UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Desarrollo de entorno on line de programación para computación natural

Máster Universitario en Investigación e Innovación en Tecnologías de la Información y las Comunicaciones (i2-TIC)

Autor: SAMI NAYYEF AL-DABBAGH, Bashar

Tutor: ORTEGA DE LA PUENTE, Alfonso

Departamento de Escuela Politécnica Superior

FECHA: 12, 2019

DESARROLLO DE ENTORNO ON LINE DE PROGRAMACIÓN PARA COMPUTACIÓN NATURAL

Approved by:		
Dr. David Camacho, Advisor School of EPS Ingeniería Informática UAM *Universidad Autónoma de Madrid*		Dr. Miguel Ángel Mora School of EPS Ingeniería Informática UAM *Universidad Autónoma de Madrid*
Dr. Marina de la Cruz School of UCM Ingeniería Informática y CIEMAT *Universidad Complutense de Madrid*		Dr. Francisco Saiz EPS Ingeniería Informática UAM *Universidad Autónoma de Madrid*
Dr. Sandra Gómez Canaval School of Departamento de Sistemas Informáticos ETSII UPM *Universidad Politécnica de Madrid*		Dr. Rafael Lahoz Beltra Matemática Aplicada UCM *Universidad Complutense de Madrid*
		Date Approved: Dec. 02 , 2019

ACKNOWLEDGEMENTS

I am overwhelmed in all humbleness and gratefulness to acknowledge my depth to all those who have helped me to put these ideas, well above the level of simplicity and into something concrete.

I would like to express my special thanks of gratitude to my Adviser who gave me a golden opportunity to do this wonderful project on the topic "DESARROLLO DE ENTORNO ON LINE DE PROGRAMACIÓN PARA COMPUTACIÓN NATURAL", which also helped me in doing a lot of Research and I came to know about so many new things. I am really thankful to him.

Any attempt at any level can’t be satisfactorily completed without the support and guidance of my parents, my wife, brothers and friends.

I would like to thank my parents who helped me a lot in gathering guiding me from time to time in making this project, despite their busy schedules, they gave me different ideas in making this project unique.

Thanking you,

Bashar Sami Nayyef Al-Dabbagh

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This work proposes a natural computer programming (for CA and NEPs) environment platform using Blockly. The platform is a web-based tool that provides simulators for two well-known natural computing systems: Cellular Automata (CA) and Network of Evolutionary Processors (NEPs). CA programming blocks presented in this work provide the ability to design and implement several types of CA including Elementary cellular automata, 2D cellular automata, and nD cellular automata. The tool also provides a graphical representation of CA's grid through projection for any CA that has 3 or more dimensions. A NEPs Blockly programming environment is presented in this work. It provides the ability to design and simulate NEPs. Blocks are used as flexible user interface to enter NEPs specifications. The blocks automatically generate a standard XML configurations code which can be sent to the server side of the simulator for implementation. The tool also provides a graphical representation for the static topology of the system.

Both CA and NEPs Blockly programming environments have been tested in several rather academic examples. The work presents an online simulation platform for natural computing algorithm using visual programing tool, namely Blockly. The proposed platform provides software engineering tools for setting up algorithms and also ease of use especially for teaching of these algorithm. The software engineering tools has been implemented on the NEPs as there is much more software tools already presented for cellular automata. The software designed for NEPs are a set of blocks to implement several types of connections between nodes. These blocks reduce time and complexity in setting up NEPs with fully connected nodes, for instance. In the other hand, cellular automata algorithm has been chosen to test the ease of the process of teaching and learning natural computing algorithms as they are much better-known model. The test has been conducted with students, teachers and researchers. Results of the experiment showed that the CA Blockly simulator outperforms traditional manual methods of implementing CA. It also showed that the proposed environment has desired features such as ease of use and decreases learning time. The NEPs part of the system has been tested against several applications. It showed that it provides a flexible designing tool for NEPs. It outperforms traditional XML coding methods in terms of ease of use and designing time. In addition we have designed specific high level constructs that automatize in some way the specific of complex NEPs' topologies by hand. They could be considered as embryonic software engineering tools to program NEPs.

Our tool is considered a generic platform for web-based implementation. It has desired features and wide range of properties that could attract the scientific community to adapt and develop in the future.

CHAPTER 1. INTRODUCTION

1.1 Motivations

Natural inspired computing has been attracting interests in the last decades. Several models and algorithms have been presented. Natural inspired computing has several advantages over conventional computing such as parallel processing of data, resiliency, and low power consumption. Network of Evolutionary Processors (NEPs) is one of the promising natural inspired algorithms [1, 2]. A NEP consists of a number of cells connected via links. The cells represent the processing units (nodes) in the system while the links are used to transfer data between these processing units. Each node contains a series of words that represent its DNA in the natural analogy. Words are processed by implementing simple operations such as insertions and deletion of symbols from the words and transmitting and receiving words to/from other nodes. NEPs are characterized by their simple structure, inherent parallel processing of data, and adaptation to solve several problems. Several versions of NEPs have been proposed [3] which have been successful applied on several problems [4]. For instance, NEPs show to be more adequate at processing NP-problems compared to conventional computing [5]. Several implementations of NEPs have been presented [6-8].

Cellular automata are another class of natural inspired computing models. They consist of a grid of cells that change their state based on pre-specified rules through a number of discrete time steps [9]. The rules are built to set the new state of a given cell based on the previous state of that cell and the neighboring cells. The rules are applied to the grid iteratively as many times as desired but to the complete grid simultaneously. Cellular automata were introduced by Von Neuman and Stanislaw Ulam in the 1940s [10]. Cellular automata have several applications in computer processors, cryptography, error correction code and more [11]. Due to its relatively early introduction compared to NEPs, cellular automata have large number of implementations.

NEPs and cellular automata are presented here in the context of developing the motivations of this research. Detailed introduction of both NEPs and cellular automata is presented in the next chapter.

Almost all of NEPs implementations take complicated syntax specific inputs inform of text. Some implementations use XML format to describe the executed NEP while others use custom designed P-lingua. Using such types of input, users are expected to learn the syntax and grammar of the input file. This leads to difficult experience for expected users and add extra requirement in learning this technology. At the same time, most of the implementations for cellular automata are not easy to access or learn.

Visual Programing Languages (VPLs) could be an alternative solution as input mechanism to NEPs. VPLs is similar to Lego bricks where programing statements is represented as blocks that can be connected in a prespecified way [12]. An example of VPLs is Scratch. It is blocks based visual programming language that has been mainly used for educational purposes [13]. Another example is Blockly which is an open source library presented by Google [14]. Blockly is a client-side visual code editor which has been used for several web and mobile applications including education, games, and others. In [15], Blockly has been utilized as a visual programming technique for multi-agent systems where the validity of using Blockly has been successfully demonstrated.

In this project, we propose an implementation for NEPs and cellular automata using Blockly. The proposed implementation is a web-based application that is free and easy to use. It is explained in detail in the rest of this document.

The hypothesis of this project is that a development tool for natural computing, accessible online and based on block languages and offering an effective and effective connection with simulators of the different paradigms should popularize the use of these computer models. The main objective of this project is to extend the current prototype by completing its functionality in the following aspects

1.2 Objectives and approach

This project reduces the complexity of using natural computing. It also makes it available online for ease of access. It focuses on simplifying the use and access of NEPs and cellular automata using VLPs. The objectives of this work can be summarized as:

- Reduces the complexity of using natural computing algorithms, specifically NEPs and cellular automata, by utilizing Blockly tool.
- Produce easy to access online simulator. If the simulators were executed in a powerful server, it might lead to reduce required execution resources for the user.
- The new introduced simulator might reduce the execution time if a powerful server is used.

The above three points are ordered based on importance which means that our main goal is ease learning and access for natural computing. Other points can be considered as secondary aims

CHAPTER 2. BACKGROUND AND RELATED WORK

2.1 Block-based programing

Block-based programing or coding is a visual methodology to write script/code in which code texts is represented as blocks [16]. Each block is analogues to a specific task, statement, or function in the text-based programing. Figure 1 shows an example of block-based programing platform [16]. The blocks are distinguished by their names, color, and shapes. Building (writing) a program is accomplished by connected a number of blocks on an order based on the purpose of the program. This code development has gained attraction especially for learning purposed. Several block-based programing platforms have been presented including Blockly [14], Scrach [16], and PencilCode [17]. These platforms have been widely adopted especially by the learning communities. They provide an easy to learn methodology to learn coding while cutting the learning curve. For Instance, AppInverter platform has been used by more than one million unique users in one month from 195 countries [18]. By the time of writing this document Scratch platform has been used by more than 46 million registered users [19]. Block-based programming has become a widely implemented approach for successful without requiring considerable experience. This approach also represents an introduction to programming in an accessible manner [20-22].

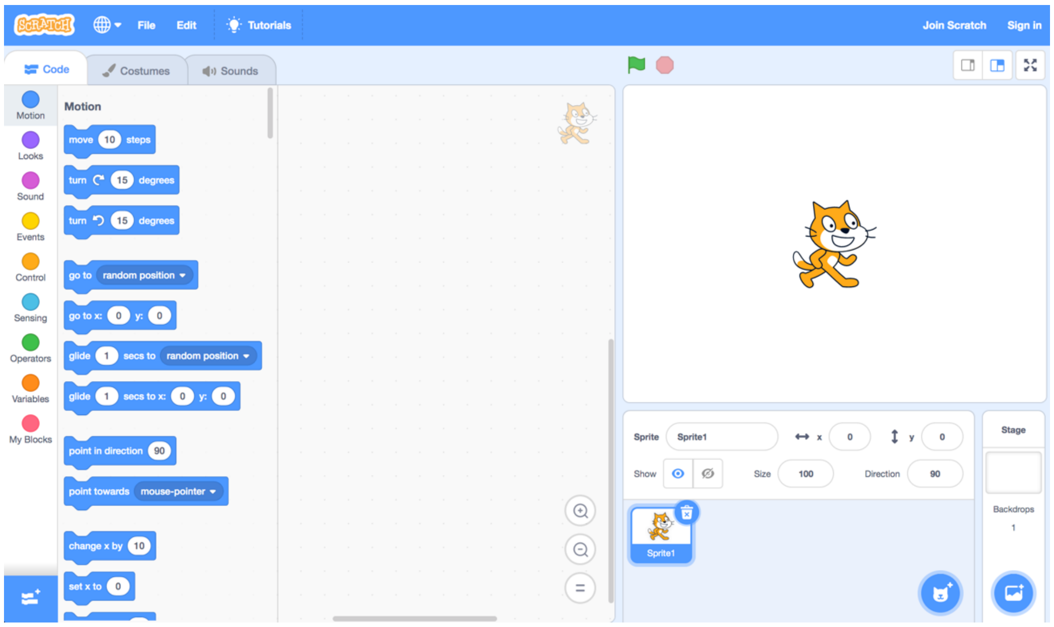
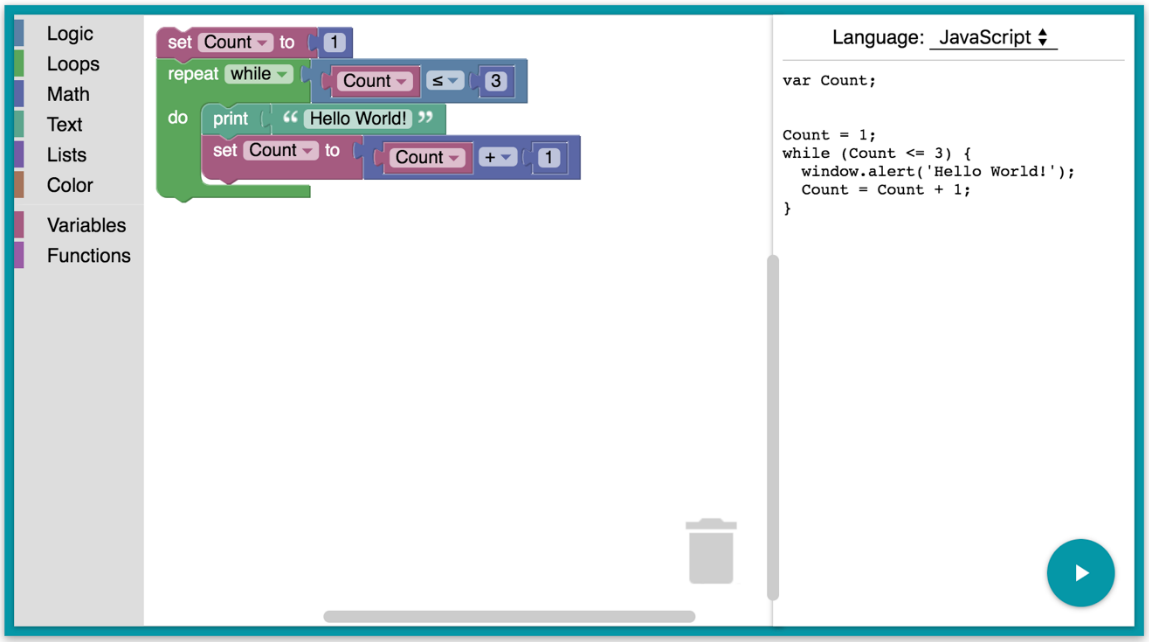


Figure 1: Example of block-based programming [16].

2.1.1 Blockly

Blockly is a block-based code editor library which can be integrated in web or phone apps [14]. It is an open source project originally developed by google. Like other similar platforms, it used interconnected graphical blocks to represent code concept such as variable definition, loops, conditional statement etc. It provides the ability to develop code by moving and connecting blocks and without the need for writing scripts. Figure 2 shows a screenshot of a code editor developed based on Blockly. In this example blocks are used to develop a simple program that type the sentence “Hello World” for a number of times. As can be seen in this figure, the blocks are interpreted into JavaScript code. In addition, this code editor can translate the blocks into several other languages such as Python, PHP, Lua, and Dart.

The Blockly platform provides several interesting features such as: uniform JavaScript open source code, it is completely client-side platform and does not requires any server-side processing. In addition, it is compatible with all major web browser and it can be customized and extended.



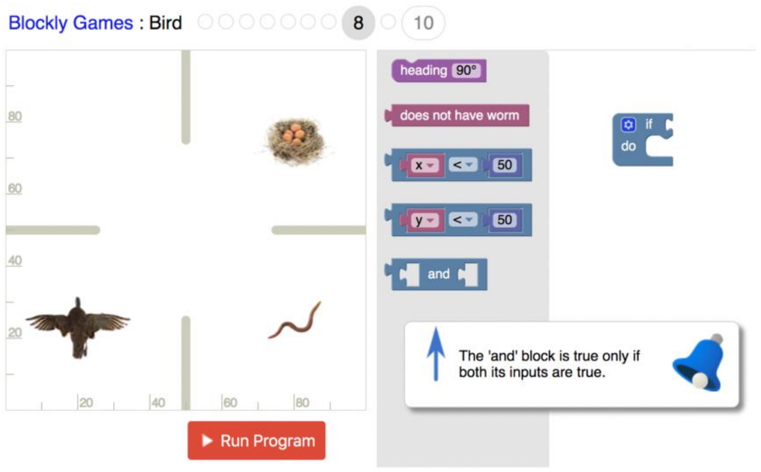
```
var Count;


Count = 1;
while (Count <= 3) {
  window.alert('Hello World!');
  Count = Count + 1;
}
```

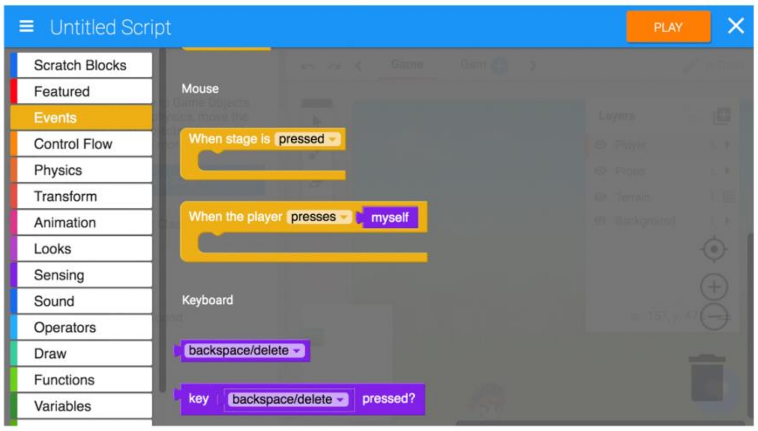
Figure 2: A screenshot of a demo application for Blockly [14].

2.1.2 Examples of Projects based on Blockly

All these features made it one of the most used block-based programing libraries. For instance, Code.org built off of Blockly platform a code editor for bother learning coding and developing apps [15]. Makewonder.com [23] uses Blockly platform for robot programing. Gamefroot.com uses Blockly for both building and running games [24]. Figure 3 shows screenshots of two projects built based on Blockly platform. The first is “blockly.games” [14], which is a website that provides several games which are played by using blocks. The second is “gamefroot.com” [24], which takes Blockly to the next level to not only playing games but also developing games.



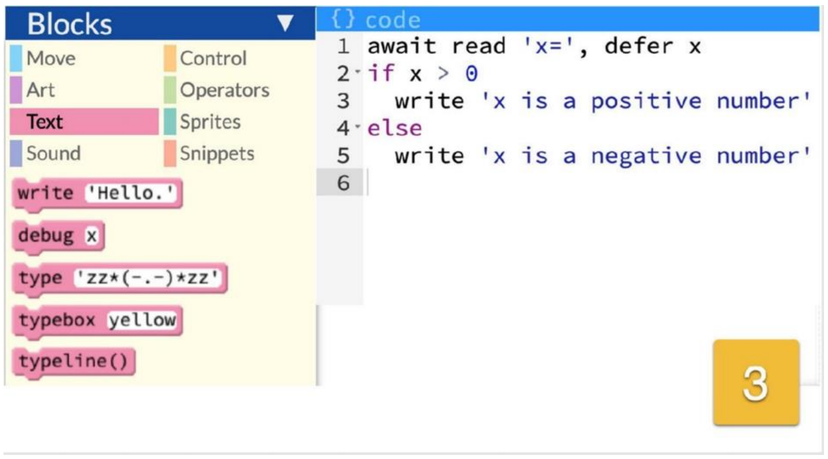
https://blockly.games/



https://make.gamefroot.com/

Figure 3: Projects build based on Blockly platform. The first example is from "blockly.games" [14] and the second is from "make.gamefroot.com" [24].

Several searchers have used Blockly to simplify programing and provide easy tools to teach programing. For example, Alrubaye has used mixed of visual and text tool to teach programing [25-26]. Figure 4 shows a screenshot of this platform. The visual tool uses Blockly while the blocks are instantly shown to the user as translated code. Results of this work showed that students can learn faster using this platform compared to text only.



```
await read 'x=', defer x
if x > 0
  write 'x is a positive number'
else
  write 'x is a negative number'
```

Figure 4: Mixed (blocks and text) platform to learn programing built based on Blockly [26].

In addition, visual programing tools have been used in several applications such as Alice [27] and AgentCubes [28] which help the user to program three-dimension simulations. Other projects have used blocks based programing to build modeling and simulation tools [29-31], playing video games [34], [35], manipulating media [36], as well as mobile application development [32], [33]. Moreover, an increasing number of tools and libraries have been developed to ensure that new block-based languages or embed block-based programming interfaces can be easily developed in existing applications [37], [38].

2.1.3 End User Programming

End user programing is writing application level scripts that simplify specific task within the application [39]. Results of this script is intended to be personal and specified for one task. For example, writing a script in Photoshop to apply filters on pictures with pre-determined parameters is considered as an example of end user programing. Another example is writing a script for processing data in a spreadsheet to calculate statistics. The syntax of such programing is usually application specific. For this reason, the syntax could be complex for the user who are not specialized in programing. Using block-based programing can simplifies the end user programing where users are not required to learn any programing skills or memories syntax.

In the context of this work, end user programing is analogues to the settings or parameters of the applications (Cellular automata and NEPs). For instance, some NEPs simulation uses XML files to set its parameters (refer to the next sections for more details). This file is written in a very specific form that is predefined and understood by the core of the simulators. Since the simulation platform proposed in this work uses block based programing, the end user programing will be highly simplified which leads to better user experience.

2.2 Cellular Automata

Cellular automata can be considered as a class of natural computing algorithm. It consists of a grid of cells each with their states. The states of the cells are changed based on prespecified rules. Several types of cellular automata have been presented in the literature such as elementary cellular automata and game of life [9]. Some of these types are presented in the next subsections.

2.2.1 Elementary Cellular Automata

The simplest cellular automata family presented is "The Elementary Cellular Automata" [9]. It is composed by a nontrivial linear set of cellular automata whose evolution is shown on 2D grids. It simulated line by line base and each cell has only two states (1 or 0). The new state of a cell on one line depends on the state of the cell in the previous line and the two adjacent cells in each side. The number of different options for the states of the cell and the two neighbor cells is $2^3 = 8$. This means that there are 8 possible patterns for the neighborhood. Figure 6 shows a number of examples of rules. The ideal is to set the cell in the current row to one if the status of the three cells in the row above matches the rule been implemented. For instance, in rule 30, the cell will be changed to one on only four cases as can be seen in Figure 6.

The number of rules of this kind of cellular automata is thus, $2^8 = 256$. This means that there are 256 different rules for the 3 neighborhoods. Tens of papers analyzing the behavior of these rules have been presented [9] [11]. Some of the rules, for instance rules showed interesting behaviors. Figure 5shows the behaviors of some of them.

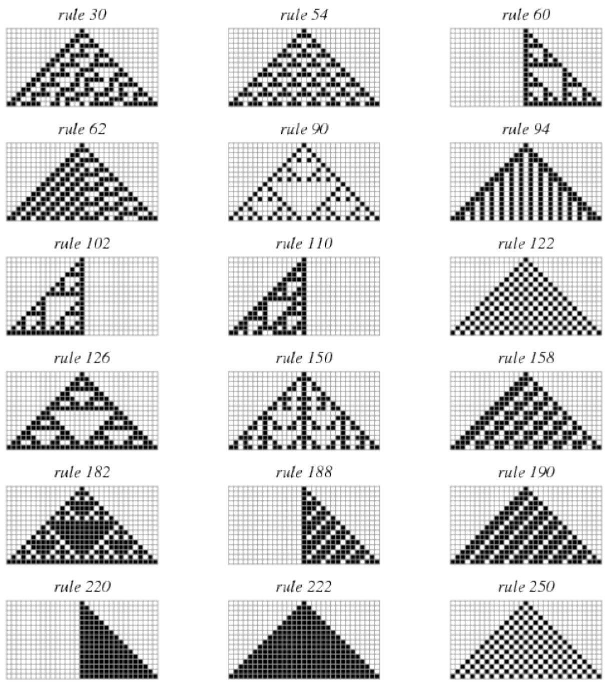


Figure 5:Examples of Elementary Cellular Automata. The rules associated with these examples are shown in [9].

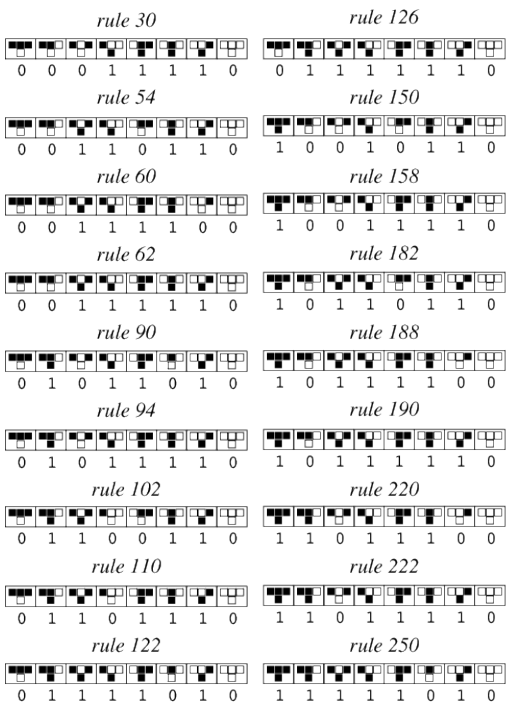


Figure 6: Example of Elementary Cellular Automata rules [9].

2.2.2 Game of Life

Game of life is a special version of cellular automata proposed by the British mathematician John Horton Conway in the 1970s [40]. It is a two dimensional cellular automata where each cell has only two state “live” or “dead”. This version of the cellular automata has no inputs except the initial states of the cells. The states of the cells evolve based on predetermined sold rules. Unlike the Elementary Cellular Automata, the game of life runs applies its rules on two dimensions. This means that the rules are applied on all cells in each time step. The rules are built considering that each cell has eight neighbors. The game of life has four main rules as follows:

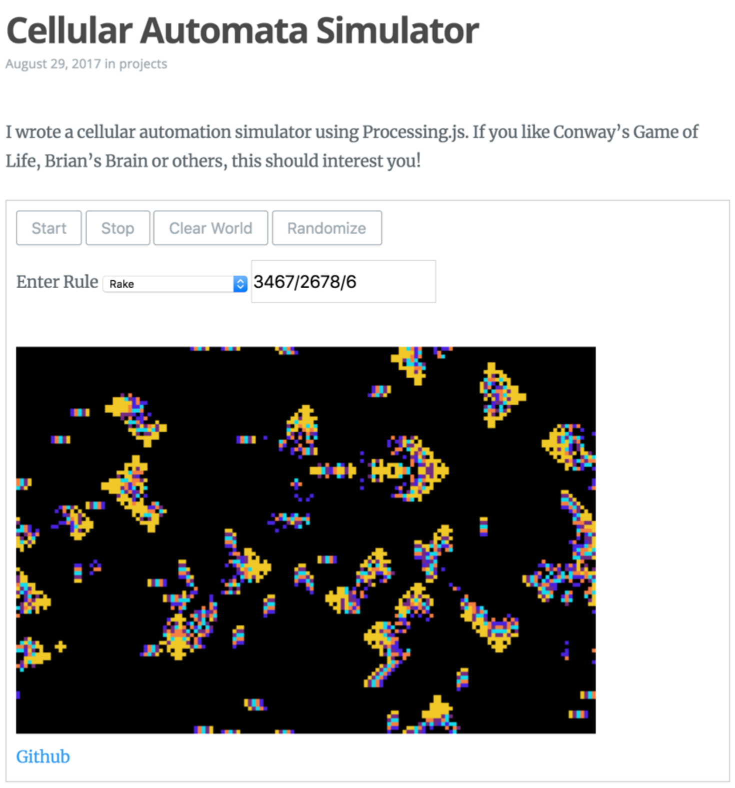
- Any live cell with fewer than two live neighbors dies.
- Any live cell with two or three live neighbors’ lives for the next generation.
- Any live cell with more than three live neighbors dies.
- Any dead cell with exactly three live neighbors’ lives.

The game of life should start from a basic configuration of the cells. Based on the configuration, the behavior of the cells in the grid will be changed. Several patterns of behavior noticed such as: Still lives, Oscillation, and Spaceship. This Wikipedia page shows life demo of some of these patterns [41].

2.2.3 Examples of Cellular Automata Simulators

Tens of cellular automata simulators have been presented throughout the past decades. Most of these simulators are available online for free. This section introduces some of these simulators while comparing their specifications with the proposed work. The comparison will be built based on a number of factors which are related for instance ease of use, covered types of cellular automata, and integration with other computational algorithms. The simulators presented here are selected based on a Google search for the keywords “cellular automata simulators”. Next is evaluation of the first three results in this search.

The first is an online open source simulator [42]. A screenshot of the website is presented in Figure 7. The simulator provides colorful animation of cellular automata grid. It also provides easy to use interface. On the other hand, it covers only one cellular automata type which is game of life. Users are allowed to choose only specific cases of this types. The tool does not provide the ability for the users to design their own rules.

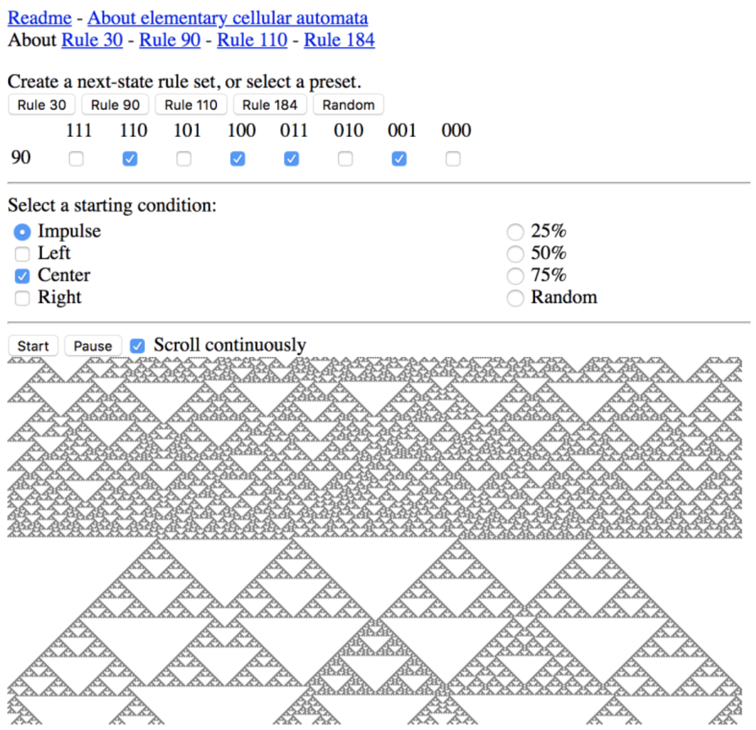

Cellular Automata Simulator

August 29, 2017 in projects

I wrote a cellular automation simulator using Processing.js. If you like Conway's Game of Life, Brian's Brain or others, this should interest you!

Figure 7: A cellular automata simulator as presented in [42].

The second simulator is presented in [43]. This simulator provides clean and easy to use interface with animation of the cellular automata grid as shown in Figure 8. However, it covers only the elementary cellular automata. Users are allowed to only choose one of the 256 rules of this kind of cellular automata. They are also provided with tools to manage the animation canvas. At the time this simulator can be considered a simple starting point for users who are interested in learning cellular automata, it does not provide flexibility for the users to develop their own cellular automata models.



Readme - About elementary cellular automata
About Rule 30 - Rule 90 - Rule 110 - Rule 184

Create a next-state rule set, or select a preset.

	111	110	101	100	011	010	001	000
90	☐	✓	☐	✓	✓	☐	✓	☐

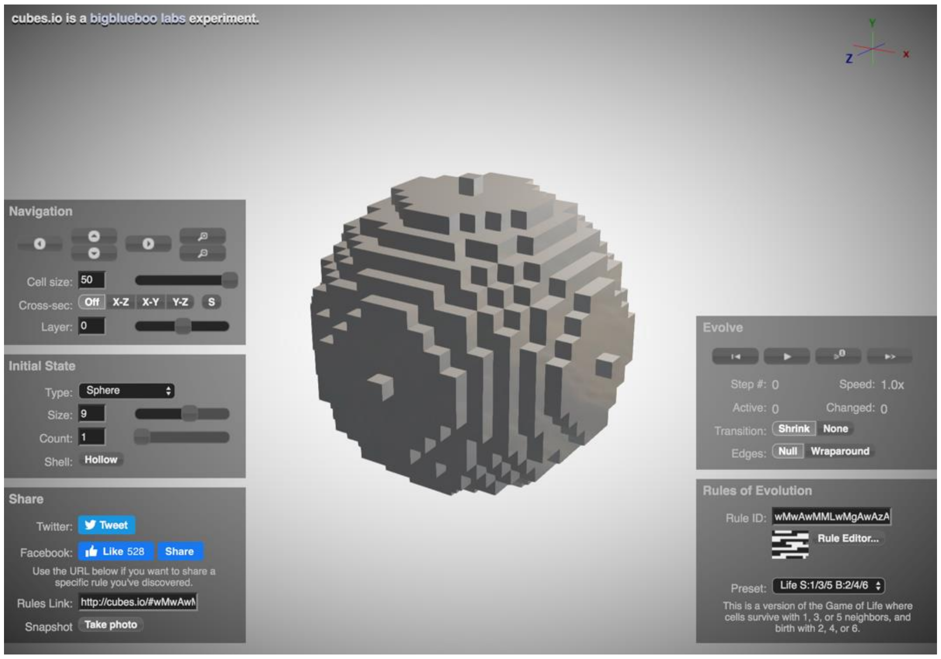
Select a starting condition:

● Impulse	○ 25%
☐ Left	○ 50%
✓ Center	○ 75%
☐ Right	○ Random

✓ Scroll continuously

Figure 8: A cellular automata simulator as presented in [43].

The simulator presented in [44] is specialized in only 3D cellular automata. Figure 9 shows a screen shot of this simulator. It provides a nice-looking 3D representation of the cellular automata grid with flexible navigation and rotation tools. However, as in the other examples, this simulator does not cover all types of cellular automata and does not provide easy to use rules designer.



cubes.io is a bigblueboo labs experiment.

Cell size: 50
Cross-sec: Off X-Z X-Y Y-Z S
Layer: 0
Type: Sphere
Size: 9
Count: 1
Shell: Hollow
Twitter:
Facebook:
Use the URL below if you want to share a specific rule you've discovered.
Rules Link: http://cubes.io/#wMwAwN
Snapshot

Step #: 0
Speed: 1.0x
Active: 0
Changed: 0
Transition: Shrink None
Edges: Null Wraparound
Rule ID: wMwAwMMLwMgAwAzA
Preset: Life S:1/3/5 B:2/4/6
This is a version of the Game of Life where cells survive with 1, 3, or 5 neighbors, and birth with 2, 4, or 6.

Figure 9: A cellular automata example as presented in [44].

The above-mentioned examples of cellular automata simulators have several interesting features such as ease of use and nice interface. These features are suitable for users who are interested in learning cellular automata especially those who are focusing on learning simple examples. However, these simulators lack several features that are necessary for comprehensive implementation of all kinds of cellular automata. They also do not provide the users with sufficient tools to design and/or analyze cellular automata models based on flexible rules. Table 1 shows features comparison of the simulators already presented in this section and the simulator proposed in this work.

Unlike the examples discussed in this section, the simulator proposed in this work is intended to cover all the features which are essentials for designing and analysis of cellular automata. It provides easy to use interface with very high flexibility. It covers all types of cellular automata including 3D cellular automata. The simulator also provides tools to design rules with colorful cell state. Above all of these features, the simulators will be implemented inside a Blockly platform which allows integrating the cellular automata models to other computational algorithms.

Table 1: Features comparison of current availed cellular automata simulators and the simulators proposed in this work. The proposed simulators overcome all other simulators.

Simulator	Easy Use	All types of CA	Rules Designer	Colorful State	3D CA	Integration with other platforms
http://robinforest.net/ [42]	Yes	No	No	Yes	No	No
http://devinacker.github.io/celldemo/ [43]	Yes	No	No	No	No	No
http://cubes.io/ [44]	Yes	No	No	No	Yes	No
Proposed simulator	Yes	Yes	Yes	Yes	Yes	Yes

2.3 Network of Evolutionary Processors (NEPs)

Network of Evolutionary Processors (NEPs) is class of the natural inspired algorithms [1, 2]. It consists of a number of small processors called “evolutionary processors”. These processors can perform very simple operations on the received data before the data is sent to another processor. It is theoretically capable of solving NP-problems with polynomial resources [5]. NEPs are characterized by their simple structure, inherent parallel processing of data, and adaptation to solve several problems. Several versions of NEPs have been proposed [3] which have been successful applied on several problems [4]. It is known for its inherently parallel processing of data. It was originally intended to be a hardware computing platform, however, most of the successful realization of NEPs are software simulation. A number of NEPs simulators are discussed later in this section.

2.3.1 NEPs Architecture

NEPs consists of a number of processors. These processors are connected via links which transmit data between processors in parallel. The transmitted data can be thought of as a DNA sequence that are sent among cells. In this analogy, the evolutionary processors can be thought of as cells or nodes. As the DNA sequence is been moved from one cell to another, it may evolve by mutations. The mutations are performed based on selected operations in the evolutionary processors. The number of processors, links, operations, and any other parameters may differ from one network to another depending on the application.

Figure 10 shows a simple architecture of one NEPs. In this figure, the NEPs consists of four processors that are connected in a ring topology. In addition to performing mutation operation, the nodes contain filters to prevent or permit specific DNA sequence to pass through. For this reason, each node might or might not be associated with filters.

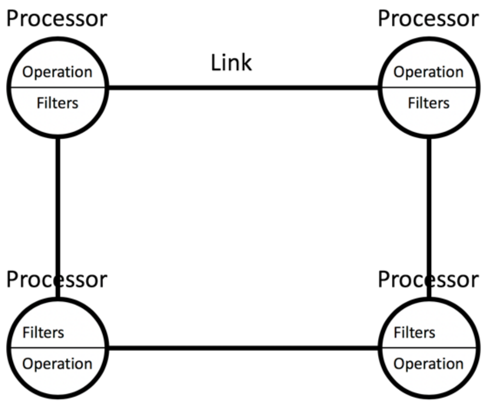


Figure 10: Network of Evolutionary Processors (NEPs) simplified architecture.

The DNA sequence or “words” are processed by implementing simple operations such as insertions and deletion of symbols from the words and transmitting and receiving words to/from other nodes. In addition to these operations, several other factors govern the work of NEPs. Next is a list of these factor:

1. **Symbol set:** These are a set of symbols which are forming the DNA or word sequence. They can be thought as the alphabet for the transferred data.
2. **Number of nodes:** The number of nodes that form the network.
3. **Connections:** The pattern based on which the nodes are connected. This pattern could be a regular network topology such as fully connected, ring, start etc. It could be also set as an irregular pattern. Selecting this pattern completely depends of the application or the problem in hand.
4. **Stopping condition:** The state of the network at which the operation stops. The condition is selected based on the nature of the problem been solved. Four stopping condition have been presented in the literature: Consecutive Config, Maximum Steps, Words Disappear, and Non-Empty Node.
5. **Rules:** The operations performed in each node. A limited number of operations have been presented in the literature such as Insertion, Substitution, Deletion, and Splicing.
6. **Filters:** The filters are used at each node to permit or prevent specific words from passing through this node.

More details about the NEPs setting parameters are presented in the next chapter.

2.3.2 Simulators Examples

Since it was presented, several software simulators for NEPs have been presented. The simulators include bother visual and textual programing methodologies. In this sub-section, some of these simulators will be presented. Early work by several research groups have presented several versions of NEPs simulators [1-3]. These simulators, however, are now considered outdated as they don't cover the general NEPs concepts that have been developed since then. A JAVA based simulators was presented in [4]. In the time this simulator made advances in terms of the improving the performance by utilizing parallel implementation, it focused only on decision making problems. In addition, flags have been raised about timing issues between evolutionary processors and communication steps. In [7] a general NEPs simulator was presented "jNEP". It is intended to simulator any NEPs in the literature where this simulation counted for all different variations of NEPs in terms of rules, filters, and connection. It also provides an option for parallel execution which utilizes the inherent parallel nature of NEPs. In addition, it is designed in an adoptable and flexible designed that make it easy to extend to include new stopping conditions, filters, or evolutionary rules. This simulator takes an XML file as an input. This file contains all necessary setting to describe any NEPs. The syntax of this file is carefully designed and formed as a BNF. The outputs of the simulators are presented as text in the terminal. This simulation was used as a base for the Blockly-based simulator presented in this work. More details about this simulator and the adaptation methodology is presented in the second section of the next chapter.

Based on jNEP another simulator "jNEPView is presented [45]. This simulator provides a visual representation of the NEPs executed with tools to analyze its performance. With this simulator users can track DNA sequence generated by each node.

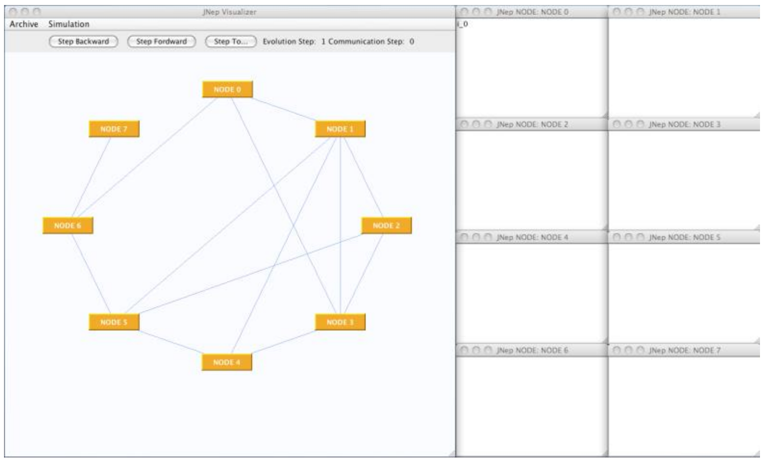


Figure 11: A screenshot of the jNEPView simulator [45].

The jNEPView provides visual representation for the network topology and the sequence output, however, it does not provide a visual programing for the setting of the NEPs and users manually write XML files to be entered to the simulators. In [46] a new simulator "AToM^3" is presented to fix this gap. This Python based simulator provides a visual methodology to setup NEPs settings. NEPs settings are entered to the simulator as UML class diagram. Figure 12 shows UML class diagram design window. In the other end, generate a visual representation for the NEPs with XML setting file. Figure 13 shows network topography of the NEPs designed in Figure 12. This simulator uses geometrical shapes to represent different aspects of the NEPs model. For instance, it uses small rectangle for stopping conditions, triangles for filters, and ovals for rules.

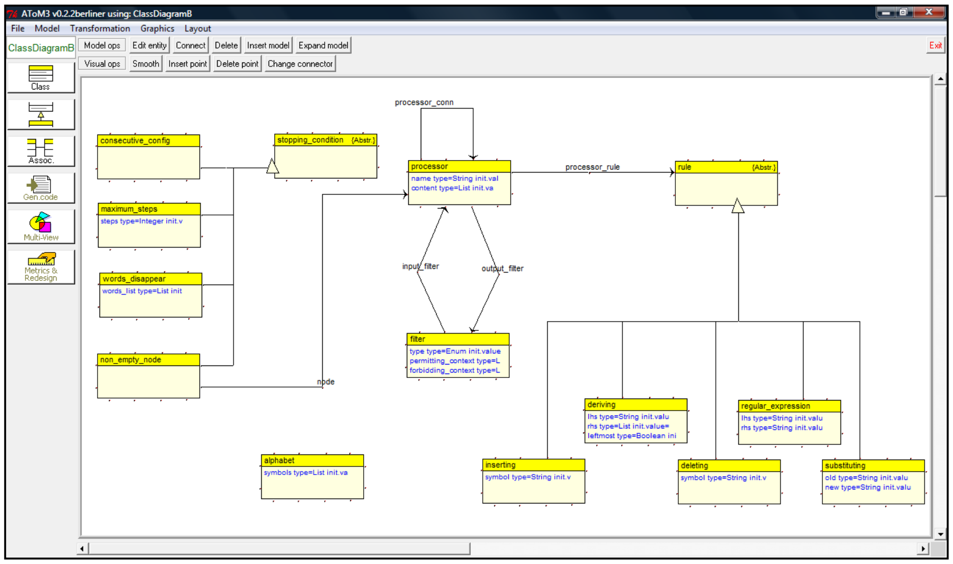

AToM3 v0.2.2berliner using: ClassDiagramB
File Model Transformation Graphics Layout
ClassDiagramB
Model ops Edit entity Connect Delete Insert model Expand model
Visual ops Smooth Insert point Delete point Change connector
Class
Assoc.
Gen.code
Multi-View
Metrics & Redesign
Exit
processor_conn
consecutive_config
stopping_condition (Abstr.)
processor
name type=String init.val
content type=List init.va
processor_rule
rule (Abstr.)
maximum_steps
steps type=Integer init.v
input_filter
output_filter
words_disappear
words_list type=List init
filter
type type=Enum init.value
permitting_context type=L
forbidding_context type=L
non_empty_node
node
deriving
lhs type=String init.valu
rhs type=List init.value=
leftmost type=Boolean ini
regular_expression
lhs type=String init.valu
rhs type=String init.valu
alphabet
symbols type=List init.va
inserting
symbol type=String init.v
deleting
symbol type=String init.v
substituting
old type=String init.valu
new type=String init.valu

Figure 12: Screenshot of AToM3 simulator. UML class diagram window [46].

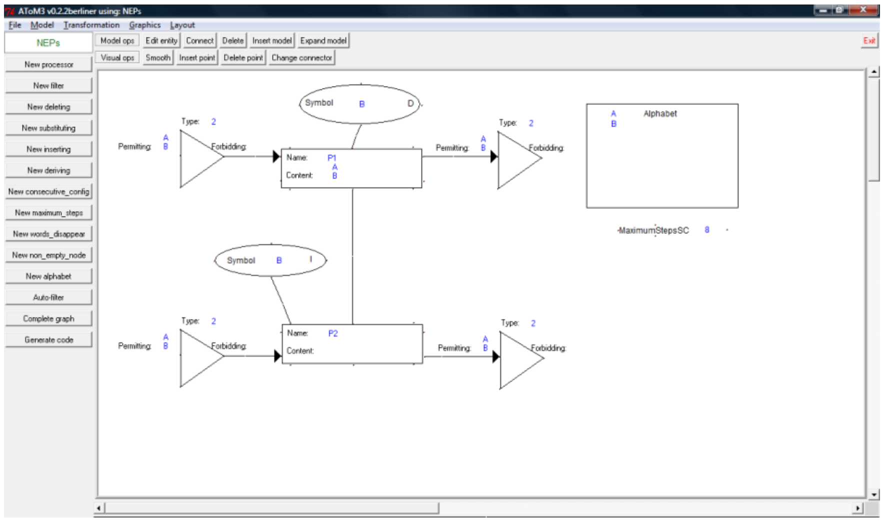


Figure 13: Screenshot of AToM3 simulator. Visual representation of the NEPs designed in previous figure [46].

While AToM^3 provides appealing visual representation for the implemented NEPs, it does not execute the model directly. Instead, it generates an XML file that is entered to jNEP for implementation. The simulator proposed in this work provides the features of all of the simulators mentioned in this section and more. It provides the following:

- Easy to use visual methodology to setup NEPs based on Blockly.

- Software solutions to automatically setup several regular network topologies.
- Visual representation of the network topography.
- Online implementation with server-side execution of the network which could dramatically increases the performance.

Details about the proposed simulators are presented in the next chapter.

CHAPTER 3. DESIGN AND METHODOLOGY

The goals of his project can be divided into two main parts. The first is designing a web based specification (programming) environment for some natural computers (NEPs and cellular automata) utilizing Blockly for testing the adequacy of block languages for this purpose and exploring some “software engineering” tools. The second is testing the system and checking the validity of our hypothesis. The first section focuses on building the simulator for Cellular automata. Blockly is used as input methodology to set up the setting for the different types of cellular automata. The simulator is built within the blocks. This means that the simulation for the cellular automata will be executed as JavaScript code in the web browser. The second section present the NEPs simulator. The blocks are also used as input methodology to set up NEPs. However, unlike cellular automata simulator, NEPs simulator is executed at the server side. This means that the inputs entered using the blocks are structured and sent to the server to be executed and the results are sent back.

3.1 Cellular Automata

In this section, we present the part of the system related with cellular automata. We will show several examples by means of which we will describe the different parts of the system.

3.1.1 Elementary Cellular Automata

Elementary cellular automata can be considered as the simplest type of cellular automata. It consists of 2D grid where each cell has only two states: on and off or 0 and 1. This type has been introduced in chapter 2.

In the next paragraphs we will describe the blocks of the system related with this type of cellular automata. Figure 14 shows the blocks in this category.

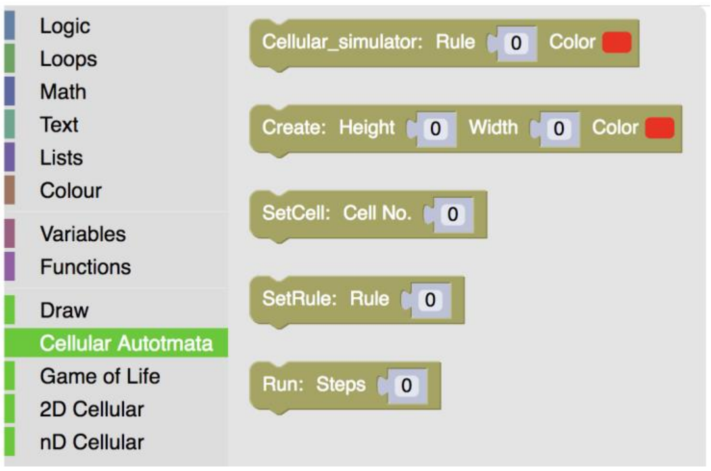


Figure 14: Blocks for implementing the Elementary Cellular Automata.

The first block is **Cellular_simulator**. This block was for learning purposes where it ran the simulation of the cellular automata in one step. The next four blocks are breakdowns of the work implemented by this block. These four blocks give flexibility for the user to design the desired cellular automata.

The second block is **Create**. The block is used to set the size (height and width) and the color of the "On" state of the cellular automata. The state could be any color out of the 70 colors provided by the color input field. The Height and Width inputs are used to set the size of the gird. These values can be larger than the size of the canvas. If these values are left to zeros, the grid is automatically created to fill the canvas. As given by Cellular Automata Law, the cells in the created grid are automatically set to be to the "Off" state. The third block is **SetCell**. This block is used to set only one cell in the first line of the grid to the "On" state. More than one SetCell blocks can be used to set more than one cell in the first line. The color of the cell set in this block is changed to the color specified in the Create Block. The block is designed to accept values that are within the size of the created grid. Otherwise, an alert message will pop up. The **SetRule** block is used to specify the rule to be applied on the cellular automata. The input value of this block could be any number between 0 and 255. The **Run** block is used to simulate the cellular automata based on the setting specified on the previous three blocks. This block has only one input which is the number of steps to be ran. In this specific case, each step means applying the rule on the next row in the grid.

Figure 15 shows an example using the blocks of this category to implement an Elementary Cellular Automata. The size of the grid is 48x48 that is ran over 40 rows using rule no. 30.

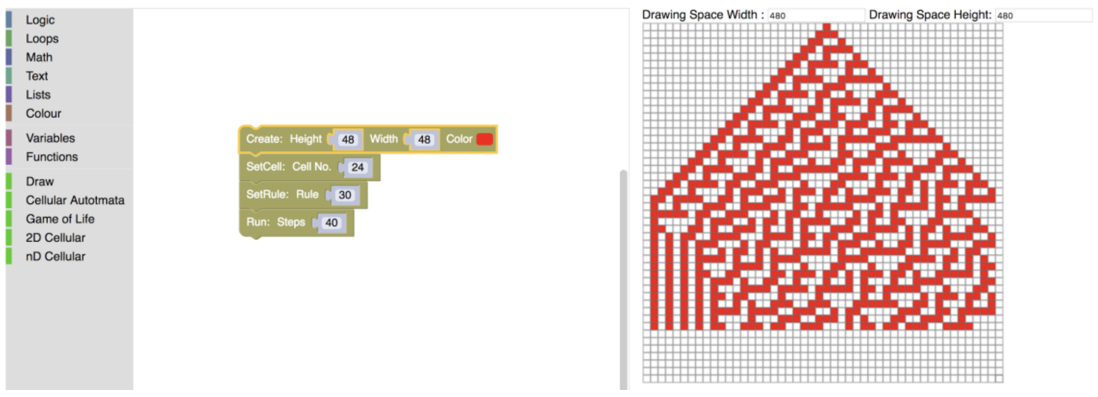


Figure 15: Elementary Cellular Automata implemented using blocks of the Cellular Automata category. The cellular automata showed applies rule no. 30 on 40 rows of the grid.

3.1.2 Game of Life

This type of cellular automata has been introduced in chapter 2. This section introduces blocks used to implement game of life. Figure 16 shows the blocks of the Game of Life Category. Three blocks only are used in this category: **Create**, **SetCell**, and **Run**. The **Create** block is similar to the create block of the previous category. It is used to create a grid of cells with specific size and color for the "Live" state. All of the cells in the created grid are set to initially be in the "dead" state. If the input size of the grid is bigger than the size of the canvas, the website will show an alert message while if the inputs are zeros, the grid will be created to full the whole canvas. The **SetCell** block is used to set one cell in the grid to "Live" state. It has two inputs to specify the position of the cell in the grid using the row and column coordinates. A number of this block can be used to set the state of any desired number of cells. The **Run** block has only one input which is the number of steps to be ran. All of the cells in the grid are tested using the game of life rule in each step at runtime.

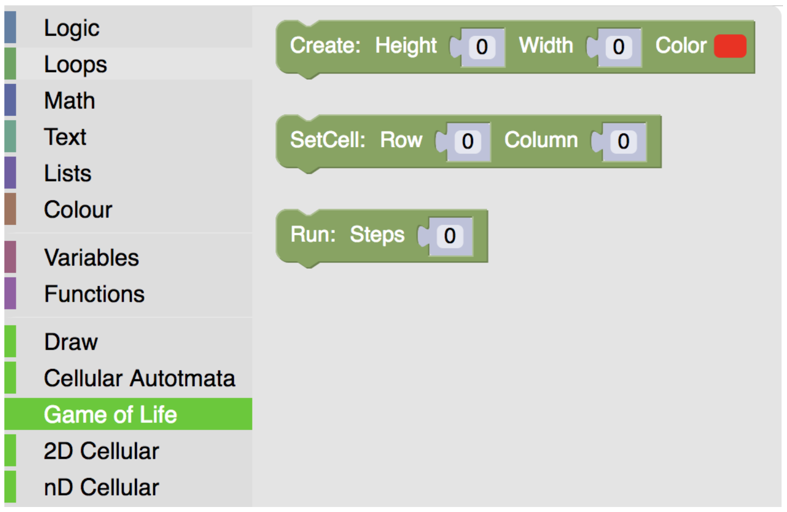
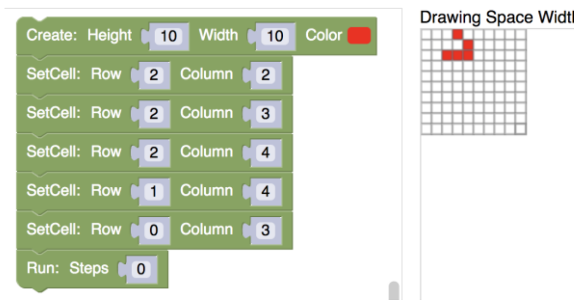


Figure 16: Blocks of the Game of life category.

Figure 17 shows a game of life example of a setting that generates a glider (refer to this website mentioned above for examples of this behavior. The grid in this figure is before applying game of life rule. Figure 18 shows the run steps of this example. It can be seen that the initial state of the grid appears again in step 4, however, shifted one row and one column due to the glider behavior of game of life.



```
Create: Height 10 Width 10 Color
SetCell: Row 2 Column 2
SetCell: Row 2 Column 3
SetCell: Row 2 Column 4
SetCell: Row 1 Column 4
SetCell: Row 0 Column 3
Run: Steps 0
```

Figure 17: Game of life example with setting that leads to spaceship behavior.

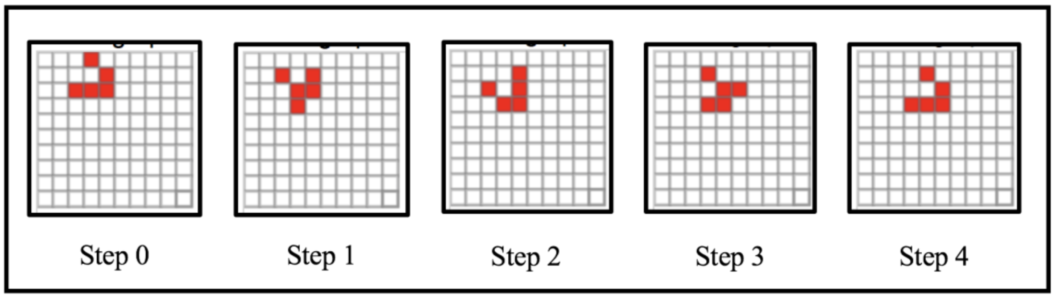


Figure 18: Game of life Example, run steps. It can be seen that the shape the example starts with in Step 0 (initial state) is repeated at shown again in step 4 shifted one row and one column.

3.1.3 General Cellular Automata that can be represented in 2D grids

In this category, blocks are designed to work on custom 2D cellular automata. The blocks give the user the ability to start a cellular automaton from scratch through designing rules that run on 2D as in game of life or in row by row as in the Elementary Cellular automata. These two different simulation schemes can be implemented on the same grid simultaneously. Such integration, provides the user the flexibility to simulate, analyze, and test different configurations of cellular automata in one platform. It is also important to mention that the possible states of the cells in this category are 70 which are the colored provide by the color input field.

Figure 19 shows the blocks in this category. Seven blocks are designed for this category. Next is the description of each block. Some of the blocks are similar to the blocks described in the above-mentioned categories such as the first and second blocks which are used to create grid and set cell in the grid. **SetRule 2D** block is used to design rules for the 2D cellular automata (in this block we assume Moore's neighborhood, where each cell has eight neighbors in addition to itself). This block takes ten inputs, nine inputs for the old state of the cell itself and the 8 surrounding neighbors while the tenth input is for the new state of the cell. As many blocks of SetRule 2D can be used to set different rules. The rules are saved in the website to be implemented using the run blocks. **Run 2D** block is similar to Run 2D block in Game of life category. Run 2D line is used to apply the rules one line at a time. This category also includes blocks that work with 1D implementation similar to the Elementary Cellular automata. The **setRule 1D** is used to design rules for this purpose. It takes six inputs, the first five are for the old state of the cell and the two neighboring cells form each side while the sixth input is for the new state of the cell. **Run 1D** block is used to apply the 1D rules. It works on a single row at a time. The two inputs of this block are to set the row and steps where the rules will be applied. It processes cells in one row from left to right based on the number of steps been entered. If the number of steps is too big, the processing will start back from the left. This will add toroidal boundary conditions to the implementation.

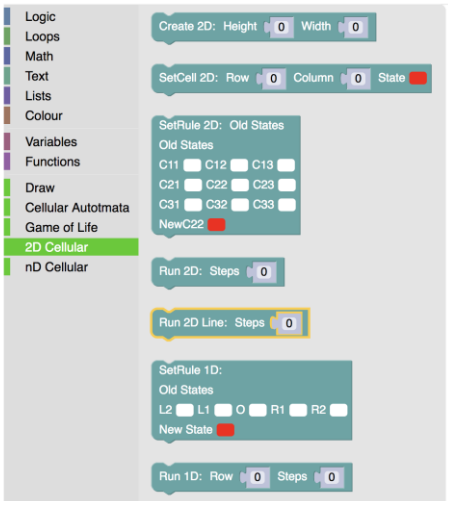


Figure 19: Blocks of 2D cellular category.

3.1.4 nD Cellular Automata

The nD cellular automata is designed to be a generic implementation of any cellular automaton. It covers any type of cellular automata with all varieties of rules and states. The grid of cells could be designed to be of any dimensions starting from 2D up to any desired number dimensions. Unlike the 2D cellular automata, the states of the cells in the nD cellular automata expanded to cover any integer number. The nD cellular automata is also expanded to cover any set of rules. The next section explains the blocks designed for the nD cellular automata, their uses, and internal implementation.

Figure 20 shows the blocks of the nD Cellular category. This category contains six blocks as follows:

Create nD: This block is used to create the nD grid of cells. It might look similar to the create blocks of the other categories, however, it has more complicated tasks. It takes two inputs: Dimensions, is used to set the number of dimensions of the cellular automata and Size: is used to set the length of the dimensions. To simplify the work of the nD cellular, it is assumed that the lengths of the dimensions of the nD cellular automata are the same. For example, using the values: Dimension = 3, Size = 10 in this block will create a 3D cellular automata that has the lengths of 10x10x10. The values of the cells of the nD cellular automata are saved in a 1D array regardless of the dimensionality of the cellular automata. This approach requires the use of appropriate transformation between nD coordinates (of the cellular automata) and 1D coordinates (of the 1D array in the code used to save the values), this process will simplify the overhead of saving grids of unspecified dimensions. Special JavaScript functions are designed to transfer the nD coordinates to 1D coordinates and vice versa. The first function is nD2oneD. This function is used to transfer nD coordinates to 1D coordinates. It takes three values as inputs: the nD coordinates, number of dimensions in the cellular, and the length of the dimensions. The transformation from nD to 1D is similar to the transformation from Binary to Decimal except that in this function will use the length of the dimensions as base to this transformation. The next Equation is used to change the indexes between nD and 1D as follows:

$$1Dindex = \sum_{i=1}^{n} \quad nDindex_i^d$$

where:

n: is the number of dimensions

d: is the length of the dimensions (the same for all dimensions).

For example, consider the coordinate {3,7,2} a 3D grid with sizes 15x15x15. The 1D coordinate (index) of this 3D coordinates is calculated as follows:

$1D_{index} = 3 * 15^0 + 7 * 15^1 + 2 * 15^2$

$1D_{index} = 3 * 1 + 7 * 15 + 2 * 225$

$1D_{index} = 558$

The second function is oneD2nD. This function is used to transfer 1D coordinates to nD coordinates. It takes three values as inputs: the 1D coordinate, the number of dimensions in the cellular, and the length of the dimensions. The transformation from 1D to nD is similar to the transformation from Decimal to Binary except that in this function will use the length of the dimensions as base to this transformation.

For example, consider the 1D index 558. The 3D coordinates of a 3D grid with size 15x15x15 is calculated as shown in the next table:

Base	1D	1D÷15	reminder
15	558	558÷15 = 37	3
15	37	37÷15 = 2	7
15	2	2÷15 =0	2
	0		

According to the above table the 3D coordinates is {3,7,2}.

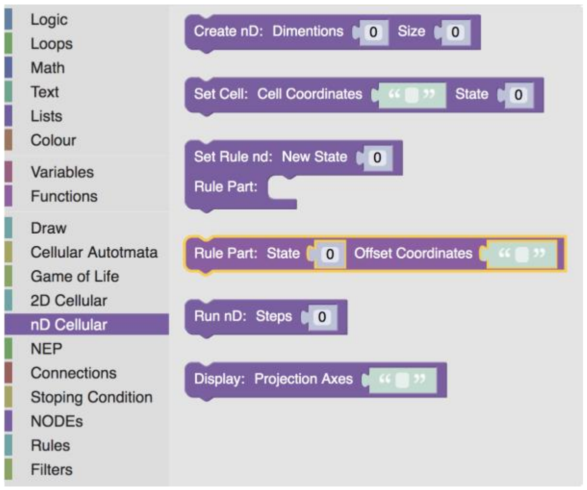
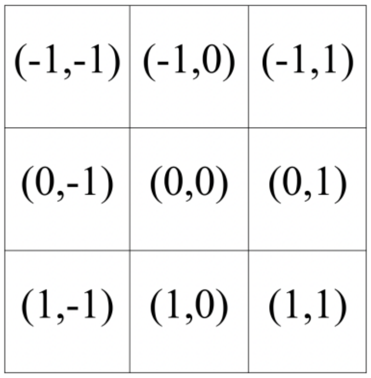


Figure 20: Bocks of nD Cellular Category.

Set Cell: This block is used to change the state of one cell in the nD grid. It takes two inputs: Cell Coordinates which is used to enter the coordinates of the cell to change its state and State which is the new state of the cell. Since the number of the dimensions of the grid is not specified until run time, the coordinates of the cell to be set is entered as text. For example, the coordinates of the 3D grid (3, 7, 2) is simply entered as “3, 7, 2” with comma separating the numbers. In JavaScript, the .split () method is used to change the entered text into a numerical array. For instance, it transfers the text “3, 7, 2” to the 1D numerical array [3, 7, 2]. Any number of this block can be used to change the states of any number of cells in the grid as desired.

Set Rule nd: This block is used to design the rules of the cellular automata. It is used in combination with the next block in this category, **Rule Part**. As explained in the previous chapter, each rule depends on the states of a number of or all of the neighboring cells in the grid. However, the numbers of neighboring cells in the nD grids varies depending on the number of the dimensions of the grid. For example, the number of neighboring cells for the 2D, 3D, and 4D grids are 8, 26, and 80 respectively. It is not possible to design block with such undetermined number of inputs. For this reason, the rules in the nD cellular automata are designed using these two blocks. The first block, Set Rule nd, serves for two purposes. The first is to take the new state of the rule while the second is to be used as a container for the second block, Rule Part. This block takes the coordinates of one neighboring cell and its state. The coordinates of the neighboring cells are specified by means of offsets from the center cell that is the current one. For instance, the cell to the left side of the center cell can be referred to in offset coordinates as {0,-1} while the sell under the center cell can be referred to in offset coordinates as {1, 0}. Figure 21 shows offset coordinates of 2D grids. This scheme can be expanded for any nD grid. Offsets are entered as text and they are processed in the same way as in the Set Cell block.



(-1,-1)	(-1,0)	(-1,1)
(0,-1)	(0,0)	(0,1)
(1,-1)	(1,0)	(1,1)

Figure 21: Offset coordinates of 2D grid. This scheme can be expanded for any nD grid.

Unlimited number of Part Rule blocks can be used inside Set Rule nd block depending on the rule and the number of dimensions of the grid. Using these two blocks, any rule can be designed for any cellular automata regardless of the number of dimensions or length of any dimension. Unlimited number of rules can be entered to the system. All of the rules will be applied to the cellular automata in the order they entered.

Run nD: This block is used to apply the nD rules to the nD cellular automata grid. This block is similar to all other Run blocks in terms of the abstract functionality. However, it is much more complicated considering that it works on nD grids. The oneD2nd and the nD2oneD functions are continuously used in the code of this block. In addition, an extra function is used: nD2oneDoffset. This function is used to specify the neighbors of the cell in rule based on the offset. It takes the coordinates of the cell and the offset of the neighbor and change it directly to the 1D index, so the run block takes the value of this neighbor from the 1D matrix. The steps of this function are: (1) adding the offset to the coordinates of the cell, and this will give the coordinates of the neighbor, (2) changing that coordinates to the 1D representation using the function nD2oneD as described above. There is a condition inserted between these two steps. This condition ensures that the neighbor cell will not be out of the cellular length. The usual toroidal boundary conditions are used in this case too.

Display: This is a unique block that is not included in all other categories. It is used to show the grid of the implemented cellular automata. If the number of dimensions of the cellular automata is 3 or higher, it is impossible to show such grid on a 2D canvas without projection. One 2D slice of the nD cellular automata will be shown at a time. The display block will be used for this purpose. The display block shows a 2D projection of any nD cellular automata. The input "Projection Axes" controls the projection. In general, for any 2D projection, two types of inputs are required. The first one is used to select the two axes that will be projected. The second one, is used to set specific values for the other axes. For example, if we want to extract 2D projection for a 3D model, that means we want to show a 2D slice of that 3D model. Let's assume that we have a 3D model in which each axis is of length 10. This would give us a cube that contains 1000 cells coming from 10 length by 10 width by 10 height. If we want to do a 2D projection of this model, it means that we will show a 10 by 10 slice of the 3D model. This slice should have a full span of two of the axes and a specific value of the third axis. So if we enter x, x, 3 in the display block, it means show the third horizontal slice of the 3D model. In the same way x, 5, x means show the fifth vertical slice in the width direction while 6, x, x means show the sixth vertical slice the length direction. To summarize, the input to the display block should contain a series of "x" characters and numbers. The length of the series should be equal to the number of dimensions of the CA. The "x" character means that this particular axis should have a full span while a number means that this axis should have that specific constant value in for this projection. There should be only two "x" characters as we are doing a 2D projection. This block also used to change the cell's states values into color so that they can be shown in the canvas as colored cells.

3.2 NEPs on Blockly

The Blockly system for NEPs is a tool that uses custom designed blocks to design NEPs, generate XML configuration structure. The resulted XML can be considered as input to another different tool developed in [6]. The system described in this document is designed with four specifications in mind:

1. The ability to design, implement and show results of the most recent update of NEPs in the literature.
2. Utilize the recent tools presented in the field to reduce development time.
3. Easiness for the user.
4. The flexibility to adapt for any future development such as add more rules, filters, and stopping conditions or connect the tool to other remote servers.

Figure 22 shows a schematic of the NEPs tool. The client side is a website designed to take user inputs in form of Blockly programs. Their blocks are designed to represent features and properties of NEPs. The website interprets the blocks into an XML configuration file for NEPs. The XML configuration file is sent in an Http request to the server side where the NEP is simulated. The output of the simulation is sent back to the client side. The output of the simulation will be shown to the user as text and as a graphical representation. In addition, the tool contains extra features such as auto-generate of edges of the NEPs.

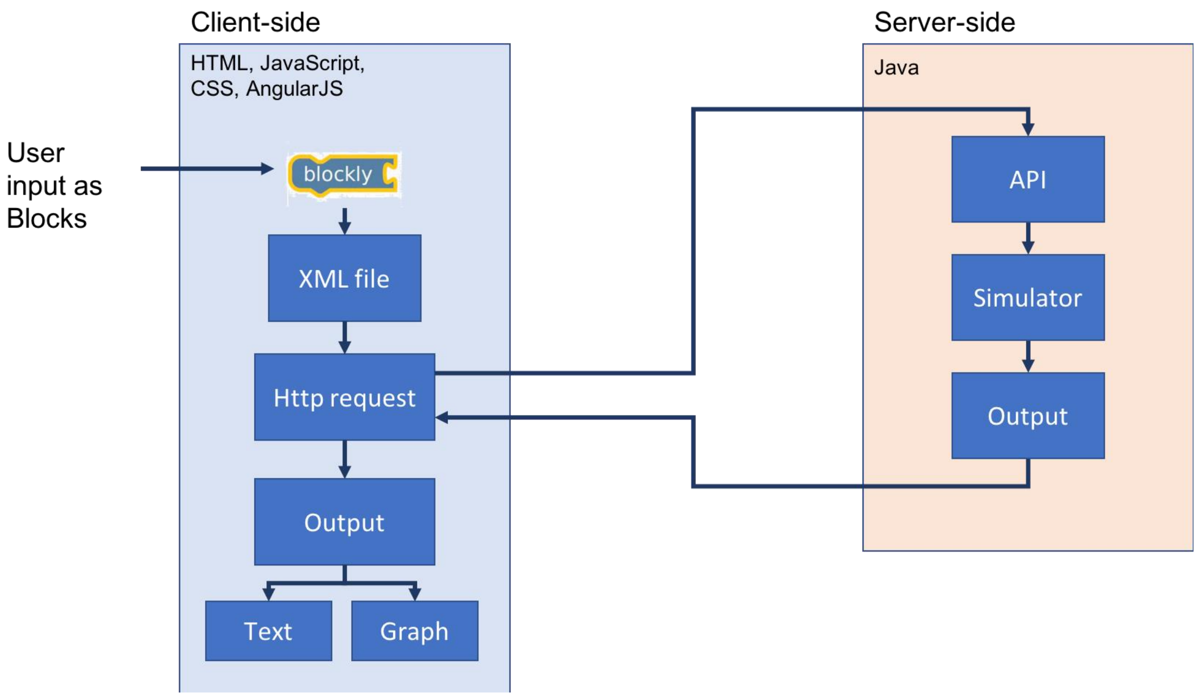


Figure 22: Complete system of proposed project. It consists of client side and server side.

From Figure 22, we can divide the workload of the development process of the tools into server side and client side. The server side includes the development of a NEPs simulator API which runs on a remote server. The client side includes the development of blocks, http request handler, and show results. This section describes the development process of all of these components in details.

3.2.1 NEPs API

The NEPs API is developed based on a tool called jNEP which was proposed by Emilio Garia [6]. jNEP is a software tool that simulates NEPs. It takes XML configuration file as input and print the output on console terminal. jNEP provides various desired features for running such type of natural computing systems. It is built to be a generic platform that can be updated as needed to adapt to any new changes or additions to the concept or implementation of NEPs. It can also be executed in a Java parallel computing platform.

The NEPs API is built by updating jNEP. The update process includes the following steps:

1. Update the main method of jNEP to a regular method that received XML configuration code as a string.
2. Test the new method with available examples from the literature.
3. Redirect the console.print to a text file instead of the operating system.
4. Test (Same as step 2)
5. Temporarily save the file on the server. And used it as a return value for the method.
6. Build Java restful API base off of the new created method. This API receive XML configuration file of a NEP as input and return a text file of the results as output.
7. Test the API locally (Same inputs as in step 2).
8. Deploy the API on an online server.
9. Test the online version of the API (Same inputs as in step 2).

The steps described above were striate forward and I faced no major challenges while performing them. However, there is only one con that I faced at deploying the API. It was very hard to find a hosting server that support Java restful API. The once that are found were rather expensive.

3.2.2 NEPs Blocks

There are two main purposes for these blocks. The first one is to receive and process users' inputs to generate the equivalent XML configuration of the desired NEPs. The second is to generate a static visual representation of the designed NEPs structure. This chapter discusses the blocks in detail. The blocks used for NEPs are classified into six categories as follows:

- NEP
- Connections
- Stopping Condition
- NODEs
- Rules
- Filters

Each one of these categories cover specific aspects of the NEPs. As it has been mentioned several times, the end goal of these blocks is to generate the XML configuration file that describes the specific NEP. The first block “NEP Create” create an XML object as a public variable. This object can be accessed by all of the blocks in the system. Each bloc can edit this XML object based on the data gained from the user. As blocks are implemented, the XML configuration code is written. It is also important to mention that this XML object is also used as a way to communicate between the blocks because it carries all of the information about the NEP while it is designed. For example, the set edge blocks can extract the number of nodes from this object. This information is vital to test whether the vertex values entered by the user are within the range of the nodes in the NEP.

The next section describes the blocks of each category:

NEP: This category includes two blocks. The first block is **NEP Create**. This block is used to create a NEP. It has only one input which is No. Nodes. It takes an integer number that represents the number of nodes in the NEP to be created. The second block is **NEP Set Symbols**.

This block is used to set the symbols of the NEP. Any number of symbols can be entered. Figure 23 shows these blocks.

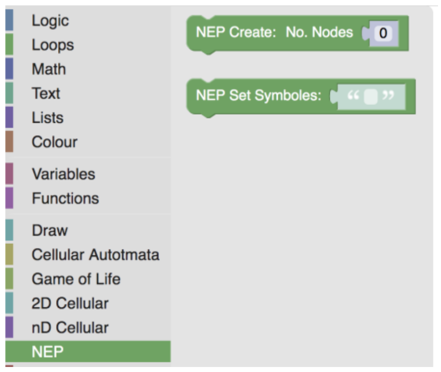


Figure 23: Blocks of NEP category. Consists of two blocks: NEP Create and NEP Set Symbols.

The blocks in the NEP category generate XML code based on the standard BNF format as follows that has been taken from [6].

- [configFile]::=<?xmlversion=”1.0”?><NEPnodes=“[integer]”>[alphabetTag][graphTag][processorsTag] [stoppingConditionsTag] </NEP>

- [alphabetTag] ::= <ALPHABET symbols=“[symbolList]”/>

Connections: This category includes a number of blocks to create edges between the nodes within the NEP. The edges in the NEP are considered as links to transferee information between nodes. In this category, ten blocks are included to create, remove and manages edges. Figure 24 shows these blocks.

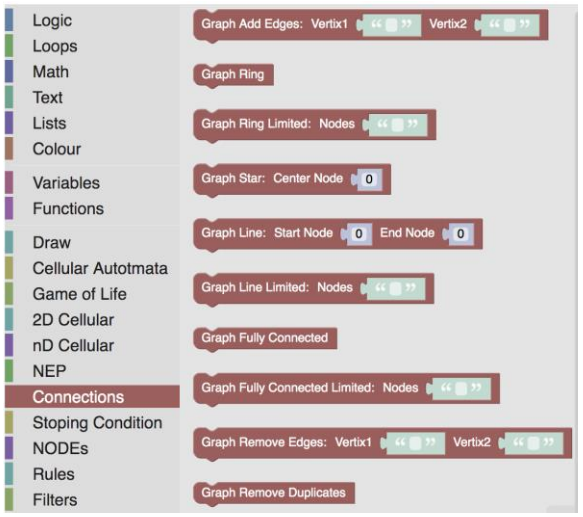

Logic
Loops
Math
Text
Lists
Colour
Variables
Functions
Draw
Cellular Autotmata
Game of Life
2D Cellular
nD Cellular
NEP
Connections
Stoping Condition
NODEs
Rules
Filters
Graph Add Edges: Vertix1 Vertix2
Graph Ring
Graph Ring Limited: Nodes
Graph Star: Center Node 0
Graph Line: Start Node 0 End Node 0
Graph Line Limited: Nodes
Graph Fully Connected
Graph Fully Connected Limited: Nodes
Graph Remove Edges: Vertix1 Vertix2
Graph Remove Duplicates

Figure 24: Blocks of Connections category in NEP. It consists of ten blocks. These blocks are used to set links between nodes of the NEP.

The blocks in this category are explained below. To see how they work you can follow the examples in Figure 25.

The first block is **Graph Add Edges**. This block is used to set one or more edges between two nodes or more nodes. It has two inputs: Vertix1 and Vertex 2 which represent the number of nodes to be connected with these edges.

With NEPs that has high number of edges, fully connected NEP for instance, it will be time consuming to use only this block to set the edges. For this reason, seven extra blocks are added to this category. These blocks are used to set a number of edges as one package in one step. The blocks cover four main types of network topologies including ring, star, line, and fully connected. These blocks will reduce the amount of work and time consumed by used to set up such highly connected network using the first block. The points 2-8 explains these blocks. These constructs are not part of the basic NEP definition. They could be considered as a kind of "software engineering" feature that automatically generates the proper connections for these types of topology.

1. **Graph Ring:** This block adds edges between all nodes of the NEP to create ring topology. It creates n-1 edges, where n is the number of nodes in the NEP. This block has no inputs.
2. **Graph Ring Limited:** This is a limited version of the previous block. It creates edges in ring fashion between specific nodes. The number of these nodes are entered to the block separated by comma.
3. **Graph Star:** This block is used to create edges between all nodes to form star network topology. It has only one input which is used to set the center node of the star topology.
4. **Graph Line:** This block creates line network topology between sequenced nodes in the NEP. It has two inputs Start Node and End Node which are used to specify the nodes at the start and end of the line topology.
5. **Graph Line Limited:** This block is similar to the previous block; however, it creates line topology between specific node. It has one input to specify the nodes included in this topology.
6. **Graph Fully Connected:** This is one of the important blocks in this category. It creates fully connected network topology between all nodes in the NEP. It is very common in the NEP literature to create fully connected NEP with a number of edges that is equal to nXn, where n is the number of nodes in the NEP. This block comes handy to auto-generate this high number of edges in one step.
7. **Graph Fully Connected Limited:** This block is similar to the previous block; however, it creates a fully connected topology between specific nodes.
8. **Graph Remove Edges:** This block is used to remove edges from the NEP.

9. **Graph Remove Duplicates:** This block is used to remove any duplicated edges. The duplicated edges can be resulted from using to auto-generate blocks.

The resulted code of all of these blocks follows the standard BNF format provided in the literature [6]:

- [graphTag]::=<GRAPH>[edge]</GRAPH>
- [edge] ::= <EDGE vertex1="[integer]" vertex2="[integer]"/> [edge]
- [edge] ::= λ

Figure 13 shows these blocks.

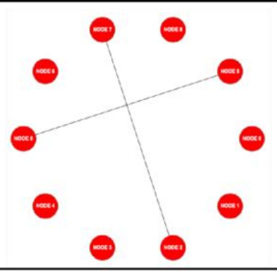
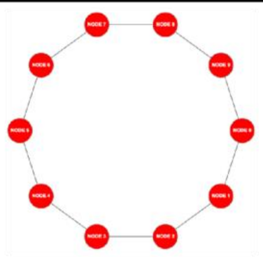
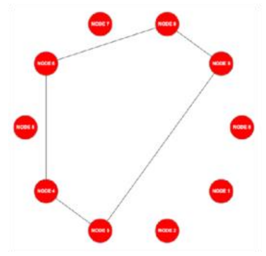
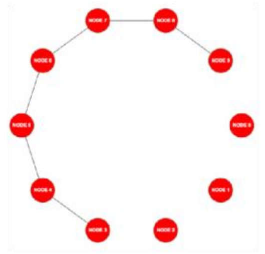
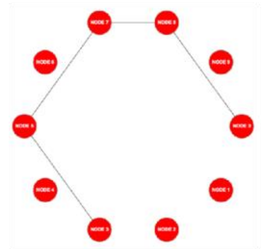
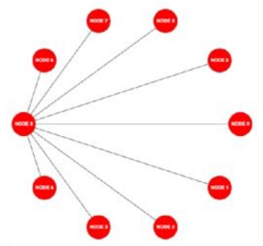
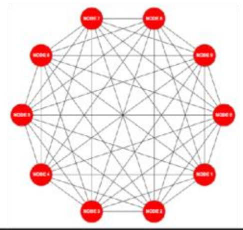
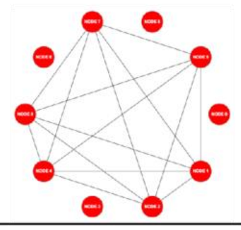
	
Graph Add Edges: Vertix1 " 2,5 " Vertix2 " 7,9 "	Graph Ring
Graph Add Edges	Graph Ring
	
Graph Ring Limited: Nodes " 3,4,6,8,9 "	Graph Line: Start Node 3 End Node 9
Graph Ring Limited	Graph Line
	
Graph Line Limited: Nodes " 3,5,7,8,0 "	Graph Star: Center Node 5
Graph Line Limited	Graph Star
	
Graph Fully Connected	Graph Fully Connected Limited: Nodes " 1,2,4,5,7,9 "
Graph Fully Connected	Graph Gully Connected Limited

Figure 25: Examples of connection blocks.

Stopping Conditions: This category contains blocks that cover all types of stopping conditions in the literature. Four blocks are designed to represent the four main stopping conditions. The blocks in this category are shown in Figure 26.

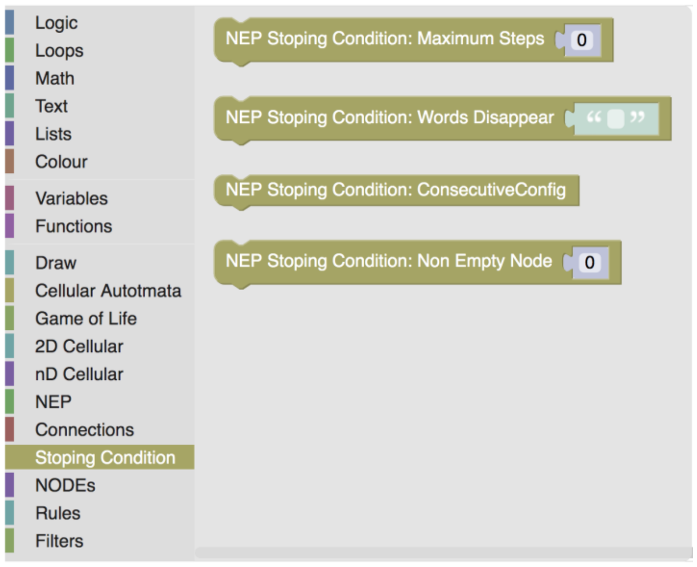


Figure 26: Blocks in Stopping Condition category.

The supported stopping conditions can be easily expanded to cover more conditions. The curretrly supported stopping conditions blocks are:

1. **Maximum Steps:** It forces the NEP to stop after running for a number of steps. It has only one input which is the number of steps as integer.
2. **Words Disappear:** It forces the NEP to stop when all of the specified words are no longer in the NEP. This block has only one input as text for the word to disappear. The words are entered with a comma separating them.

3. **ConsecutiveConfig:** This block is to represent the ConsecutiveConfig stopping condition. This block has no inputs. According to this type of Stopping condition, the NEP stops the first time a configuration is repeated in two consecutive cycles of the NEP.
4. **Non-Empty Node:** This stopping condition forces the NEP to stop if the specified node is not empty. Such stopping condition is used for NEP that has an output node.

Using any of the above block will add its related code to the XML file. The syntax of the resulted specifications of the stopping condition in the next BNF format taken from [6] is as follows:

-[stoppingConditionsTag] ::= <STOPPING CONDITION> [conditionTag] </STOPPING CONDITION> - -
[conditionTag]::=<CONDITIONtype="MaximumStepsStoppingCondition"maximum="[integer] "/> [conditionTag]

-
[conditionTag]::=<CONDITIONtype="WordsDisappearStoppingCondition"words="[wordList]" /> [conditionTag]

- [conditionTag] ::= <CONDITION type="ConsecutiveConfigStoppingCondition"/> [condition-Tag]

-
[conditionTag]::=<CONDITIONtype="NonEmptyNodeStoppingCondition"nodeID="[integer]"/ > [conditionTag]

- [conditionTag] ::= λ

NODEs: This category contains only one block which is named **NEP Set Node**. Figure 27 shows this block. This block is used to add nodes to the NEP. Each block of this type that is added to the system produce a node in the resulted XML. This block has to inputs: initCond and auxiliaryWords which are used to set the initial condition and auxiliary words of the node respectively. The node also accepts two type of internal blocks: Rules and Filters. These rules are used to set the rules and filters of the generated node. The Rules and Filters blocks are discussed in the next subsections.

The BNF syntax for this block taken from [6] is as follows:

- [nodeTag] ::= <NODE initCond="[wordList]" [auxWordList]> [evolutionaryRulesTag] [node-FiltersTag] </NODE> [nodeTag]

- [nodeTag] ::= λ

- [auxWordList] ::= auxiliaryWords="[wordList]" | λ

- [evolutionaryRulesTag]::=<EVOLUTIONARY RULES>[ruleTag]</EVOLUTIONARY RULES>

- [ruleTag] ::= <RULE ruleType="[ruleType]" actionType="[actionType]" symbol="[symbol]" newSymbol="[symbol]"/> [ruleTag]

-
[ruleTag]::=<RULEruleType="splicing"wordX="[symbolList]"wordY="[symbolList]"wordU="[symbolList]" wordV="[symbolList]"/> [ruleTag]

-
[ruleTag]::=<RULEruleType="splicingChoudhary"wordX="[symbolList]"wordY="[symbolList]" wordU="[symbolList]" wordV="[symbolList]"/> [ruleTag]

- [ruleTag] ::= λ
- [ruleType] ::= insertion | deletion | substitution - [actionType] ::= LEFT | RIGHT | ANY

[nodeFiltersTag] ::= <FILTERS>[inputFilterTag] [outputFilterTag]</FILTERS>
[nodeFiltersTag] ::= <FILTERS>[inputFilterTag]</FILTERS>
[nodeFiltersTag] ::= <FILTERS>[outputFilterTag]</FILTERS> [nodeFiltersTag] ::= <FILTERS></FILTERS>

- [inputFilterTag] ::= <INPUT [filterSpec]/>
- [outputFilterTag] ::= <OUTPUT [filterSpec]/>
-
[filterSpec]::=type=[filterType]permittingContext="[symbolList]"forbiddingContext="[symbolList]" - [filterSpec] ::= type="SetMembershipFilter" wordSet="[wordList]"
- [filterSpec] ::= type="RegularLangMembershipFilter" regularExpression="[regExpression]"
- [filterType]::=1|2|3|4

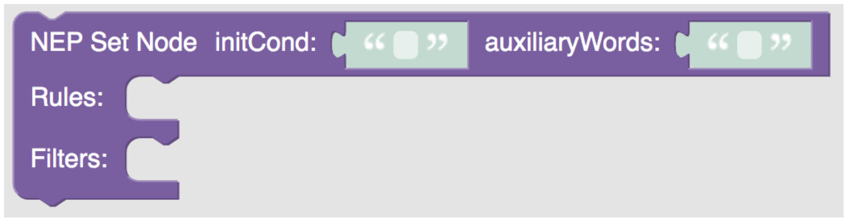


Figure 27: NEP Set Node block. This block is located inside the NODEs category.

Rules: This category contains blocks that cover all types of rules available in the literature. Figure 28 shows the blocks included in this category. The blocks are discussed below:

1. **NEP Set Rule Insertion:** This block represents the insertion rule of the NEP. It has three inputs: a dropdown list to select the action type (left, right, any), symbol, and the symbol itself.
2. **NEP Set Rule Substitution:** This block represents the substitution rule. It has two inputs: the old and the new symbols.
3. **NEP Set Rule Deletion:** This block represents the Deletion rule and has only one input to specify the symbol to be deleted.
4. **NEP Set Rule Splicing:** This block can be used to represent the two types of Splicing rule: Splicing and Splicing Choudhary. It has five inputs: a dropdown list to select the type of splicing rules and four extra inputs for the four words involved in the rule, that are, word x, word y, word U, and wordV.

Figure 16 shows these blocks.

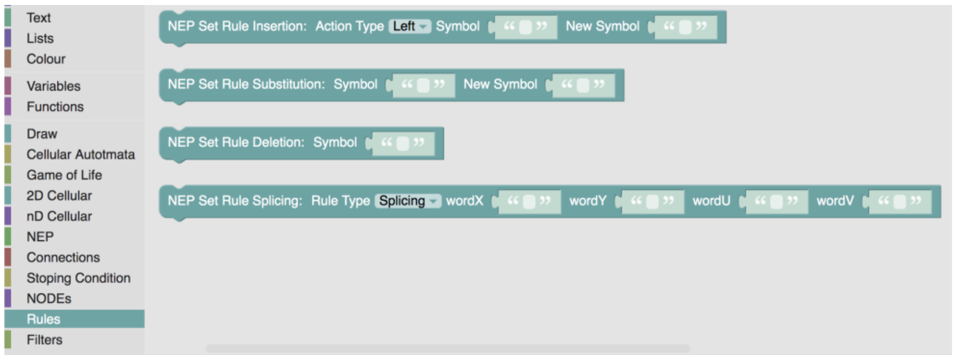

Text
Lists
Colour
Variables
Functions
Draw
Cellular Autotmata
Game of Life
2D Cellular
nD Cellular
NEP
Connections
Stoping Condition
NODEs
Rules
Filters
NEP Set Rule Insertion: Action Type Left Symbol " " New Symbol " "
NEP Set Rule Substitution: Symbol " " New Symbol " "
NEP Set Rule Deletion: Symbol " "
NEP Set Rule Splicing: Rule Type Splicing wordX " " wordY " " wordU " " wordV " "

Figure 28: Rules blocks. These blocks cover all types of rules which have been adopted in the literature.

As in the case of stopping condition, the system can easily be expanded to cover any new types of rules.

Filters: The filter category contains three blocks that represent the four standards filters and two membership filters. The blocks in this category are shown in Figure 29 The blocks are discussed in detail as follows:

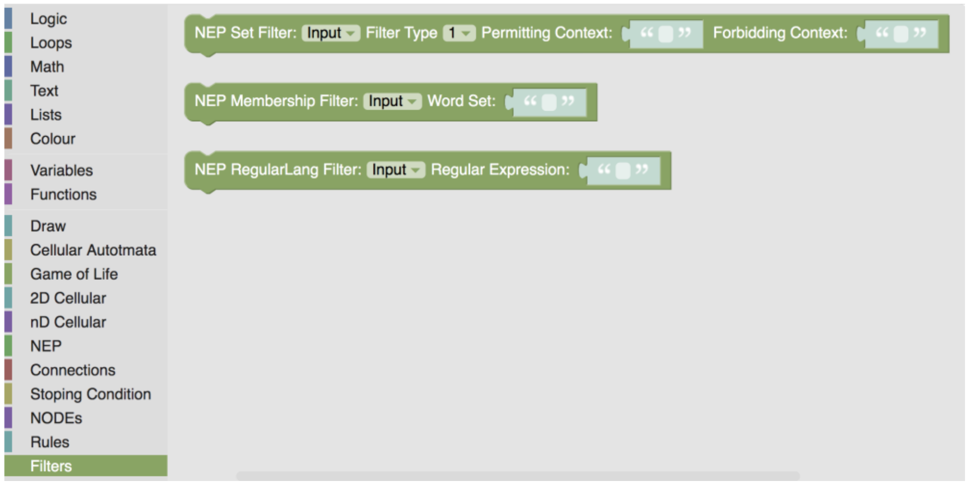

Logic
Loops
Math
Text
Lists
Colour
Variables
Functions
Draw
Cellular Autotmata
Game of Life
2D Cellular
nD Cellular
NEP
Connections
Stoping Condition
NODEs
Rules
Filters
NEP Set Filter: Input Filter Type 1 Permitting Context: " " Forbidding Context: " "
NEP Membership Filter: Input Word Set: " "
NEP RegularLang Filter: Input Regular Expression: " "

Figure 29: Filter blocks.

1. **NEP Set Filter:** This block is used to set any of the four standard types of filter. In has four inputs: two dropdown lists to select the direction (input, output) and the Filter type while the other two to set the Permitting and Forbidding Contexts.
2. **NEP Membership Filter:** It represents the membership filter and has only one input to specify the set of word to be permitted.
3. **NEP RegularLang Filter:** It is used to set regular expressions based filters. This block has two inputs: a dropdown list to set the direction and another input to set the regular expression.

3.2.3 NEPs Visual Representation

The proposed Blockly tool for NEPs provides a visual representation of the static NEP topology while the user is designing it. The topology includes the nodes and the edges that are connecting them. Figure 18 shows an example of the visual representation for the NEPs. The tool represents the nodes as red circles. Each node is labeled with its integer number that is also its sequential number. The edges are represented as black lines connecting the nodes.

In previous figures we have shown several examples of this graphic NEPs representation. Figure 30 shows one more example.

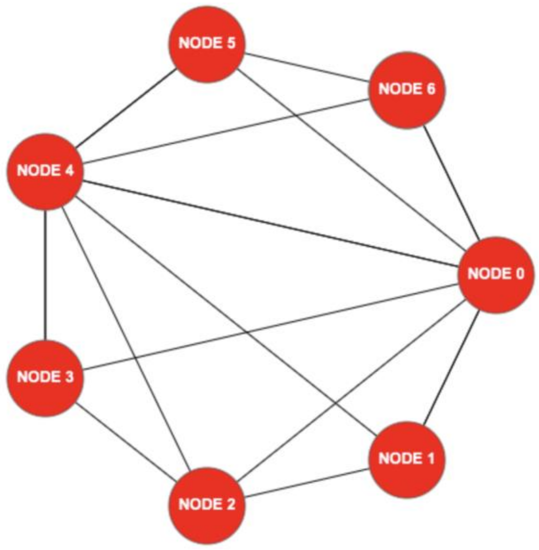
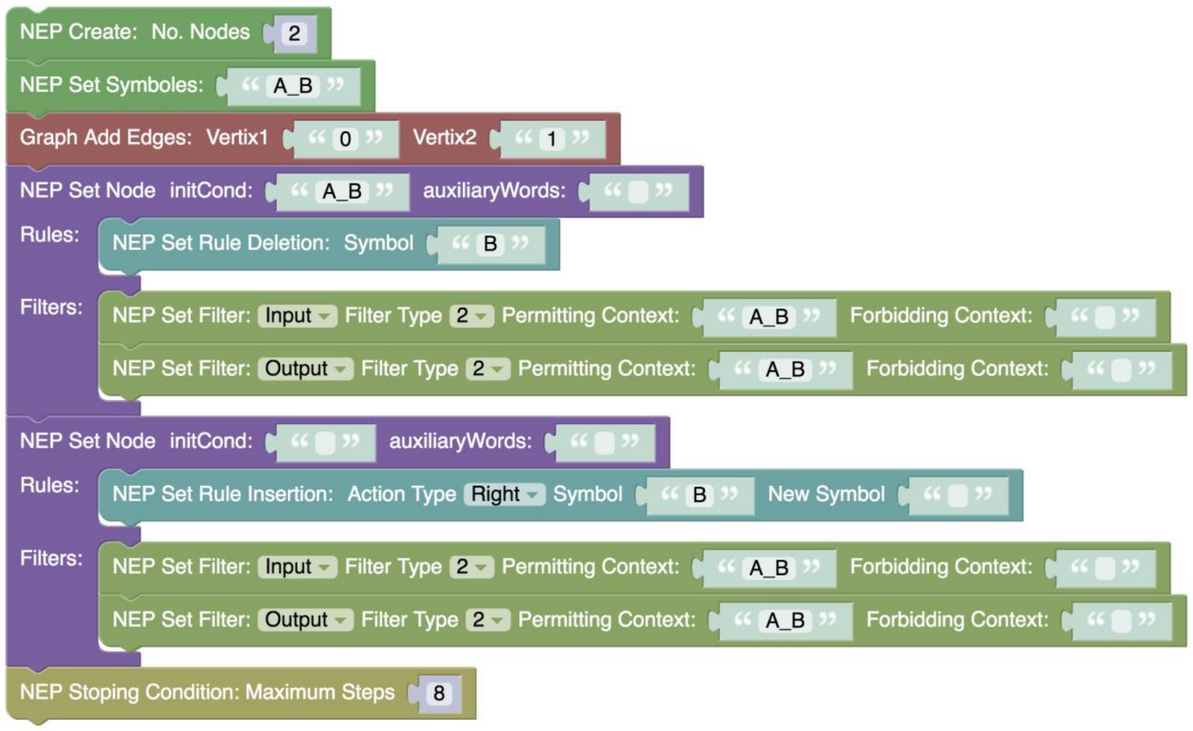


Figure 30: Example of the visual representation of the NEPs.

3.2.4 Example

In this example we demonstrate the system ability to auto generate XML configuration file for a simple NEP. This example was originally presented in [6] and it was used to test the jNEP simulator. In this section, the same example is used to test the new proposed tool. This NEP consists of two nodes that are connected. The first node inserts the symbol B while the second one deletes it. The system has two symbols A and B. The first node starts with the word A_B as an initial condition while the second node has no initial condition. The NEP stops after executing 8 steps. Using the proposed Blockly system, we will design this example. Figure 31 shows the Blockly design for this example.



```
NEP Create: No. Nodes 2
NEP Set Symboles: " A_B "
Graph Add Edges: Vertix1 " 0 " Vertix2 " 1 "
NEP Set Node initCond: " A_B " auxiliaryWords: " "
Rules:   NEP Set Rule Deletion: Symbol " B "
Filters: NEP Set Filter: Input Filter Type 2 Permitting Context: " A_B " Forbidding Context: " "
         NEP Set Filter: Output Filter Type 2 Permitting Context: " A_B " Forbidding Context: " "
NEP Set Node initCond: " " auxiliaryWords: " "
Rules:   NEP Set Rule Insertion: Action Type Right Symbol " B " New Symbol " "
Filters: NEP Set Filter: Input Filter Type 2 Permitting Context: " A_B " Forbidding Context: " "
         NEP Set Filter: Output Filter Type 2 Permitting Context: " A_B " Forbidding Context: " "
NEP Stoping Condition: Maximum Steps 8
```

Figure 31: Simple NEP example on Blockly.

After running this Blockly program. The system successfully generated the desired XML file with accurate configuration. Next is the generated XML code:

```
<?xml version="1.0"?>
<!--NEP Config file-->
<!--This xml file is autogenerated from blockly-->
<NEP nodes="2">
    <ALPHABET symbols="A_B"/>
    <GRAPH>
        <EDGE vertex1="0" vertex2="1"/>
    </GRAPH>
    <EVOLUTIONARY_PROCESSORS>
```

```
        <NODE initCond="A_B">
            <EVOLUTIONARY_RULES>
                <RULE  ruleType="deletion"  actionType="ANY"  symbol="B"
newSymbol=" "/>
            </EVOLUTIONARY_RULES>
            <FILTERS>
                <INPUT         type="2"         permittingContext="A_B"
forbiddingContext=""/>
                <OUTPUT        type="2"         permittingContext="A_B"
forbiddingContext=""/>
            </FILTERS>
        </NODE>
        <NODE initCond="">
            <EVOLUTIONARY_RULES>
                <RULE      ruleType="insertion"      actionType="RIGHT"
symbol="B" newSymbol=""/>
            </EVOLUTIONARY_RULES>
            <FILTERS>
                <INPUT         type="2"         permittingContext="A_B"
forbiddingContext=""/>
                <OUTPUT        type="2"         permittingContext="A_B"
forbiddingContext=""/>
            </FILTERS>
        </NODE>
    </EVOLUTIONARY_PROCESSORS>
    <STOPPING_CONDITION>
        <CONDITION type="MaximumStepsStoppingCondition" maximum="8"/>
    </STOPPING_CONDITION>
```

```
</NEP>
```

3.2.5 Run on Server using http request

Once the NEP is designed and a complete XML configuration code is generated, the designed NEP can be executed on the server. The generated XML is sent to the server using an http request. The user can run NEPs on the server by clicking on the button labeled "RUN ON SERVER". This http request is implemented as a function that is triggered by clicking on this button. When the results are back from the server, it is shown on a separate text area.

Next is the result of running the XML code of the example from the previous sub-section:

```
XML CONFIGURATION FILE LOADED AND PARSED SUCCESSFULLY...

GRAPH INFO PARSED SUCCESSFULLY...

STOPPING CONDITIONS INFO PARSED SUCCESSFULLY...

EVOLUTIONARY PROCESSORS INFO PARSED SUCCESSFULLY...

NEP RUNNING...


***************                            NEP  INITIAL  CONFIGURATION
***************

     --- Evolutionary Processor 0 ---

A_B

     --- Evolutionary Processor 1 ---





evolving: 16
```

```
*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 1
***************

      --- Evolutionary Processor 0 ---

A

      --- Evolutionary Processor 1 ---




output filtering: 0

delivering and input filtering: 0

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:
2 ***************

      --- Evolutionary Processor 0 ---


      --- Evolutionary Processor 1 ---

A



evolving: 0

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 3
***************

      --- Evolutionary Processor 0 ---


      --- Evolutionary Processor 1 ---

A_B
```

```
output filtering: 0

delivering and input filtering: 0

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:
4 ***************

     --- Evolutionary Processor 0 ---

A_B

     --- Evolutionary Processor 1 ---




evolving: 0

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 5
***************

     --- Evolutionary Processor 0 ---

A

     --- Evolutionary Processor 1 ---




output filtering: 0

delivering and input filtering: 0

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:
6 ***************

     --- Evolutionary Processor 0 ---
     --- Evolutionary Processor 1 ---

A




evolving: 0

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 7
***************

     --- Evolutionary Processor 0 ---



     --- Evolutionary Processor 1 ---

A_B




output filtering: 14

delivering and input filtering: 0

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:
8 ***************

     --- Evolutionary Processor 0 ---

A_B

     --- Evolutionary Processor 1 ---





---------------------- NEP has stopped!!! -----------------------
```

```
Stopping                        condition                        found:
net.e_delrosal.jnep.stopping.MaximumStepsStoppingCondition



-------------------------------------------------------------------


We are glad you used jNEP


Orignally designed by: Emilio del Rosal

Made available online through blockly by: Bashar Sami
```

CHAPTER 4. RESULTS AND ANALYSIS

This chapter discusses and evaluates the contributions of this work. The work presents an online developing platform for natural computing algorithm using visual programing tool, namely Blockly. The proposed platform provides software engineering tools for setting up algorithms and we try to also ease the teaching-learning process of these models of computation. The software engineering tools has been implemented mainly on the NEPs part as there is much more software tools already presented for cellular automata. The software designed for NEPs are a set of blocks to implement several types of connections between nodes. These blocks reduce time and complexity in setting up NEPs with fully connected nodes, for instance. In the other hand, cellular automata algorithm has been chosen to test the ease of the process of teaching and learning natural computing algorithms as they are much better-known model. The test has been conducted with students, teachers and researchers. The software tools designed for NEPs have been already presented in the previous chapter with examples. This chapter introduces the experiment conducted to examine the ease of teaching of natural computing algorithm using the proposed Blockly based simulator compared to conventional manual implementation.

4.1 Cellular Automata Blockly Simulator

The Cellular Automata system on Blockly provides an easy methodology for researchers in natural computing projects. The validity of the system was tested by means of an experiment which includes two groups of users. The first group (test group) will be asked to implement CA examples using the proposed Blockly system. The second group (control group) will be asked to implement the same CA examples manually on paper. After the test, both of the groups will be asked to fill a questionnaire that is used to evaluate the ease of use. The data collected from the two groups will be analyzed and compared to each other. The hypothesis is that the first group which uses the proposed Blockly environment will need less time to implement their CA examples. They will also be more likely to use this system in the future in their research. To make sure that the participants are not biased, it is suggested that they should have some sort of experience in IT and that this is not their first time they are introduced to programing languages.

The steps of the experiment can be summarized as follows:

1. Give an introduction about CA and make sure that the participants understand it.
   The introduction includes:
   - The definition of CA.
   - Applications in the industry with examples
   - Natural computing algorithms inspired by CA
   - Simple examples of CA (Elementary Cellular Automata and Game of life). This step includes implementation of simple Elementary Cellular Automata by hand on the white board.
2. Introduce the proposed CA environment using Blockly (This step is for the test group only)
   This step should focus on the ease of implementing CA using the proposed Blockly system. A number of Elementary CA and GoL configurations will be implemented using the direct implementation tools available in the proposed system while the participants

will be asked to implement the example(s) included in the experiment using the generic nD Blockly tool. For this reason, the nD tool will be also presented for the participants of the test group.

3. Introduce the example(s) that they should implement.
4. Participants will implement one example.
5. Implement the example (the proposed example for this experiment is presented in detail in the next section).
6. Collect results of the examples in the form of screenshots and/or text.
7. Present the questionnaire for both groups.
8. Collect the filled forms.
9. Analyzed the data.

4.1.1 Proposed example for the experiment

As discussed above, the experiment will be conducted both, with Blockly and by hand. Implementing CA by hand on paper could be sometimes complicated and might take a long time. Considering this fact, we propose the Elementary Cellular Automata for this experiment. The proposed Blockly system for CA includes direct implementation of ECA where the rules are internally designed, and users need only to select one out of 255 rules available in this type of CA. However, in this experiment, participants of the test group will be asked to design and implement this type of cellular automata using the generic nD tool. Figure 1 and Figure 2 show examples of the Elementary Cellular Automata with the rules' breakdown. One of these examples can be used for this experiment.

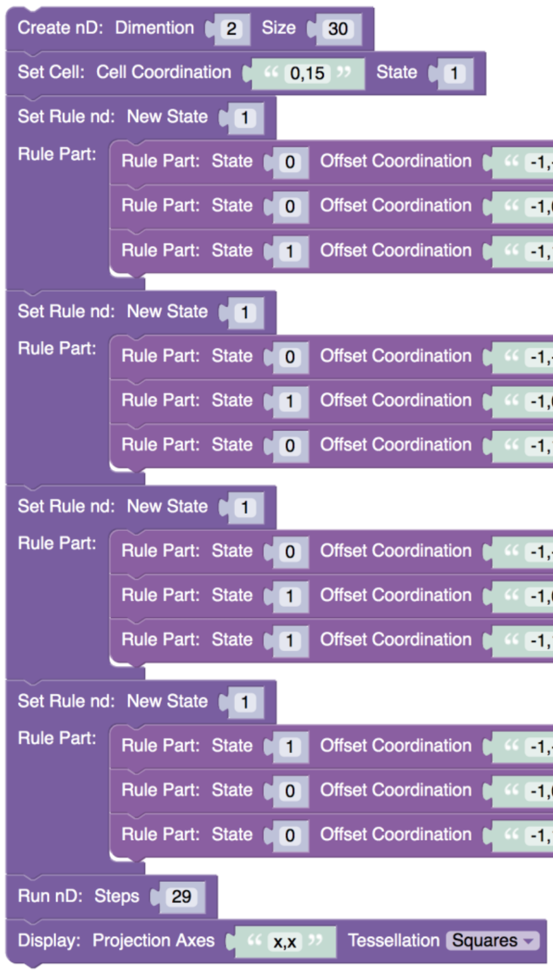
Both of the participants groups will be asked to implement one of the examples shown in Figure 1 (same example for both groups). The test group will implement it using nD Blockly tool while the control group will implement it by hand on paper.

The next sections explain the idle solutions of both groups for Elementary Cellular Automata with rule 30, run it for 29 steps on grid of 30x30.

Ideal Solution for Test Group (using Blockly) for ECA with Rule 30

This group is expected to implement this ECA using the nD Blockly tool. After they implement it, they will be asked to provide a screenshot of the blocks and screenshot of the resulted grid.

Figure 32 and Figure 33 show the blocks and results of this examples respectively.



```
Create nD: Dimention 2 Size 30
Set Cell: Cell Coordination " 0,15 " State 1
Set Rule nd: New State 1
Rule Part:  Rule Part: State 0 Offset Coordination " -1,-1 "
            Rule Part: State 0 Offset Coordination " -1,0 "
            Rule Part: State 1 Offset Coordination " -1,1 "
Set Rule nd: New State 1
Rule Part:  Rule Part: State 0 Offset Coordination " -1,-1 "
            Rule Part: State 1 Offset Coordination " -1,0 "
            Rule Part: State 0 Offset Coordination " -1,1 "
Set Rule nd: New State 1
Rule Part:  Rule Part: State 0 Offset Coordination " -1,-1 "
            Rule Part: State 1 Offset Coordination " -1,0 "
            Rule Part: State 1 Offset Coordination " -1,1 "
Set Rule nd: New State 1
Rule Part:  Rule Part: State 1 Offset Coordination " -1,-1 "
            Rule Part: State 0 Offset Coordination " -1,0 "
            Rule Part: State 0 Offset Coordination " -1,1 "
Run nD: Steps 29
Display: Projection Axes " x,x " Tessellation Squares
```

Figure 32: Blocks for implementation of Elementary Cellular Automata using the nD tool. The CA is implemented with rule 30.

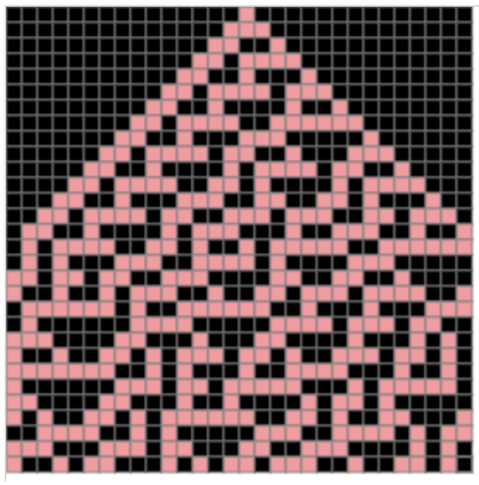


Figure 33: Results of running of the Elementary Cellular Automata in figure 19 for 29 steps on 30x30 grid.

Ideal Solution for Control Group (by hand on paper) for ECA with Rule 30

This group will be asked to implement the Cellular Automata by hand on paper. They will be provided with a 30x30 grid printed on paper to implement the example. An example of the grid is shown in Figure 34. The expected results on this grid should be similar to the grid shown on Figure 33.

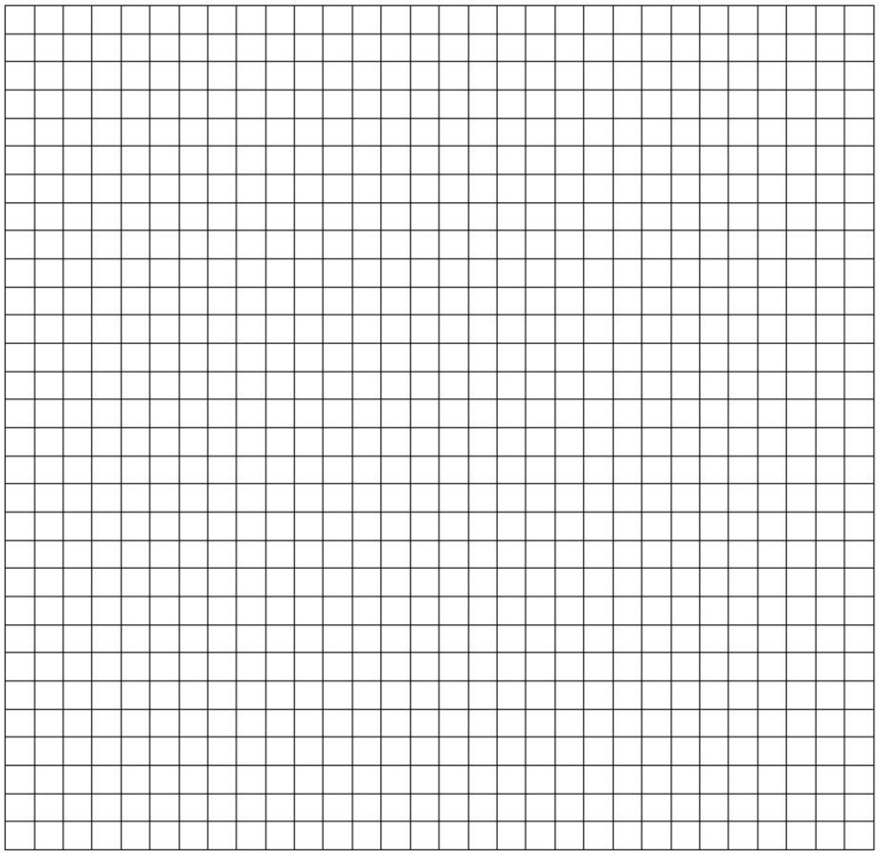


Figure 34: 30x30 empty template grid. This grid will be provided printed on paper for the control group (by hand on paper).

4.1.2 Survey Questions

At the end of the experiment, the participants of both groups (Test and Control) were asked to answer a few questions about their experience. A complete copy of the questionnaire can be found in Appendix A. The questions in the survey are divided into four parts. Next is an explanation about these parts:

1. The questions in the first part collect basic information about the participants like their age, profession, and gender.
2. The second part is “Previous Skills”. In this part information about participants’ previous skills is collected like whether they have heard of CA before or not.

3. The third part is to evaluate the participants' understanding about CA.
4. The fourth part is about the example in the experiment. There are two versions of this part one for each group (Test and Control).

4.2 Conducting the experiment

The experiment was conducted in collaboration with the Computer Science Department at Al-Nahrain University, Baghdad, which is one of the highly ranked universities in Iraq. The participants are mainly faculty and staff of the Computer Science Department. The experiment was conducted at the same department on Jun 12, 2019.

Ten participants were included in this experiment. The participants were divided into two groups (Test and Control). Four participants were allocated to the control group (working by hand on paper) while the other 6 were allocated to the test group (use Blockly tool). Participants enrolled each group following their personal preferences.

The experiment followed the same steps as explained in the beginning of this section. The next section presents and analyzes the results of this experiment.

4.2.1 Experiment's Results

This section discusses the results of the experiment. It is divided into four sub-sections. Each of them presents and analyzes the results of a specific aspect of the experiment. The first, is devoted to participants' previous background and experience while the other three talks about participants answers to the survey's questions.

4.2.1.1 Participants Answers

As presented in the previous section, participants were asked to solve a question in nD cellular automata. The example was selected to be from rule 30 of elementary cellular automata. Four participants were asked to solve the example by hand on paper and the 6 remainder participants were asked to solve it in Blockly using nD cellular tool. Only one out of the four participants who tried to solve the example on paper was successful in solving the example while the other three failed in solving the example. On the other hand, all of the six participants how tried solving the example in Blockly were able to correctly solve it. At the end of the experiment the four participant who tried solving the example on paper were asked to resolve it in Blockly. All of these four participants correctly solved the example when they tried Blockly.

4.2.1.2 Basic Information and Previous Skills

Table 2 shows the results of the first two parts of the survey (basic information and previous skills). The data in the table shows that all of the participants are of age above 30 and they can be categorized into faculty, staff, and one grad student. The data also show that the participants are almost equal in gender category where 40 % of the participants are male and 60% are female. All of the participants have programming language skills. Java programing language appeared to be common among the participants with 80% while other programing languages appeared in the table with 30% or less. It was also found that 60% of the participants have not learned about CA before the time of this experiment.

Table 2: Results of the first two parts of the survey (basic information and previous skills).

Participants	**Age**	**Profession**	**Gender**	**Programming Language**	**Know CA Before**
1	31	Programmer	Male	C#, java script	No
2	33	Eng.	Male	C#, java script, java	No
3	40	Teacher Assistant	Female	Sql, ASP, C++	No
4	38	Teacher	Female	Matlab, java, C++	Yes
5	36	Grad Student	Male	python, java, C#	Yes
6	38	Professor	Female	Visual Basic, C, Java	No
7	44	Teacher Assistant	Female	C, Java, Visual basic	No
8	34	Teacher	Male	MatLab, MathCad, R	No
9	34	Teacher	Female	java, python, C++	Yes
10	34	Teacher Assistant	Female	Java, C, Visual Basic	Yes
Analysis	Mean: 36.2	Faculty and Staff	40% Male 60% Female	Java: 80% C, C++, C#: 30% each Visual Basic: 30%	Yes: 40% No: 60%

4.2.1.3 Evaluation of Understanding

This part collects information about participants understanding of CA after they have already conducted the experiment. The goal is to conduct cross-groups comparison of participants understanding of CA. Two multiple choices questions included in this part as follows:

Q1: How do you evaluate your understanding of CA? (choices 1-5 where 5 is good understanding and 1 is basic understanding)

Q2: How do you evaluate the easiness of CA? (choices 1-5 where 5 is very easy and 1 is difficult)

Table 3 shows the results of these two questions for both groups. Results show that there is no statistically significant difference between the two groups. This could be a result of the way the experiment was conducted where both of the groups existed in the same room while the researcher was presenting CA. This means that they all received the exact same explanation of the topic. The other reason is that the control group (those who used paper) were asked to redo the experiment using Blockly after they finished working on paper. This probably lead to enhance their understanding of CA because they used both ways. Even though, the results of this section show no advantages when using Blockly but the results of the last section of the survey shows significant advantage to Blockly.

Table 3: Results of the questions in the third part of the survey for both groups.

Participants	**Group** **Test (Blockly)** **Control (by hand)**	**Q1: Understanding of CA**	**Q2: Easiness of CA**
1	Control	4	4
2	Control	4	5
3	Control	5	4
4	Test	2	1
5	Control	2	4
6	Test	3	4
7	Test	3	4
8	Test	2	3
9	Test	4	5
10	Test	4	4
Analysis	Test mean: 60% Control mean: 40%	Test mean: 3 Control mean: 3.75	Test mean: 3.5 Control mean: 4.25

4.2.1.4 Evaluation of Ease of Use

The last part of the survey tests how using Blockly CA tool affected participants experience with CA. This part includes three questions that targets time consumption, effects of tool on understanding, and future interest in CA. The questions are:

Q1: How long did you spend in solving the question in today's experiment? (minutes)

Q2: How do you think (Blockly or on paper) affected your understanding of Cellular Automata?

Q3: Based on today's experience, do you think that you will use Cellular Automata in your work in the future?

Table 4 shows the results of the questions of the last part in survey for both groups. It can be easily seen how Blockly has significant advantages over paper group. It can be also seen that all participants tried using Blockly. This is because the participants who solved the example on paper were asked to resolve it using Blockly. For this reason, this table shows that all participants answered the questions related to Blockly while only four answered the questions related to paper use. Next is analysis of this results in terms of time consumption, understanding of CA, and future interest.

Time consumption: The time consumed to solve the example in Blockly is in order of a few minutes (mean 2.9 minutes) while the time consumed to solve the same example on paper is in orders of hours (mean 1:33 hours). These numbers show that solving the example in Blockly about 32 times faster than solving it on paper. Numbers also showed that participants who tried solving the example on paper first took longer time when they tried to solve it in Blockly compared to those who directly started using Blockly. This could mean that they were confused because of using of the manual try on paper.

Understanding of CA: Results showed that all participants who tried to solve the example by hand on paper think that it made it difficult for them to understand CA while those who tried using Blockly think that using Blockly made it easier for them to understand CA. It can be also seen from the results that the 4 participants who tried the paper way all change their answers from “made it difficult” to “made it easy” when they switched to using Blockly.

Future interest: Results showed that 60% of participants who tried Blockly expressed their interests in using CA in their work in the future. On the other hand, all participants from the control group (on paper) showed no interest in using CA in the future. However, three out of four participants change their interest when they tried Blockly the second time.

Table 4: Effects of Blockly on time consumption, understanding of CA, and future interest in CA.

Participants	**Q1: Time consumed**		**Q2: Effect of understanding**		**Q3: Use in the future**	
	Blockly	Paper	Blockly	Paper	Blockly	Paper
1	8 min	2:15 hour	made it easy	made it very difficult	Yes	No
2	6 min	2:30 hour	made it easy	made it difficult	Yes	No
3	3 min	1 hour	made it easy	made it difficult	Yes	No
4	1 min	x	made it easy	x	Yes	x
5	4 min	28 min	made it super easy	made it difficult	No	No
6	1 min	x	made it easy	x	No	x
7	1 min	x	made it easy	x	No	x
8	2 min	x	made it easy	x	No	x

9	1 min	x	made it easy	x	Yes	x
10	2 min	x	made it easy	x	Yes	x
Analysis	Blockly mean: 2.9 min Paper mean: 1:33 hours		Blockly mean: Made it easy Paper mean: Made it difficult		Blockly: Yes 60% Paper: No 100%	

CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

This work proposed a natural computer programming (for CA and NEPs) environment platform using Blockly. The platform is a web based tool that provides simulators for two major natural computing systems: Cellular Automata (CA) and Network of Evolutionary Processors (NEPs). CA is a grid of cells that changes their state based on pre-specified rules through a number of discrete time steps. CA has been used in several applications such as modeling of physical system, traffic control, and flood propagation. CA programming blocks presented in this work provide the ability to design and implement several types of CA including Elementary cellular automata, 2D cellular automata, and nD cellular automata. The nD CA is a generic simulation for any type of CA. It provides flexibility for the user in terms of number of dimensions, size, and the rules. The nD tools also provide potentially unlimited number of states that can be represented in the CA grid. The tool also provides a graphical representation of CA's grid through projection for any CA that has 3 or more dimensions. NEPs is one of the promising natural inspired computing models. It consists of a finite number of small processors (nodes). The processors are connected through edges (links) to transfer data between the processors. Data is a series of symbols called words which are processed inside the nodes by simple operations such insertion, deletion or substitution. NEPs is powerful at processing NP-complete problems compared to conventional computers. A NEPs Blockly programming environment is presented in this work. It provides the ability to design and simulate NEPs. Blocks are used as flexible user interface to enter NEPs specifications. The blocks automatically generate a standard XML configurations code which can be sent to the server side of the simulator for implementation. The tool also provides a graphical representation for the static topology of the system.

Both CA and NEPs Blockly programming environments have been tested in several rather academic examples. We have tested our hypothesis about improving the learning process of natural programming by means of blocks programming languages by means of a real activity in the classroom. Results of the experiment showed that the CA Blockly simulator outperforms traditional manual methods of implementing CA. It also showed that the proposed environment has desired features such as ease of use and decreases learning time. The NEPs part of the system has been tested against several applications. It showed that it provides a flexible designing tool for NEPs. It outperforms traditional XML coding methods in terms of ease of use and designing time. In addition we have designed specific high level constructs that automatize in some way the specific of complex NEPs' topologies by hand. They could be considered as embryonic software engineering tools to program NEPs.

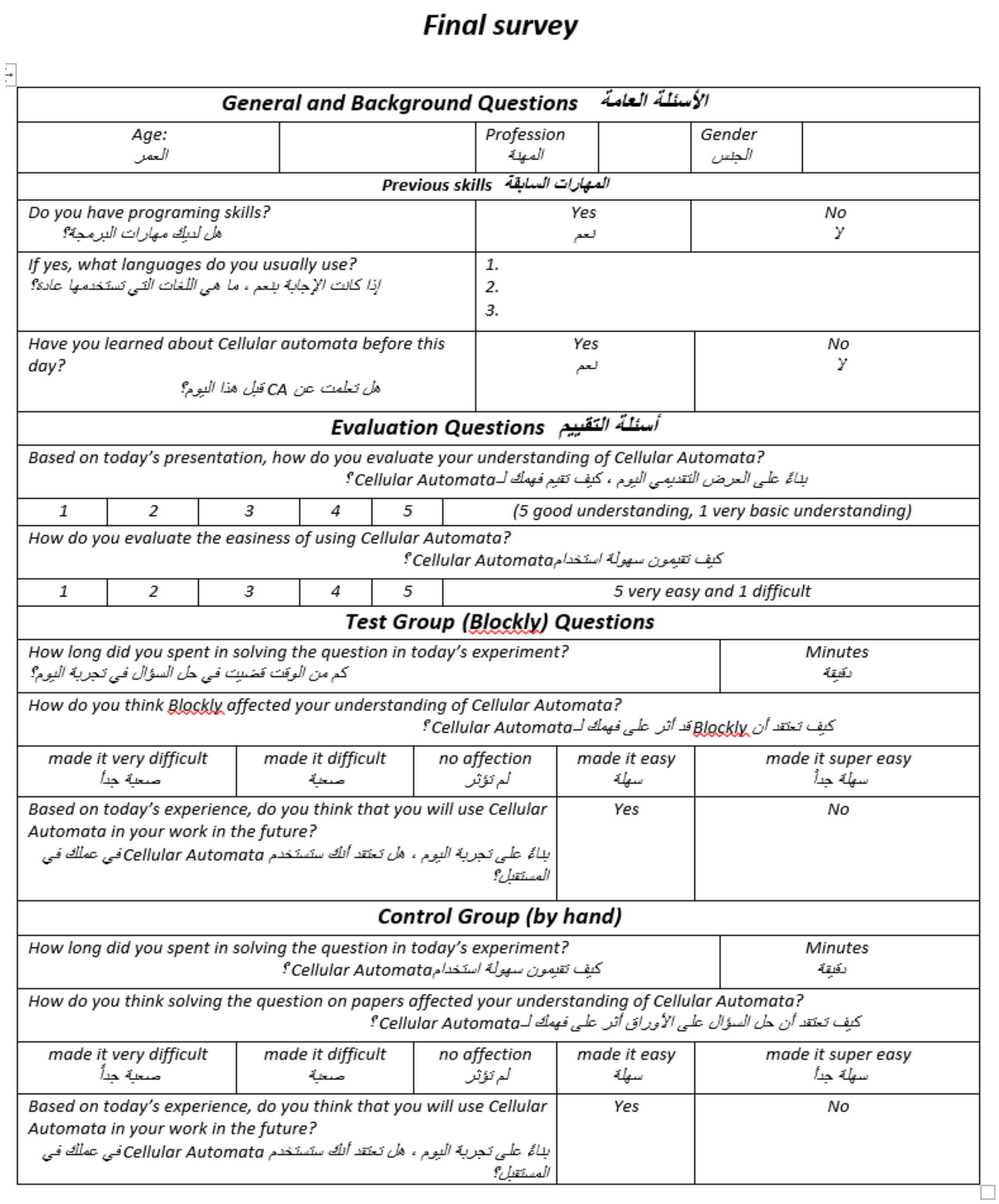
5.2 Future Work

Our environment can be considered as a generic platform for CA and NEPs. They can be expanded in several directions. The CA part can be expanded to cover grids with different tessellations such as triangles and hexagons. The current CA simulator executes the work locally inside the web browser which limits the performance due to the constraints in memory and processing power. A server-side simulator can be added to the tool to improve the performance. A parallel processing server can be used to accelerate processing speed.

The NEPs environment can also be expanded in several directions. As in CA simulator, a parallel server-side simulator is highly recommended to improve the performance of the NEPs simulator. The tool is designed with such expansion is in mind. The current server-side simulator sends results back to the client side as one big chunk at the end of the execution. This can be updated to send results back to the client side in small chunks while the current job is executed instead of waiting until the whole job completely executed. Such updated would dramatically decrease the communication latency which improves the performance of the whole tool.

APPENDIX A QUESTIONNAIRE FOR EXPERIMENT

This questionnaire is used during the experiment to test the ease of teaching and learning of cellular automata using the proposed simulator using Blockly.



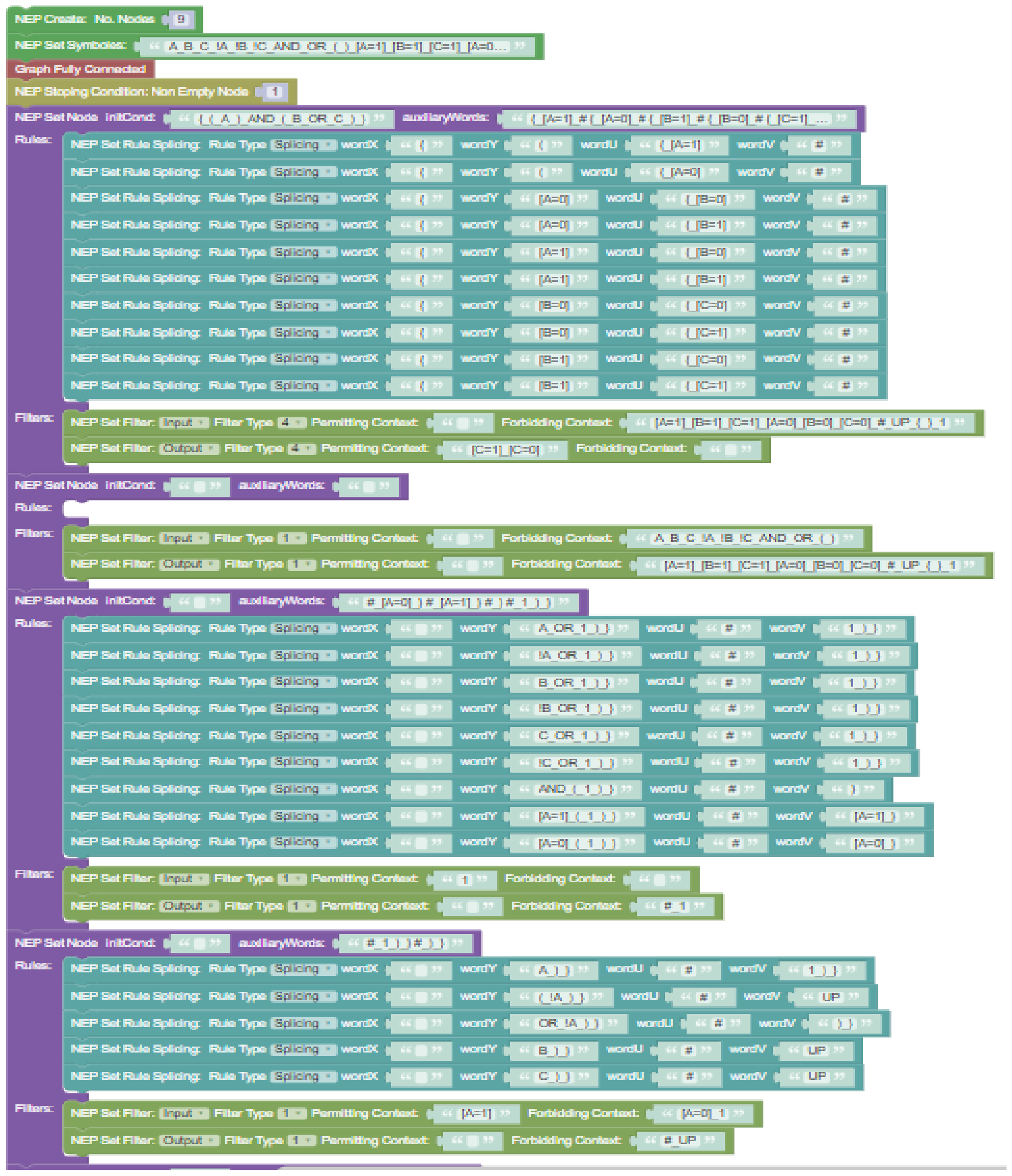
Final survey

General and Background Questions الأسئلة العامة					
Age: العمر		Profession المهنة		Gender الجنس	
Previous skills المهارات السابقة					
Do you have programing skills? هل لديك مهارات البرمجة؟		Yes نعم		No لا	
If yes, what languages do you usually use? إذا كانت الإجابة بنعم ، ما هي اللغات التي تستخدمها عادة؟		1. 2. 3.			
Have you learned about Cellular automata before this day? هل تعلمت عن CA قبل هذا اليوم؟		Yes نعم		No لا	
Evaluation Questions أسئلة التقييم					
Based on today's presentation, how do you evaluate your understanding of Cellular Automata? بناءً على العرض التقديمي اليوم ، كيف تقيم فهمك لـCellular Automata؟					
1	2	3	4	5	(5 good understanding, 1 very basic understanding)
How do you evaluate the easiness of using Cellular Automata? كيف تقيمون سهولة استخدامCellular Automata؟					
1	2	3	4	5	5 very easy and 1 difficult
Test Group (Blockly) Questions					
How long did you spent in solving the question in today's experiment? كم من الوقت قضيت في حل السؤال في تجربة اليوم؟				Minutes دقيقة	
How do you think Blockly affected your understanding of Cellular Automata? كيف تعتقد أن Blockly قد أثر على فهمك لـCellular Automata؟					
made it very difficult صعبة جداً	made it difficult صعبة	no affection لم تؤثر	made it easy سهلة	made it super easy سهلة جداً	
Based on today's experience, do you think that you will use Cellular Automata in your work in the future? بناءً على تجربة اليوم ، هل تعتقد أنك ستستخدم Cellular Automata في عملك في المستقبل؟			Yes	No	
Control Group (by hand)					
How long did you spent in solving the question in today's experiment? كيف تقيمون سهولة استخدامCellular Automata؟				Minutes دقيقة	
How do you think solving the question on papers affected your understanding of Cellular Automata? كيف تعتقد أن حل السؤال على الأوراق أثر على فهمك لـCellular Automata؟					
made it very difficult صعبة جداً	made it difficult صعبة	no affection لم تؤثر	made it easy سهلة	made it super easy سهلة جداً	
Based on today's experience, do you think that you will use Cellular Automata in your work in the future? بناءً على تجربة اليوم ، هل تعتقد أنك ستستخدم Cellular Automata في عملك في المستقبل؟			Yes	No	

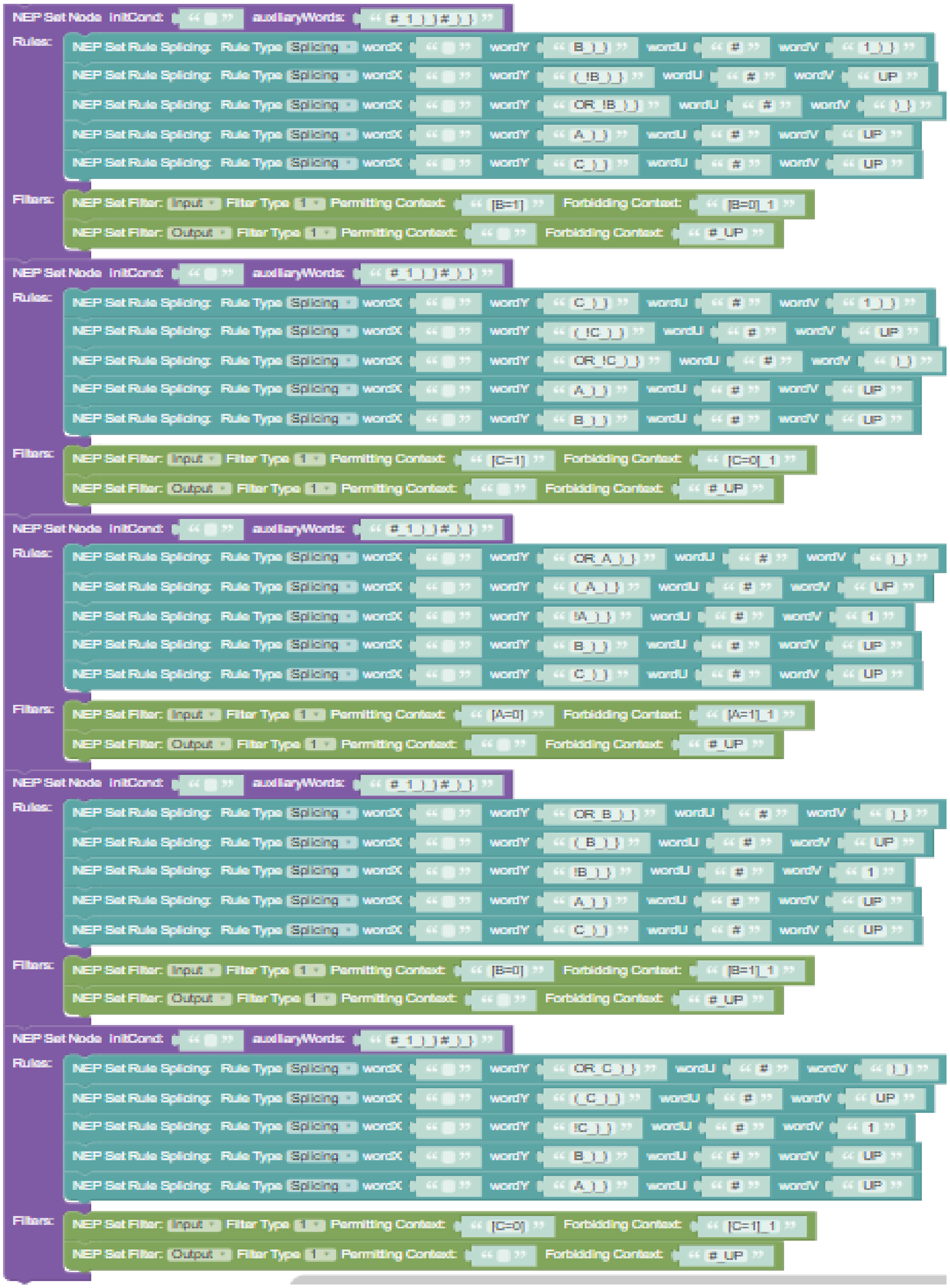
APPENDIX B. NEPs BLOCKLY EXAMPLES

These examples have been introduced in [6] using manual writing of XML configurations. In this appendix, we introduce the them using Blockly implementation.

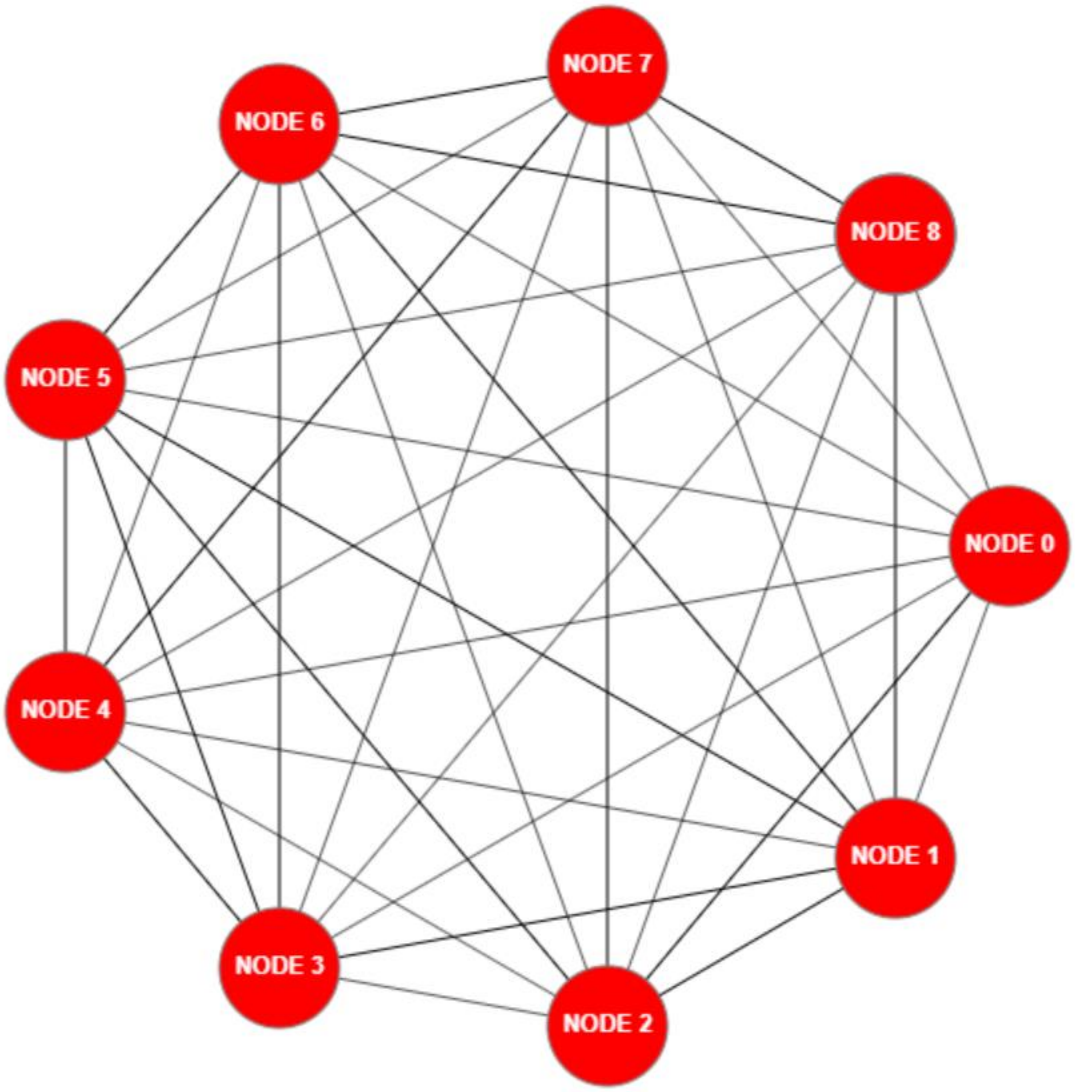
1. SAT problem



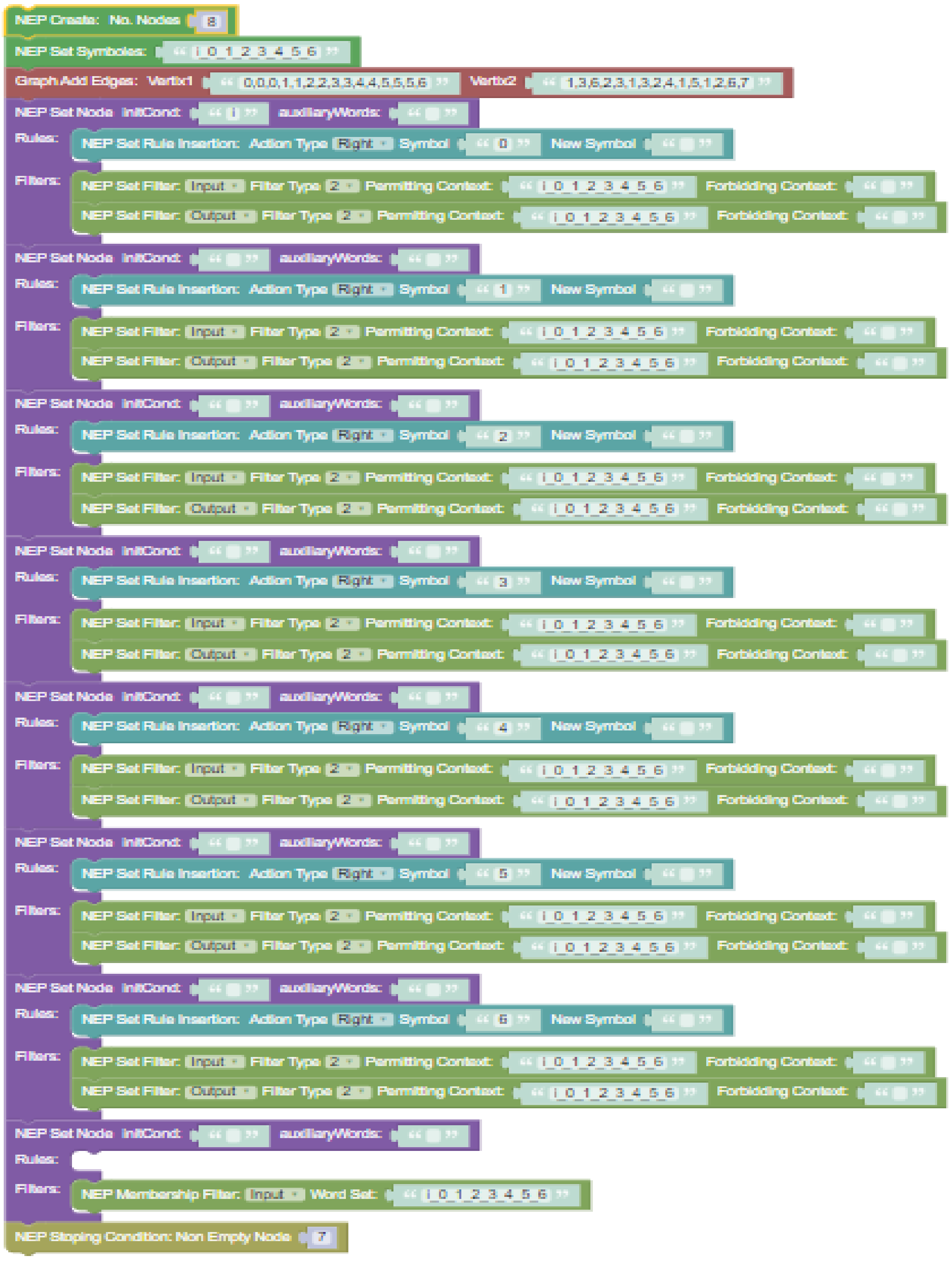
```
NEP Create: No. Nodes 9
NEP Set Symboles: " A_B_C_!A_!B_!C_AND_OR_(_)_[A=1]_[B=1]_[C=1]_[A=0... "
Graph Fully Connected
NEP Stoping Condition: Non Empty Node 1
NEP Set Node InitCond: " {_(_A_)_AND_(_B_OR_C_)_} " auxiliaryWords: " {_[A=1]_#{_[A=0]_#{_[B=1]_#{_[B=0]_#{_[C=1]_... "
Rules: NEP Set Rule Splicing: Rule Type Splicing wordX " { " wordY " ( " wordU " {_[A=1] " wordV " # "
       NEP Set Rule Splicing: Rule Type Splicing wordX " { " wordY " ( " wordU " {_[A=0] " wordV " # "
       NEP Set Rule Splicing: Rule Type Splicing wordX " { " wordY " [A=0] " wordU " {_[B=0] " wordV " # "
       NEP Set Rule Splicing: Rule Type Splicing wordX " { " wordY " [A=0] " wordU " {_[B=1] " wordV " # "
       NEP Set Rule Splicing: Rule Type Splicing wordX " { " wordY " [A=1] " wordU " {_[B=0] " wordV " # "
       NEP Set Rule Splicing: Rule Type Splicing wordX " { " wordY " [A=1] " wordU " {_[B=1] " wordV " # "
       NEP Set Rule Splicing: Rule Type Splicing wordX " { " wordY " [B=0] " wordU " {_[C=0] " wordV " # "
       NEP Set Rule Splicing: Rule Type Splicing wordX " { " wordY " [B=0] " wordU " {_[C=1] " wordV " # "
       NEP Set Rule Splicing: Rule Type Splicing wordX " { " wordY " [B=1] " wordU " {_[C=0] " wordV " # "
       NEP Set Rule Splicing: Rule Type Splicing wordX " { " wordY " [B=1] " wordU " {_[C=1] " wordV " # "
Filters: NEP Set Filter: Input Filter Type 4 Permitting Context: "  " Forbidding Context: " [A=1]_[B=1]_[C=1]_[A=0]_[B=0]_[C=0]_#_UP_(_)_1 "
         NEP Set Filter: Output Filter Type 4 Permitting Context: " [C=1]_[C=0] " Forbidding Context: "  "
NEP Set Node InitCond: "  " auxiliaryWords: "  "
Rules:
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: "  " Forbidding Context: " A_B_C_!A_!B_!C_AND_OR_(_) "
         NEP Set Filter: Output Filter Type 1 Permitting Context: "  " Forbidding Context: " [A=1]_[B=1]_[C=1]_[A=0]_[B=0]_[C=0]_#_UP_(_)_1 "
NEP Set Node InitCond: "  " auxiliaryWords: " #_[A=0]_)_#_[A=1]_)_#_)_#_1_)_) "
Rules: NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " A_OR_1_)_) " wordU " # " wordV " 1_)_) "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " !A_OR_1_)_) " wordU " # " wordV " 1_)_) "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " B_OR_1_)_) " wordU " # " wordV " 1_)_) "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " !B_OR_1_)_) " wordU " # " wordV " 1_)_) "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " C_OR_1_)_) " wordU " # " wordV " 1_)_) "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " !C_OR_1_)_) " wordU " # " wordV " 1_)_) "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " AND_(_1_)_) " wordU " # " wordV " ) "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " [A=1]_(_1_)_) " wordU " # " wordV " [A=1]_) "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " [A=0]_(_1_)_) " wordU " # " wordV " [A=0]_) "
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: " 1 " Forbidding Context: "  "
         NEP Set Filter: Output Filter Type 1 Permitting Context: "  " Forbidding Context: " #_1 "
NEP Set Node InitCond: "  " auxiliaryWords: " #_1_)_)#_)_) "
Rules: NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " A_)_) " wordU " # " wordV " 1_)_) "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " (_!A_)_) " wordU " # " wordV " UP "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " OR_!A_)_) " wordU " # " wordV " )_) "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " B_)_) " wordU " # " wordV " UP "
       NEP Set Rule Splicing: Rule Type Splicing wordX "  " wordY " C_)_) " wordU " # " wordV " UP "
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: " [A=1] " Forbidding Context: " [A=0]_1 "
         NEP Set Filter: Output Filter Type 1 Permitting Context: "  " Forbidding Context: " #_UP "
```



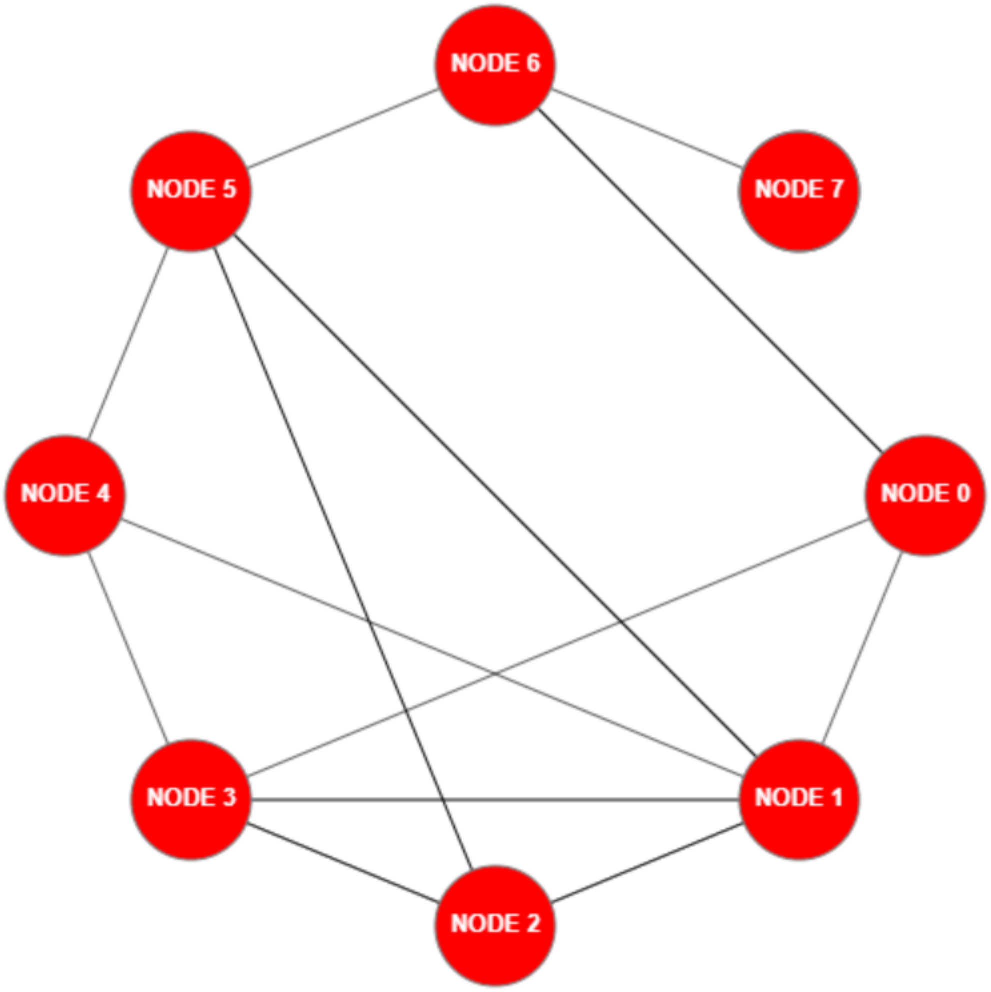
```
NEP Set Node  InitCond: " "  auxiliaryWords: " #_1_}_}#_}_} "
Rules:  NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " B_}_} "  wordU " # "  wordV " 1_}_} "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " (_!B_}_} "  wordU " # "  wordV " UP "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " OR_!B_}_} "  wordU " # "  wordV " }_} "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " A_}_} "  wordU " # "  wordV " UP "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " C_}_} "  wordU " # "  wordV " UP "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " [B=1] "  Forbidding Context: " [B=0]_1 "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " "  Forbidding Context: " #_UP "
NEP Set Node  InitCond: " "  auxiliaryWords: " #_1_}_}#_}_} "
Rules:  NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " C_}_} "  wordU " # "  wordV " 1_}_} "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " (_!C_}_} "  wordU " # "  wordV " UP "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " OR_!C_}_} "  wordU " # "  wordV " }_} "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " A_}_} "  wordU " # "  wordV " UP "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " B_}_} "  wordU " # "  wordV " UP "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " [C=1] "  Forbidding Context: " [C=0]_1 "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " "  Forbidding Context: " #_UP "
NEP Set Node  InitCond: " "  auxiliaryWords: " #_1_}_}#_}_} "
Rules:  NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " OR_A_}_} "  wordU " # "  wordV " }_} "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " (_A_}_} "  wordU " # "  wordV " UP "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " !A_}_} "  wordU " # "  wordV " 1 "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " B_}_} "  wordU " # "  wordV " UP "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " C_}_} "  wordU " # "  wordV " UP "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " [A=0] "  Forbidding Context: " [A=1]_1 "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " "  Forbidding Context: " #_UP "
NEP Set Node  InitCond: " "  auxiliaryWords: " #_1_}_}#_}_} "
Rules:  NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " OR_B_}_} "  wordU " # "  wordV " }_} "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " (_B_}_} "  wordU " # "  wordV " UP "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " !B_}_} "  wordU " # "  wordV " 1 "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " A_}_} "  wordU " # "  wordV " UP "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " C_}_} "  wordU " # "  wordV " UP "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " [B=0] "  Forbidding Context: " [B=1]_1 "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " "  Forbidding Context: " #_UP "
NEP Set Node  InitCond: " "  auxiliaryWords: " #_1_}_}#_}_} "
Rules:  NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " OR_C_}_} "  wordU " # "  wordV " }_} "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " (_C_}_} "  wordU " # "  wordV " UP "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " !C_}_} "  wordU " # "  wordV " 1 "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " B_}_} "  wordU " # "  wordV " UP "
        NEP Set Rule Splicing:  Rule Type Splicing  wordX " "  wordY " A_}_} "  wordU " # "  wordV " UP "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " [C=0] "  Forbidding Context: " [C=1]_1 "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " "  Forbidding Context: " #_UP "
```


NODE 7
NODE 6
NODE 8
NODE 5
NODE 0
NODE 4
NODE 1
NODE 3
NODE 2

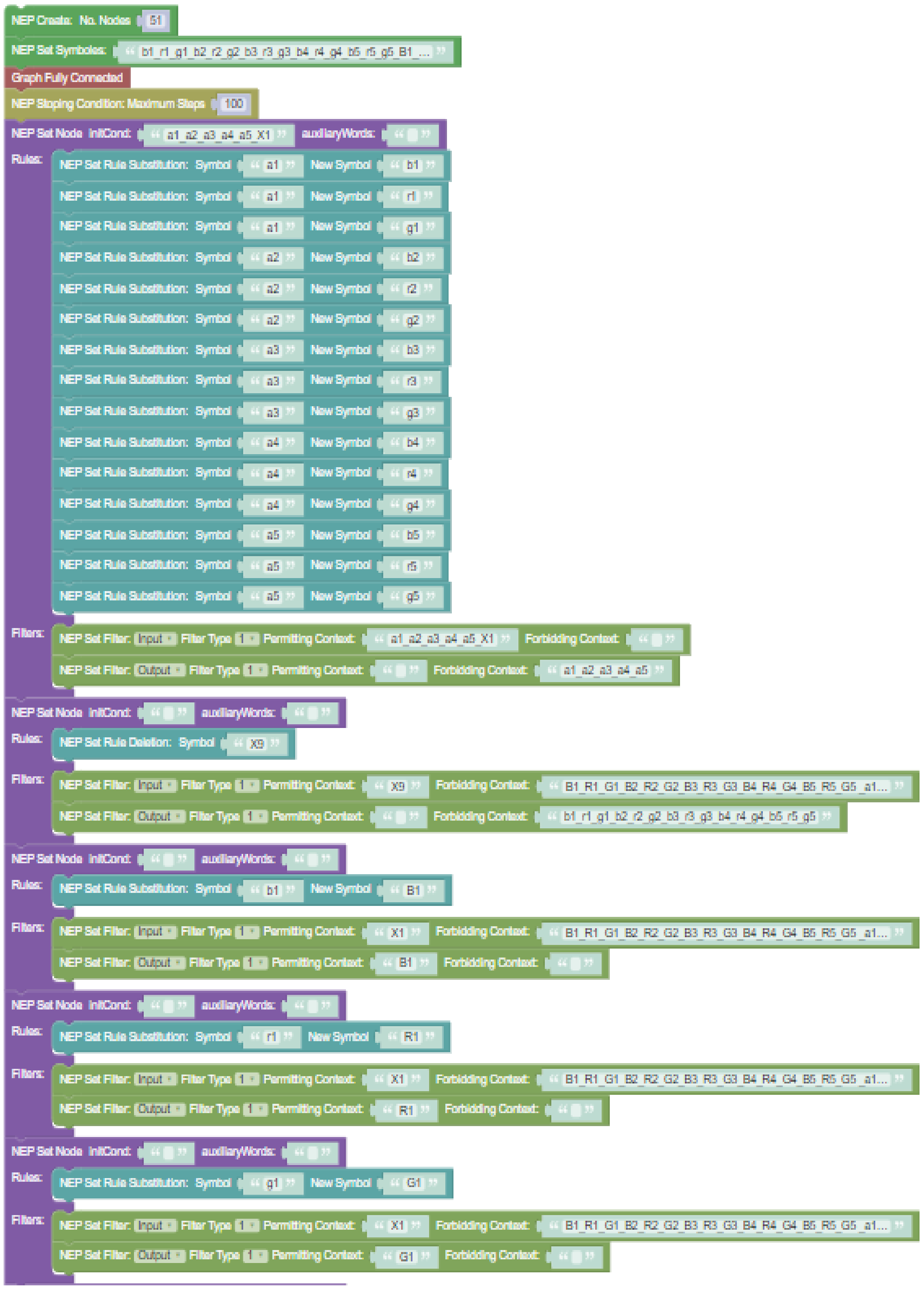
2. Hamiltonian path problem



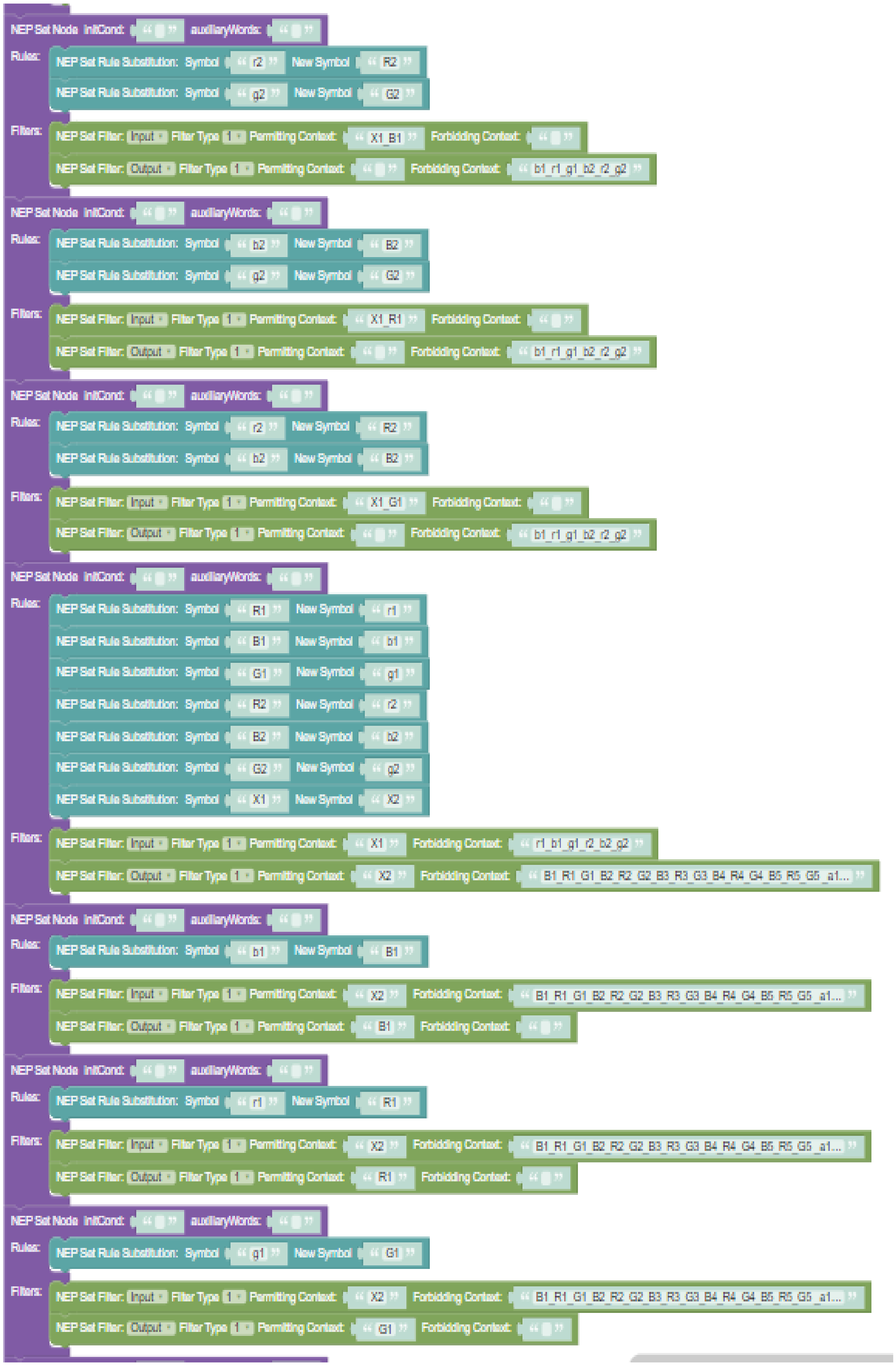
```
NEP Create: No. Nodes 8
NEP Set Symboles: " I 0 1 2 3 4 5 6 "
Graph Add Edges: Vertix1 " 0,0,0,1,1,2,2,3,3,4,4,5,5,5,6 " Vertix2 " 1,3,6,2,3,1,3,2,4,1,5,1,2,6,7 "
NEP Set Node InitCond: " I " auxiliaryWords: " "
  Rules:   NEP Set Rule Insertion: Action Type Right Symbol " 0 " New Symbol " "
  Filters: NEP Set Filter: Input Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
           NEP Set Filter: Output Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
NEP Set Node InitCond: " " auxiliaryWords: " "
  Rules:   NEP Set Rule Insertion: Action Type Right Symbol " 1 " New Symbol " "
  Filters: NEP Set Filter: Input Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
           NEP Set Filter: Output Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
NEP Set Node InitCond: " " auxiliaryWords: " "
  Rules:   NEP Set Rule Insertion: Action Type Right Symbol " 2 " New Symbol " "
  Filters: NEP Set Filter: Input Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
           NEP Set Filter: Output Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
NEP Set Node InitCond: " " auxiliaryWords: " "
  Rules:   NEP Set Rule Insertion: Action Type Right Symbol " 3 " New Symbol " "
  Filters: NEP Set Filter: Input Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
           NEP Set Filter: Output Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
NEP Set Node InitCond: " " auxiliaryWords: " "
  Rules:   NEP Set Rule Insertion: Action Type Right Symbol " 4 " New Symbol " "
  Filters: NEP Set Filter: Input Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
           NEP Set Filter: Output Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
NEP Set Node InitCond: " " auxiliaryWords: " "
  Rules:   NEP Set Rule Insertion: Action Type Right Symbol " 5 " New Symbol " "
  Filters: NEP Set Filter: Input Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
           NEP Set Filter: Output Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
NEP Set Node InitCond: " " auxiliaryWords: " "
  Rules:   NEP Set Rule Insertion: Action Type Right Symbol " 6 " New Symbol " "
  Filters: NEP Set Filter: Input Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
           NEP Set Filter: Output Filter Type 2 Permitting Context: " I 0 1 2 3 4 5 6 " Forbidding Context: " "
NEP Set Node InitCond: " " auxiliaryWords: " "
  Rules:
  Filters: NEP Membership Filter: Input Word Set: " I 0 1 2 3 4 5 6 "
NEP Stoping Condition: Non Empty Node 7
```


NODE 6
NODE 5
NODE 7
NODE 4
NODE 0
NODE 3
NODE 1
NODE 2

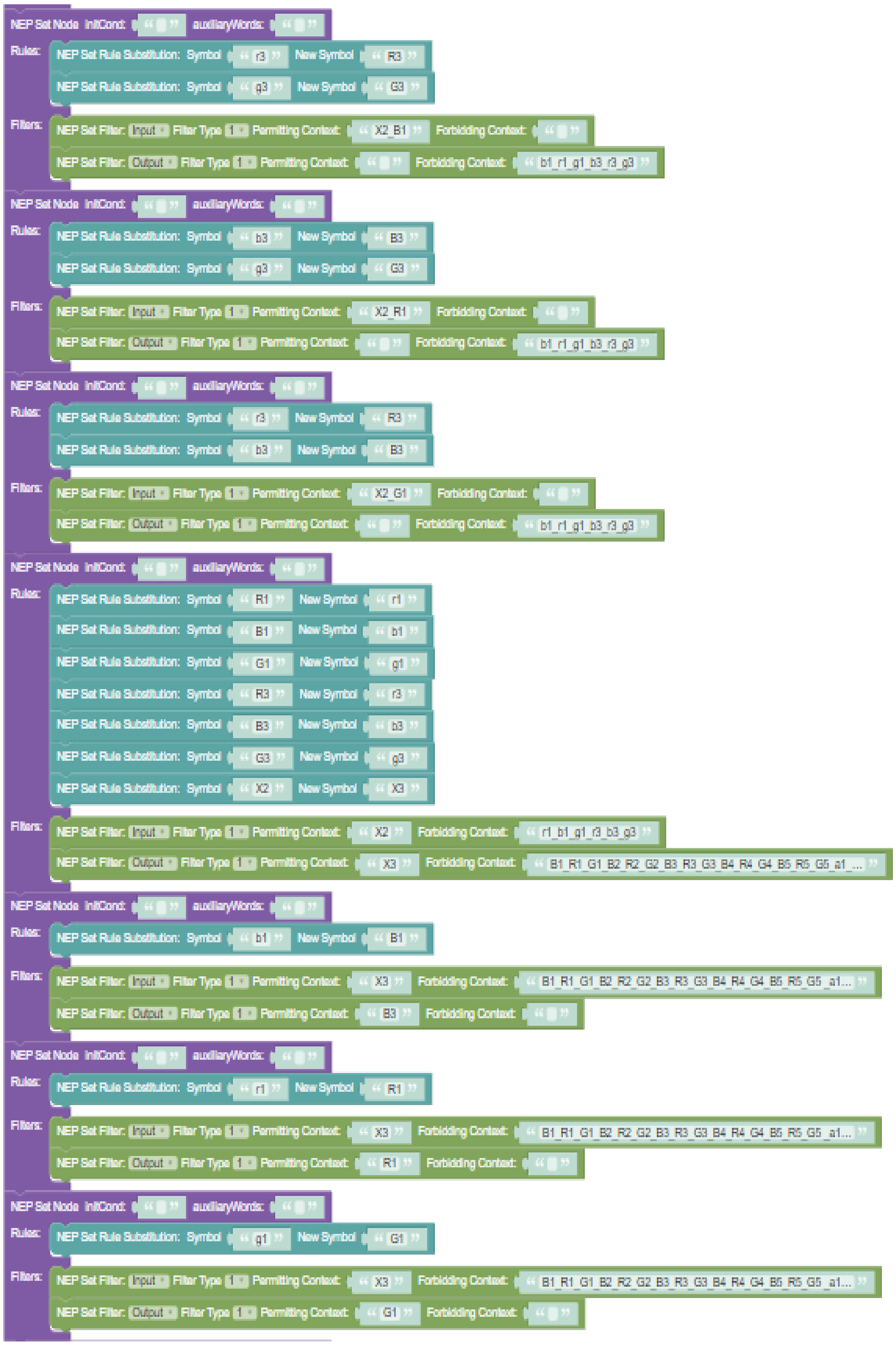
3. 3Color



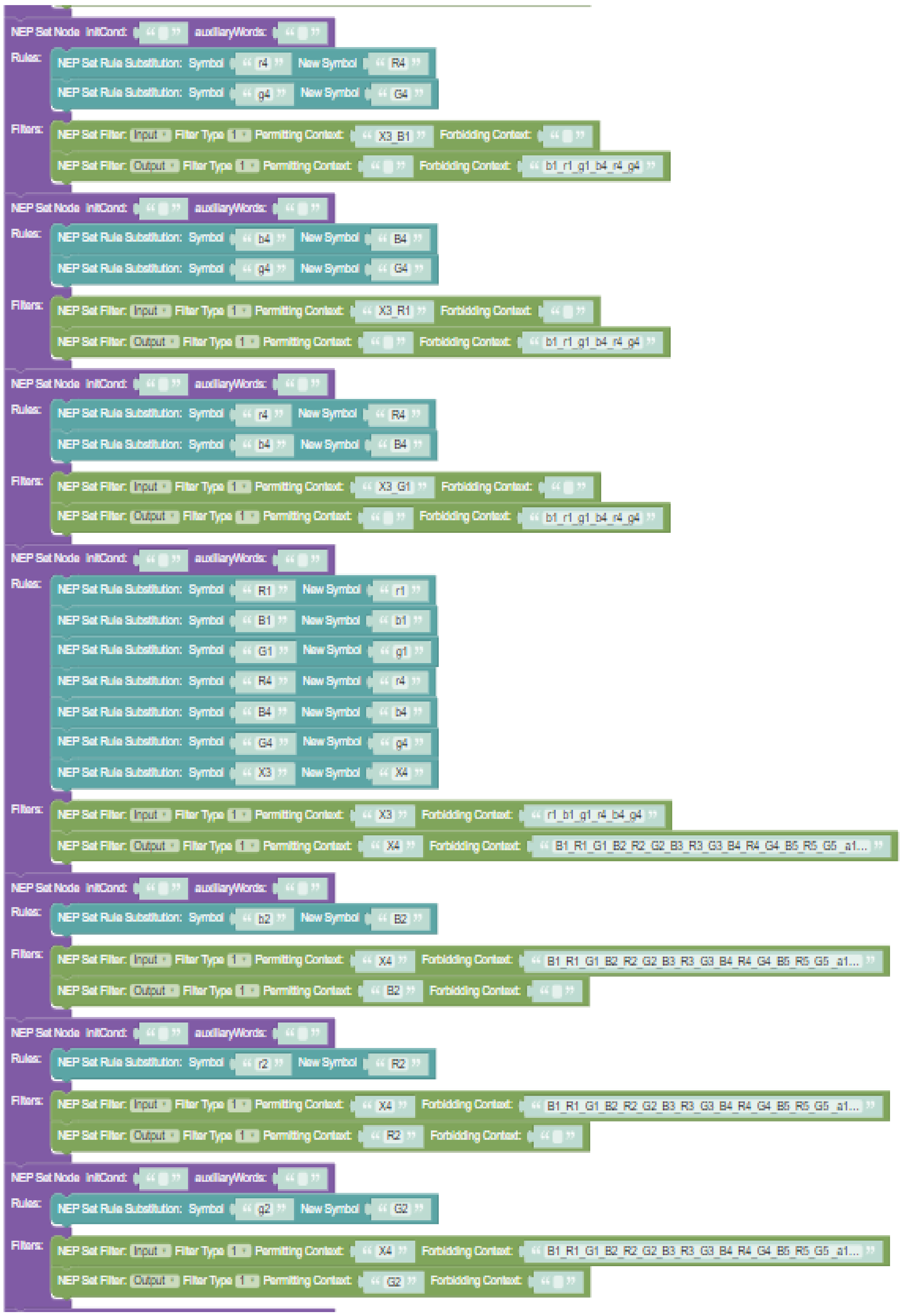
```
NEP Create: No. Nodes 51
NEP Set Symboles: " b1_r1_g1_b2_r2_g2_b3_r3_g3_b4_r4_g4_b5_r5_g5_B1_... "
Graph Fully Connected
NEP Stoping Condition: Maximum Steps 100
NEP Set Node InitCond: " a1_a2_a3_a4_a5_X1 " auxiliaryWords: "  "
Rules:  NEP Set Rule Substitution: Symbol " a1 " New Symbol " b1 "
        NEP Set Rule Substitution: Symbol " a1 " New Symbol " r1 "
        NEP Set Rule Substitution: Symbol " a1 " New Symbol " g1 "
        NEP Set Rule Substitution: Symbol " a2 " New Symbol " b2 "
        NEP Set Rule Substitution: Symbol " a2 " New Symbol " r2 "
        NEP Set Rule Substitution: Symbol " a2 " New Symbol " g2 "
        NEP Set Rule Substitution: Symbol " a3 " New Symbol " b3 "
        NEP Set Rule Substitution: Symbol " a3 " New Symbol " r3 "
        NEP Set Rule Substitution: Symbol " a3 " New Symbol " g3 "
        NEP Set Rule Substitution: Symbol " a4 " New Symbol " b4 "
        NEP Set Rule Substitution: Symbol " a4 " New Symbol " r4 "
        NEP Set Rule Substitution: Symbol " a4 " New Symbol " g4 "
        NEP Set Rule Substitution: Symbol " a5 " New Symbol " b5 "
        NEP Set Rule Substitution: Symbol " a5 " New Symbol " r5 "
        NEP Set Rule Substitution: Symbol " a5 " New Symbol " g5 "
Filters: NEP Set Filter: Input Filter Type 1 Permiting Context: " a1_a2_a3_a4_a5_X1 " Forbidding Context: "  "
         NEP Set Filter: Output Filter Type 1 Permiting Context: "  " Forbidding Context: " a1_a2_a3_a4_a5 "
NEP Set Node InitCond: "  " auxiliaryWords: "  "
Rules:  NEP Set Rule Deletion: Symbol " X9 "
Filters: NEP Set Filter: Input Filter Type 1 Permiting Context: " X9 " Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output Filter Type 1 Permiting Context: "  " Forbidding Context: " b1_r1_g1_b2_r2_g2_b3_r3_g3_b4_r4_g4_b5_r5_g5 "
NEP Set Node InitCond: "  " auxiliaryWords: "  "
Rules:  NEP Set Rule Substitution: Symbol " b1 " New Symbol " B1 "
Filters: NEP Set Filter: Input Filter Type 1 Permiting Context: " X1 " Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output Filter Type 1 Permiting Context: " B1 " Forbidding Context: "  "
NEP Set Node InitCond: "  " auxiliaryWords: "  "
Rules:  NEP Set Rule Substitution: Symbol " r1 " New Symbol " R1 "
Filters: NEP Set Filter: Input Filter Type 1 Permiting Context: " X1 " Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output Filter Type 1 Permiting Context: " R1 " Forbidding Context: "  "
NEP Set Node InitCond: "  " auxiliaryWords: "  "
Rules:  NEP Set Rule Substitution: Symbol " g1 " New Symbol " G1 "
Filters: NEP Set Filter: Input Filter Type 1 Permiting Context: " X1 " Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output Filter Type 1 Permiting Context: " G1 " Forbidding Context: "  "
```



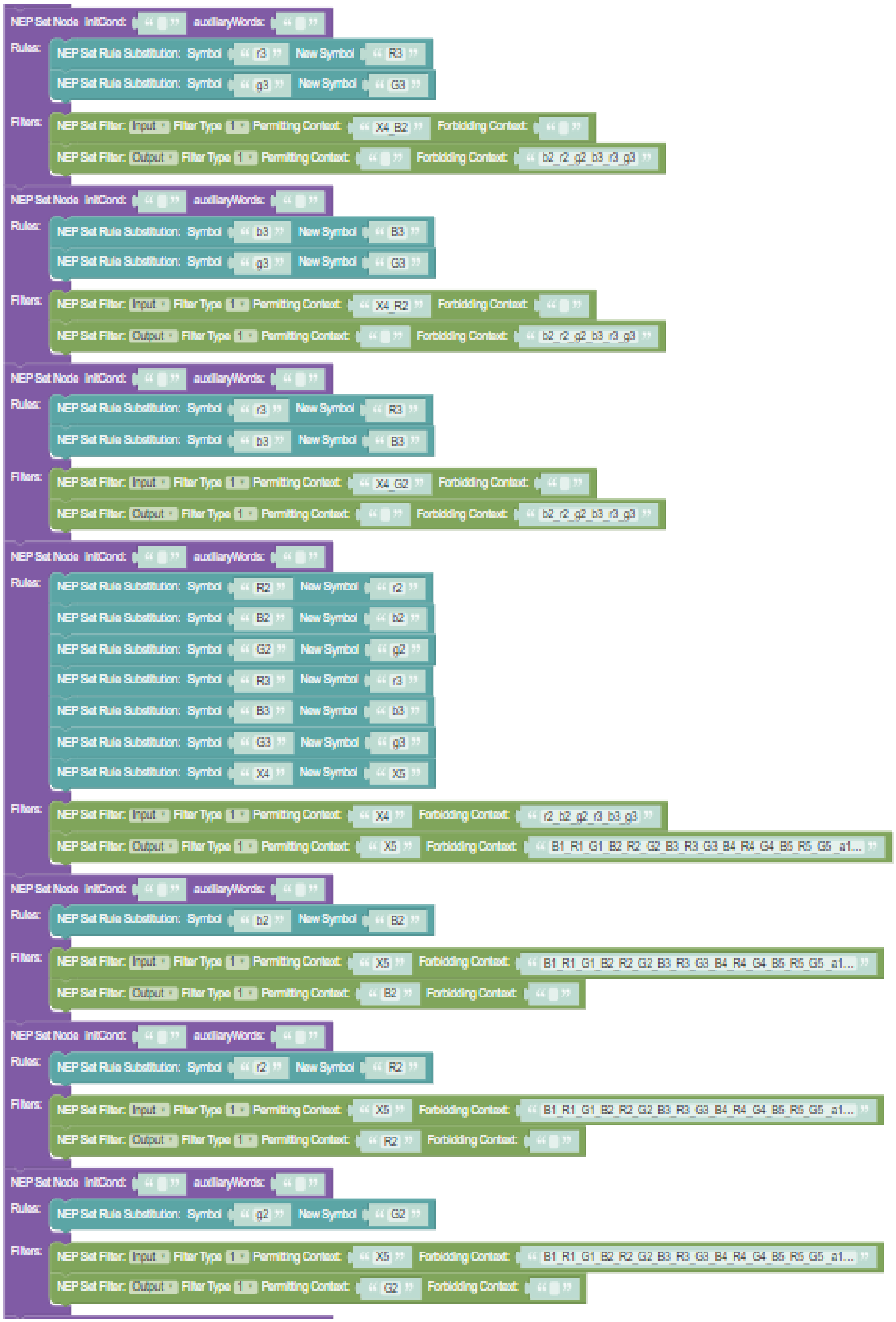
```
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:   NEP Set Rule Substitution:  Symbol " r2 "  New Symbol " R2 "
         NEP Set Rule Substitution:  Symbol " g2 "  New Symbol " G2 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X1_B1 "  Forbidding Context: " "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " "  Forbidding Context: " b1_r1_g1_b2_r2_g2 "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:   NEP Set Rule Substitution:  Symbol " b2 "  New Symbol " B2 "
         NEP Set Rule Substitution:  Symbol " g2 "  New Symbol " G2 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X1_R1 "  Forbidding Context: " "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " "  Forbidding Context: " b1_r1_g1_b2_r2_g2 "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:   NEP Set Rule Substitution:  Symbol " r2 "  New Symbol " R2 "
         NEP Set Rule Substitution:  Symbol " b2 "  New Symbol " B2 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X1_G1 "  Forbidding Context: " "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " "  Forbidding Context: " b1_r1_g1_b2_r2_g2 "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:   NEP Set Rule Substitution:  Symbol " R1 "  New Symbol " r1 "
         NEP Set Rule Substitution:  Symbol " B1 "  New Symbol " b1 "
         NEP Set Rule Substitution:  Symbol " G1 "  New Symbol " g1 "
         NEP Set Rule Substitution:  Symbol " R2 "  New Symbol " r2 "
         NEP Set Rule Substitution:  Symbol " B2 "  New Symbol " b2 "
         NEP Set Rule Substitution:  Symbol " G2 "  New Symbol " g2 "
         NEP Set Rule Substitution:  Symbol " X1 "  New Symbol " X2 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X1 "  Forbidding Context: " r1_b1_g1_r2_b2_g2 "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " X2 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:   NEP Set Rule Substitution:  Symbol " b1 "  New Symbol " B1 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X2 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " B1 "  Forbidding Context: " "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:   NEP Set Rule Substitution:  Symbol " r1 "  New Symbol " R1 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X2 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " R1 "  Forbidding Context: " "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:   NEP Set Rule Substitution:  Symbol " g1 "  New Symbol " G1 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X2 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " G1 "  Forbidding Context: " "
```



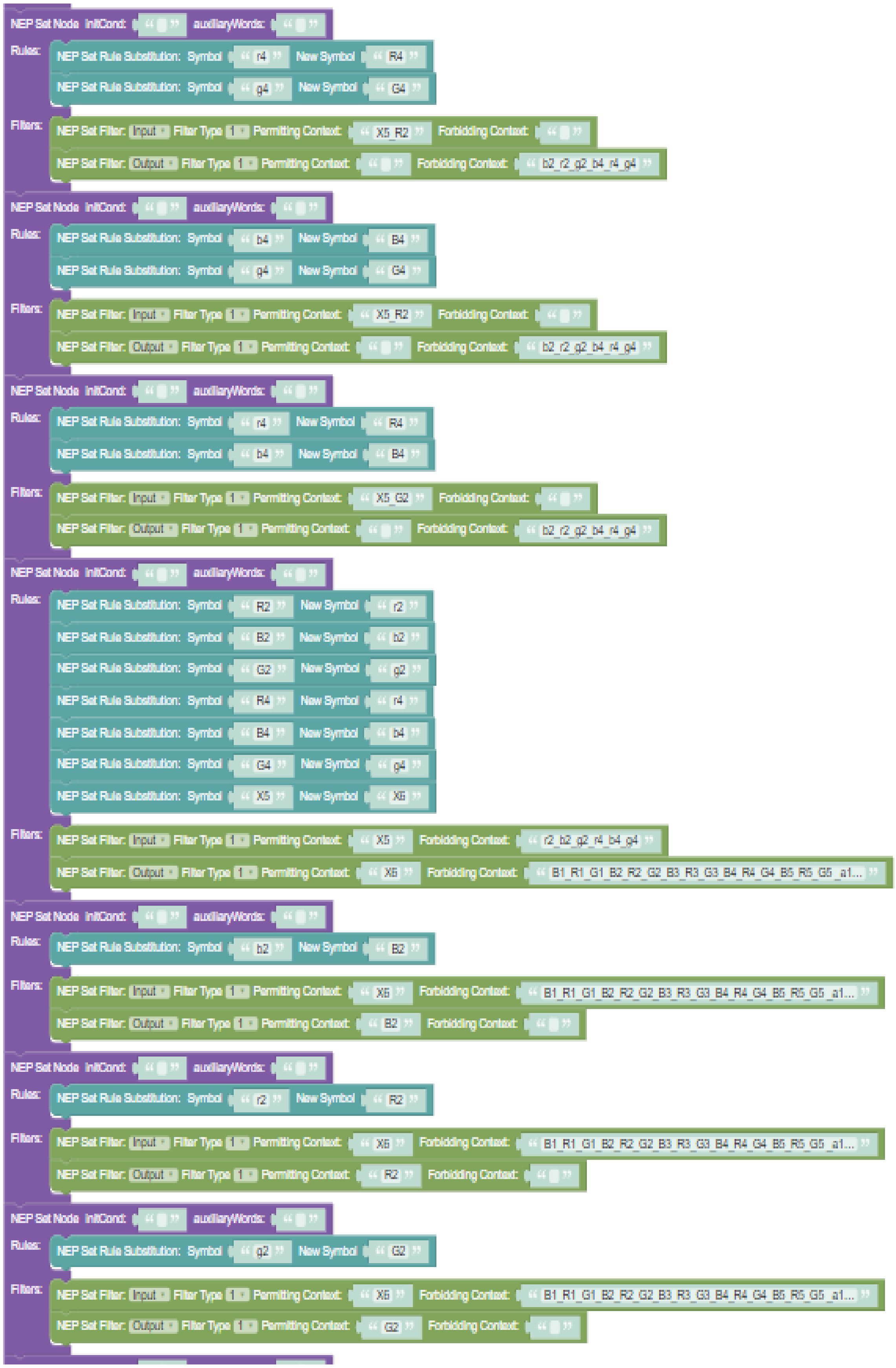
```
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “ r3 ” New Symbol “ R3 ”
         NEP Set Rule Substitution: Symbol “ g3 ” New Symbol “ G3 ”
Filters: NEP Set Filter: Input  Filter Type 1 Permitting Context: “ X2_B1 ” Forbidding Context: “ ”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “ ” Forbidding Context: “ b1_r1_g1_b3_r3_g3 ”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “ b3 ” New Symbol “ B3 ”
         NEP Set Rule Substitution: Symbol “ g3 ” New Symbol “ G3 ”
Filters: NEP Set Filter: Input  Filter Type 1 Permitting Context: “ X2_R1 ” Forbidding Context: “ ”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “ ” Forbidding Context: “ b1_r1_g1_b3_r3_g3 ”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “ r3 ” New Symbol “ R3 ”
         NEP Set Rule Substitution: Symbol “ b3 ” New Symbol “ B3 ”
Filters: NEP Set Filter: Input  Filter Type 1 Permitting Context: “ X2_G1 ” Forbidding Context: “ ”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “ ” Forbidding Context: “ b1_r1_g1_b3_r3_g3 ”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “ R1 ” New Symbol “ r1 ”
         NEP Set Rule Substitution: Symbol “ B1 ” New Symbol “ b1 ”
         NEP Set Rule Substitution: Symbol “ G1 ” New Symbol “ g1 ”
         NEP Set Rule Substitution: Symbol “ R3 ” New Symbol “ r3 ”
         NEP Set Rule Substitution: Symbol “ B3 ” New Symbol “ b3 ”
         NEP Set Rule Substitution: Symbol “ G3 ” New Symbol “ g3 ”
         NEP Set Rule Substitution: Symbol “ X2 ” New Symbol “ X3 ”
Filters: NEP Set Filter: Input  Filter Type 1 Permitting Context: “ X2 ” Forbidding Context: “ r1_b1_g1_r3_b3_g3 ”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “ X3 ” Forbidding Context: “ B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1_... ”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “ b1 ” New Symbol “ B1 ”
Filters: NEP Set Filter: Input  Filter Type 1 Permitting Context: “ X3 ” Forbidding Context: “ B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... ”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “ B3 ” Forbidding Context: “ ”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “ r1 ” New Symbol “ R1 ”
Filters: NEP Set Filter: Input  Filter Type 1 Permitting Context: “ X3 ” Forbidding Context: “ B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... ”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “ R1 ” Forbidding Context: “ ”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “ g1 ” New Symbol “ G1 ”
Filters: NEP Set Filter: Input  Filter Type 1 Permitting Context: “ X3 ” Forbidding Context: “ B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... ”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “ G1 ” Forbidding Context: “ ”
```



```
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution: Symbol " r4 "  New Symbol " R4 "
        NEP Set Rule Substitution: Symbol " g4 "  New Symbol " G4 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X3_B1 "  Forbidding Context: " "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " "  Forbidding Context: " b1_r1_g1_b4_r4_g4 "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution: Symbol " b4 "  New Symbol " B4 "
        NEP Set Rule Substitution: Symbol " g4 "  New Symbol " G4 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X3_R1 "  Forbidding Context: " "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " "  Forbidding Context: " b1_r1_g1_b4_r4_g4 "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution: Symbol " r4 "  New Symbol " R4 "
        NEP Set Rule Substitution: Symbol " b4 "  New Symbol " B4 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X3_G1 "  Forbidding Context: " "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " "  Forbidding Context: " b1_r1_g1_b4_r4_g4 "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution: Symbol " R1 "  New Symbol " r1 "
        NEP Set Rule Substitution: Symbol " B1 "  New Symbol " b1 "
        NEP Set Rule Substitution: Symbol " G1 "  New Symbol " g1 "
        NEP Set Rule Substitution: Symbol " R4 "  New Symbol " r4 "
        NEP Set Rule Substitution: Symbol " B4 "  New Symbol " b4 "
        NEP Set Rule Substitution: Symbol " G4 "  New Symbol " g4 "
        NEP Set Rule Substitution: Symbol " X3 "  New Symbol " X4 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X3 "  Forbidding Context: " r1_b1_g1_r4_b4_g4 "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " X4 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution: Symbol " b2 "  New Symbol " B2 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X4 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " B2 "  Forbidding Context: " "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution: Symbol " r2 "  New Symbol " R2 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X4 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " R2 "  Forbidding Context: " "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution: Symbol " g2 "  New Symbol " G2 "
Filters: NEP Set Filter: Input  Filter Type 1  Permitting Context: " X4 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output  Filter Type 1  Permitting Context: " G2 "  Forbidding Context: " "
```



```
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “r3” New Symbol “R3”
         NEP Set Rule Substitution: Symbol “g3” New Symbol “G3”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X4_B2” Forbidding Context: “ ”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “ ” Forbidding Context: “b2_r2_g2_b3_r3_g3”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “b3” New Symbol “B3”
         NEP Set Rule Substitution: Symbol “g3” New Symbol “G3”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X4_R2” Forbidding Context: “ ”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “ ” Forbidding Context: “b2_r2_g2_b3_r3_g3”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “r3” New Symbol “R3”
         NEP Set Rule Substitution: Symbol “b3” New Symbol “B3”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X4_G2” Forbidding Context: “ ”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “ ” Forbidding Context: “b2_r2_g2_b3_r3_g3”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “R2” New Symbol “r2”
         NEP Set Rule Substitution: Symbol “B2” New Symbol “b2”
         NEP Set Rule Substitution: Symbol “G2” New Symbol “g2”
         NEP Set Rule Substitution: Symbol “R3” New Symbol “r3”
         NEP Set Rule Substitution: Symbol “B3” New Symbol “b3”
         NEP Set Rule Substitution: Symbol “G3” New Symbol “g3”
         NEP Set Rule Substitution: Symbol “X4” New Symbol “X5”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X4” Forbidding Context: “r2_b2_g2_r3_b3_g3”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “X5” Forbidding Context: “B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1...”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “b2” New Symbol “B2”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X5” Forbidding Context: “B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1...”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “B2” Forbidding Context: “ ”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “r2” New Symbol “R2”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X5” Forbidding Context: “B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1...”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “R2” Forbidding Context: “ ”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:   NEP Set Rule Substitution: Symbol “g2” New Symbol “G2”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X5” Forbidding Context: “B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1...”
         NEP Set Filter: Output Filter Type 1 Permitting Context: “G2” Forbidding Context: “ ”
```



NEP Set Node InitCond: “ ” auxiliaryWords: “ ”
Rules: NEP Set Rule Substitution: Symbol “r4” New Symbol “R4”
NEP Set Rule Substitution: Symbol “g4” New Symbol “G4”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X5_R2” Forbidding Context: “ ”
NEP Set Filter: Output Filter Type 1 Permitting Context: “ ” Forbidding Context: “b2_r2_g2_b4_r4_g4”

NEP Set Node InitCond: “ ” auxiliaryWords: “ ”
Rules: NEP Set Rule Substitution: Symbol “b4” New Symbol “B4”
NEP Set Rule Substitution: Symbol “g4” New Symbol “G4”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X5_R2” Forbidding Context: “ ”
NEP Set Filter: Output Filter Type 1 Permitting Context: “ ” Forbidding Context: “b2_r2_g2_b4_r4_g4”

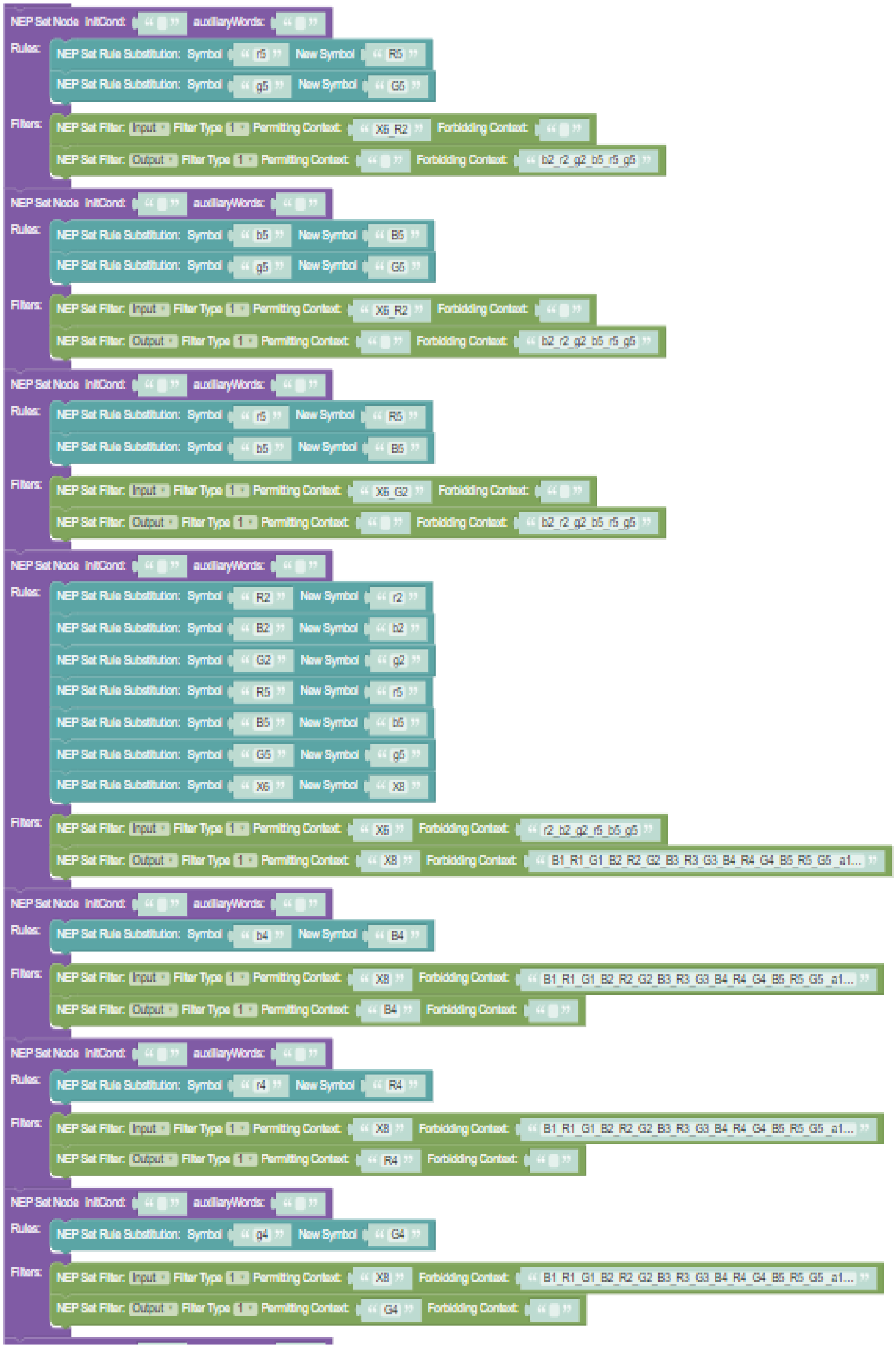
NEP Set Node InitCond: “ ” auxiliaryWords: “ ”
Rules: NEP Set Rule Substitution: Symbol “r4” New Symbol “R4”
NEP Set Rule Substitution: Symbol “b4” New Symbol “B4”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X5_G2” Forbidding Context: “ ”
NEP Set Filter: Output Filter Type 1 Permitting Context: “ ” Forbidding Context: “b2_r2_g2_b4_r4_g4”

NEP Set Node InitCond: “ ” auxiliaryWords: “ ”
Rules: NEP Set Rule Substitution: Symbol “R2” New Symbol “r2”
NEP Set Rule Substitution: Symbol “B2” New Symbol “b2”
NEP Set Rule Substitution: Symbol “G2” New Symbol “g2”
NEP Set Rule Substitution: Symbol “R4” New Symbol “r4”
NEP Set Rule Substitution: Symbol “B4” New Symbol “b4”
NEP Set Rule Substitution: Symbol “G4” New Symbol “g4”
NEP Set Rule Substitution: Symbol “X5” New Symbol “X6”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X5” Forbidding Context: “r2_b2_g2_r4_b4_g4”
NEP Set Filter: Output Filter Type 1 Permitting Context: “X6” Forbidding Context: “B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1...”

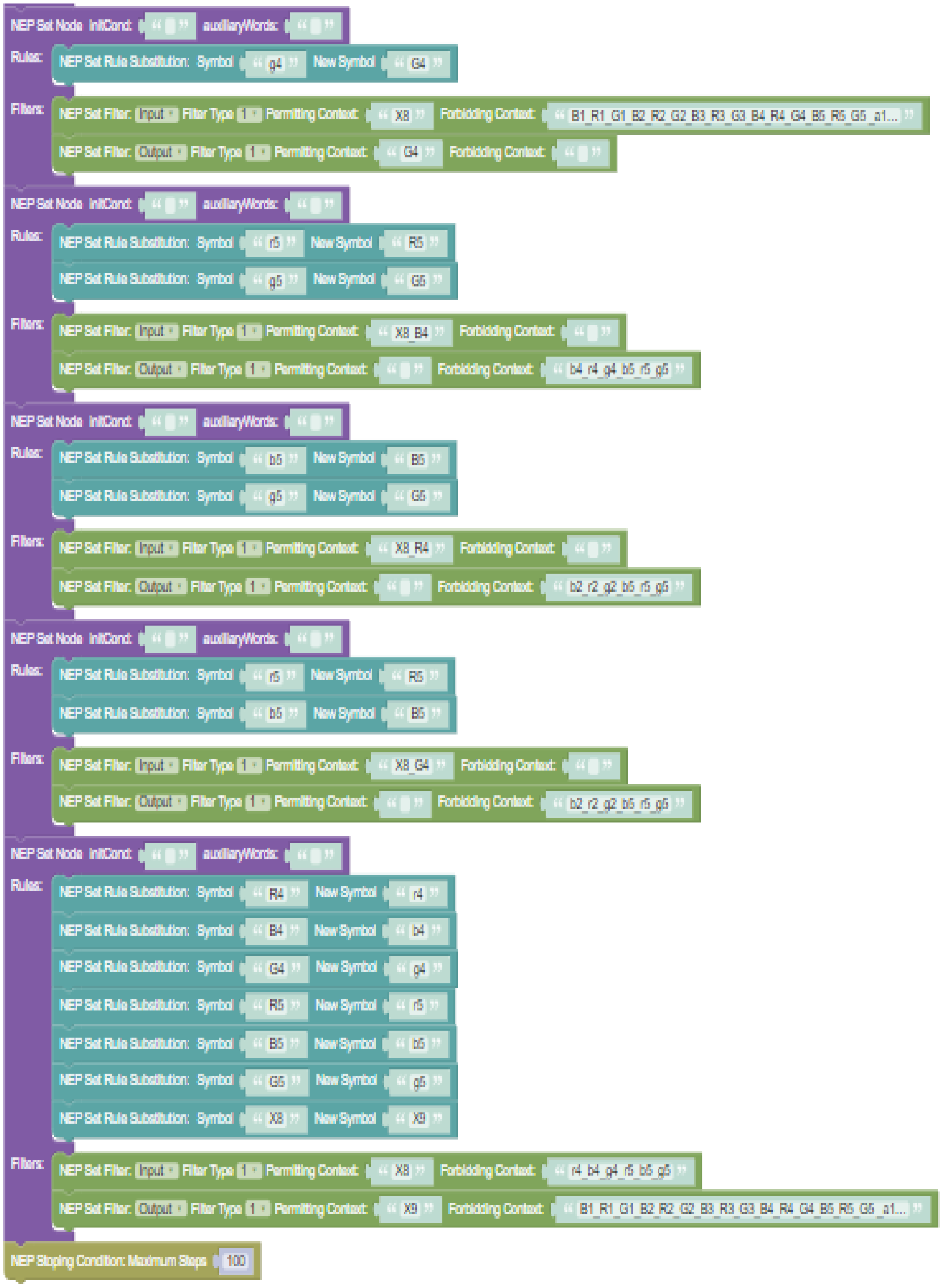
NEP Set Node InitCond: “ ” auxiliaryWords: “ ”
Rules: NEP Set Rule Substitution: Symbol “b2” New Symbol “B2”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X6” Forbidding Context: “B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1...”
NEP Set Filter: Output Filter Type 1 Permitting Context: “B2” Forbidding Context: “ ”

NEP Set Node InitCond: “ ” auxiliaryWords: “ ”
Rules: NEP Set Rule Substitution: Symbol “r2” New Symbol “R2”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X6” Forbidding Context: “B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1...”
NEP Set Filter: Output Filter Type 1 Permitting Context: “R2” Forbidding Context: “ ”

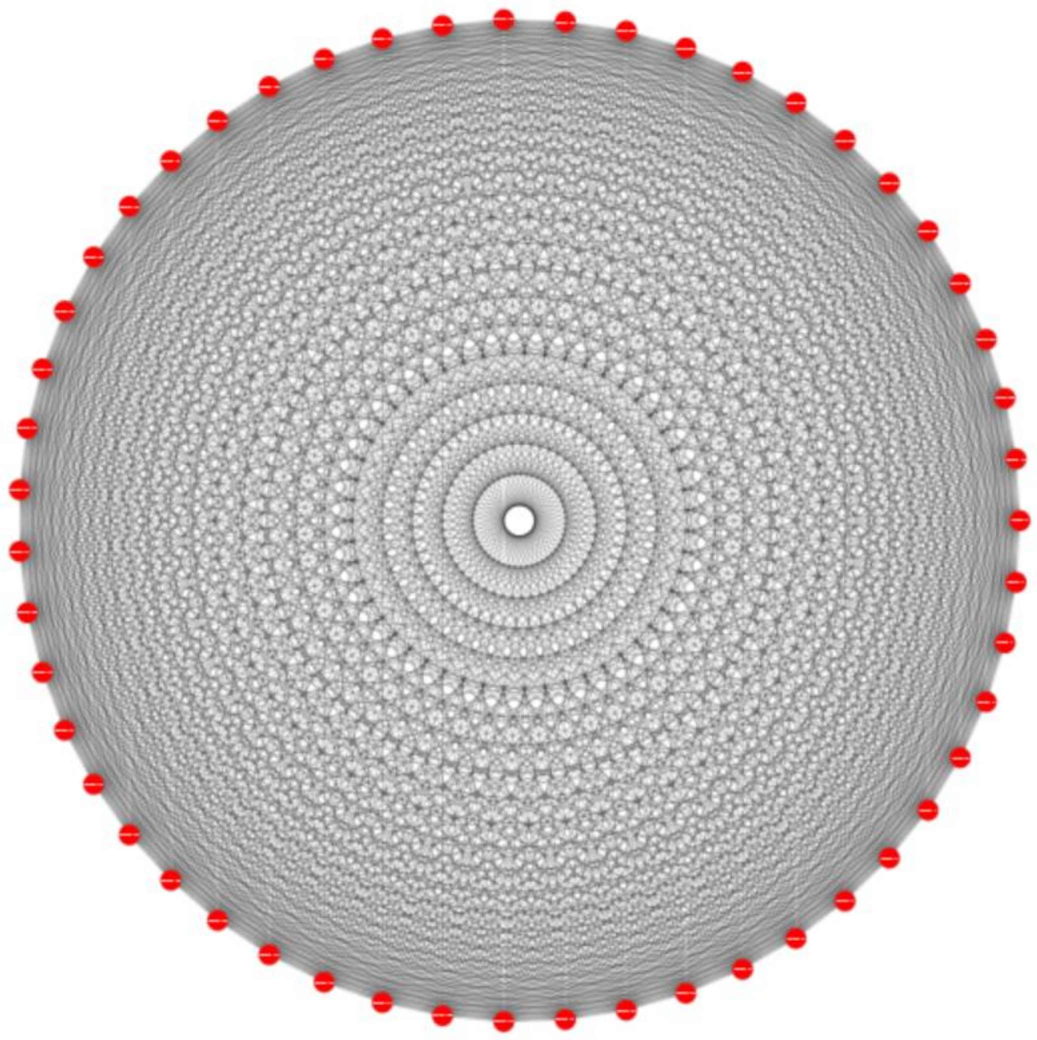
NEP Set Node InitCond: “ ” auxiliaryWords: “ ”
Rules: NEP Set Rule Substitution: Symbol “g2” New Symbol “G2”
Filters: NEP Set Filter: Input Filter Type 1 Permitting Context: “X6” Forbidding Context: “B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1...”
NEP Set Filter: Output Filter Type 1 Permitting Context: “G2” Forbidding Context: “ ”



```
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution:  Symbol " r5 "  New Symbol " R5 "
        NEP Set Rule Substitution:  Symbol " g5 "  New Symbol " G5 "
Filters: NEP Set Filter: Input  Filter Type 1  Permiting Context: " X6_R2 "  Forbidding Context: " "
         NEP Set Filter: Output  Filter Type 1  Permiting Context: " "  Forbidding Context: " b2_r2_g2_b5_r5_g5 "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution:  Symbol " b5 "  New Symbol " B5 "
        NEP Set Rule Substitution:  Symbol " g5 "  New Symbol " G5 "
Filters: NEP Set Filter: Input  Filter Type 1  Permiting Context: " X6_R2 "  Forbidding Context: " "
         NEP Set Filter: Output  Filter Type 1  Permiting Context: " "  Forbidding Context: " b2_r2_g2_b5_r5_g5 "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution:  Symbol " r5 "  New Symbol " R5 "
        NEP Set Rule Substitution:  Symbol " b5 "  New Symbol " B5 "
Filters: NEP Set Filter: Input  Filter Type 1  Permiting Context: " X6_G2 "  Forbidding Context: " "
         NEP Set Filter: Output  Filter Type 1  Permiting Context: " "  Forbidding Context: " b2_r2_g2_b5_r5_g5 "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution:  Symbol " R2 "  New Symbol " r2 "
        NEP Set Rule Substitution:  Symbol " B2 "  New Symbol " b2 "
        NEP Set Rule Substitution:  Symbol " G2 "  New Symbol " g2 "
        NEP Set Rule Substitution:  Symbol " R5 "  New Symbol " r5 "
        NEP Set Rule Substitution:  Symbol " B5 "  New Symbol " b5 "
        NEP Set Rule Substitution:  Symbol " G5 "  New Symbol " g5 "
        NEP Set Rule Substitution:  Symbol " X6 "  New Symbol " X8 "
Filters: NEP Set Filter: Input  Filter Type 1  Permiting Context: " X6 "  Forbidding Context: " r2_b2_g2_r5_b5_g5 "
         NEP Set Filter: Output  Filter Type 1  Permiting Context: " X8 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution:  Symbol " b4 "  New Symbol " B4 "
Filters: NEP Set Filter: Input  Filter Type 1  Permiting Context: " X8 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output  Filter Type 1  Permiting Context: " B4 "  Forbidding Context: " "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution:  Symbol " r4 "  New Symbol " R4 "
Filters: NEP Set Filter: Input  Filter Type 1  Permiting Context: " X8 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output  Filter Type 1  Permiting Context: " R4 "  Forbidding Context: " "
NEP Set Node  InitCond: " "  auxiliaryWords: " "
Rules:  NEP Set Rule Substitution:  Symbol " g4 "  New Symbol " G4 "
Filters: NEP Set Filter: Input  Filter Type 1  Permiting Context: " X8 "  Forbidding Context: " B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1... "
         NEP Set Filter: Output  Filter Type 1  Permiting Context: " G4 "  Forbidding Context: " "
```



```
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:
    NEP Set Rule Substitution:  Symbol “g4”  New Symbol “G4”
Filters:
    NEP Set Filter: Input  Filter Type 1  Permitting Context: “X8”  Forbidding Context: “B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1...”
    NEP Set Filter: Output  Filter Type 1  Permitting Context: “G4”  Forbidding Context: “ ”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:
    NEP Set Rule Substitution:  Symbol “r5”  New Symbol “R5”
    NEP Set Rule Substitution:  Symbol “g5”  New Symbol “G5”
Filters:
    NEP Set Filter: Input  Filter Type 1  Permitting Context: “X8_B4”  Forbidding Context: “ ”
    NEP Set Filter: Output  Filter Type 1  Permitting Context: “ ”  Forbidding Context: “b4_r4_g4_b5_r5_g5”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:
    NEP Set Rule Substitution:  Symbol “b5”  New Symbol “B5”
    NEP Set Rule Substitution:  Symbol “g5”  New Symbol “G5”
Filters:
    NEP Set Filter: Input  Filter Type 1  Permitting Context: “X8_R4”  Forbidding Context: “ ”
    NEP Set Filter: Output  Filter Type 1  Permitting Context: “ ”  Forbidding Context: “b2_r2_g2_b5_r5_g5”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:
    NEP Set Rule Substitution:  Symbol “r5”  New Symbol “R5”
    NEP Set Rule Substitution:  Symbol “b5”  New Symbol “B5”
Filters:
    NEP Set Filter: Input  Filter Type 1  Permitting Context: “X8_G4”  Forbidding Context: “ ”
    NEP Set Filter: Output  Filter Type 1  Permitting Context: “ ”  Forbidding Context: “b2_r2_g2_b5_r5_g5”
NEP Set Node  InitCond: “ ”  auxiliaryWords: “ ”
Rules:
    NEP Set Rule Substitution:  Symbol “R4”  New Symbol “r4”
    NEP Set Rule Substitution:  Symbol “B4”  New Symbol “b4”
    NEP Set Rule Substitution:  Symbol “G4”  New Symbol “g4”
    NEP Set Rule Substitution:  Symbol “R5”  New Symbol “r5”
    NEP Set Rule Substitution:  Symbol “B5”  New Symbol “b5”
    NEP Set Rule Substitution:  Symbol “G5”  New Symbol “g5”
    NEP Set Rule Substitution:  Symbol “X8”  New Symbol “X9”
Filters:
    NEP Set Filter: Input  Filter Type 1  Permitting Context: “X8”  Forbidding Context: “r4_b4_g4_r5_b5_g5”
    NEP Set Filter: Output  Filter Type 1  Permitting Context: “X9”  Forbidding Context: “B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5_a1...”
NEP Stoping Condition: Maximum Steps 100
```



REFERENCES

[1] Juan Castellanos, Carlos Martín-Vide, Victor Mitrana, and José M Sempere. Networks Of evolutionary processors.Acta Informatica, 39(6-7):517–529, 2003.

[2] Juan Castellanos, Carlos Martin-Vide, Victor Mitrana, and Jose M Sempere. Solvingnp-complete problems with networks of evolutionary processors. InInternational Work-Conference on Artificial Neural Networks, pages 621–628. Springer, 2001.

[3] Erzsébet Csuhaj-Varjú, Carlos Martín-Vide, and Victor Mitrana. Hybrid networks ofevolutionary processors are computationally complete.Acta Informatica, 41(4-5):257–272, 2005.

[4] Nuria Gomez Blas, Miguel Angel Diaz, Juan Castellanos, and Francisco Serradilla.Networks of evolutionary processors (nep) as decision support systems. 2008.

[5] Castellanos, J., Martin-Vide, C., Mitrana, V., & Sempere, J. M. (2001, June). Solving NP-complete problems with networks of evolutionary processors. In International Work-Conference on Artificial Neural Networks (pp. 621-628). Springer, Berlin, Heidelberg.

[6] Emilio del Rosal García. Real life applications of bio-inspired computing models: Eap and neps. 2013. PhD thesis.

[7] Emilio Del Rosal, Rafael Nunez, Carlos Castaneda, and Alfonso Ortega. Simulating neps in a cluster with jnep. InProceedings of International Conference on Computers,Communications and Control, ICCCC. Citeseer, 2008.

[8] Carmen Navarrete Navarrete, Marina de la Cruz Echeandia, Eloy Anguiano Rey, Al-fonso Ortega de la Puente, and Jose Miguel Rojas. Parallel simulation of neps onclusters. In2011 IEEE/WIC/ACM International Conferences on Web Intelligence andIntelligent Agent Technology, volume 3, pages 171–174. IEEE, 2011.

[9] Stephen Wolfram.A new kind of science, volume 5. Wolfram media Champaign, IL,2002.

[10] Clifford A Pickover.The math book: from Pythagoras to the 57th dimension, 250 milestones in the history of mathematics. Sterling Publishing Company, Inc., 2009.

[11] S. Wolfram.Cellular Automata And Complexity: Collected Papers. CRC Press, 2018.

[12] Martin Erwig, Karl Smeltzer, and Xiangyu Wang. What is a visual language?Journalof Visual Languages & Computing, 38:9–17, 2017.

[13] https://scratch.mit.edu/, Accessed October 2019.

[14] https://developers.google.com/blockly/guides/overview, Accessed October 2019.

[15] https://code.org , Accessed October 2019.

[16] https://en.scratch-wiki.info/wiki/Block-Based_Coding, Accessed October 2019.

[17] https://pencilcode.net/ , Accessed October 2019.

[18] http://appinventor.mit.edu/, Accessed November 2019.

[19] https://scratch.mit.edu/statistics/, Accessed November 2019.

[20] Franklin, Diana, et al. "Using upper-elementary student performance to understand conceptual sequencing in a blocks-based curriculum." Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. ACM, 2017.

[21] Grover, Shuchi, Roy Pea, and Stephen Cooper. "Designing for deeper learning in a blended computer science course for middle school students." Computer Science Education 25.2 (2015): 199-237.

[22]Weintrop, David, and Uri Wilensky. "Comparing block-based and text-based programming in high school computer science classrooms." ACM Transactions on Computing Education (TOCE) 18.1 (2017): 3.

[23] https://www.makewonder.com/, Accessed, October 2019.

[24] https://make.gamefroot.com/games/new, Accessed, October 2019.

[25] Alrubaye, Hussein. Comparison of visual programming and hybrid programming environments in transferring programming skills. Diss. Rochester Institute of Technology, 2017.

[26] Alrubaye, H., Ludi, S., & Mkaouer, M. W. (2019, November). Comparison of block-based and hybrid-based environments in transferring programming skills to text-based environments. In Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering (pp. 100-109). IBM Corp.

[27] Cooper, Stephen, Wanda Dann, and Randy Pausch. "Alice: a 3-D tool for introductory programming concepts." Journal of Computing Sciences in Colleges 15.5 (2000): 107-116.

[28] Ioannidou, Andri, Alexander Repenning, and David C. Webb. "AgentCubes: Incremental 3D end-user development." Journal of Visual Languages & Computing 20.4 (2009): 236-251.

[29] A. Begel, E. Klopfer, "Starlogo TNG: An introduction to game development", J. E-Learn., 2007.

[30] M. S. Horn, C. Brady, A. Hjorth, A. Wagh, U. Wilensky, "Frog pond: a codefirst learning environment on evolution and natural selection", Proceedings of IDC, pp. 357-360, 2014.

[31] M. H. Wilkerson-Jerde, U. Wilensky, "Restructuring Change Interpreting Changes: The DeltaTick Modeling and Analysis Toolkit", Proc. of the Constructionism 2010 Conference, 2010.

[32] W. Slany, "Tinkering with Pocket Code a Scratch-like programming app for your smartphone", Proc. of Constructionism Austria, 2014.

[33] D. Wolber, H. Abelson, E. Spertus, L. Looney, App Inventor 2: Create Your Own Android Apps, Beijing:O'Reilly Media, 2014.

[34] S. Esper, S. R. Foster, W. G. Griswold, "CodeSpells: embodying the metaphor of wizardry for programming", Proceedings of the 18th ACM ITiCSE, pp. 249-254, 2013.

[35] D. Weintrop, U. Wilensky, "RoboBuilder: A program-to-play constructionist video game", Proceedings of the Constructionism 2012 Conference, 2012.

[36] J. Maloney, M. Nagle, J. Monig, "GP: A General Purpose Blocks-Based Language", Proceedings of the 2017 ACM SIGCSE, pp. 739-739, 2017.

[37] D. Bau, "Droplet a blocks-based editor for text code", J. Comput. Sci. Coll., vol. 30, no. 6, pp. 138-144, 2015.

[38] N. Fraser, "Ten things we've learned from Blockly", 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond), pp. 49-50, 2015.

[39] Ko, Andrew J., et al. "The state of the art in end-user software engineering." ACM Computing Surveys (CSUR) 43.3 (2011): 21.

[40] Gardner, Martin. "Mathematical games-The fantastic combinations of John Conway's new solitaire game, Life, 1970." *Scientific American, October*: 120-123

[41] https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life , Accessed, October 2019

[42] http://robinforest.net/post/cellular-automata/, Accessed, October 2019.

[43] http://devinacker.github.io/celldemo/, Accessed, October 2019.

[44] http://cubes.io/, Accessed, October 2019.

[45] del Rosal, E., & Cuéllar, M. (2009, June). jNEPView: A graphical trace viewer for the simulations of nEPs. In International Work-Conference on the Interplay Between Natural and Artificial Computation (pp. 356-365). Springer, Berlin, Heidelberg.

[46] Jimenez, A., del Rosal, E., & de Lara, J. (2010). A visual language for modelling and simulation of networks of evolutionary processors. In Trends in Practical Applications of Agents and Multiagent Systems (pp. 411-418). Springer, Berlin, Heidelberg.