Tests

The following methods are used when testing the program.

Unit Test

A Java framework, JUnit5, has been used within the process. You may consult to `com.luox6.conway.test` for the written test cases.

Testing scenarios

Cell

Cell represents a single cell in the map

- clone a new cell with exact same state
- set status of the cell
- record survival times

ConwayMap

ConwayMap is the a map represents the status of cells

- row/col length getters
- class constructor
- `getCell` cell status with coordinate wrapping
- `neighbour` live cell counts around specific coordinates
- `toString` string representation/output format
- count cell statuses

MapCollection

MapCollection is a collection of ConwayMap, represent continuous progress from a single map

- map index increments
- collection reset
- identify calculated indexes

MapModel

MapModel include extra functionality over Collection, where view required. E.g. current viewing map index

- step actions (go back and forth)
- identify if simulation has started
- set (can calculate, if necessary) index to given location
- input error handling

The above have tested model class's construction, mutations, and expected string representation.

Run Unit Test

You may use your favourite IDE integration with JUnit. Please consult JUnit manual However, if you would like to use `ConsoleLauncher`, after compiling the program following the instruction in `readme.md`, try the following command under project root directory for unit test result.

```
$ sh scripts/test.sh
```

or windows platform

```
.\scripts\test.cmd
```

Please consult `test-log.txt` for complete log.

Manual Test

However, the unit test cannot cover some corner cases, e.g.:

- Seed file parsing
- IO/Input validation
- GUI functionality
- …

CLI & Data Representation

In order to test these cases, here we have some crafted files, which are also listed in `manual->Example Args` section. These cases have covered the following topics:

- example1.txt -> Given test file from homework description
- example2.txt -> Wrapping, neighbour counts, simple, mostly static case
- example3.txt -> Wrapping, neighbour counts, complex moving case
- invalid1.txt -> Seed file with < 2 rows
- invalid2.txt -> Seed file with < 2 columns
- invalid3.txt -> Seed file with undefined char
- invalid4.txt -> Seed file with inconsistent rows
- invalid5.txt -> Empty seed file

Commands to run these examples have also listed in the manual. It is believed that these cases cover most of the common mistakes that users would make.

Game rules are also tested at `example2.txt` and `example3.txt` by stepping through the generated files.

You may find the expected output in `tests/result` section. Use `diff` for the file generated from the program with corresponded command – these should be identical, or it means something goes wrong.

Some examples inspired by Wikipedia.

GUI & Action test strategies

As the underlying data representation has both tested by the unit test and the manual test section, for the GUI part, it is mostly focused whether the data bindings are working, or if target behaviors are reached.

Before simulation You may find the status bar on the left bottom shown as "Ready to start"

- cell color button color and text changed when pressed
- calculated simulation status on the right bottom should be 0
- top stepper number should be 0

Simulation in progress

- warning when cell is pressed
- calculated simulation status on the right bottom should be a positive number
- top stepper number will change as step forward and step back
- the number of live/dead cell on the bottom middle should update with the map

Notice: it is normal that the calculated simulation is 1 less than the cell survival time. The App calculated the survival at stage 0 with times 1

Toolbar

- `Set as Begin` should clear all cells' survival time (0 for dead, 1 for live)
- `New Map` should pop up a dialog asking for row and give error if value cannot be parsed or less than 1
- `Reset` should set everything of the map to 0 but keep the row and col as the same
- `Configuration` should invoke configuration panel
- toolbar can be dragged out of the window and restore with no problem (by clicking the exit button of the pane)

Save file

- file with given name will appear and can be open via a menu -> open file
- range saving button result and override notice
- filename format

in the test folder, `example2-result` contains the result where saved range from 0 to 32, as a reference

Configuration

- setting persistence (after application closed)
- setting update should immediately reflect to the board
- cell color changes accordingly
- hide/show cell survival time
- cell shade changes accordingly

Warnings

- at any time, textfields that expected number but failed to parse will give warning
- at any time, if the action of textfield failed, the value will be restored to its previous value