Full Guide - HCA Simulator 2

Table of Contents

1 Introduction

This guide's goal is to make users more comfortable using this application, by understanding each and every functionality, but also for developers that would like to continue the work on this simulator.

We will also cover some of the results we achieved with this simulator, as well as some errors encountered and finally the limits of such experiments.

Through the document, you may find boxes with different kind of informations. The lime boxes are for tips, the orange ones are for warning, and the red ones are for very important informations.

All of the **Optional** sections and subsections are required for developers and people that want to access and/or modify the source code. If you are only using the project from a .jar file for example, you can skip these sections.

2 Prerequisites

We will discuss in this section the tools you will need in order to correctly use the simulator.

Compatibility Warning

This simulator was developed under Ubuntu 20.04 and only launched on this environment. The compatibility of this simulator with other operating systems in unsure (especially Windows).

2.1 OpenJDK

As this application is made using Java, it is pretty obvious that you will need OpenJDK installed on your machine to make the simulator run. Regarding the version, no version is really needed, but the latest ones will do the job. Here is how to install OpenJDK 17 on your machine (Linux).

```
$ sudo apt-get install openjdk-17-jdk
```

OpenJDK Tip

For Windows users, please check the official documentation of OpenJDK for more information. But as stated above, if you are running Windows, you may as well install a Linux Virtual Machine or make a dual boot to run the simulator safely.

In order to verify your Java installation, you may run the following command on your terminal.

```
$ java -version
```

2.2 Eclipse (Optional)

For Developers

This section is optional and is required only for developers using and upgrading the simulator.

The whole simulator has been developed under the Eclipse IDE, although it is not required to be under this one, it is recommended. We will explain in this subsection how to install the IDE and clone the project inside of Eclipse.

2.2.1 Eclipse - IDE Installation

Here are the required steps to install Eclipse IDE. You can also follow the Eclipse Documentation if needed, which may be more accurate than this.

- Go on the download page on the Eclipse website and download the installer. Current version is 2021-09, you should pick the latest stable version of it, as recommended.
- You may now extract the downloaded archive to get to the installer.
- Run the **eclipse-inst** file located at **eclipse-inst-linux64/eclipse-installer** by right clicking it and selecting Run.
- Select **Eclipse IDE for JAVA Developers**, then **Install** and finally **Launch**.

2.2.2 Eclipse - Project cloning

Gitlab Tip

This subsubsection is dedicated to the usage of Git inside of the Eclipse IDE, if you do not wish to use Git inside the IDE, you can safely skip this, and go to the next subsection.

You should follow these steps to clone the project inside of the Eclipse IDE.

- Select the following inside Eclipse : **Window** → **Show View** → **Other**. Then type in **Git Repositories** and select it.
- You now need to get the Gitlab URL (with HTTPS). You can it from the Gitlab home page of the project by clicking on the **Clone** button.
- On the bottom left of your Eclipse window, you should now be able to click on **Clone a Repository**.
- Once selected, you should have this interface, as shown in *Figure 2.1*. You must fill in the form **exactly** as shown in the picture.

  Eclipse Git Warning

  The **Location** and **Connection** sections of the form should be filled automatically already. You still may need to manually verify it.

- You now have to fill the Authentication section with your **Telecom Paris personal Email** and **Telecom Paris personal Email password**.

  Authentication Warning

  You **may** need to change your Gitlab password before doing this procedure, as Gitlab could ask for a modification to authenticate you. This error is not detailed in Eclipse if the process fails !

- Your project should now successfully be cloned inside of your Eclipse IDE.

2.3 Git (Optional)

For Developers - Gitlab Warning

If you already did the previous step by adding Gitlab to the Eclipse IDE, this subsection is useless. Also, this section is optional is required only for developers using and upgrading the simulator.

If you chose not to use the builtin Gitlab integration of Eclipse IDE, you will have to use Git in your terminal. Here is the terminal command needed to install Git on your machine.

```
$ sudo apt install git
```

In this guide, we assume that you have already an average understanding of how Git works, and how to work with it. If not, here are some resources for you to learn Git.

- Official Git Documentation
- First Time Git Setup
- Gitlab guide for Git command line

Windows Git Tip

If you decided to stay on Windows, you can use a great alternative, which is Git Bash. Other options can be used as WSL for exemple.

3 Open The Project (Optional)

For Developers

This section is optional and is required only for developers using and upgrading the simulator.

Once your project is cloned, you have to locate it. It should be the place where you launched the command line for cloning (if you followed that way). You want to move the whole project (the **hca** folder) into your **eclipse-workspace** folder, which is the one Eclipse opens by default.

Once you moved it into the workspace, go back to your Eclipse IDE, and click on the following: **File** → **Import** → **Existing Project Into Workspace**.

An interface should pop up, and the root directory should already be filled with the root of the project you just cloned. If this is not the case, you will have to do it manually.

Opening Projects Warning

You **have** to select the **Search for nested projects** option, otherwise only one of the projects contained in the directory would be opened in Eclipse.

You should now see all the projects listed on the left side of your Eclipse IDE.

4 Create a Running Configuration (Optional)

In Eclipse, in order to run the project inside of the IDE you need a Run Configuration. This section will explain you how to create the correct one.

You need to click on **Run** → **Run Configurations** on the top menu of your Eclipse window while you're on the project. An interface should pop up, then right click on **Java Application**, then click on **New Configuration**.

Then, you need to fill the name of that configuration, a good name would be **simulator_main**. Your project should be set to **simulator** and you main class should be **main.Main**. No arguments are needed for this application, so you can save and launch this configuration when you want to start the simulator.

5 Main Menu Parameters

This section will focus on the main menu parameters you can select before starting the simulation.

5.1 Start and Exit Buttons

The Start button will launch the simulation. We will explain later in this guide the simulation itself, for now the only thing to understand is that the start button takes the parameters of the menu, and launches accordingly a simulation.

The exit button obviously exits the program.

5.2 Rules Selection

The Rules selection on the menu is very easy to understand. The buttons on the line **Alive** represent the alive rules, and the ones on the line **Dead** the dead rules. Each button represents the behavior of a cell of the bottom automaton in a specific case.

For example, if the button of line **Alive** and column **0** is enabled, an alive cell with 0 neighbors will stay alive in the next iteration of the simulation. Or if the button of line **Dead** and column **5** is enabled, a dead cell with 5 neighbors will become alive in the next iteration of the simulation.

Default Rules Tip

When starting the program, there are default rules already enabled. Those rules are the ones of the Game Of Life.

5.3 Initial Pattern Selection

The simulation makes sense only if some cells are alive at the first iteration, to see how it goes. In order to make this simpler, some basic Game Of Life patterns are pre-loaded in the simulator. You can select one on the **Select Initial Pattern** gliding menu. Here are the available patterns.

- The Glider.
- The Blinker.
- The Block.
- The Herschel.
- The Glider Gun (Gospel).

5.4 Initial State Selection

In some cases, you do not want to have a simple pre-loaded pattern, but a custom initial state. In the gliding menu **Select Initial Pattern**, if you choose **Personal**, you can do so. In fact, if you select it, it will take your own grid.

To access this grid, you simply have to click the **Initial State Selection** button, which will open a pop-up. You can click any cell of this grid to light it up or shut it down. When you are done selecting your custom initial state, you can just click **Save**. The **Clear** button will erase all alive cells of your personal grid.

The simulation will then use your own grid as initial state for the simulation.

Pattern Overriding Warning

You have to be careful when making your own custom initial state. If you make it but do not select **Personal** as Pattern, the selected pattern will **override** your custom grid and launch the simulation anyways.

5.5 Iteration Number Selection

The iteration number selection is only the number of iterations the simulator will execute before stopping and analyzing the grid. It can obviously not be 0 or lower. Its default value is 100.

5.6 Middle and Top Abstraction Methods

This simulator is a Hierarchical Cellular Automaton Simulator (HCA), so it has multiple levels, 3 for now. The level 0 only computes its state from the rules indicated by the users beforehand. The automata of Level 1 and Level 2 deduce their state by looking at the grid of the automaton of Level n-1. For example, a Level 1 automaton will get access to the Level 0 automaton's grid, and via a abstraction method, deduce its own state. The user is the one who is responsible for the selection of these methods. You can select none, resulting in a simulation containing only the Level 0 automaton, or all of them for more than 6 automatons on 3 levels.

5.7 Square Size Selection

In order to determine the size of the middle automaton, you have to divide the bottom automaton in multiple squares. Each of these will represent a single cell of the middle automaton. There are four possible square sizes.

- **2x2 Square** - This will make each Middle Automaton cell represent a 2x2 cell square of the Bottom Automaton.
- **4x4 Square** - This will make each Middle Automaton cell represent a 4x4 cell square of the Bottom Automaton.
- **8x8 Square** - This will make each Middle Automaton cell represent a 8x8 cell square of the Bottom Automaton.
- **10x10 Square** - This will make each Middle Automaton cell represent a 10x10 cell square of the Bottom Automaton.

Square Size Warning

The size of the bottom automaton can be selected by hard-coding the size in the existing code. You have to be sure the size of the automaton is divisible by the square size you select ! Otherwise errors will rise, as this case is not checked in the simulator.

5.7.1 Middle Abstraction

The Middle Abstraction Methods take as input the bottom automaton's grid, and computes the state of the middle ones. The only possible abstraction available at the moment is a threshold of cells.

In fact, for every cell of the middle automaton, the program will check the state of the related square cells in the Bottom Automaton, and if the number of alive cells is higher or equal to the threshold, the cell will be set alive for the next iteration.

In order to add a middle automaton, simply enter the threshold in the correct text area in the menu interface. If you want multiple abstractions on the same simulation (ie select multiple thresholds), seperate each threshold with a comma, **without space**, as shown below.

Example of multiple seleted thresholds : $\boxed{\textbf{1,2,3,10}}$

Middle Abstractions Tip

If you want to launch a simulation without any abstraction / middle automaton, leave the textbox empty.

Middle Abstraction Methods Warning

Warning, for every abstraction method selected, a new middle automaton will be create. This means that there can be as many middle automata as there are middle abstraction methods.

5.7.2 Top Abstraction

The Top Abstraction work just as the Middle one. As the top automaton is only composed of a single cell, there is no such thing as mapping squares from the automaton right under it. You only have to select one or multiple threshold (with the syntax shown in the previous section). The simulator will go through the middle automaton, and detect if the cell threshold is reached; if it is, the unique cell of the top automaton will be set alive, otherwise set dead.

Top Abstraction Methods Warning

Warning, for every abstraction method selected, a new top automaton will be created for every existing middle automaton. This means that the number of top automata displayed will be the number of middle automata existing times the number of top abstraction methods selected.

5.8 Detailed Log Button

This button enables the detailed logger. By default the logger will only output warnings and errors if some occur. Enabling the detailed logging will allow the logger to output informative data while processing. It can be used for debug if needed.

6 Simulation

Now that the process of filling up parameters in the main menu is clear, we can explain the simulation itself, and what does everything mean.

6.1 Alive and Dead cells

The most important thing to understand is that the alive cells are the one that are coloured. The dead cells are in dark gray color. The alive cells always have one of the following colors : Magenta, Green, Cyan, Yellow, Blue and Red. We will explain just after this why we need multiple colors to indicate alive cells.

Here is an example of a Glider in the simulator. Its cells are colored in Magenta. You can see this in *Figure 6.1*.

6.2 Blocks

This is here that the different colors make sense. We're going to explain what are block and how they are determined.

We call **Block** a group of cell that are adjacent one to another. Blocks can be as small as a single cell and as big as the whole grid. Each block has its own color, this is why there are multiple colors to represent alive cells. Here is an example of a single Blinker, recognized as a single block in *Figure 6.2*.

Now what if we got multiple blinkers on the same simulation. Here is the result in *Figure 6.3*. We can see that, being different blocks, the cells are coloured accordingly to each block.

Blocks Tips

Each block has a unique block ID, from which we deduce the assigned color for its cells. Also, having only 6 different colors for alive cells, if there are more than 6 different blocks in the simulator, some will have the same color.

Block division (if a block at iteration n is divided in 2 or more distinct blocks at iteration n+1) and block merging (2 or more blocks merging into a new one) are handled in the simulator. At each iteration, blocks are re-calculated by the simulator. This simply follow the following rules.

- If a block is divided into two or more new blocks, one of the resulting blocks will keep the same ID and history. If a block is divided into n separated blocks, it will result into the creation of n-1 blocks. One of them will stay the original block with its ID and history.
- When blocks are merging, all the old blocks are removed, and a new one is created. There is no such thing as a block "eating" another one. If a merge of n blocks occurs, all of the n blocks will be removed and a new one will be created.

Block Merge and Division

Warning, it is possible that a division and a merge happen at the same time on the same cells. The previously explained rules are still valid. Please check the code for detailed explanation.

6.3 Mathematical Analysis

After the number of iteration selected by the user in the menu GUI before starting the simulation, the simulation stops, and the mathematical analysis of blocks start.

The goal of this mathematical analysis is to turn every block's movement into a mathematical function. In order to do this, we apply the Linear Least Squares Regression method. The benefit of using this method is that the mathematical function resulting will always be affine (of form : ax + b).

For each block, the function is computed by the simulator, and displayed on the console. You can see this in *Figure 6.4*. Simultaneously, a graph will pop up with the mathematical functions drawn, to visually see the movements of blocks. Here is an example on *Figure 6.5*.

Graph Tip

The coordinates for the X and Y axis represent the cells coordinates on the grid. Also, each block has a legend on the right side and has a unique ID, to link the graph to the console output.

Static Block Warning

Warning, if a block is static and has no movement, it will not be possible to find a mathematical function to identify its movement. On the console, the following error message will be output, see *Figure 6.6*. The point on the graph will still be displayed nonetheless.

7 Simulations Results and Interpretations

This part needs to be completed.

8 Creating Jars from Eclipse

Here are the detailed steps you should follow to make a .jar file of the simulator from the Eclipse IDE.

- Right click on your project on the left menu of Eclipse IDE.
- Select **Export**.
- Select **Java**, then **Runnable JAR file**. Just as shown in *Figure 8.1*.

You now have your .jar file of the simulator project that you can launch with :

```
$ java -jar simulator.jar
```

9 Limits Of The Simulator

This section will focus on the limits of the current version of the simulator. We will have a talk about some of next functionalities that could be great to implement in the simulator. In any case, for all details, you should see the Gitlab, the code and comments.

9.1 Lisibility of the Simulation

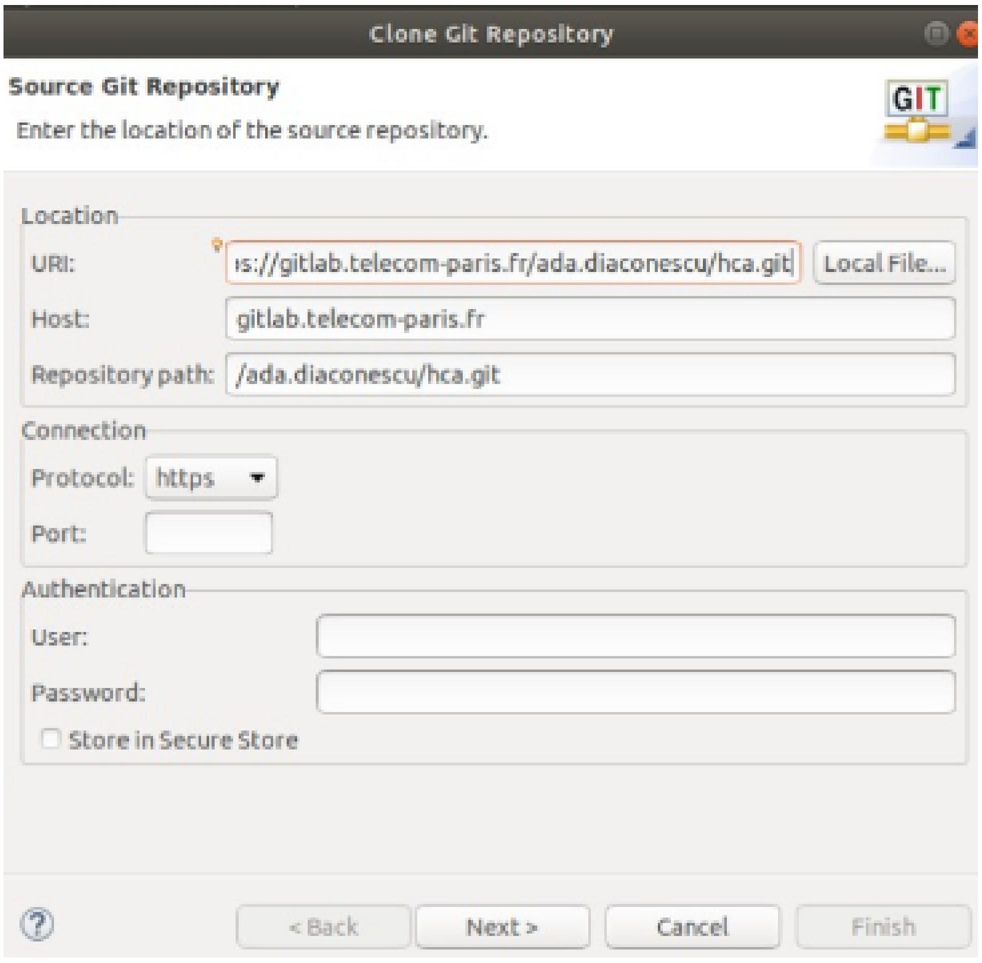
9.2 Middle Block Handling

9.3 Additional Options

9.4 Real-Time Graph

9.5 General Optimization

10 Annexes


Clone Git Repository

Source Git Repository

Enter the location of the source repository.

Location

URI: ɪs://gitlab.telecom-paris.fr/ada.diaconescu/hca.git

Host: gitlab.telecom-paris.fr

Repository path: /ada.diaconescu/hca.git

Connection

Protocol: https

Port:

Authentication

User:

Password:

Store in Secure Store

Figure 2.1 - Clone Git Repository Menu Eclipse.

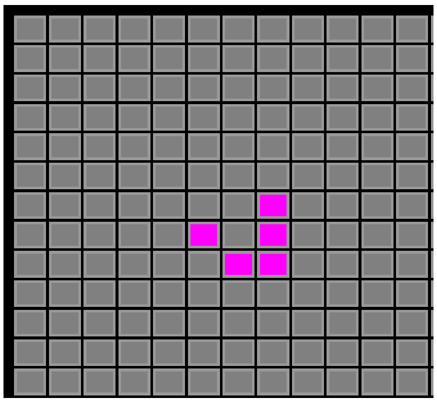


Figure 6.1 - Glider alive cells in the simulator.

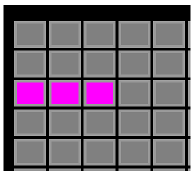


Figure 6.2 - Blinker block alone in the simulator.

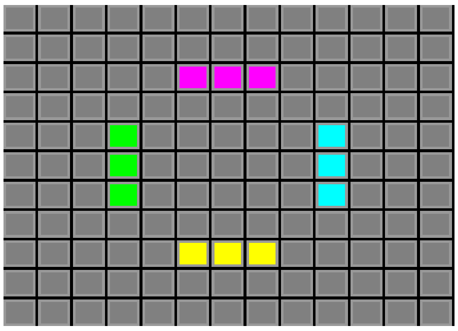
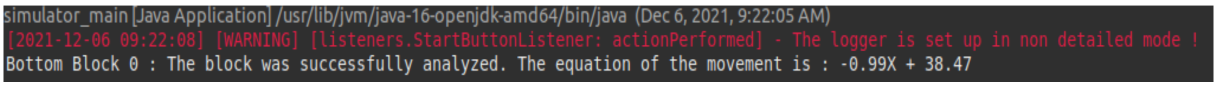


Figure 6.3 - Four blinkers being four blocks in the simulator.



```
simulator_main [Java Application] /usr/lib/jvm/java-16-openjdk-amd64/bin/java  (Dec 6, 2021, 9:22:05 AM)
[2021-12-06 09:22:08] [WARNING] [listeners.StartButtonListener: actionPerformed] - The logger is set up in non detailed mode !
Bottom Block 0 : The block was successfully analyzed. The equation of the movement is : -0.99X + 38.47
```

Figure 6.4 - Result mathematical function of default glider.

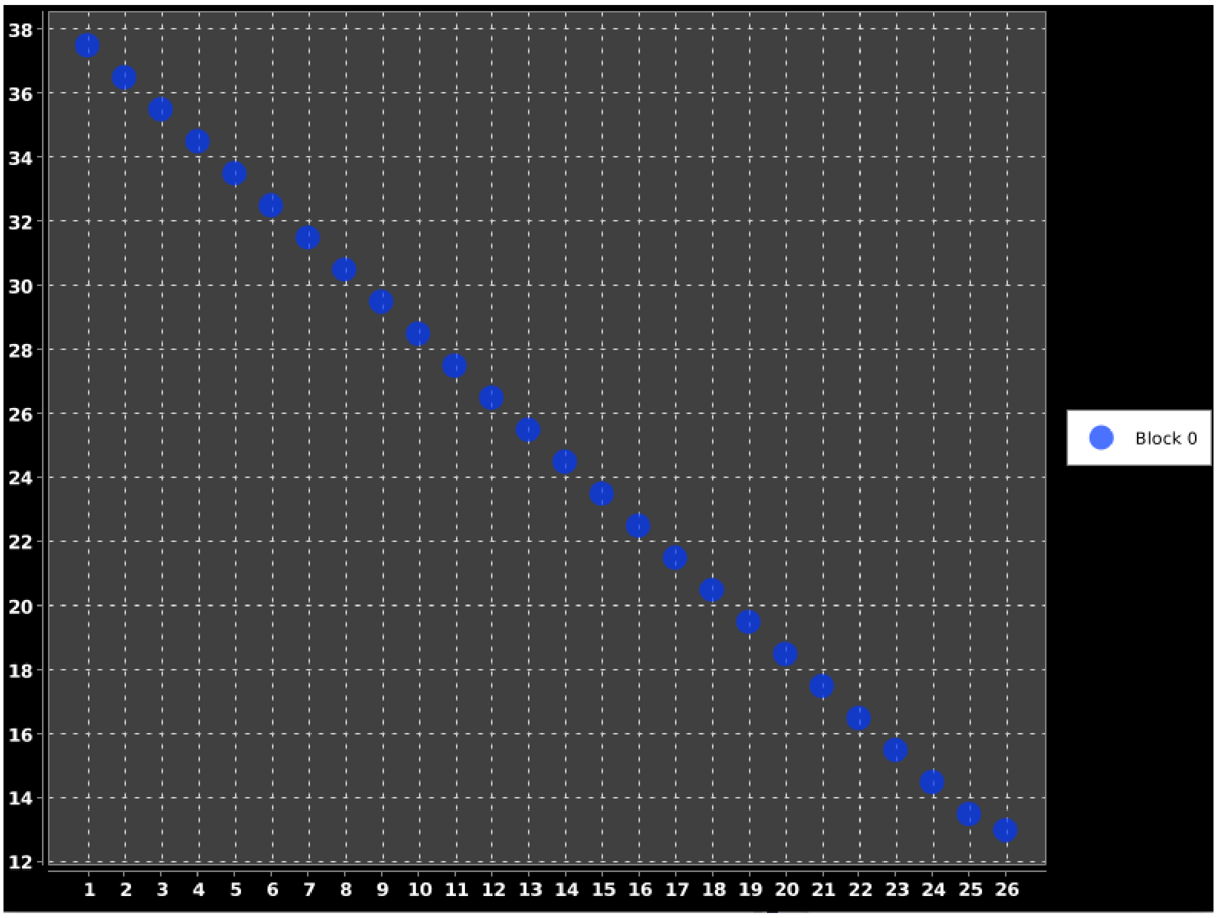
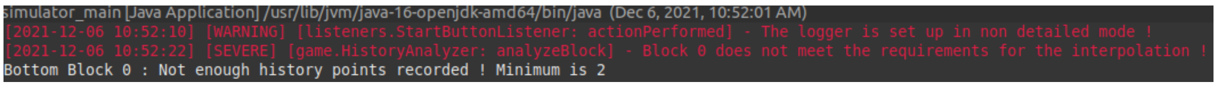
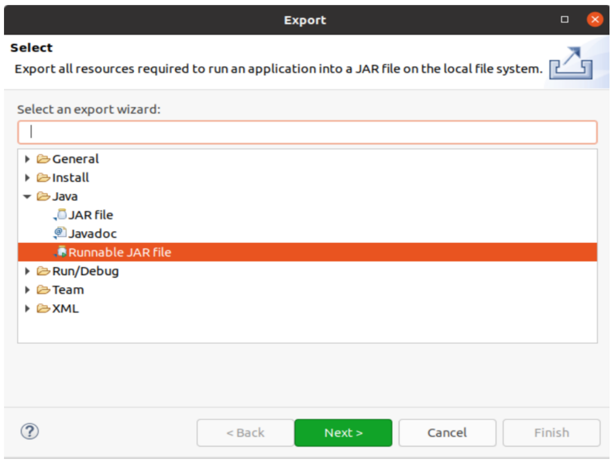


Figure 6.5 - Graph resulting of a default glider movement.



```
simulator_main [Java Application] /usr/lib/jvm/java-16-openjdk-amd64/bin/java  (Dec 6, 2021, 10:52:01 AM)
[2021-12-06 10:52:10] [WARNING] [listeners.StartButtonListener: actionPerformed] - The logger is set up in non detailed mode !
[2021-12-06 10:52:22] [SEVERE] [game.HistoryAnalyzer: analyzeBlock] - Block 0 does not meet the requirements for the interpolation !
Bottom Block 0 : Not enough history points recorded ! Minimum is 2
```

Figure 6.6 - Static block mathematical analysis - Blinker case.


Export

Select

Export all resources required to run an application into a JAR file on the local file system.

Select an export wizard:

- General
- Install
- Java
  - JAR file
  - Javadoc
  - Runnable JAR file
- Run/Debug
- Team
- XML

Figure 8.1 - Export project as .jar file.