Instituto Politécnico Nacional

Escuela Superior de Cómputo

Evolutionary Computing

Laboratory session #09: Cellular Automaton (CA)

Student: Vargas Romero Erick Efrain

Professor: Rosas Trigueros Jorge Luis

Practice completion date: November 15, 2021

Report delivery date: November 15, 2021

1

Theoretical framework

1.1 Cellular Automaton

The *Cellular Automaton* (CA) are simple constructions that appeared during the 50s. *John von Neumann* (img. 1.1) tried to develop an abstract model of biological self-reproduction, this was a very discussed topic during that age. First of all, John von Neumann thought about 3-dimensional models which were described with partial differential equations, then he changed his point of view and he thought about robotics and imagined that would be possible implement an example using a building game [1].

Figure 1.1: John von Neumann

During 1951 suggested by *Stanislaw Ulam*, von Neumann simplified his model and finally he created a 2-dimensional CA. That CA was built in 1952 and it had 29 colors per cell and very complex rules, these rules allowed emulate the operations of computer components and other mechanic devices. To test mathematically the possibility of self-reproduction, von Neumann described the consturction of a configuration with 200,000 cells, they should reproduce themselves [1].

The CA are mathematical idealizations of natural systems. A CA has a discrete web of cells, each of them takes an integer finite value. The values of the cells are get in discrete steps according to deterministic rules which specifies the value of a cell in terms of the value of their neighbors. You can see an example below of the rule 110 discovered by Stephen Wolfram (img. 1.2).
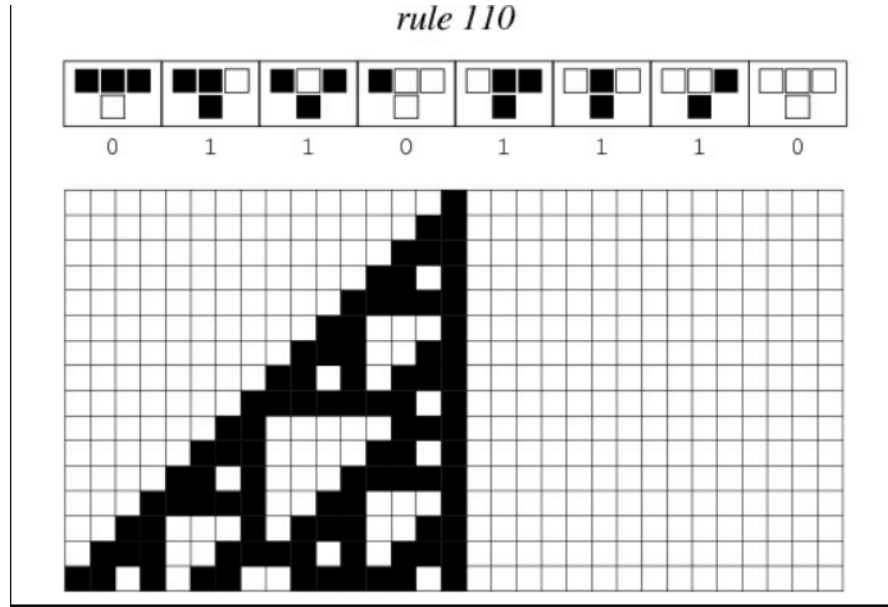

Figure 1.2: Rule 110

Be $\mathbb{Z}$ the set of integers and $\mathbb{Z}^+$ the set of positive integers, then the evolution space in one dimension is represented as a sucession of elements that will determine an 1-dimensional array; be c_i the $i-th$ position in the array where $i \in \mathbb{Z}$, each of that positions in the array is called cell and those cells take elements of Σ set, additionally $\Sigma \in \mathbb{Z}^+$. This set represents the quantity of elements of Σ. We called *configuration* to a set of cells. Wolfram represents the 1-dimensional CA with two parameters, k and r where k is the number of states of Σ and r is the number of neighbors of the central cell c_i. The set of neighbours to the left and to the right of c_i is denominated as *neighbour*. So summarizing, we can represent a CA as a Quadruple:

$$\Sigma, r, \Phi, C_i \qquad (1.1)$$

Where:

- Σ is the states set

- r is the number of neighbors of a central cell (c_i)

- Φ is the transition function

- C_i is the initial configuration

Is possible to known the number of states of Σ if we calculate $|\Sigma| = k$, therefore $k \in \mathbb{Z}^+$ and Σ is enumerable, it means, $\Sigma = 0, 1, \dots, k-1$

2

Material and equipment

The necessary material for this practice is:

- A computer with the latest *Python* stable version installed

- PyQt5 instlled with the target of create a GUI

- A text editor
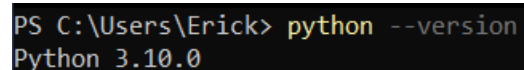
Or is possible to use the google site `https://colab.research.google.com/` that allows us to use a virtual machine with an *Ubuntu* operative system with *Python* installed.

3

Practice development

3.1   Cellular Automaton

To develop this practice we used *VisualStudio Code* as text editor, Python and PyQt5, this one is a library focused in create desktop applications. We can verify if we hace python installed in our machine using the next command:

```
python --version
```

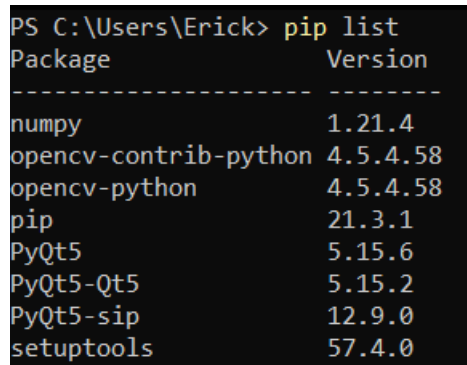If we run this command in our terminal using *PowerShell, CMD, bash, etc.* We must see something like this:

Figure 3.1: Verifying python version

And to check the packages installed we can use the next command:

```
pip list
```

Again if we run this command in our terminal we must see something like this:

```
PS C:\Users\Erick> pip list
Package                  Version
------------------------ --------
numpy                    1.21.4
opencv-contrib-python    4.5.4.58
opencv-python            4.5.4.58
pip                      21.3.1
PyQt5                    5.15.6
PyQt5-Qt5                5.15.2
PyQt5-sip                12.9.0
setuptools               57.4.0
```

Figure 3.2: Listing Python Packages

3.2 2-dimensional Cellular Automaton

Instructions: Consider a 2D CA defined as follows:

- S = infinite rectangular grid
- N = {Closest Neighbors} ∪ {self}
- Q = {0, 1, 2}

$\Phi = (b1, b2, s1, s2, u1, u2, r1, r2)$

- If the cell is in state 0, and the number of closest neighbors in state 1 is between b1 and b2 (b stands for born) the state of the cell will change to 1.
- If the cell is in state 1,
    - If the number of closest neighbors in state 1 is between s1 and s2 (s stands for survival) the state of the cell will remain in 1.
    - Else: If the number of closest neighbors in state 2 is between u1 and u2 (u stands for upgrade) the state of the cell will change to 2.
- If the cell is in state 2,
    - If the number of closest neighbors in state 2 is between r1 and r2 (r stands for remain), the state of the cell will remain in 2.
- Otherwise, the state of the cell will change to 0.

Find either an oscillator, a still life or a glider that visits all three states and uses in Φ at least two digits in your student number.

For this practice we build a software to simulate 2-dimensional CA, you can see the GUI in img. 3.3
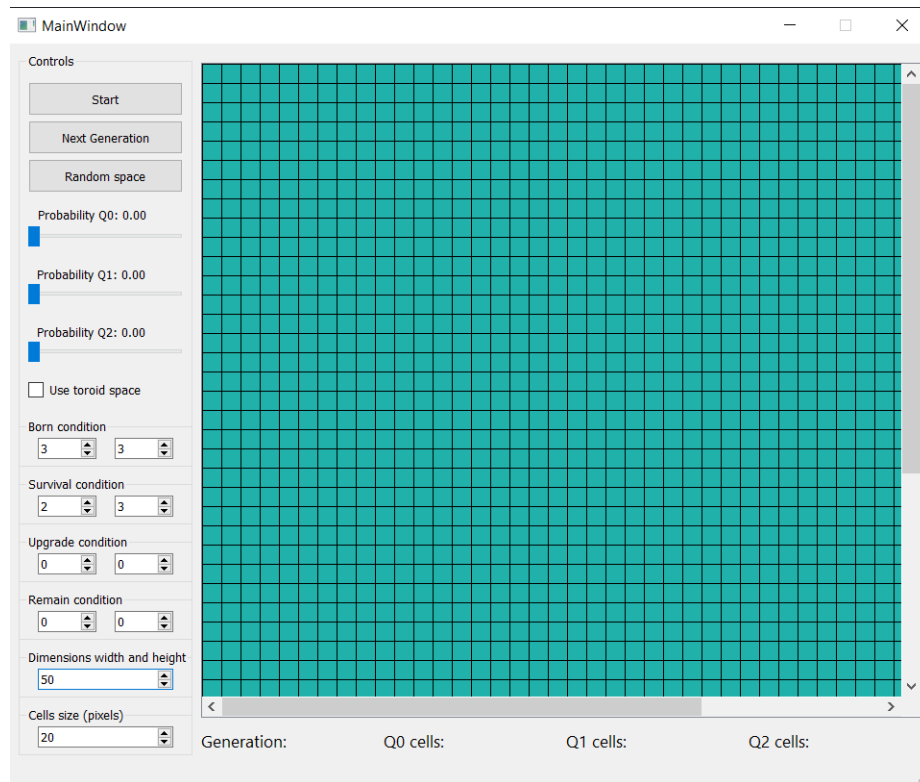
Figure 3.3: 2D CA Simulator

This software has so useful features:

1. Generate a random evolution space

2. Set the probability of a certain state appears

3. Use an open space during the simulation (cartesian plane)

4. Use a closed space during the simulation (toroid)

5. Set the transition list or conditions (values between 0 to 8 only)

6. Set the dimensions of the evolution space

7. Set the cells size

8. Display the current generation

9. Display the current number of Q0 cells

10. Display the current number of Q1 cells

11. Display the current number of Q2 cells

12. Get the next generation

13. Start the simulation (every generation is calculated every 0.1s)

14. Stop the simulation

15. Click in a cell to change the state (left and right click of your mouse.)

To try to find some interesting patterns we decided to set random rules and test with different densities in the Q0, Q1 and Q2 states generating random initial conditions. We find an interesting pattern using the rule $B3S23U12R1$ and my student id is 2016601770 so as you can see my student id and the rule share the digits 2 and 1. The patter found was an oscilator with the next behaviour:

First of all as you can see in img. 3.4 we require of the three states to build our oscilator, four cells of the state Q1, two of state Q2 and the rest of the evolution space are just of Q0 state.
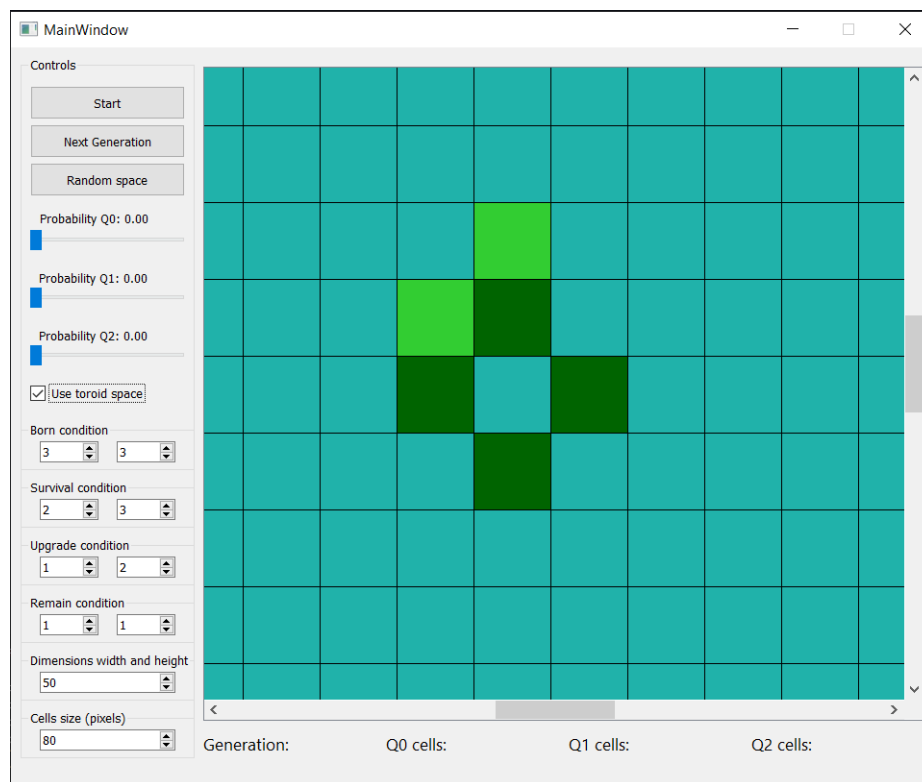

Figure 3.4: Oscilator generation 0

If we get the next generation the center of the oscilator changes by a state Q1 and the rest is the same img. 3.5.
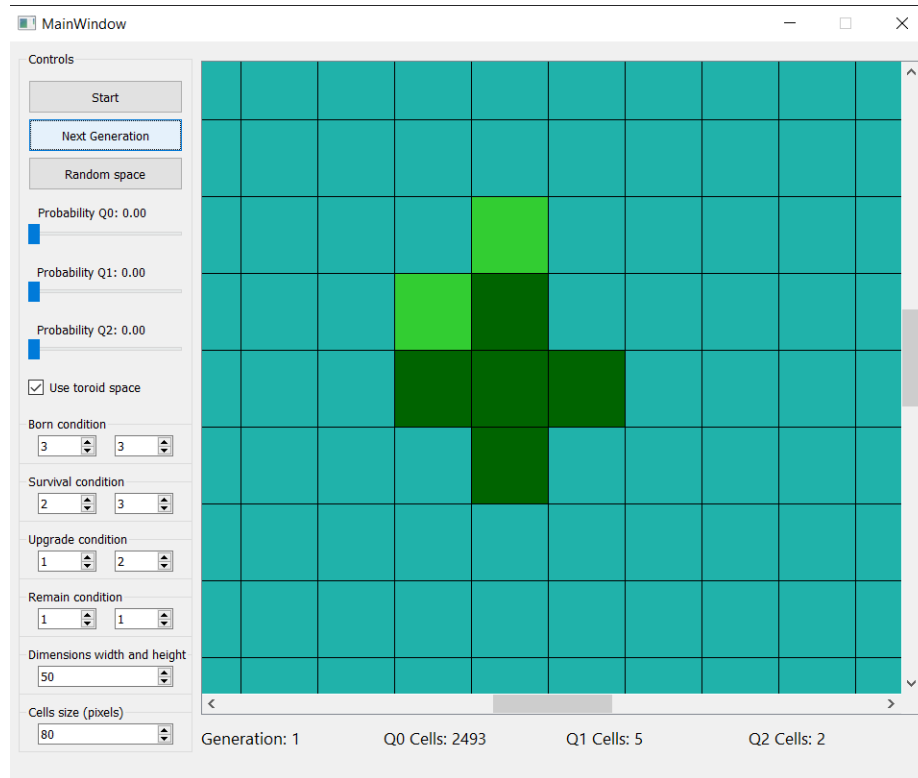

Figure 3.5: Oscilator generation 1

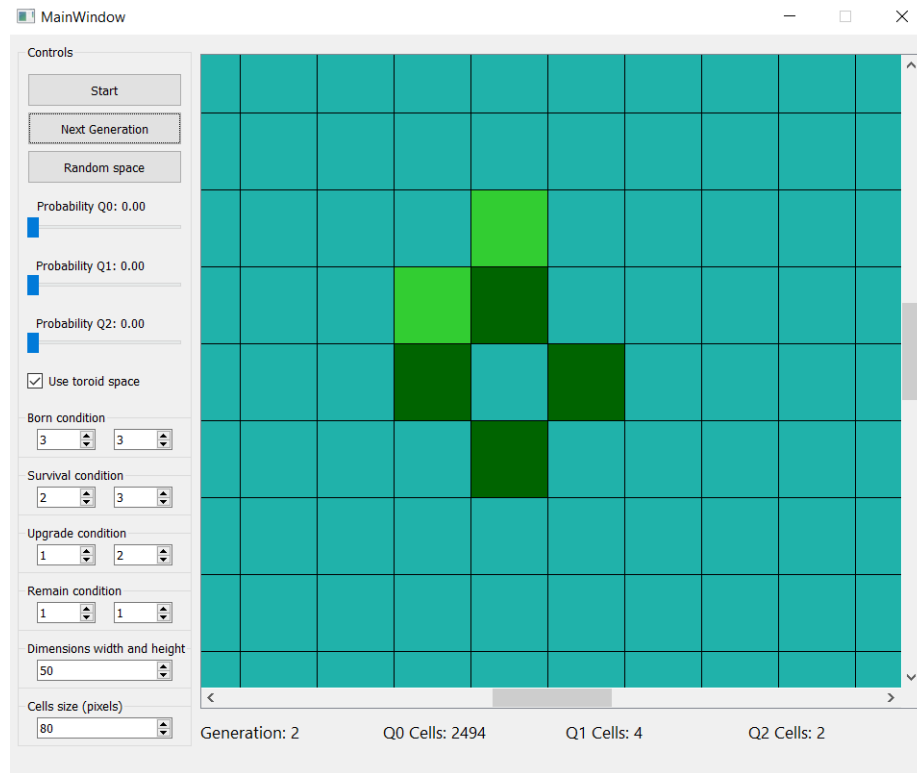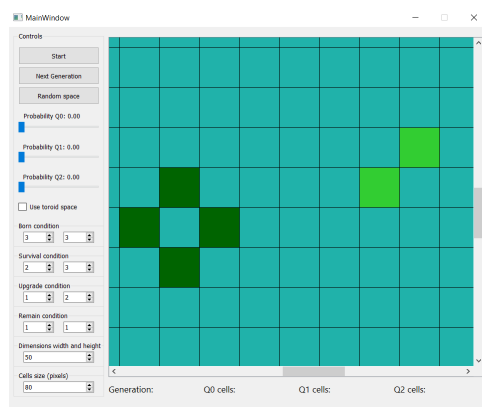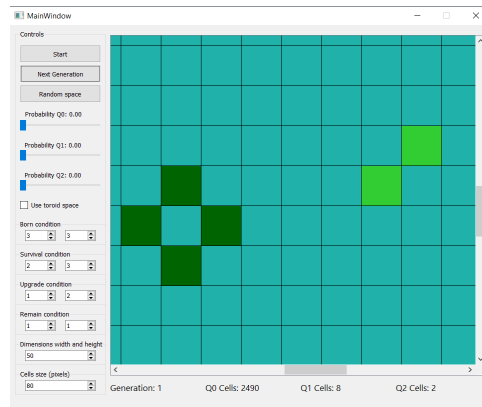Finally, if we move to the next generation we get the original configuration img. 3.6.

Figure 3.6: Oscilator generation 2

Something else interesting about this rule is that if you remove the Q2 cells the pattern changes to an still life and the same case if you remove the Q1 cells.

(a) Still lifes generation 0

(b) Still lifes generation 1

3.3 1-dimensional Cellular Automaton

Instructions: Consider a 1D CA defined as follows:

- S = Infinite 1-D grid

- N = {First 2 layers of closest neighbors} $\cup$ {self} ($|N| = 5$)

- $Q = \{0, 1\}$

Provide a graphical representation of the evolution of this CA with

- $\Phi = Wxy0$

- $\Phi = Wxy0R$

where x, y are nonzero digits from your student number. Provide evolutions that start with a small number of cells in 1 and with random configuration.

Similarly to the previous problem we built a simple software to simulate 1-dimensional Cellular Automaton, you can see the GUI in img. 3.7.
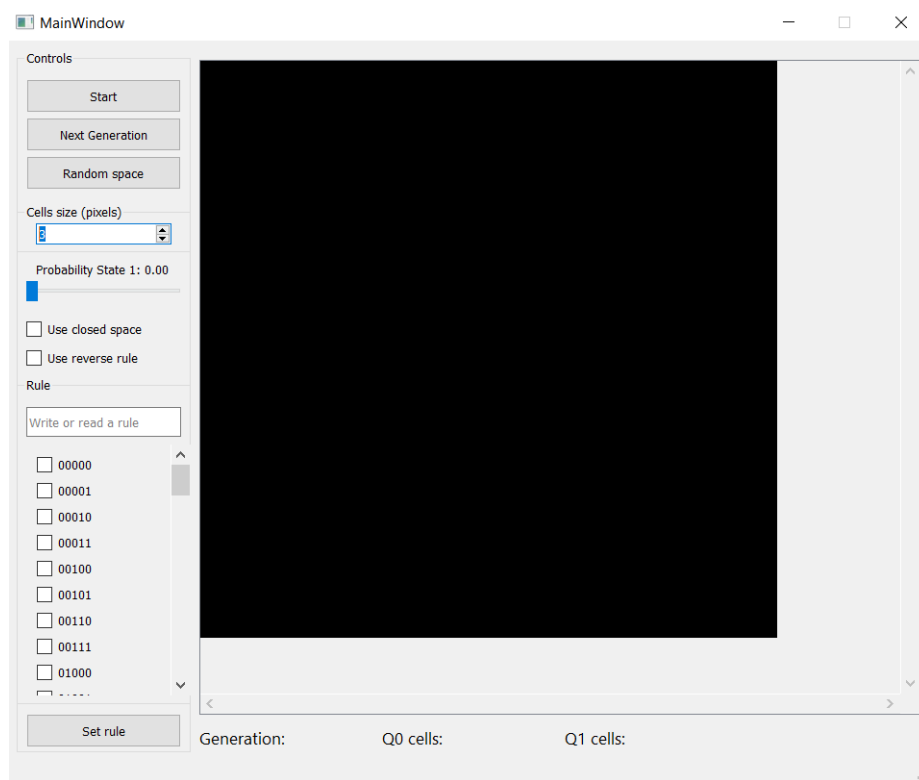

Figure 3.7: 1D CA Simulator

This software has the next features:

1. Generate a random evolution space

2. Set the probability of state 1 appearing

3. Use an open space during the simulation (like a tape)

4. Use a closed space during the simulation (like a ring

5. Set the transition rule using an integer

6. Set the transition rule choosing between the possible configurations available

7. Set the cells size

8. Display the current generation

9. Display the current number of Q0 cells

10. Display the current number of Q1 cells

11. Define as a reverse rule

This part of the practice is easier than the previous one, we choose as the test rule the whole id student (2016601770) and we found interesting behaviours. First of all if we sumulate this rule in a *"normal"* way we found some patters when some cells collide with others. To this experiment we defined a density of 70% of probability to appear for state Q1. And as you can appreciate in img. 3.8 most of the cells are Q1 and the interesting part comes when we run the simulation until the generation 199.
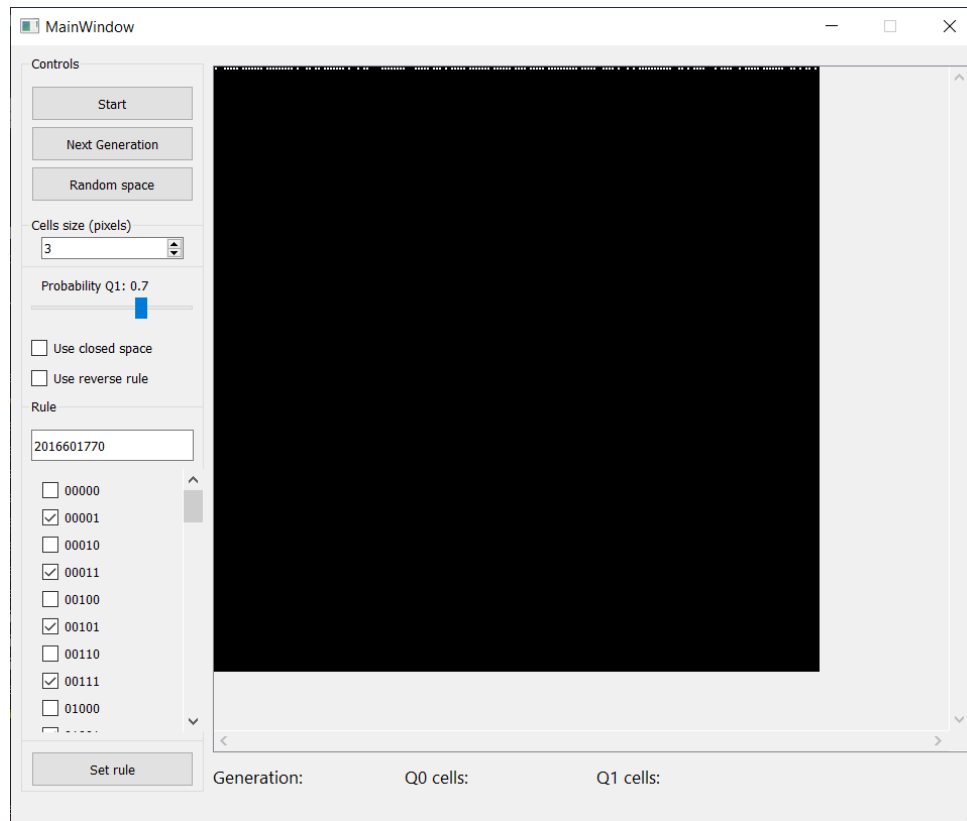
Figure 3.8: Rule 2016601770 Gen. 0

Now as you can see at the end of the simulation looks like we generate a 3D figure, at the begining we do not have a uniform behaviour looks like chaotic but at the end we have an uniform figure.
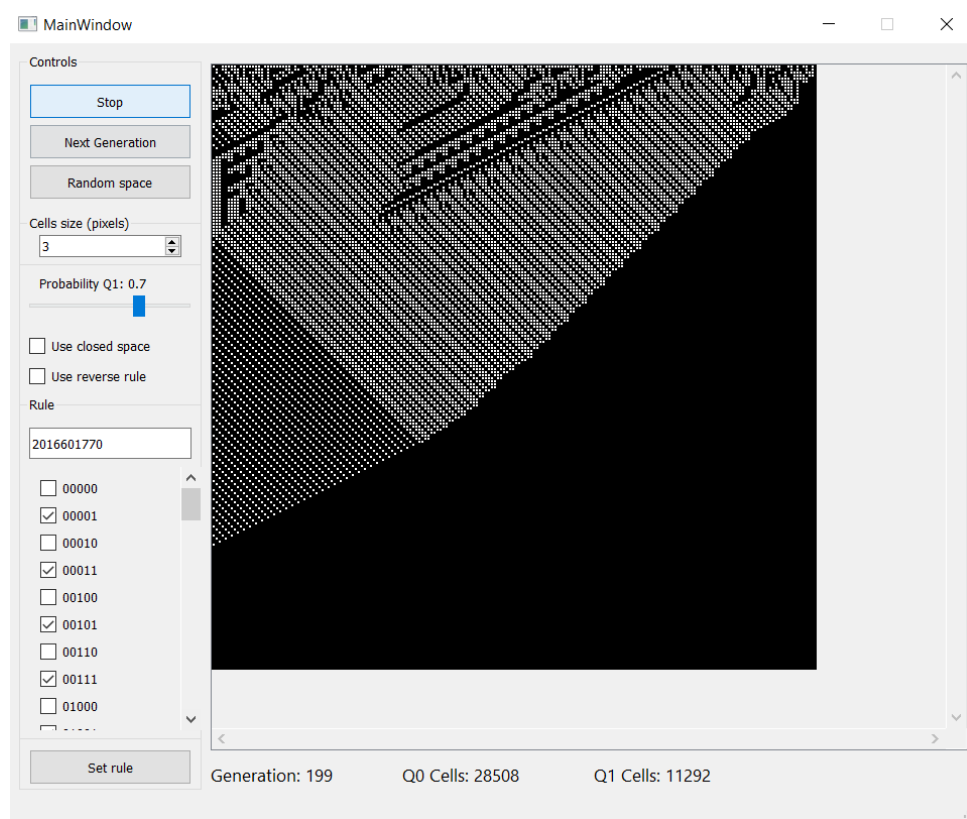
Figure 3.9: Rule 2016601770 Gen. 199

When we use the same rule but using $W2016601770R$ we find a completely different behaviour. In this case instead of use a high probability in Q1 we only used a cell as you can see in img 3.10.
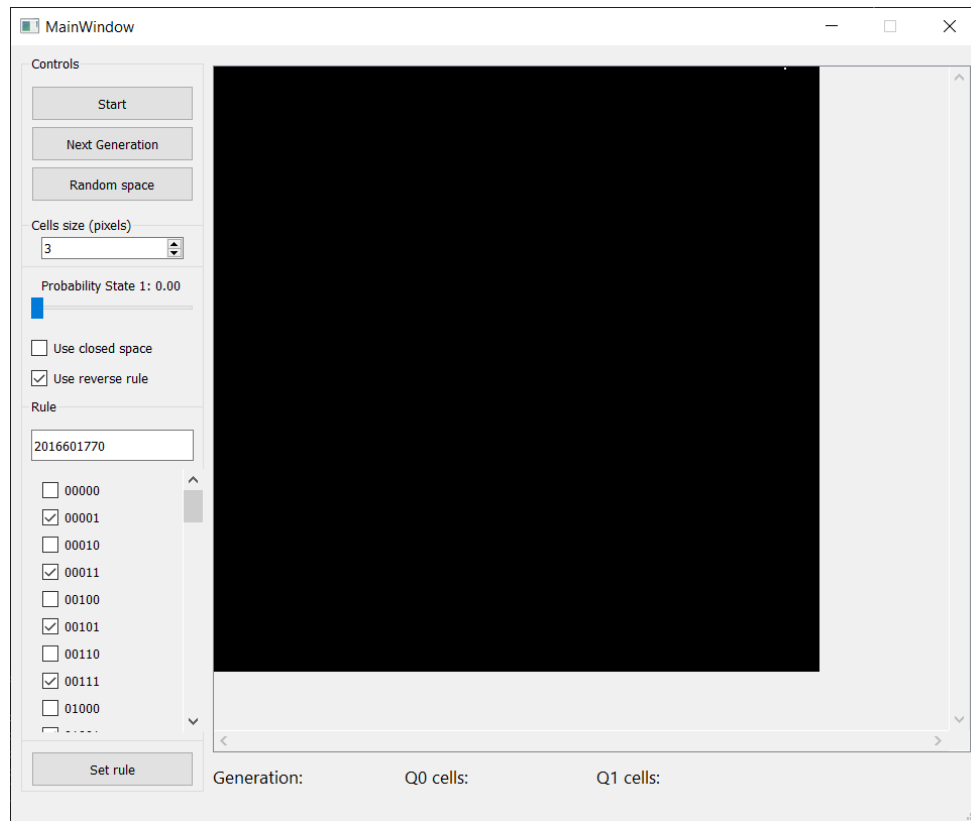
Figure 3.10: Rule 2016601770 Gen. 0

And if we run the simulation we discovered an interesting pattern formed by only triangles, to be more accurate we found a fractal.
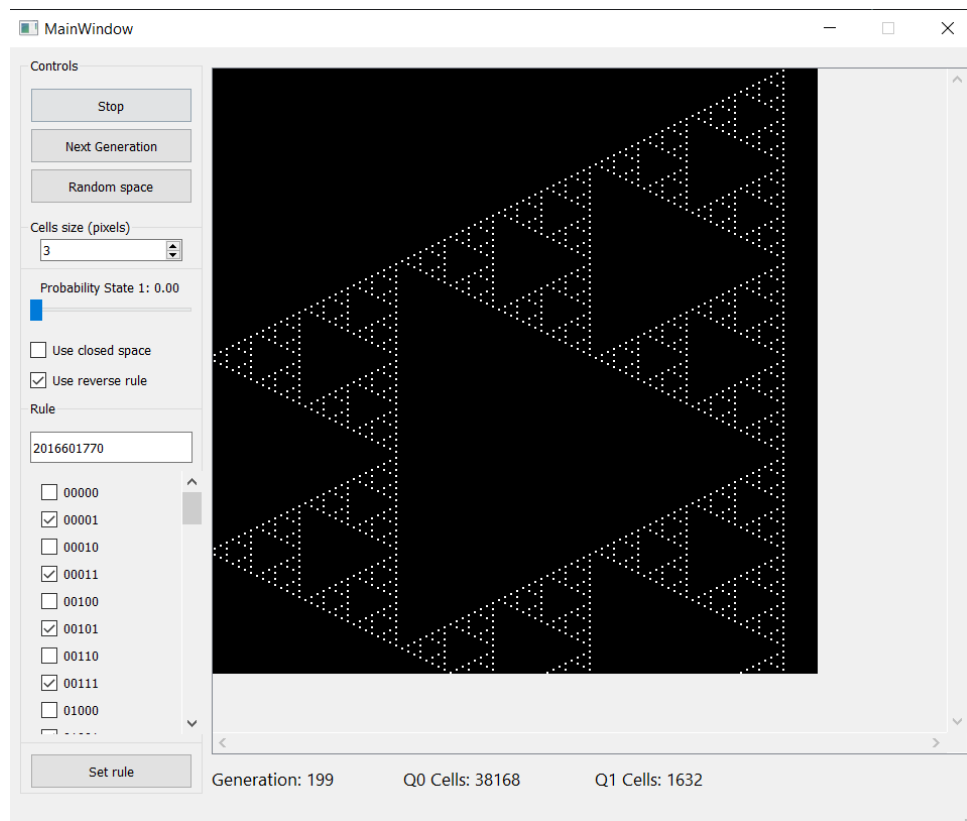
Figure 3.11: Rule 2016601770 Gen. 199

Finally, if we use a high density of cells in the initial configuration we see like a chaotic rule that is why we decided to try using only a cell. As you can see here we used a density of 70% instead of a cell and we can not see a pattern or figure like in the previous case.
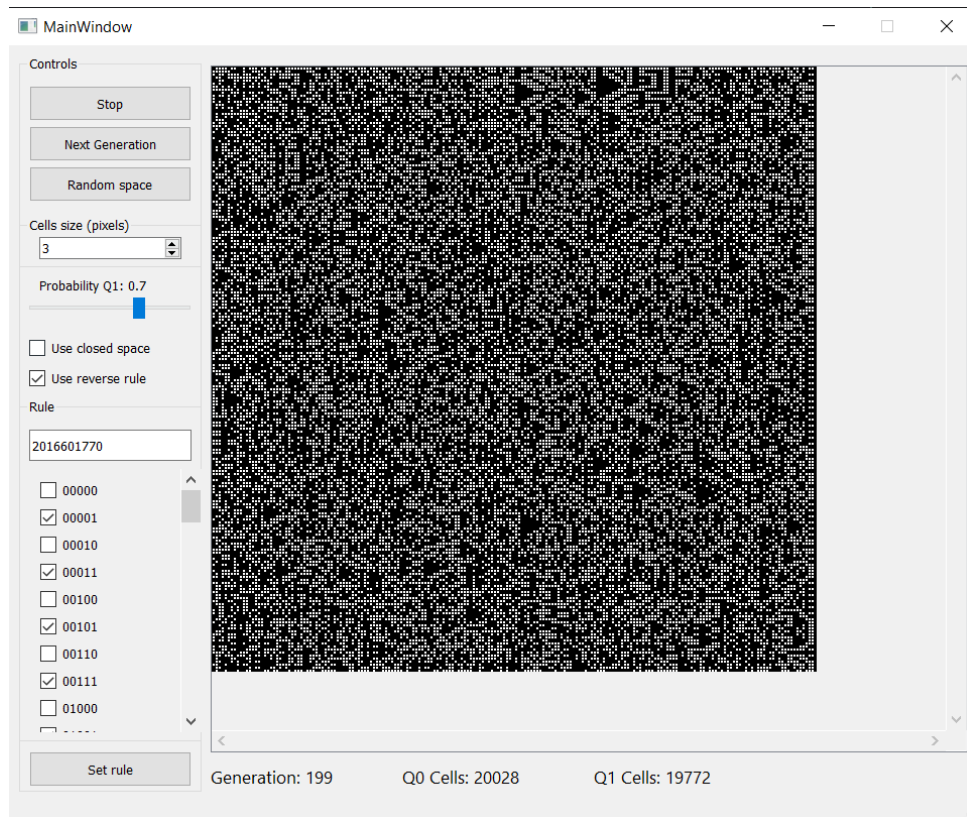
Figure 3.12: Rule 2016601770 Gen. 199 High density

4

Conclusions

This practice has been very interesting first of all because I consider that cellular automaton (CA) are very powerful tools to analize different things like, natural behaviours. Something really amazing about CA is the possibility of find incredible and interesting patterns, as we saw in the *Practice development* we discover a fractal with the rule 2016601770 using an elemental CA but increasing the neighbour from only the left and right neighbors to take two neighbors to each side. Try to find a rule or configuration that describes somethig is complicated because we have a lot of possibilities, we can create different rules and different initial conditions, not only using a small group of cells, but we can create a very complex initial configuration that builts something incredible. This practice helped me a lot to improve my skills using Python because I learnt how to build graphical applications using PyQT5 and is necessary to mention that I already used this library but with $C++$.

Bibliography

[1] A. Esquivel Valdez, E. Torres Hernández and E. E. Vargas Romero *Exploración de universos discretos complejos con autómatas celulares en dos dimensiones*. Accessed on November 14th, 2021. Available online: `http://www.comunidad.escom.ipn.mx/genaro/Papers/Thesis_files/tesisEricEduardoAlbertoTT2019-B067.pdf`