RW711
Cellular Automata Framework

Shaun Schreiber

March 10, 2014

Contents

1 Problem Statement

The purpose of this assignment is to design a framework that can simulate two dimensional cellular automata. This framework must be general enough such that it can handle any two dimensional cellular automaton. An implementation of Conway's Game of Life using this framework must also be handed in.

2 Design

The framework is divided into 3 sections, cell §2.1, lattice §2.2 and cellular automata §2.3.

2.1 Cell

The cell basis element of the framework. Each cell will define its own set of rules. This was done to ensure that hybrid cellular automata can be implemented on this framework. Each cell must also define what its neighbourhood is. This was done for experimental reasons. Each cell is only allowed to have one value. Each cell will determine its next value by applying its rules. Each cell will be able to return its current value, its next value and an array of indices, each index is not the actual index of each neighbour, but the offset from the current cell, identifying it neighbours.

2.2 Lattice

A lattice consists of a grid of cells. The framework will allow a lattice of any dimension. The lattice will be able to update itself, by iterating through the grid of cells and applying their individual rules. A lattice will be able to return an instance of itself, a cell given a certain index and its dimensions.

2.3 Cellular Automata

The task of this section is to simulate any give cellular automata that abides by the guide lines and functionality stated in sections §2.1 and §2.2. This section will be able to retrieve the next step[1] for a given lattice. It will be able to start and stop the simulation of a cellular automaton. It will include a mechanism that will allow other objects to attach to the current cellular automaton and be notified when the the current lattice has changed or updated. This mechanism must allow other objects to detach from the current cellular automaton. It will also return the current state of the lattice before an update is applied.

3 Implementation

Cell and lattice both specify a criteria to which some class must abide for the cellular automata abstract class to use it, because of this Cell and lattice where both made interfaces.

3.1 Cell

The interface of cell abides to all of the specifications set in the design section 2.1. This makes it easy to add new cells with different rules and neighbourhoods.

- `Getneigbours` returns a list of array indices. These indices are the offsets of the neighbours from the current position.

[1] Step here refers to the new lattice that is formed after each cell has determined their new value.

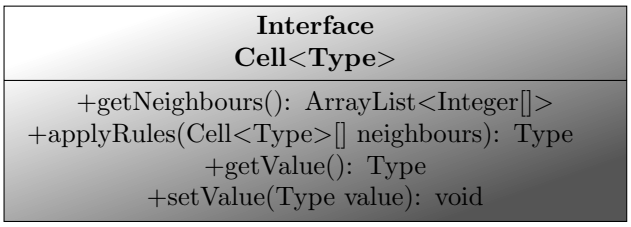

Figure 1: Cell class diagram

- `ApplyRules` applies the rules of the current cell and return its new value.
- `GetValue` returns the current value of the cell.
- `SetValue` stores the given value as the new value of the cell.

.

3.2 Lattice

The interface of cell abides to all of the specifications set in the design section 2.1. This makes it easy to add new cells with different rules and neighbourhoods.

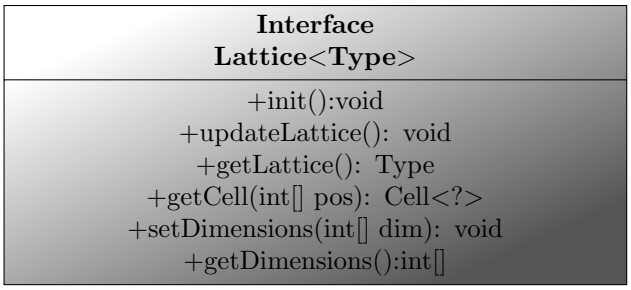

Figure 2: Lattice class diagram

- `Init` Initializes the current lattice.
- `UpdateLattice` Iterates through all the cells contained by the lattice and apply their rules to determine each cells new value. All of the new values are stored replacing the old values.
- `GetCell` returns the cell at the given position.
- `SetDimensions` stores the given dimensions as the new dimensions of the current lattice.
- `GetDimensions` returns the dimensions of the current lattice.

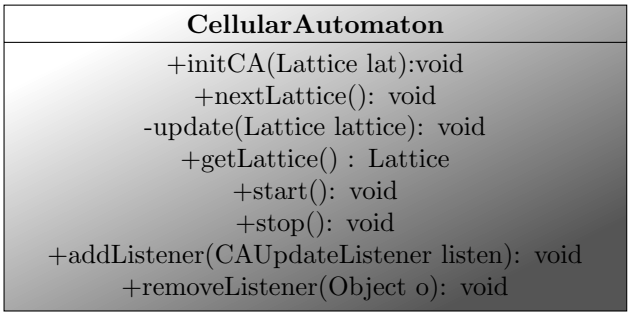

Figure 3: Cellular Automata abstract class diagram

3.3 Cellular Automata

- `InitCA` Initializes the lattice and set the isRunning boolean false.
- `NextLattice` Stores the new lattice after the current lattice was advanced by one tick.
- `Update` this function runs through a list of listeners and invokes each listeners update method.
- `Start` changes the value of isRunning to true thus allowing a nextLattice function call to update the lattice.
- `Stop` changes the value of isRunning to false thus stopping a nextLattice function call to update the lattice.
- `AddListener` adds a CAUpdateListener to the list of listeners.
- `removeListener` removes the first occurrence of given object from the list.

4 CAUpdateListener

Figure 4: CAUpdateListener class diagram

- `Update` is called when a lattice is successfully updated to the next tik.

4.1 Game Of Life(2D)

Conway's Game of Life was implemented with the basic rules, but making use of wrap around at the borders and not padding it with zero's. An additional function was added to the game of life lattice. This function was used to read a cellular automaton from a file.The GUI can display and simulate a given cellular automaton. Also the GUI can pause, edit and continue.

4.2 Game of Life(3D)

cellular automaton Conway's Game of Life was implemented with the rules B5/S4,5[2] and making use of wrap around at the borders and not padding it with zero's. An additional function was added to the 3D game of life lattice. This function was used to read a cellular automaton from a file. The GUI can only display and simulate a cellular automaton.

5 Testing(2D)

5.1 Test1

Reason: This testcase tests to see if the wrap around works.
input:
7 7
0000000
0111000
0001000
0010000
0000000
0000000
0000000
pass: True

5.2 Test2

Reason: This testcase tests will to see if the rules work properly.
input:
11 11
00000000000
00000000000
00000000000
00000000000
00001110000
00011011000
00001110000
00000000000
00000000000
00000000000
00000000000
pass: True

5.3 Test3

Reason: This testcase tests will to see if the rules are properly applied.
input:
3 3 000
000

[2]The B5 means that a dead cell will be revived if exactly 5 neighbours are alive an the S4,5 means the between 4 and 5 neighbours, inclusive, a living cell my stay alive.

000
pass: True

5.4 Test4

Reason: This testcase tests will to see if the rules are properly applied.
input:
3 3 111
111
111
pass: True

5.5 Test5

Reason: This testcase tests to see if the wrap around works.
input:
15 17
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000111100000000
0001000100000000
0000000100000000
0001001000000000
0000000000000000
0000000000000000
0000000000000000
pass: True

5.6 Test6

Reason: This testcase tests will to see if the rules work properly. This is a long repeating pattern.
input:
11 11
00000000000
00000000000
00000000000
00000000000
00001110000
00010000000
00010000000
00010000000
00000000000
00000000000

00000000000
pass: True

6 Testing(3D)

There are tests for the 3D game of life but these tests are fairly large thus are omitted from the document.

References

[1] jMonkeyEngine.org — jMonkeyEngine Community

In-text: (Hub.jmonkeyengine.org, 2014)

Bibliography: Hub.jmonkeyengine.org. 2014. jMonkeyEngine.org — jMonkeyEngine Community. [online] Available at: http://hub.jmonkeyengine.org/ [Accessed: 10 Mar 2014].