` which provides necessary functionality for the game itself.
- (`Game`). The class `game.CatanGame` provides a concrete instantiation of the game. It makes extensive use of classes in the package `model`.

In addition to the above files, supplementary code is present in `catanGUI.jar`. **The source code for this library is not provided, as you will not need to modify it, nor import it into eclipse**.

Part 1 — `HexNode` (worth 50%)

Before we can implement the full Catan game, we need to develop a basis for it. A skeleton implementation of `Node` is provided called `HexNode` which currently does not implement all the required functionality. You need to complete its implementation as necessary. When you've done this, you should find the tests in `Part1Tests` all pass. A valid graph should represent an hexagonal grid, this means that for example:

- A node can not have itself as a neighbour.
- If going WEST from node1 we reach node2, then going EAST from node2 we reach node1. That is, the graph is non-directional: if node1 is connected to node2, then node2 is connected to node1 in the opposite direction.
- If going EAST from node1 we reach node2, and going NORTHEAST from node1 we reach node3, then going SOUTHEAST from node3 we reach node2.
- the method `HexNode.isValid()` contains useful assertions checking that the graph is valid according to those criteria. We encourage you to use `isValid()` or a similar personalized method to check if your code is preserving the validity of the graph.

We suggest you to implement the methods in the following order:

1. `hasNeighbor(Direction)`
2. `add(Direction, Node<Direction, V>)`
3. `isConnected(Direction, Node<Direction, V>)`
4. `connect(Direction, Node<Direction, V>)`
5. `fillNeighborhood()`
6. `generate(Integer)`
7. `toList()`
8. `stream()`
9. `clockwiseStream()`

Carefully read the documentation of what those methods should do on the interface `Node`. The last two methods allows to stream the nodes:

- `stream`: nodes are streamed from left to right, and top to bottom starting at the top leftmost node and ending at the bottom rightmost node.
- `clockwiseStream`: nodes are streamed clockwise starting from the top leftmost node, and ending at the center node.

NOTE: while there are complex but efficient ways to create streams without materializing the list of elements, this is not required by this assignment, and you can simply create a list with the elements in the right order and then create a stream from such list.

3 Part 2 (worth 40%)

`CatanTile` and `CatanGame` have many methods that needs to be implemented in this phase. For the methods `CatanTile.addSettlement(Settlement, Location)` and `CatanTile.addRoad(Road, Direction)`, observed that a tile *shares* it's settlement and road locations with neighbouring tiles.

In `CatanGame`, there are several methods which we suggest you implement in the following order:

1. `toString(Stream<Node<K, V>> stream, Function<V, String> mapper>`. This method aims to provide a rich mechanism for printing out information from the graph. This method makes good use of generic types to ensure a flexible API.

   NOTE: `CatanGame` already has an implementation of the basic `toString()` method. This returns a string of the ids of all the nodes in the correct order. *You may find it useful to examine this method.*

   HINT: After implementing this method, you should find that test `toStringTest1()` now passes.

2. `setResources(List<Resources> pool)`: taking a pool of resources we need to distribute them in a left to right, top to bottom manner as previously described. *The implementation of this method is provided as an example.*

   HINT: To get tests `testSetResources1/2/3()` to pass, you will need to implement methods in `CatanTile`.

3. `setResourceCounters(List<ResourceCounter> pool)`: taking a pool of resource counters, we need to distribute them on the board in a clockwise fashion, starting from the top leftmost tile, spiralling into the center. If we encounter any `DESERT` tiles, we assign that tile a counter with the value `NONE`. *This method needs to be implemented.*

   HINT: Reading `testSetResourceCounters1` may help you to understand the required behaviour better.

4. `distributeResources(Integer diceRoll)`: taking a dice roll of between 2 and 12, we need to distribute any resources gained to players that have settlements on tiles with that number. It is a good strategy to implement `distributeResources` last. A correct implementation for `CatanTile.addSettlement(Settlement, Location)` is crucial to be able to complete `distributeResources(Integer diceRoll)`. *This method needs to be implemented.*

When you have implemented those methods, you should find the tests in `Part2Tests` all pass.

Submission

Your lab solution should be submitted electronically via the *online submission system*, linked from the course homepage. The minimum set of required files is:

```
game/HexNode.java
game/CatanTile.java
game/CatanGame.java
```

You must ensure your submission meets the following requirements (which are needed for the automatic marking script):

1. **Your submission is packaged into a jar file, including the source code**. *Note, the jar file does not need to be executable.* See the following Eclipse tutorials for more on this:

   `http://ecs.victoria.ac.nz/Support/TechNoteEclipseTutorials`

2. **The names of all classes, methods and packages remain unchanged**. That is, you may add new classes and/or new methods and you may modify the body of existing methods. However, you may not change the name of any existing class, method or package. *This is to ensure the automatic marking script can test your code.*

3. **All testing mechanism supplied with the assignment remain unchanged.** Specifically, you cannot alter the way in which your code is tested as the marking script relies on this. However, this does not prohibit you from adding new tests. *This is to ensure the automatic marking script can test your code.*

4. **You have removed any debugging code that produces output, or otherwise affects the computation.** *This ensures the output seen by the automatic marking script does not include spurious information.*

Note: Failure to meet these requirements could result in your submission being reject by the submission system and/or zero marks being awarded.

Assessment

This assignment will be marked as a letter grade (A+ ... E), based primarily on the following criteria:

- **Correctness of Part 1 (50%)** — does submission adhere to specification given for Part 1.
- **Correctness of Part 2 (40%)** — does submission adhere to specification given for Part 2.
- **Style (10%)** — does the submitted code follow the style guide and have appropriate comments (inc. Javadoc)

As indicated above, part of the assessment for the coding assignments in SWEN221 involves a qualitative mark for style, given by a tutor. Whilst this is worth only a small percentage of your final grade, it is worth considering that good programmers have good style.

The qualitative marks for style are given for the following points:

- **Usage of streams and lambdas**. For this particular assignment, tutors will also check that code is using `streams` and `lambdas` where appropriate.

- **Division of Concepts into Classes**. This refers to how *coherent* your classes are. That is, whether a given class is responsible for single specific task (coherent), or for many unrelated tasks (incoherent). In particular, big classes with lots of functionality should be avoided.
- **Division of Work into Methods**. This refers to how well a given task is split across methods. That is, whether a given task is broken down into many small methods (good) or implemented as one large method (bad). The approach of dividing a task into multiple small methods is commonly referred to as *divide-and-conquer*.
- **Use of Naming**. This refers to the choice of names for the classes, fields, methods and variables in your program. Firstly, naming should be consistent and follow the recommended Java Coding Standards (see `http://g.oswego.edu/dl/html/javaCodingStd.html`). Secondly, names of items should be descriptive and reflect their purpose in the program.
- **JavaDoc Comments**. This refers to the use of JavaDoc comments on classes, fields and methods. We certainly expect all `public` and `protected` items to be properly documented. For example, when documenting a method, an appropriate description should be given, as well as for its parameters and return value. Good style also dictates that `private` items are documented as well.
- **Other Comments**. This refers to the use of commenting within a given method. Generally speaking, comments should be used to explain what is happening, rather than simply repeating what is evident from the source code.
- **Overall Consistency**. This refers to the consistent use of indentation and other conventions. Generally speaking, code must be properly indented and make consistent use of conventions for e.g. curly braces.

Finally, in addition to a mark, you should expect some written feedback highlighting the good and bad points of your solution.

Java FX Notes

We advise you to just use `java -jar catanGUI.jar` to run the gui, but some of you may be trying to import `catanGUI.jar` into eclipse as a separate project, dependent over `catan.jar`.

In some instances eclipse does not recognise parts of JavaFX (used in the ui) as part of a public api. If you have this error, you can get around it by adding an accessibility rule for the build path of your project. To do this follow the following steps.

1. Go to the build path configuration screen for your project in eclipse by selecting your project, and opening '`Properties`' from the '`Project`' menu at the top.
2. Navigate to the '`Build Path`' tab on the left.
3. Select the '`Libraries`' tab.
4. Expand the '`JRE System Library`' entry.
5. Select '`Access Rules`' and press the '`Edit`' button.
6. Add a rule with Resolution '`Accessible`' and Rule Pattern '`javafx/**`'
7. Apply changes and close.