MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION

FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION

"NOVOSIBIRSK NATIONAL RESEARCH UNIVERSITY

STATE UNIVERSITY"

(NOVOSIBIRSK STATE UNIVERSITY, NSU)

09.03.01 - Informatics and Computer Engineering

Focus (profile): Software Engineering and Computer Science

TERM PAPER

*Martynov Bodgan Alekseevich*

*Belov Danil Valerievich*

**"Game of Life"**

Table of contents

[Introduction 3](#_Toc166356795)

[Problem Statement 4](#_Toc166356796)

[Hardware 5](#_Toc166356797)

[Software 14](#_Toc166356798)

[User Guide 21](#_Toc166356799)

[Conclusion 24](#_Toc166356800)

# Introduction

We have chosen and made a “Conway’s game of life” as project-game using Logisim and CdM-8.

The Game of Life, often referred to simply as Life, stands as a testament to the genius of British mathematician John Horton Conway, who introduced it in 1970. This remarkable cellular automaton operates independently, devoid of player interaction, where its progression hinges solely on the initial state provided, requiring no external input thereafter. Engaging with the Game of Life involves crafting an initial configuration and then observing its organic evolution.

Rules:

The universe of the Game of Life is an infinite, two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, live or dead. Every cell interacts with its eight neighbors, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbors die, as if by underpopulation.

2. Any live cell with two or three live neighbors lives on to the next generation.

3. Any live cell with more than three live neighbors dies, as if by overpopulation.

4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

# Problem Statement

We have made one change from the basic technical task:

1. The editor of the rules of “the game of life”. User can change rules.

First of all, we have made work plan and divided the project into subtasks:

1. Divide the project into the Logisim part and Assembler (CdM8) part.
2. Realize Logisim part
3. Realize Assembler part
4. Connect both parts
5. Check for bags and fix it
6. Optimize it

As mentioned earlier, our project has two parts: Logisim and Assembler (CdM8).

The Logisim part of project will be responsible for main functions of game:

* Display
* System of generation cells
* Changing rules

Assembler (CdM8). The CdM8 part of the project will be responsible for some functions:

* Management - whether it is minimized, its current state and the ability to pause it.
* The control of four cursor keys - arrows down up left and right, pause and status buttons - invert, set death and set life.

# Hardware

Let's start with small schemes that perform the functions of helpers.

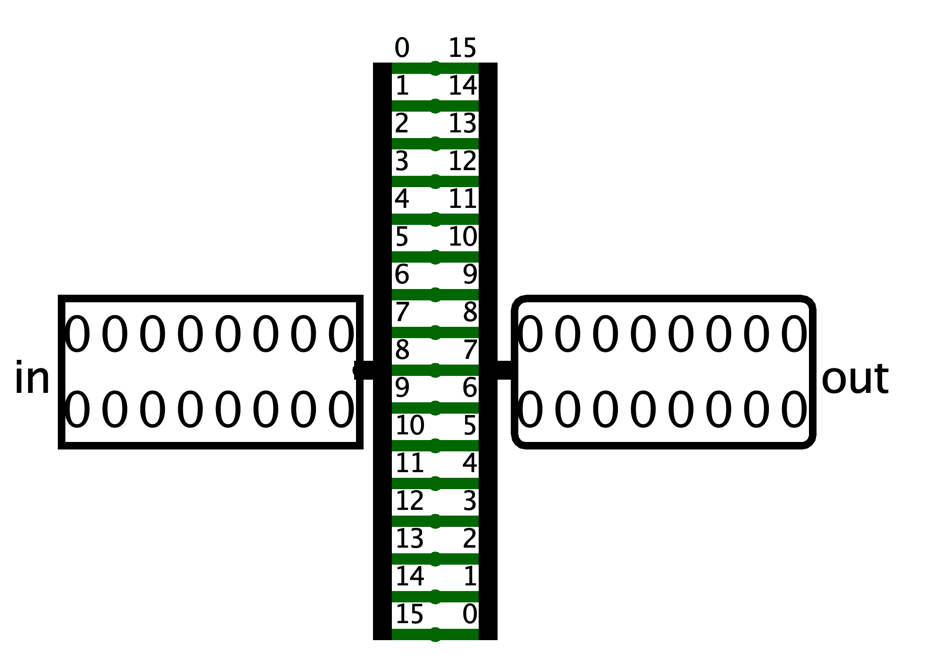
|  |  |
| --- | --- |
| BS-Helper | BS-Helper(0->31) |
|  |  |

It is used only in the main scheme. It is needed for convenient editing of rules.

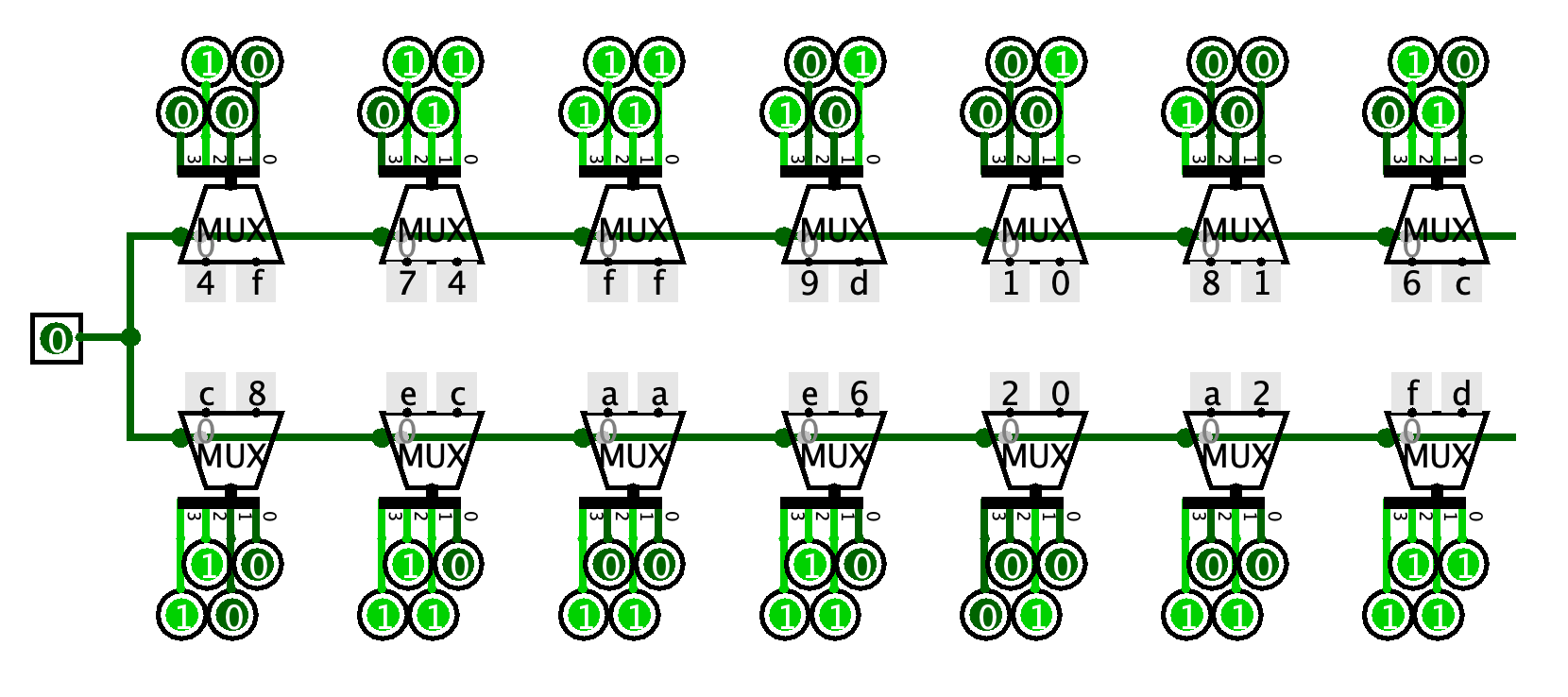
State-Helper:

If the cursor (i, j) is on the selected cell, transfer the new state to the alive scheme. Otherwise, pass the state (00) — old, that is, do not change the cell (since the cursor is not on it)

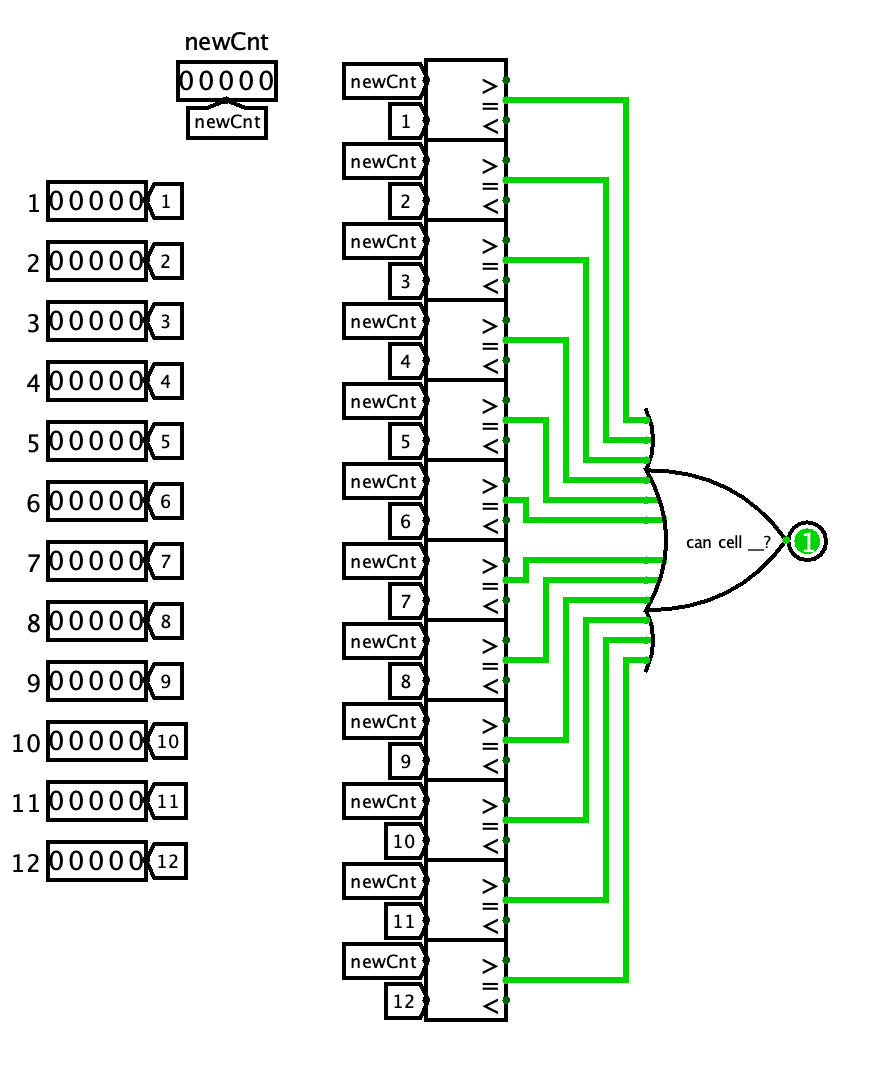
Reverse16Bits: reverse the number.



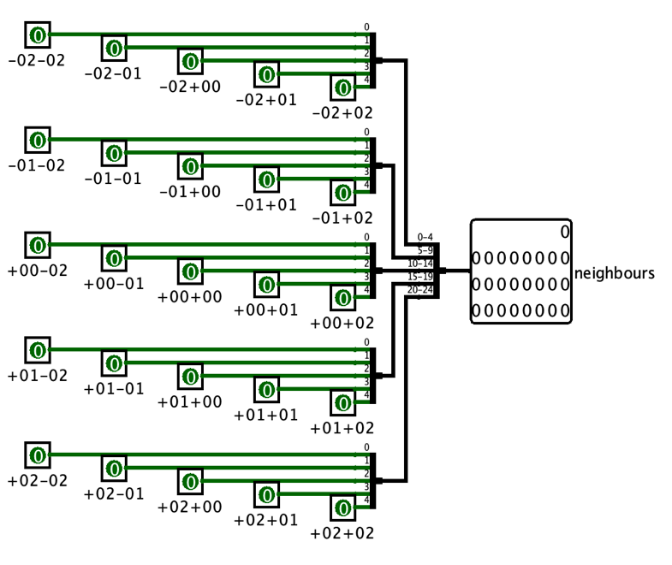
Text scheme contains constant values for text output.



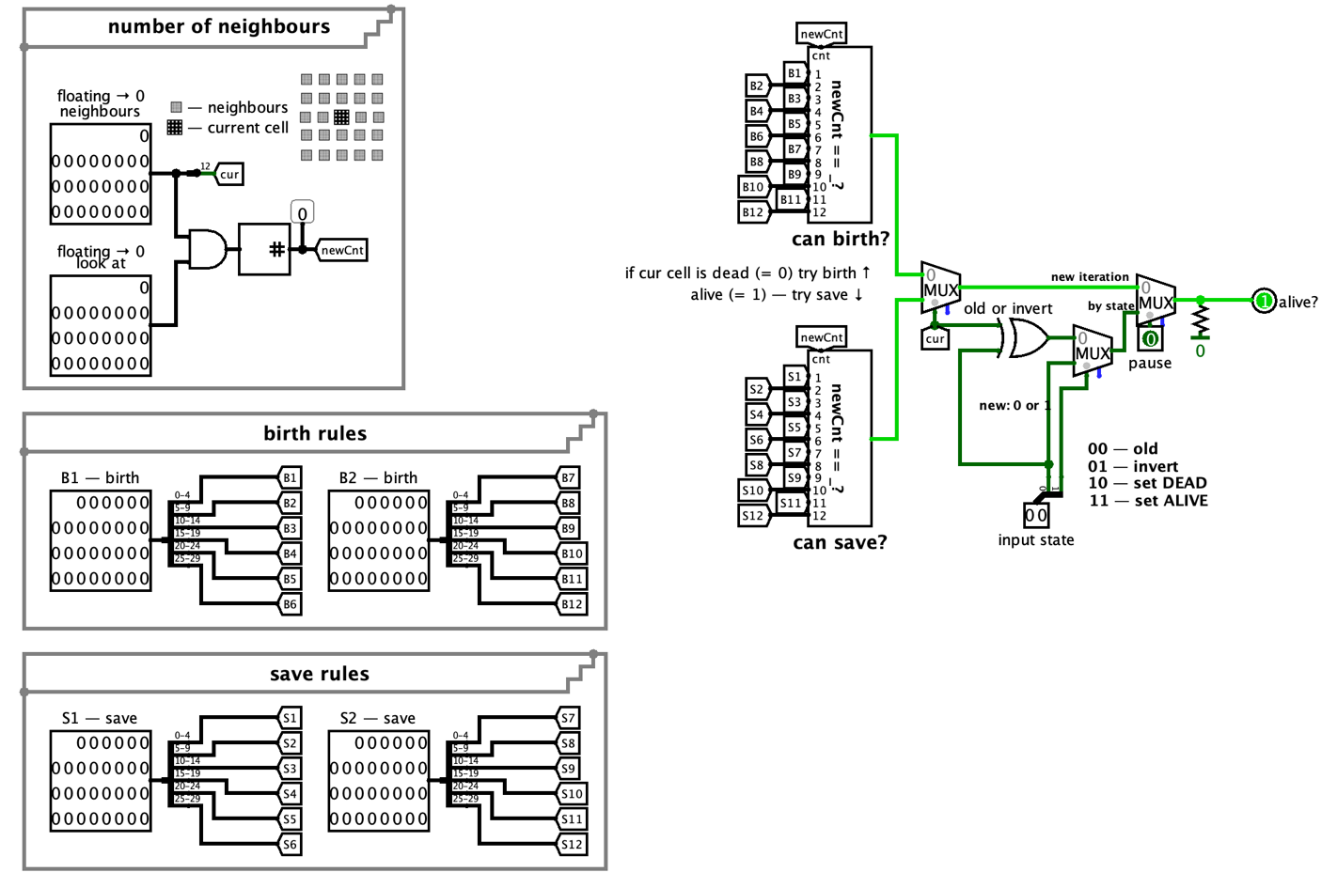
OR12. This scheme gets 13 numbers: rules(birth or save) and count of neighbours.



The first of the basic schemes is neighbours scheme. Neighbours scheme gets all cell’s neighbours and passes to alive scheme.

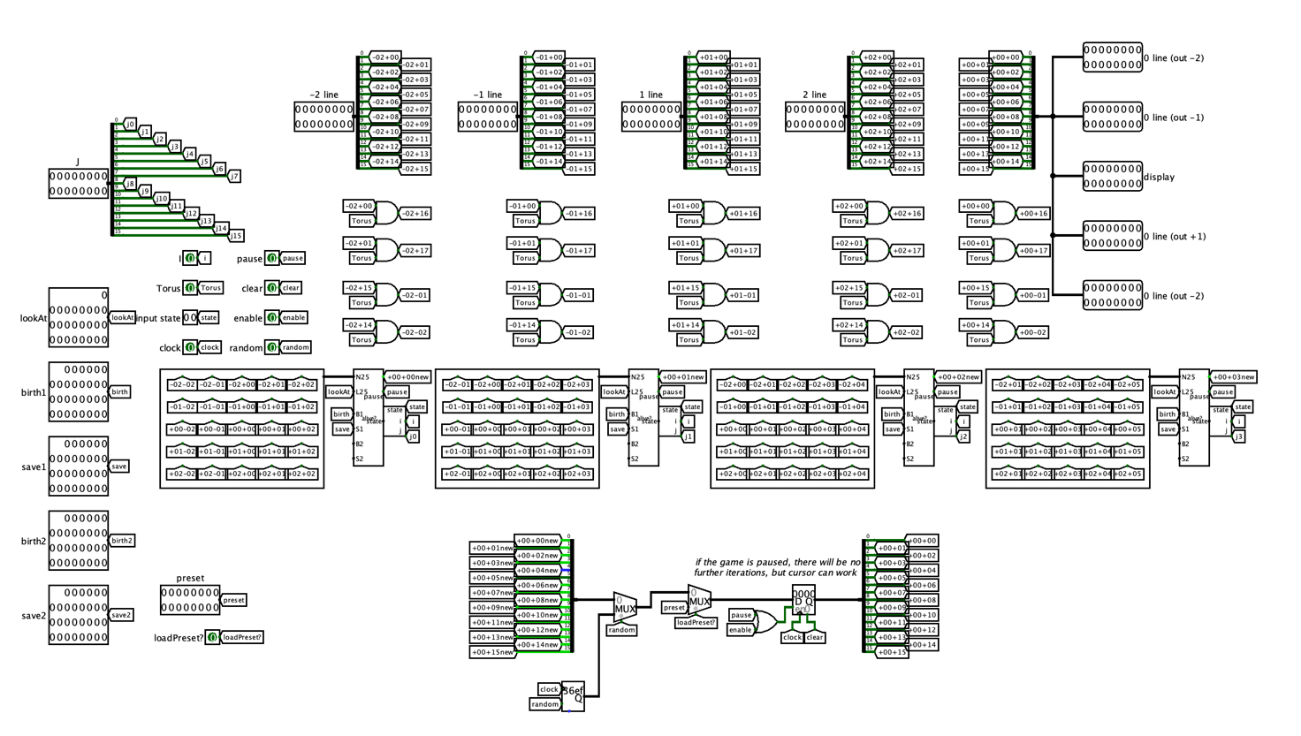


Next is alive scheme. Alive scheme gets all cell’s neighbours, and outputs the next state (live or dead) of the cell based on the rules and the received state.



The following scheme is line. It has 16 cells: neighbours, alive and state schemes.

It receives the cursor, the torus value, the input state for the cursor, rules, clock, the pause value, the clear value, the enable value, the random value, upper and lower lines.

Scheme displays the current status of the line and passes its status to lines below and above.  


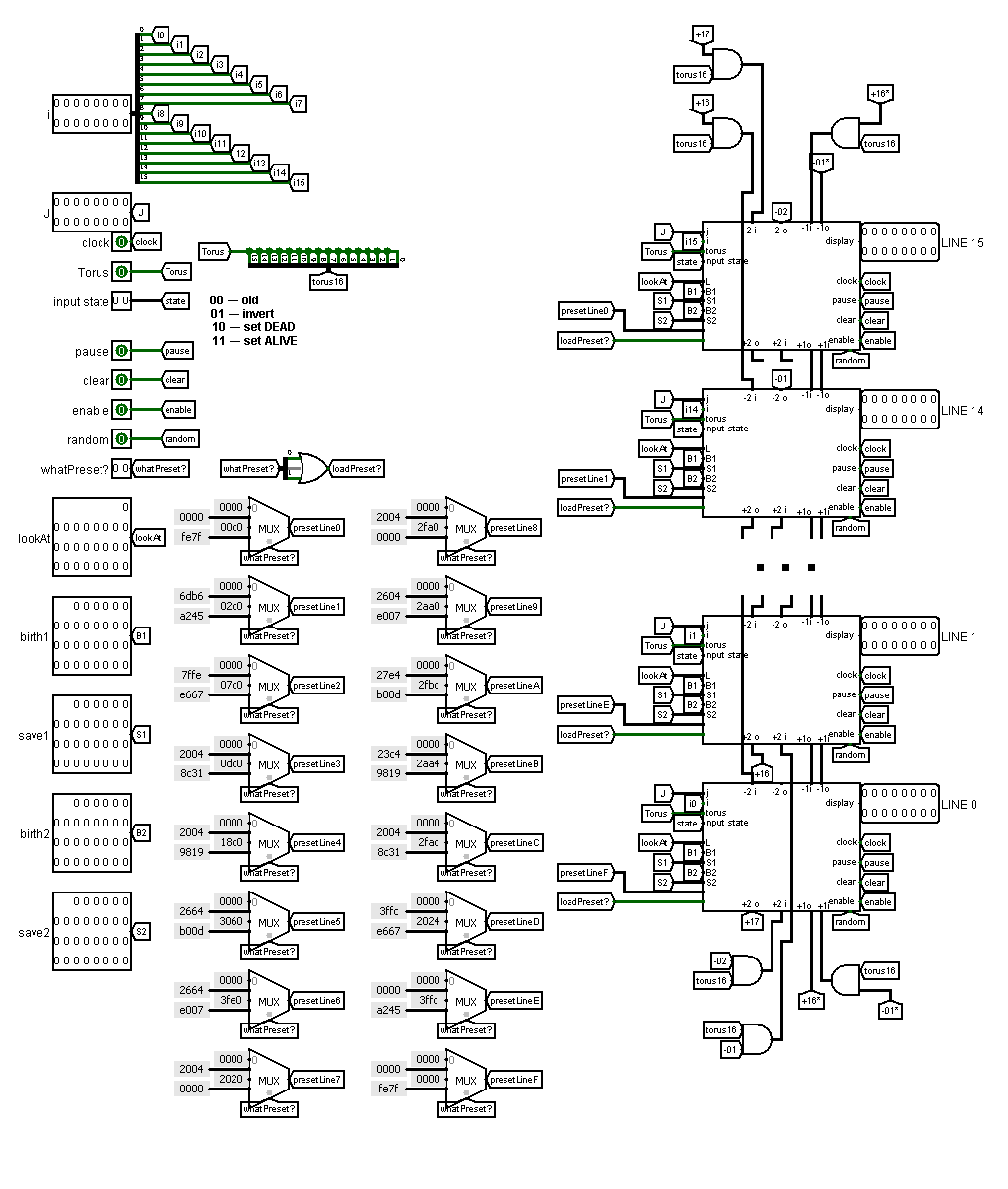
Next scheme is matrix. It has 16 lines schemes.

It receives the cursor, the torus, the pause, the state, the clear, the random, the enable, the preset, the clock values and rules. Scheme displays the field (16 lines) to the LED Matrix.

We divide the line cursor into 16 lines.

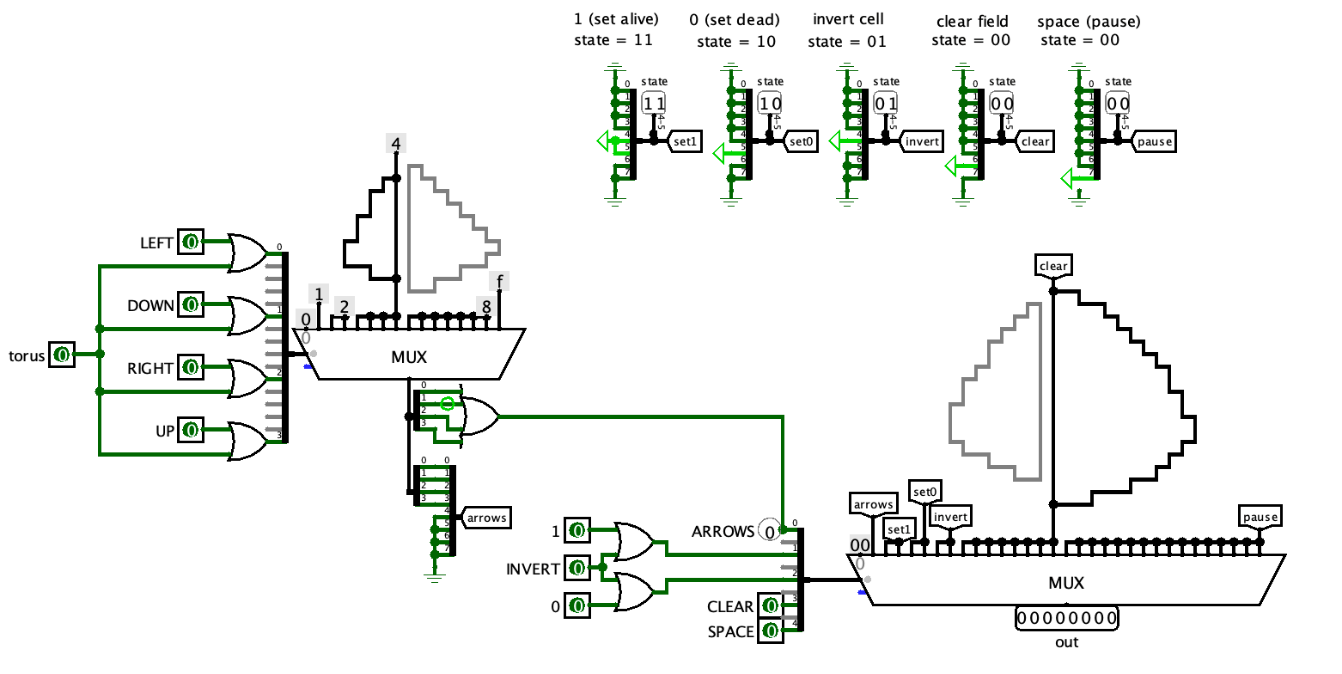
Creating a 16-bit number, each bit of which is equal to the value of the torus. It is necessary for the correct folding of the field.

The number “whatPreset?” defines the preset that will be set on the next beat. There are also multiplexers with constant values in the circuit, storing 3 presets.



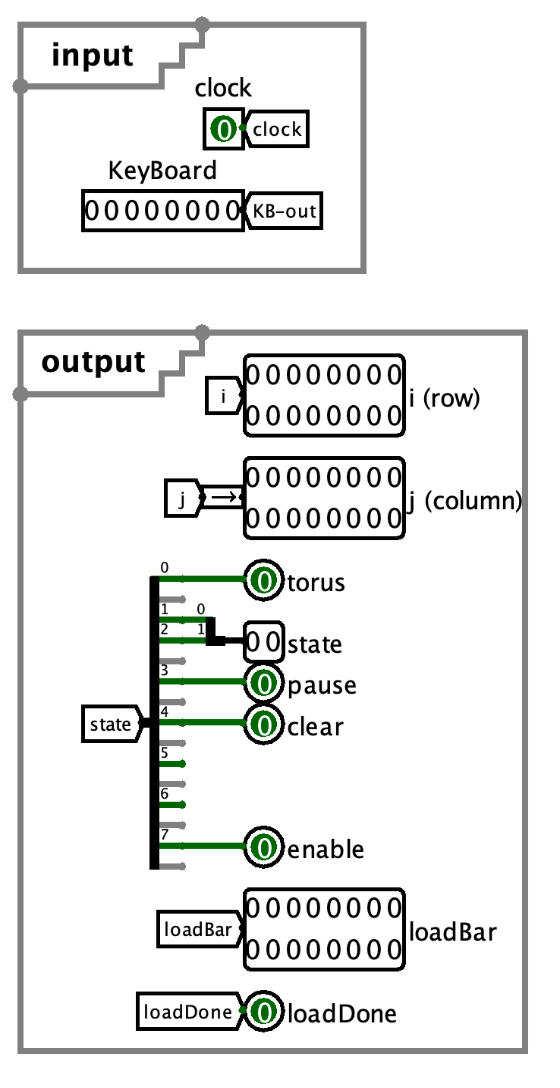
Keyboard scheme gets 10 true / false values and passes 8-bit values to processor.

It decides which button is preferable. We have made this to speed up the processing of keystrokes by the processor.

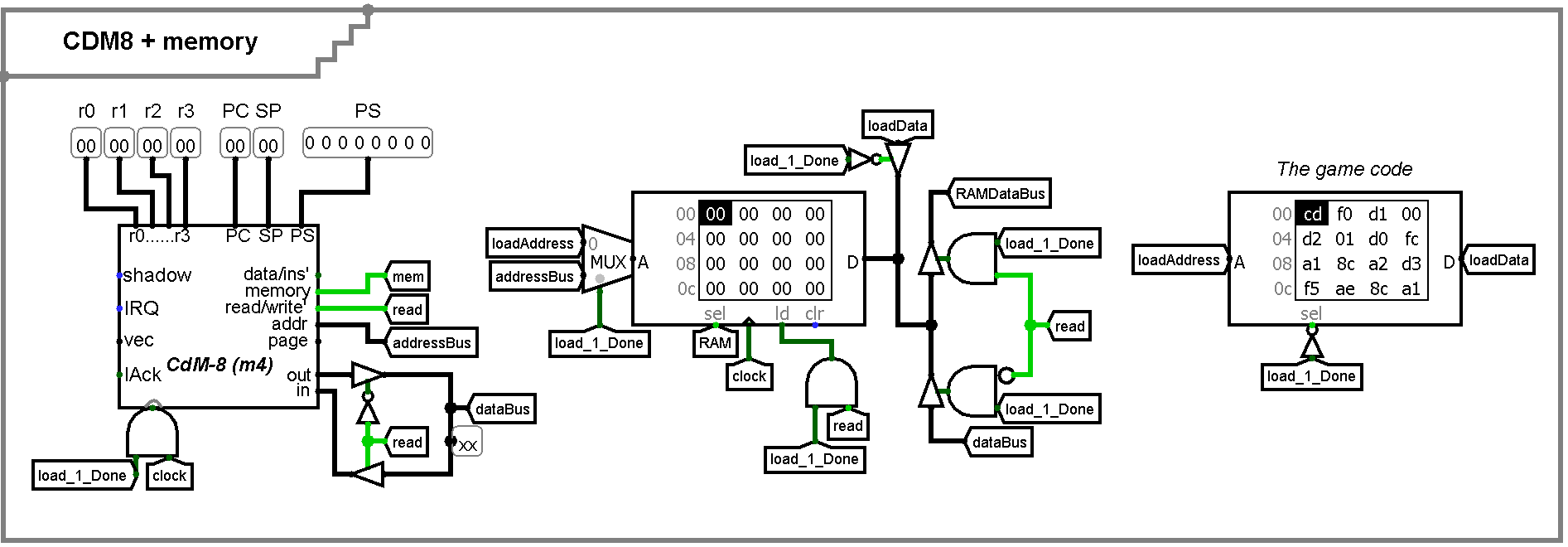


CDM8+ scheme.

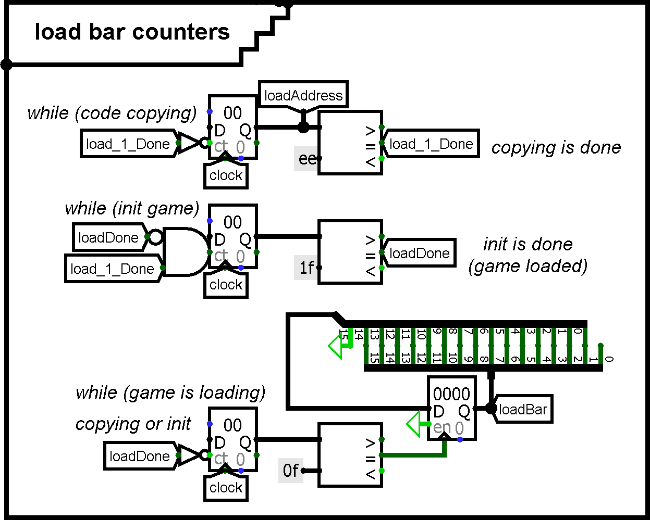
It receives only clock and keyboard’s 8-bit value.



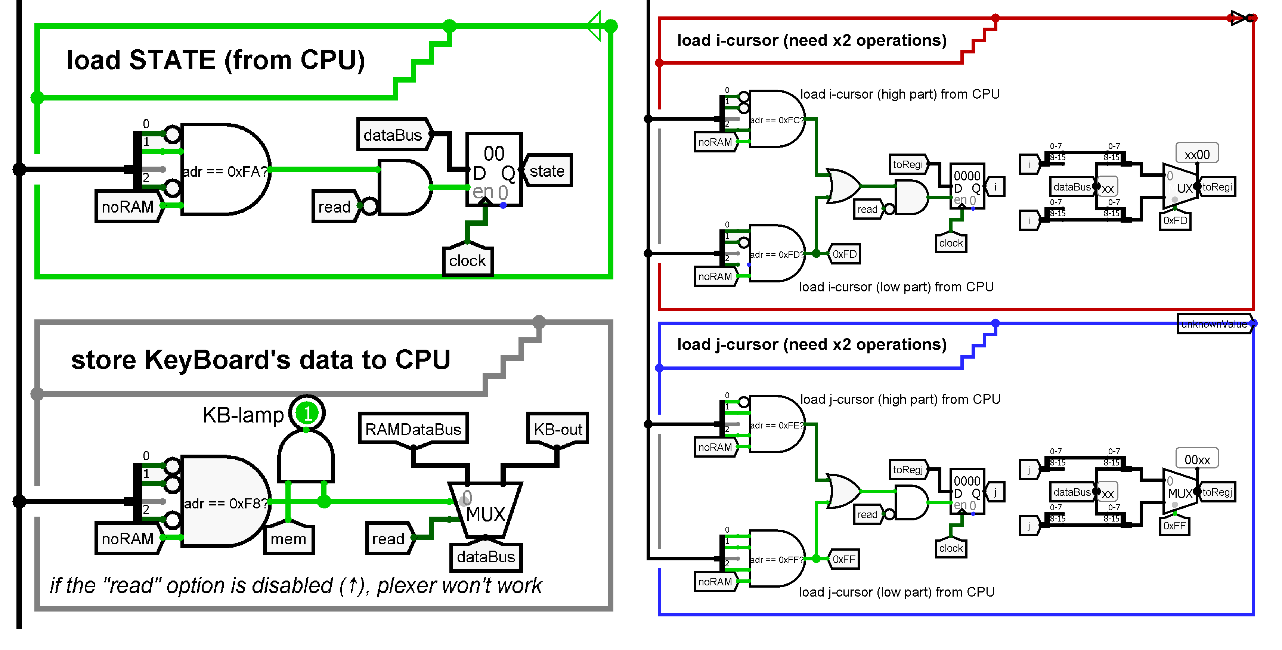
We have used CdM-8-mark4.circ as processor and standard logisim’s RAM and ROM.



Progress bar counters is part of CDM8+ scheme. It is responsible for loading bar.

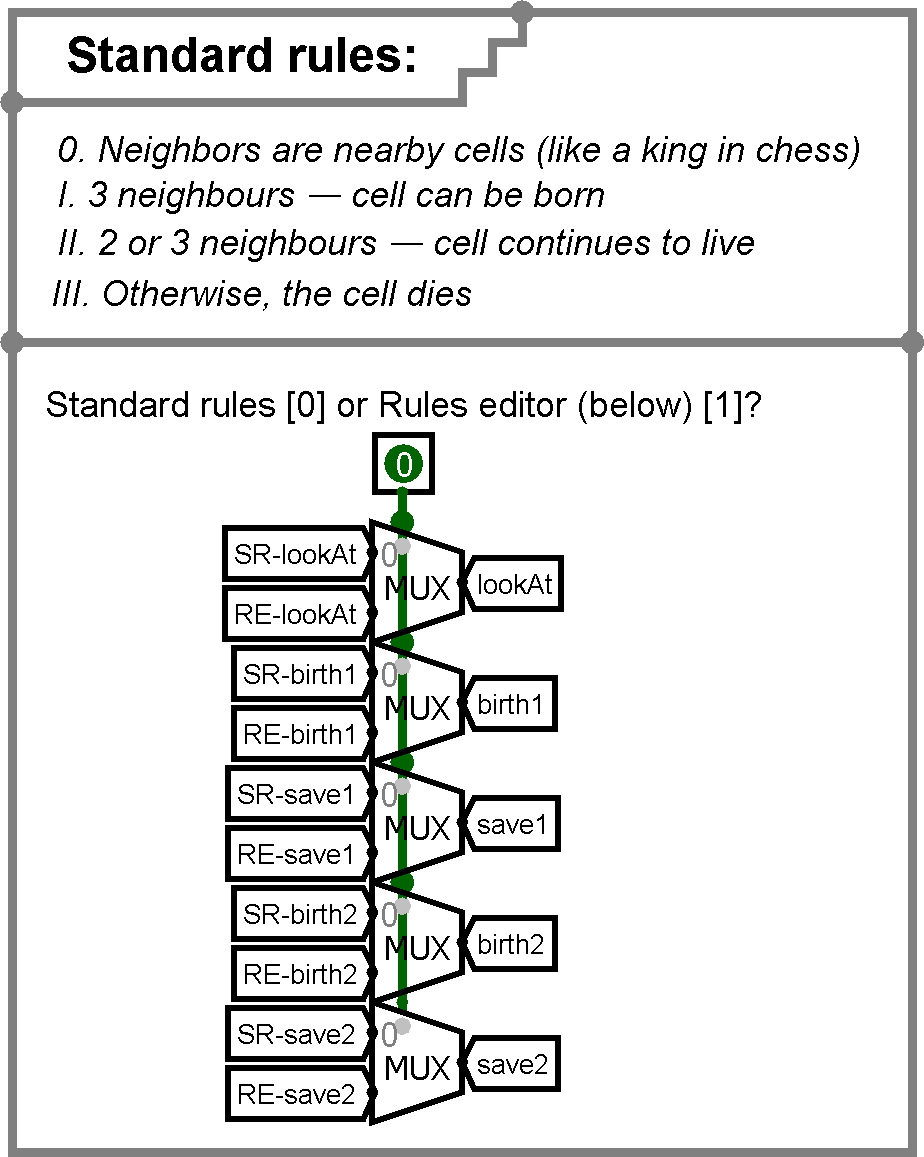


Next, we need a part of scheme for “reading” from keyboard or “writing” to matrix.

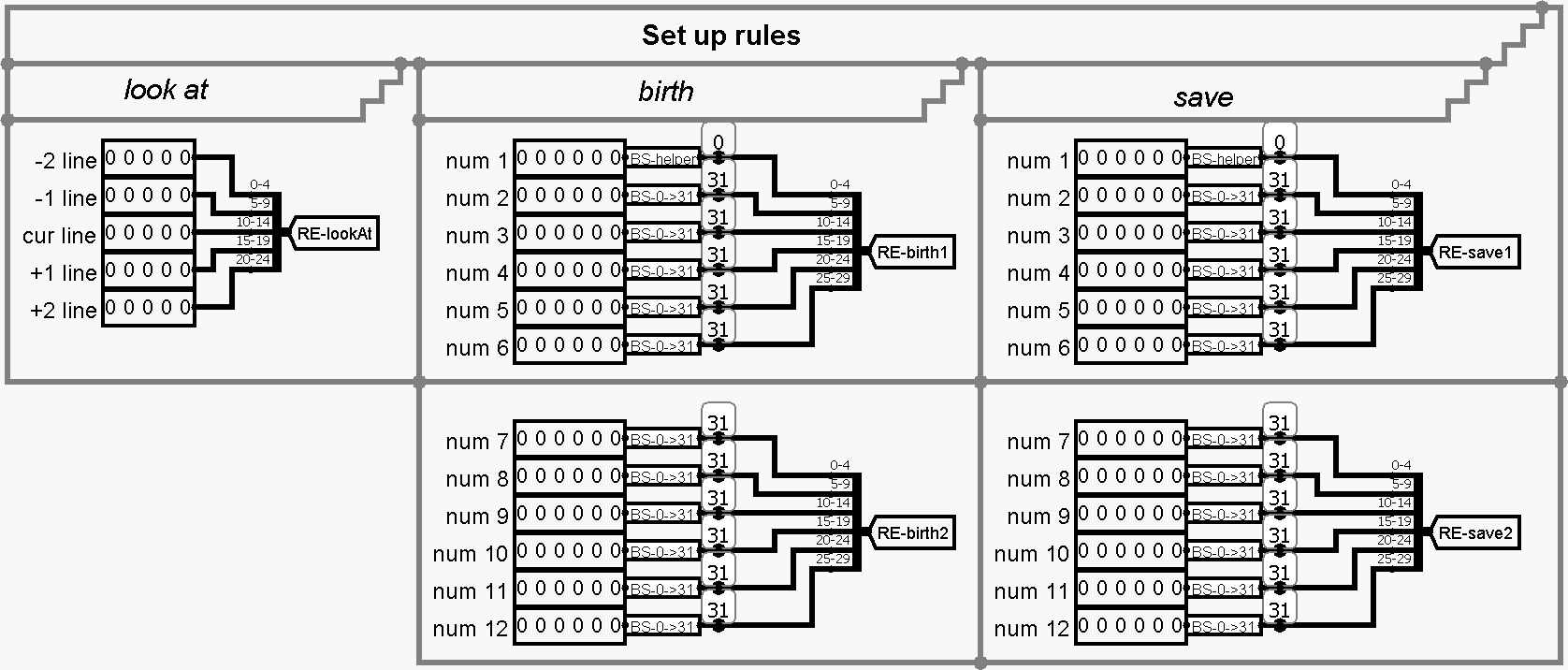


And main scheme. The user will play here.

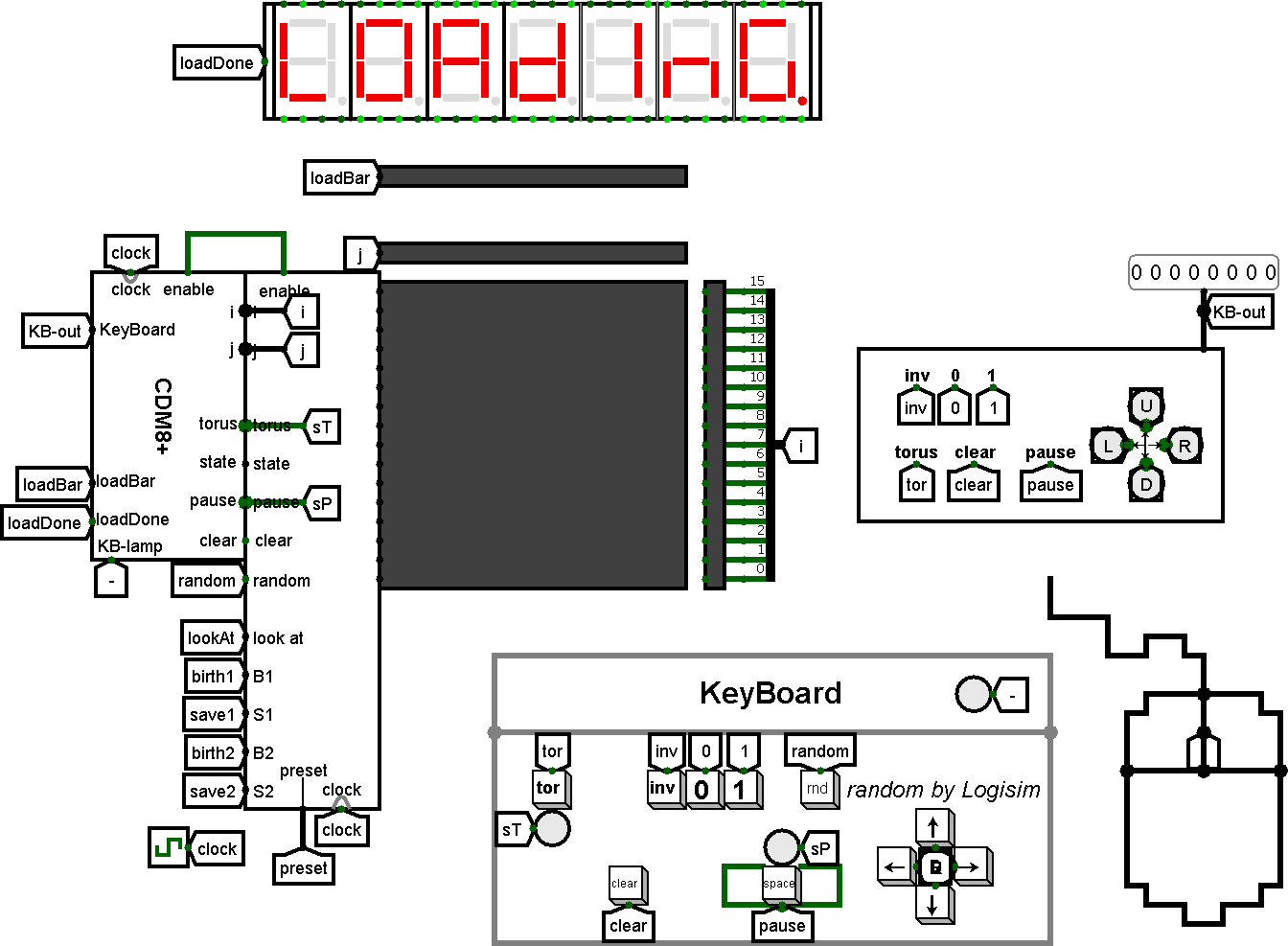
It has some information about standard rules and button to select standard rules or rules editor.



Below, the user can set up rules in the rules editor and see standard rules.



The keyboard with lights and matrix are on the right.



# Software

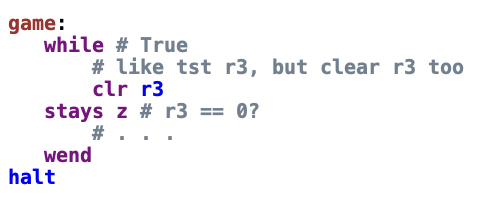
Software is CdM8 code. Our assembly code is responsible for the game management.

This function is responsible for initialization a game process. This code sets the cursor and the state in the start position.

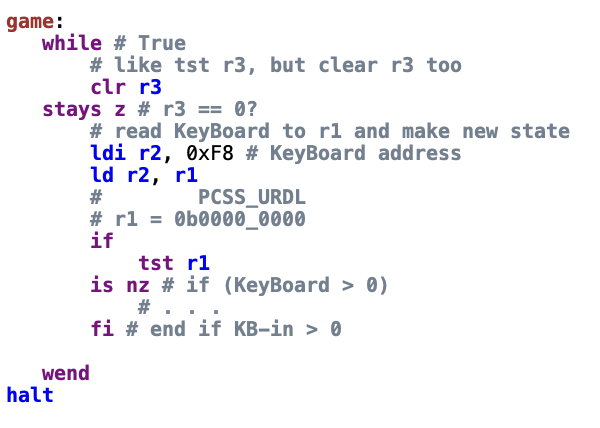


Main function in our code is “game”.

We use cycle while true because our game is infinite.



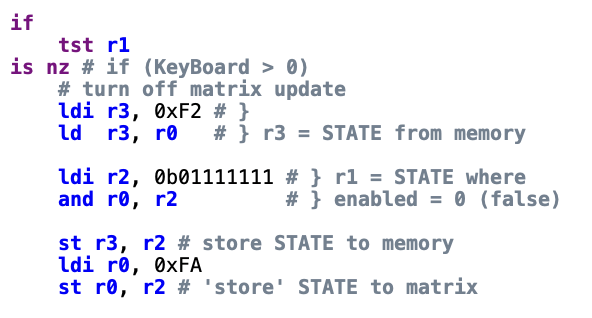
Read keyboard’s data to register 1 (r1) and make a new state.



Next, we process the data that was read from the keyboard.

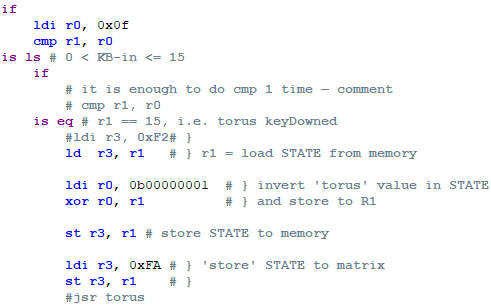
Importantly, when we are reading data from the keyboard, we are disabling matrix update.

We wanted to speed up the program, so we split the branches. It looks like a binary tree. We check whether the lower or higher 4 bits are involved.

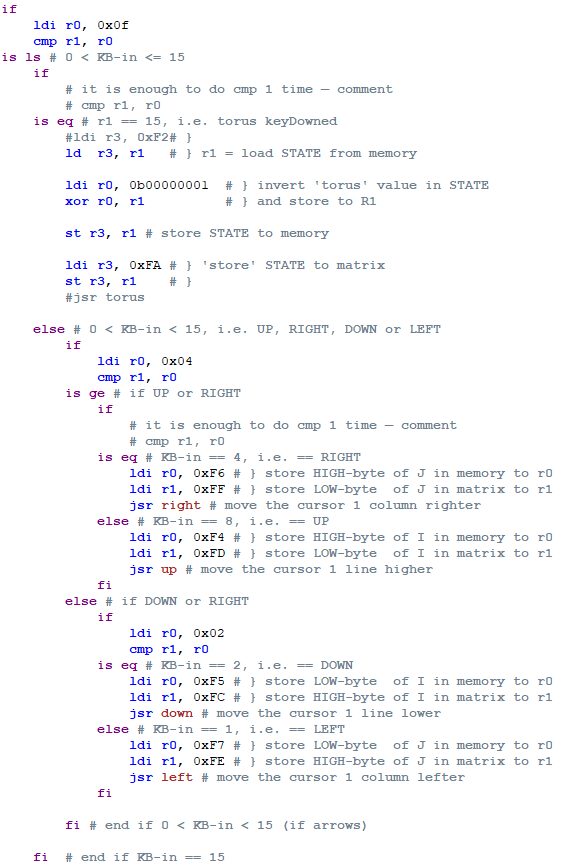


If the value is 15 or less, then user wants to use arrows or torus buttons.

If the value we got from the keyboard is 15, then the user wants to switch torus (toroidal cycled field).

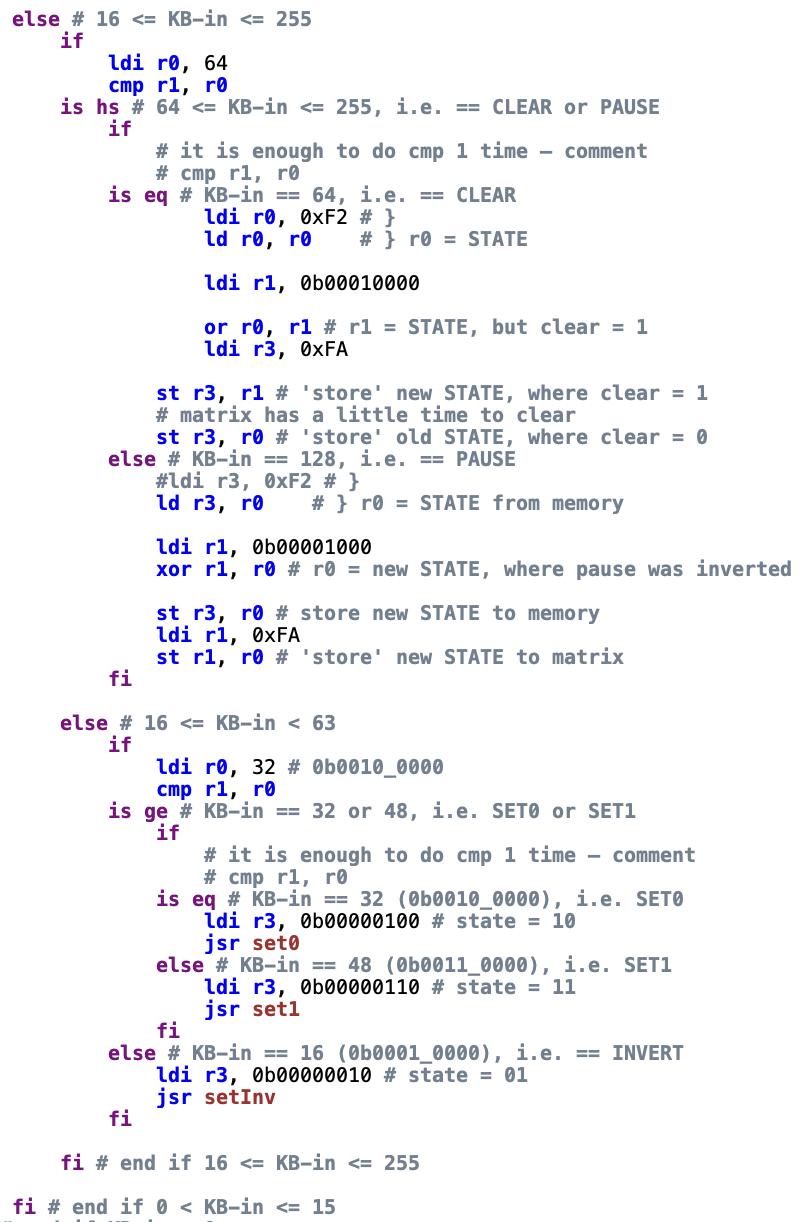


Otherwise, it means that the value is less than 15, then the user interacts with the arrows. The arrows move the cursor over the field.

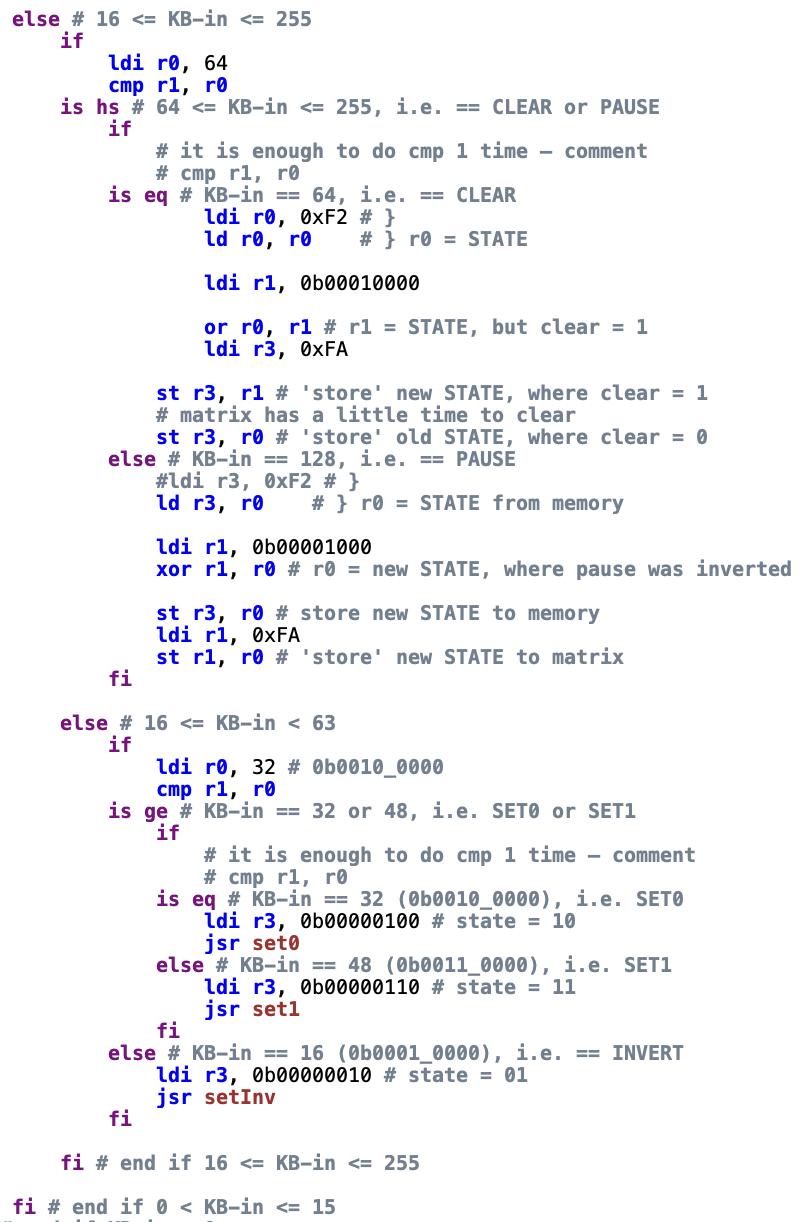


If the value is 16 or more, then user wants to change a cell (invert , set 0, set 1), switch pause or clear the field.

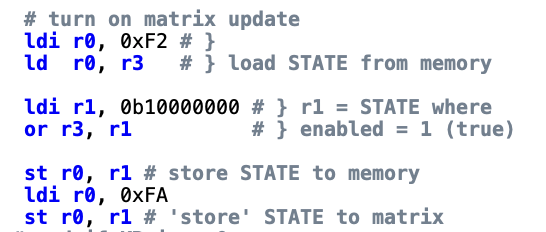
Next, if the value is in the range from 64 to 255, then the user has clicked on pause or clearing the field. If the number is 64, then clear the field, otherwise pause.



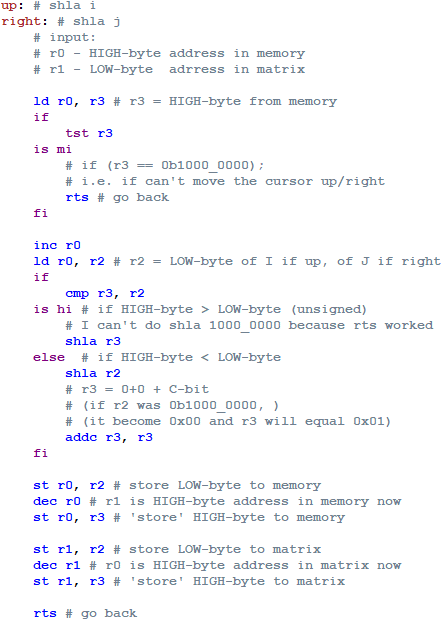
If the value from the keyboard is in the range from 16 to 64, not inclusive, then we change the state of the cell.



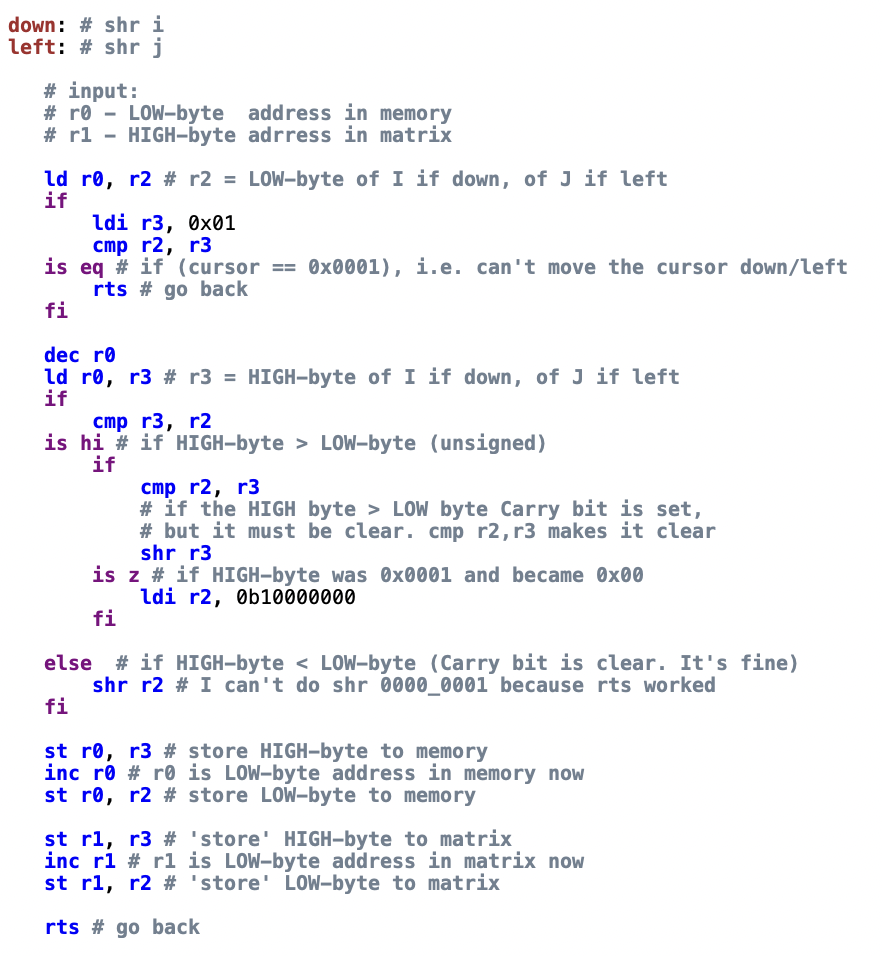
And when we have finished receiving data from the keyboard, we turn on the matrix update again.

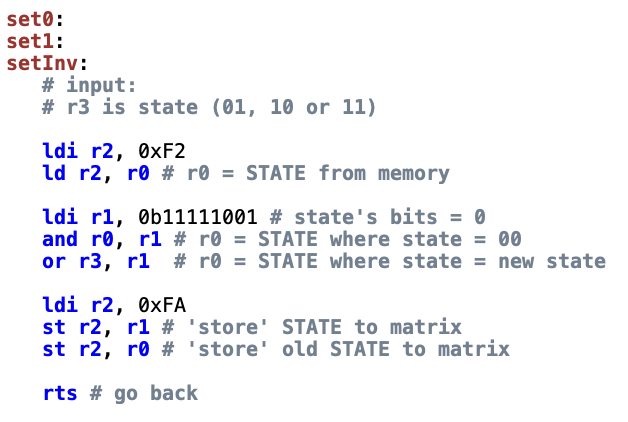


The up/right, down/left subroutines move the cursor. If the user wants to move the cursor up or to the right, we need to shift the number to the left by 1 (unsigned multiply by 2).



Otherwise if the user wants to move the cursor down or to the left, we need to shift the number to the right by 1 (unsigned divide by 2).



And the last subroutine is to change the state of the cell.

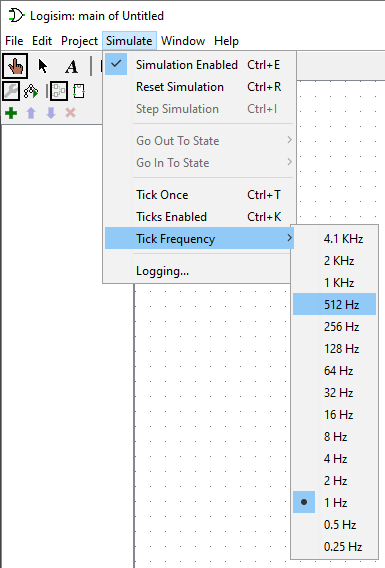
# User Guide

1. Open main.circ in Logisim.
2. Set the frequency to 512+ Hz.

**1**

**2**

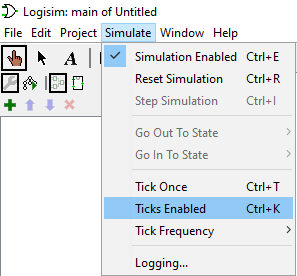
**3**



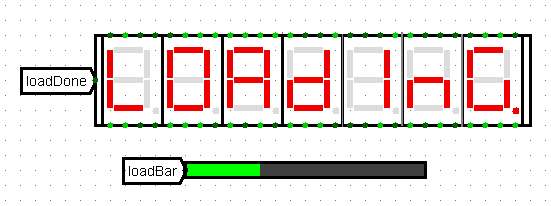
1. Turn on ticks.

**1**

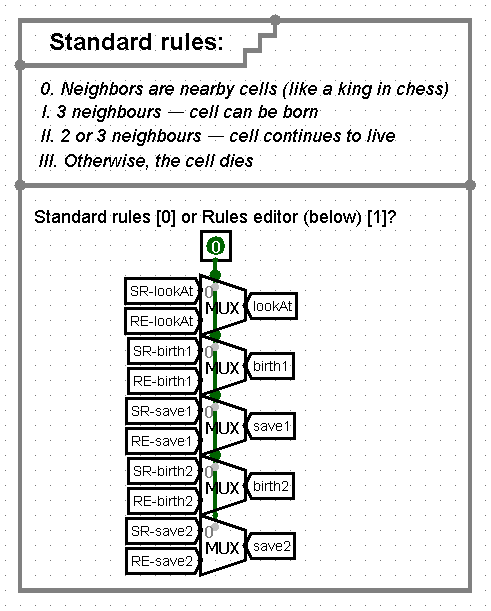
**2**



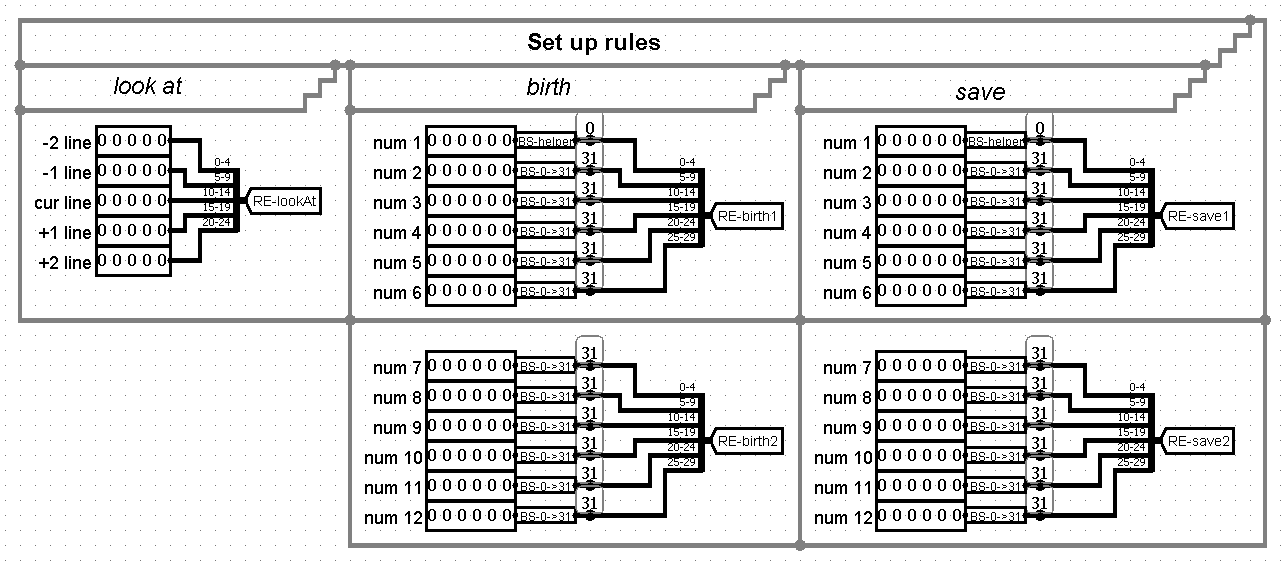
1. Wait until the download bar is full and the “play it” label appears.



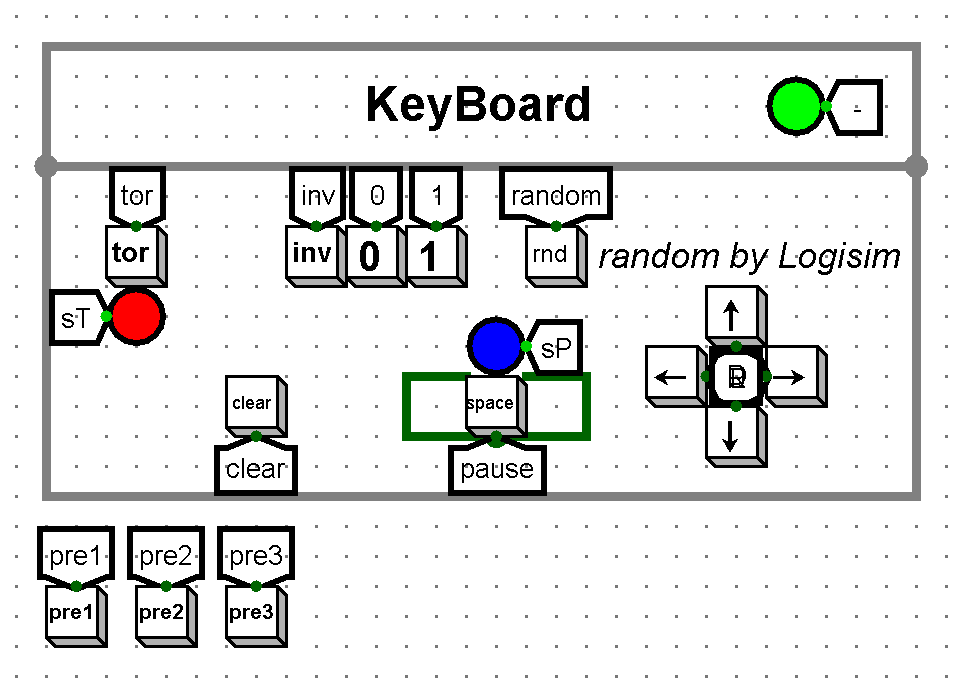
1. On the left in this scheme, you can turn on the rules editor by clicking on the button. (Default is standard rules).



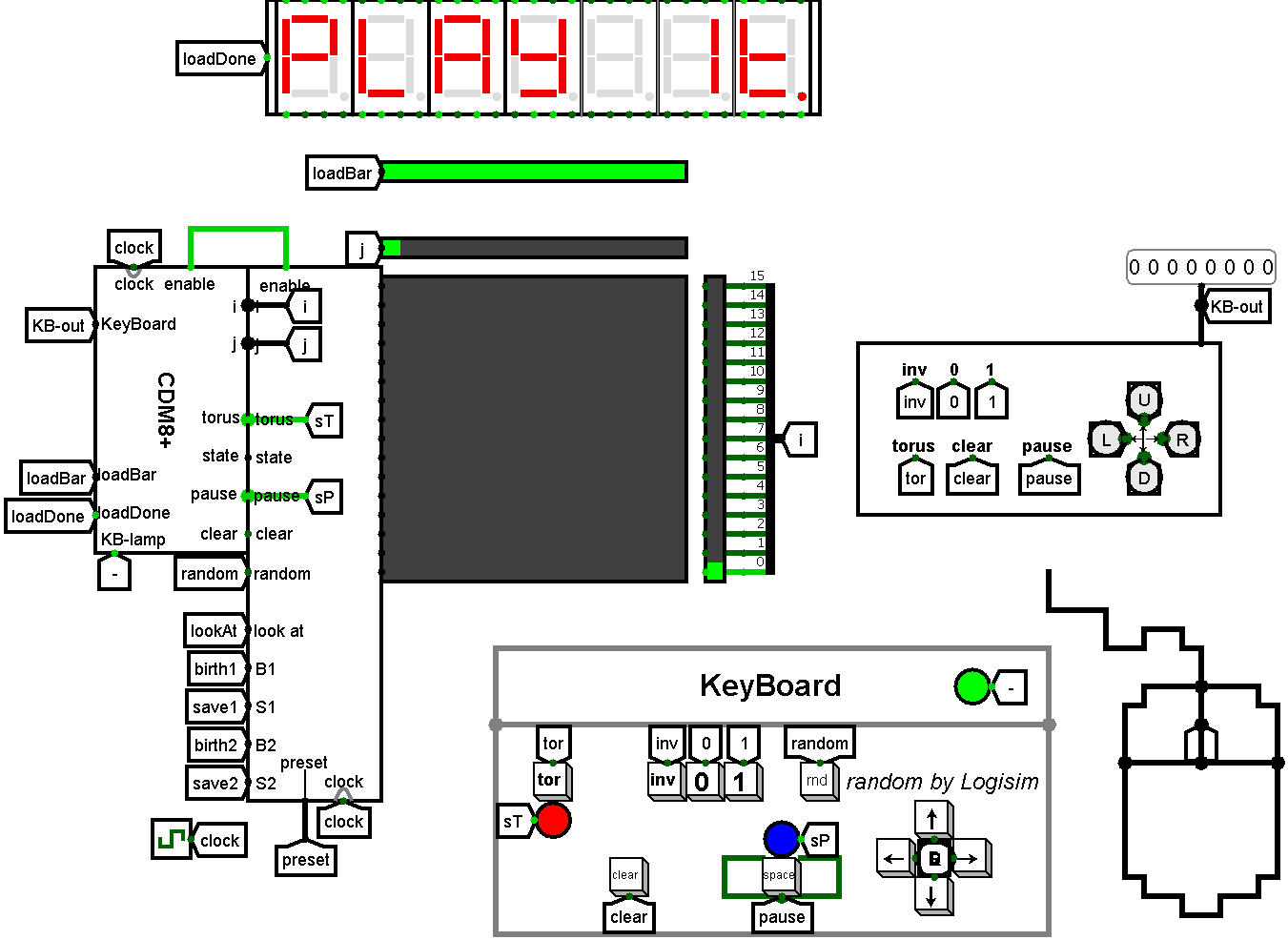
1. If the rules editor is selected, configure it below.



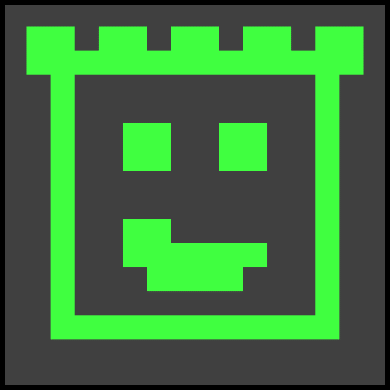
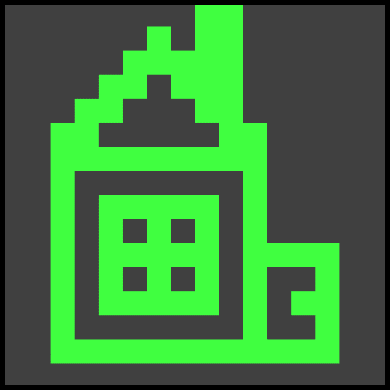
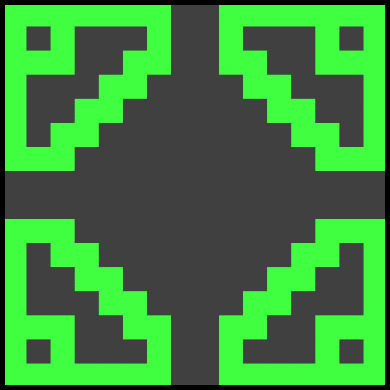
1. If the blue indicator is on, it means a pause in the game. If red light is on, then torus is on. When green light is on, processor reads keyboard’s data.



1. Play it!



1. You can also find three presets below the keyboard.

That’s all, you can enjoy the game!

# Conclusion

The end result of our project was the realization of the «Game of Life».

During the implementation of the project, we studied the history of the creation of the game, the essence of the game. With this knowledge, we were able to implement this game in CdM8 code and Logisim schemes.

In the course of our work, we fixed some bugs and have looked for the best ways to solve problems to optimize the game.

Teamwork was useful for soft skills. We correctly assigned the roles and performed the work efficiently.

Ultimately, we improved our programming, circuit engineering and communication skills.