Cellular Automaton

Technical documentation

By

Sajjad Hassanpour

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Document metric** | | | | | | | |
| Project | Cellular Automaton | | | Company | | WUT | |
| Name | Technical document of – Cellular Automaton | | | | | | |
| Topics | Requirements Specification | | | | | | |
| Author | Sajjad Hassanpour | | | | | | |
| File | Requirement Specification for - Life Board | | | | | | |
| Version no | 0.6 | Status | working | | Opening date | | 7.04 |
| Summary | Detailed description and illustration of the requirements for the Life Board application | | | | | | |
| Authorized by |  | | | Last modification | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Who** | **Description** |
| 0.6 | 7.04.2015 | Sajjad Hassanpour | Algorithm summary and conclusion |
| 0.5 | 5.04.2015 | Sajjad Hassanpour | Completing class Diagram |
| 0.4 | 4.04.2015 | Sajjad Hassanpour | Activity diagram and class diagram |
| 0.3 | 03.04.2015 | Sajjad Hassanpour | Basic idea of data structure and rule class |
| 0.2 | 01.04.2015 | Sajjad Hassanpour | Solution for rule conflict |
| 0.1 | 28.03.2015 | Sajjad Hassanpour | Introduction and idea of multi-threading of application |

**Table of content**

1. Busyness analysis by Mr. Michał Słupczyński

1.1. Summary from Busyness analysis

1.2. Development strategy

1.3. Technologies

2. Introduction of technical solution

3. Architecture

3.1.1. Use-case Diagram

3.1.2. Use-case Diagram Description

3.2.1. Activity Diagram

3.2.2. Activity Diagram Description

3.3.1. Class Diagram

3.3.2. Class Diagram Description

4. Data management

4.1. Data save and load

5. Algorithm Description

6. Conclusion

1.1. Summary from Business analysis

The history and development of cellular automata is attributed to polish-born Stanisław Ulam (1909 – 1984) and John von Neumann (1903 – 1957) of Hungarian origin, dates back to the 1940s and was based on re-search on crystal growth modelling and self-replicating robots. Cellular automata are relevant to the study of biology, chemistry, physics and many branches of science. The main goal of this application is to allow a legible and easy to use method of modelling and visualizing cellular automata.

A cellular automaton consists of a rectangular grid of cells, which can be in a specific state, a set of rules that defines the creation of a new generation in the system and an initial configuration.

1.2. Development strategy

As it was recommended I use waterfall methodology for development of this project which is effective in individual projects. Since this project is individual it makes sense to use this strategy and do the project in sequential order (as it is in waterfall model.)

1.3. Technologies

I will use Java as development technology of this project. I mostly target java 8 platform but since java 7 is still in use I do the best to have compatibility with it also. Java is one of the effective and popular technology in software development and choosing this technology can extends the use of the application into Linux operating system which is a good option.

3. Architecture

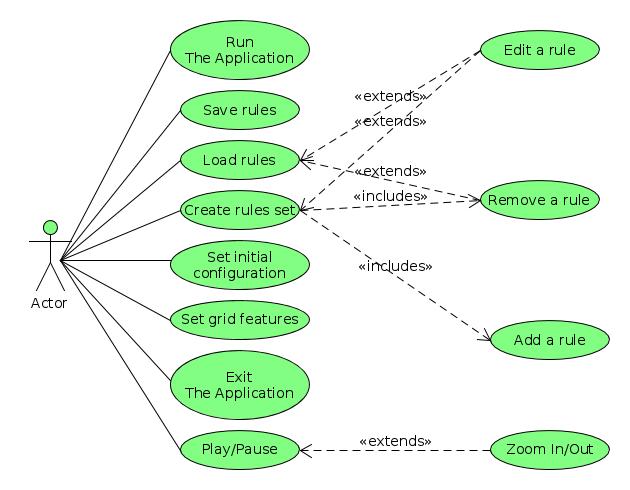
3.1.1. Use-Case Diagram

This diagram sum up business analysis to help us in technical procedure of application design. Specifically, illustration of use case helps me to come up with the next step “activity diagram”.

3.1.2. Use-Case Diagram Description

Basically, all the action the user can perform is mentioned in diagram; includes: Run the application, create rule set, save rule set, load rule set, start and pause game, …

*[USE-CASE DIAGRAM]*

**

3.2.1. Activity Diagram

The Activity Diagram is the most important part of technical report and solution of application. The flow of activity will be performed with user is illustrated here and also the solution for rules conflicts is started here and extended by more details in class diagram and class diagram description.

3.2.2. Activity Diagram Description

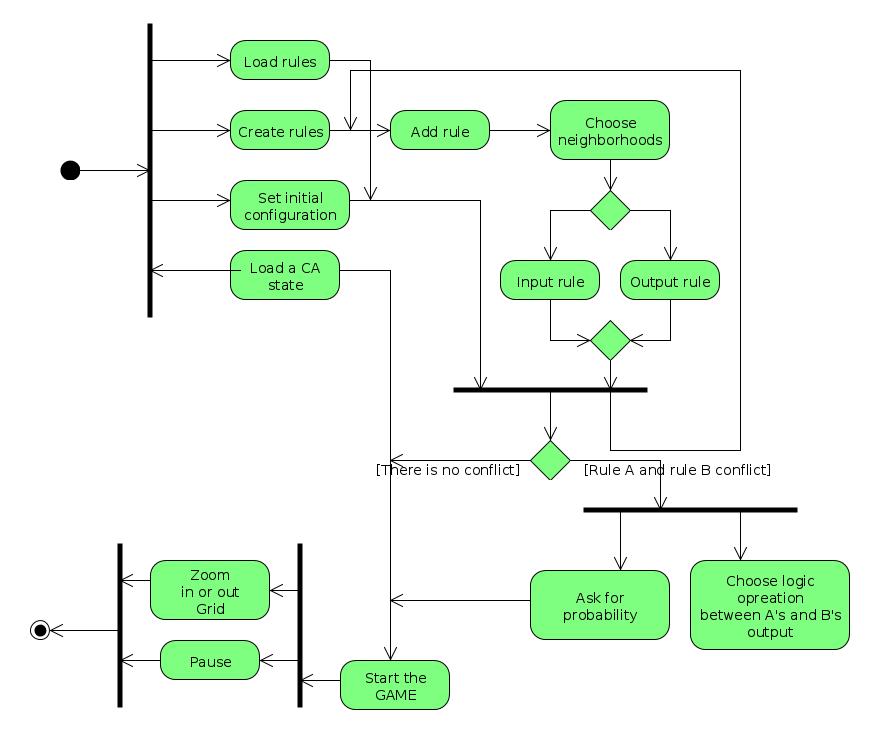
The user after run the application in order to start the game has to provide the application with an initial configuration and most important a set of rules. As it is shown in diagram rules can be created or loaded from previous time. It is trivial in order to load a rule or rule set the application has to provide user with save option. The same action is possible for configuration.

In creation of a rule the type of neighborhood has to be specified as also input and output state of the rule. In the next step according to the business analysis rules can be added in technical part we have to come up with solution for get and store a rule or a rules set and also a solution for solve the conflict between the rules. The first problem has the full solution in class diagram, just to mention the class rule will solve this problem. The second problem, conflict between rules, is the biggest issue of this report. The conflict can be considered the situation in which two rules have same state and the output implies by the rules is different. Obviously it would be confusion for automaton which output should be considered. In the Business analysis by Mr. Michał Słupczyński recommended solution is solve this issue with giving an option to user between either choosing a rule over the others or ask for a logic operation to apply among rules and consider the result of operation as final output. This solution is considered as an option since I am following the busyness analysis. This solution requires to have kind of prediction in advance of game start. In rule set class a method will implemented to recognize such conflicts.

~~The other solution for conflict problem can be to find conflict in advance like previous solution and give some suggestion to user and ask to edit a rule or rules to solve the conflict. For instance in following case, Rule A (a cell with more than 1 alive neighbors in fist row will die or stay died) and rule B (a cell with exactly two alive neighbors in first row will live or come to alive) are conflict in situation we have exactly two alive cells in first row because both rules can be applied and results are different. We can give such a suggestion to user for editing rule A to solve possible conflict (suggestion 1: instead of more than, use less than).~~

~~The next solution can be considered here is two ask user for a number as probability to keep a cell alive or dead in case of conflicts and apply that probability in case of conflict. For example in conflict between Rules A and B user ask to keep alive the cells with probability 60% the application with that probability will try to choose the rule which keeps the cell alive.~~

*[ACTIVITY DIAGRAM]*



Class Diagram

As is mentioned in Technology section I am going to use java for implementation of this application. The class diagram blew designed according to java class behavior. Despite classes I will have at least two interface in order to move between Jframes which is not mentioned in the diagram in case of simplicity.

Class Diagram Description

In the class diagram below GridFrame, RuleEditorFrame, RuleConflictFrame classes are window frame of application. The GridFrame is the window which user will see as first and main window. I will use GridBagLayout to layout the these Jframe classes and java swing components.

* The class Rule

This class is the solution to get, store and apply a rule during the runtime of application.

* + The class field HowManySign can has value -1, 0, +1 which represent in order at least, exactly and at most in a rule.
  + The class field HowManyNom indicate the number of alive or dead cell in a rule.
  + The class field OnType may has value 'R', 'C', 'DM', 'DMN' which representing Row, Column, Diagonal Main (or in main diagonal direction), Diagonal Not Main (or not in direction of not main diagonal) corresponding to the rule.
  + The class field OnPos indicate the OnType field number.
  + There are different method in this class to set and get attribute of the class to have the encapsulation.
* The class RuleSet

This class help to have set of rules in organized manner in the application implementation.

* + This class contains an important method which will find rules which are conflict with each other. This method has to be called after we get all rules from user and show the result in RuleConflictFrame. This result should have the solution for solve Rules conflict (which previous explained).
  + Also we have two methods to access rules in different way.
* The GridPanel class

This class is child class of Jpanel and I will draw the Game Grid on it. It ~~more or less~~ contains everything to control grid like: zoom in and out in grid etc.

* The Rule Engine class

This class considered for idea of using multi-thread in application to have a better performance in the application. General idea of this class is to have bunch of thread except just one thread in game. This idea could be extended in two approaches. The first one is to divide the rule set among these threads. So each thread will look for not all rules but just one or small number of the rules in game's grid and will find the output. Since this idea is just theory I faced with kind of difficulty. Imagine the case we have two conflicting rule in two different thread to be applied. If we use first approach to solve conflict (either do the logic operation or ask user to solve the conflict by giving him/her suggestion) there would not be any problem but in the case we keep the conflict and will choose between them with some probability looks communication among those threads is needed. In this solution in case of we had so many rules we can divide rules to some sub rules set and assign to each thread a set of rules except just one rule.

The second approach to have multi-thread application in order to have better performance is to divide the grid between threads. So each threads will just look to a part of grid not all of it which also helps performance. The decision of choosing one of the above approaches will more depends on what is going to happen later on during implementation.

* The Cell class

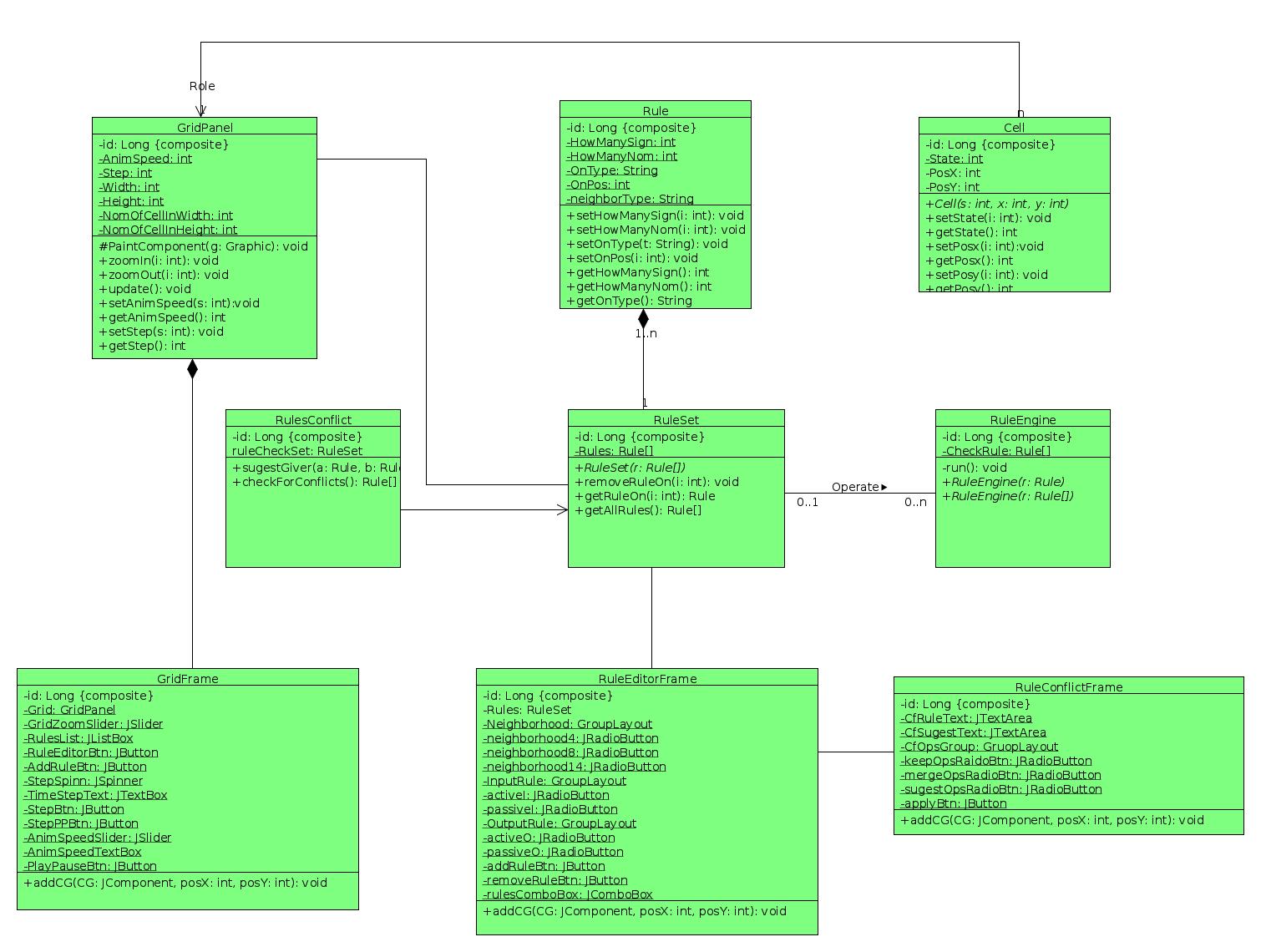
This class will provide all property that we need for a cell in our grid for a cell like neighborhoods, position of cell and state of a cell.

* RuleConflictFrame

This frame has important rule in application since it is dealing with one our issues (rule conflicts) in the application. This frame will appear after application finds out about conflicts in rule set. There are three possible solution will appear in the frame as first on the solution from busyness analysis merge rule's output by logic operation, second, suggestion to user in order to edit rules and at last ask user about the probability of keep system “alive” or “dead”.

* The RuleConflict class

All the solution algorithm about solving conflict issue which mentioned above will be implemented in this class.



4. Data management

4.1. Data save and load

Most crucial part of data management in this application is the part to store rules and state of the automaton. As I mentioned in class diagram the is two class related to get a logic sentence rule one as singular the other as set of rules. So here in order to store the objects of such classes we would not have big problems. I will use SQLite as database technology to save and load data more specific rules or a set of rules.

The state of automaton is quite easy to save when we have the rules saved. The state of automaton will considered as sequence of “1” and “0” which corresponded to alive and dead cells in addition with width and height of the grid which are just two integer values.

5. Algorithm Description

There is nothing new to describe in this part because all solution algorithm is already discussed so here I just sum up all solutions that I came up with in this document. Solution to store a rule and set of the rules is in class Rule and RulesSet. Solution in order to solve conflicts in class RuleConflict and RuleConflictFrame mentioned and solved.

The class ruleEngine which meant to work as thread in different rules was our solution have better performance in the application. The selected solution between two solutions proposed before is to initialize each thread with small number of the rules in the way all conflicting rules will be in one thread to be applied.

6. Conclusion

here the version division of application will presented as follow:

* In first version all window frames will be created rule and rule set class RuleConflict class with merge and probability method and moving grid.
* RuleEngine class and RuleConflict class with suggestion method.
* The save and load will added to the application which requires database.