SpaceChem Beginner Guide

Welcome to SpaceChem Beginner Guide.

Date of Release: May 31, 2023

Version: 1.0

All screenshots originate from the SpaceChem game. The logo originates from Wikipedia. Copyright: Zachtronics

Table of Contents

Introduction **3**
About SpaceChem 4
Game structure 5
Goals of the game 7
Programming mindset 9
Game mechanics **11**
Reactor structure 12
Reactor interface 14
Basic instructions 16
Advanced instructions 17
Special objects 18
Gameplay **19**
Assignments 20
Programming a reactor – part 1 21
Programming a reactor – part 2 29
Production assignment interface 39
Defense assignment interface 41
Setting up production and defense 43
Help **45**
Rules and tips 46
Troubleshooting 48
How to improve performance 50
Bibliography 51
Glossary 52

Introduction

This chapter includes general information about SpaceChem, its structure and goals you are supposed to accomplish.

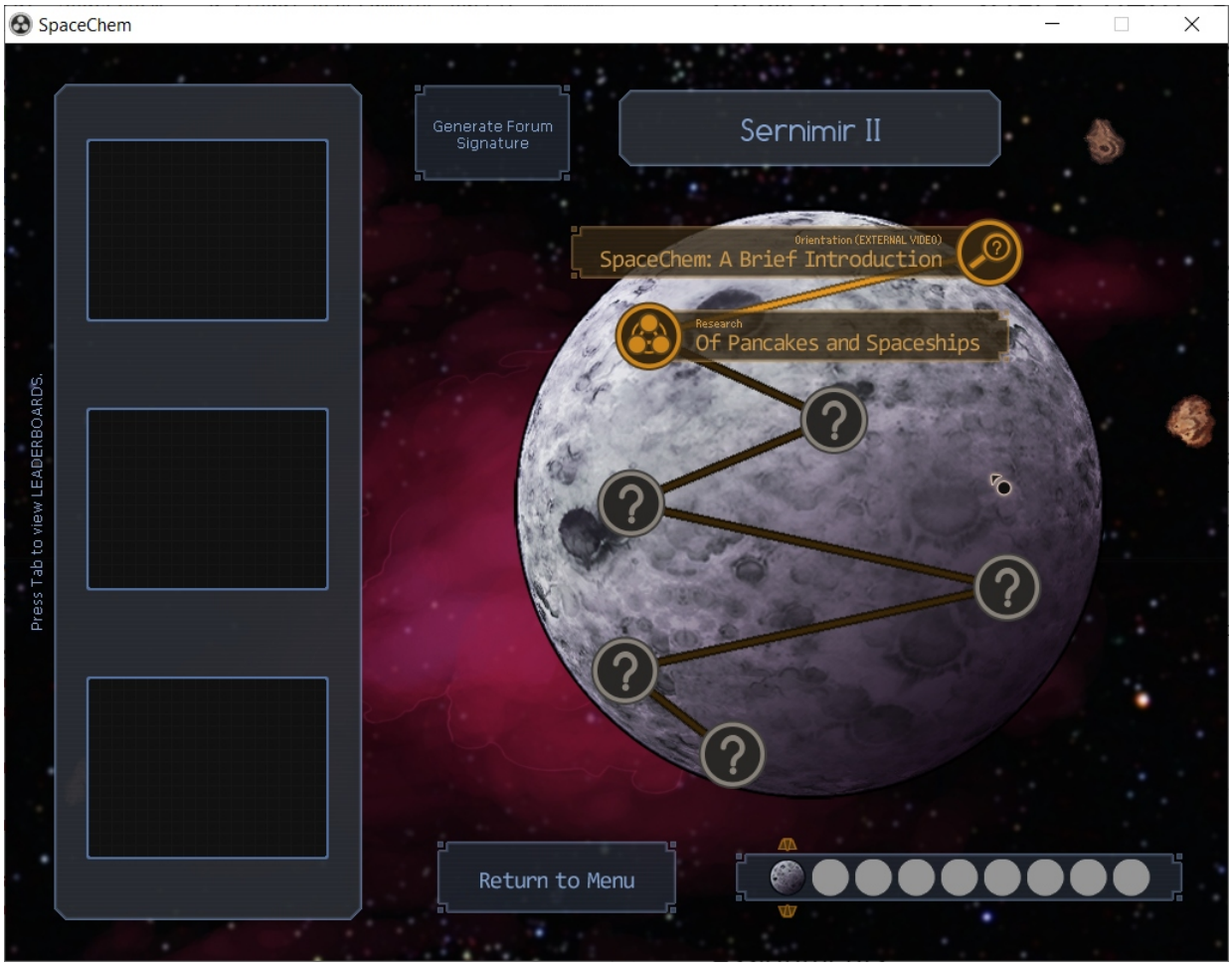
- About SpaceChem
- Game structure
- Goals of the game
- Programming mindset

About SpaceChem

SpaceChem opening screen

SpaceChem is a puzzle game by Zachtronics, which was released in 2011. The plot of the game revolves around SpaceChem, a fictional chemical engineering company dealing with extraction of various more or less real minerals on fictional planets, in a distant future. The player is employed as a Reaction Engineer Trainee and is assigned tasks related to chemical engineering and programming. Some of them involve programming reactors in order to execute chemical reactions, while others consist in combining the outputs of several reactors. The player uses a simple Turing-complete visual programming language to create algorithms.

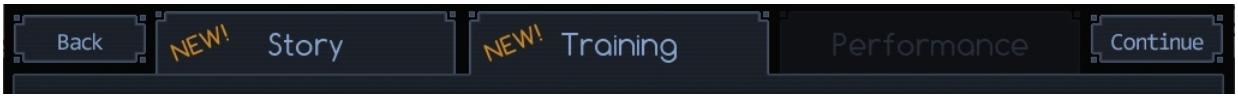
Game structure



First planet opening screen

The game is divided into 9 planets, on which the player solves individual levels, or assignments. The game progression is predominantly linear – in most cases, after solving a level, a new one is unlocked. In rare cases, after completing a level, more than one level is unlocked or two levels must be beaten to unlock a new one. From the third planet on, the final level on each planet is a "boss fight" – in other words, a defense assignment. Apart from the above-mentioned types of assignments there are challenges. These are more difficult optional assignments, so they can be skipped.

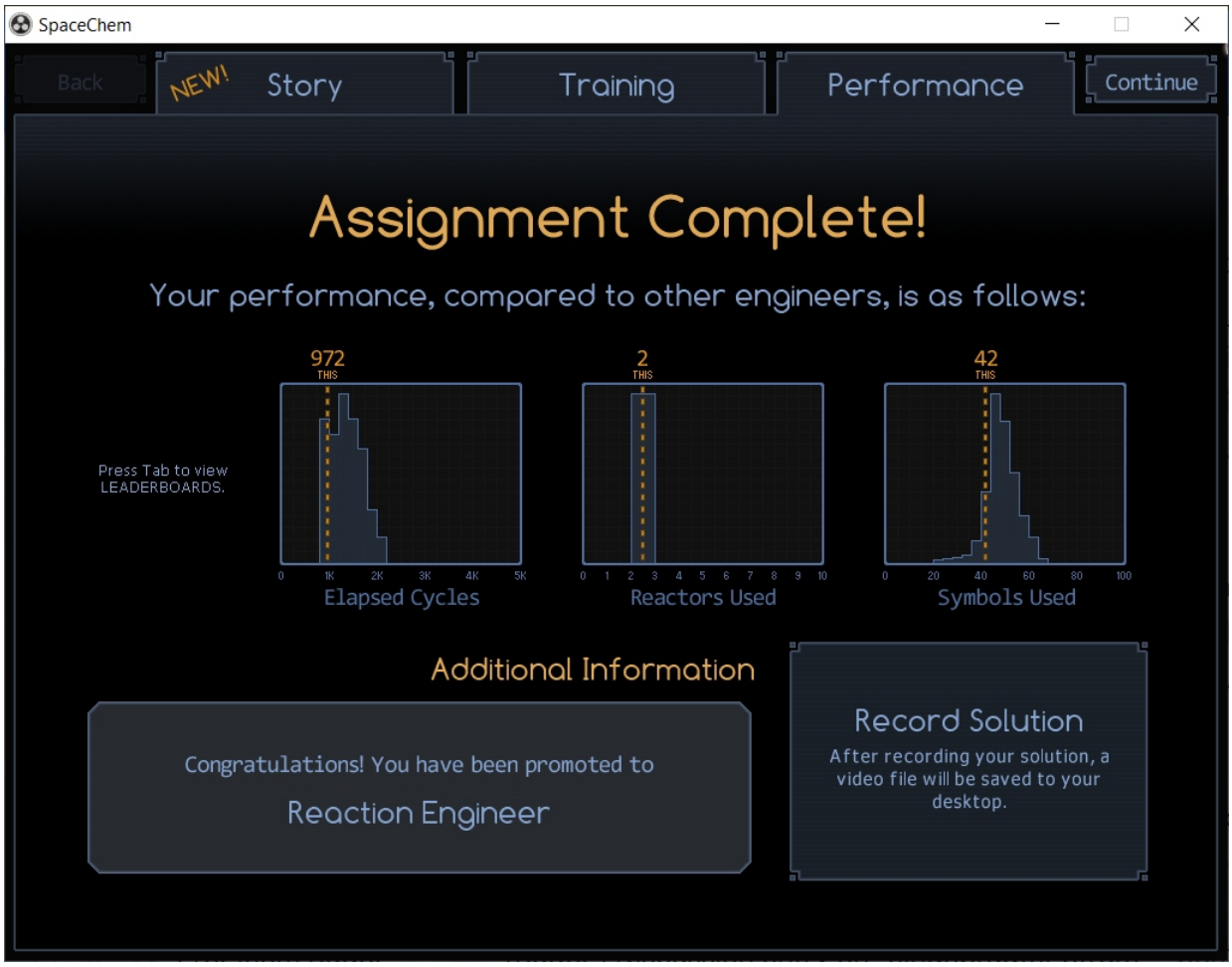
Apart from assignments where you do the actual programming work, the game contains also some background information. It is divided into three tabs: **Story**, **Training**, and **Performance**.



Background information tabs

- **Story** presents successive episodes of the storyline, which explain what is happening at the moment and the assignments at hand.
- **Training** contains a brief description of new objects, instructions or assignments.
- **Performance** presents your result of the completed assignment compared to other players. It is active only for completed assignments.

Goals of the game



Performance tab with histograms

The ultimate goal of the game is to solve all levels on all planets. But apart from that, the goal is continuous development. At the beginning, you learn to program the reactor by making simple reactions, for example bonding two atoms together. Level after level, more complex reactions, new instructions and new types of reactors are introduced and the assignments are no longer simple. You have to not only know how to use instructions to create an algorithm but also think how the required chemical compound can be synthesized. You have to consider what processes should be applied (for example, adding or removing bonds) and in what order, and additionally how to fit the algorithm within the constraints of the reactor. The space inside is limited and your initial design might be simply too extensive.

It is a challenge in itself to solve a level but you are also motivated to improve your solutions. The game provides histograms presenting the number of cycles, reactors and symbols compared to other players, or engineers. This way you are prompted to rethink and rework your solutions so that they are faster and/or simpler – in most cases only one of these parameters can be improved at the same time. Go to How to improve performance to get some useful information about improvement of performance. Sometimes a completely new design can be developed, because almost all levels can be solved in various ways. Last but not least, the game includes special challenges, for example to create a solution that uses fewer than the specified number of cycles. Sure, you have beaten the level but can you do it even better?

Programming mindset

SpaceChem applies a visual programming language, which means that you don't have to write code to create algorithms. All commands (called instructions in-game) are in the form of symbols, which are placed on the grid in a desired sequence. Thanks to that, the game is relatively easy to grasp even for a person without any programming knowledge and experience.

Bear in mind that you will not learn how to program by playing SpaceChem. However, by solving successive levels you have an opportunity to acquire a programming mindset, which might help you become a programmer in the future, if you wish so.

What you can learn

- **Every solution is based on inputs and outputs.**

  The input is what you receive from outside (for example, data entered by a website user while logging in) and the output is what your program produces based on the input and the instructions you add.

- **You can get different inputs, which require different solutions.**

  You have to process the input such that the required output is obtained. That is why if you get two different atoms in the same input area, you have to recognize them and create separate algorithms for them so that they can be processed in the required manner. When using the real life example mentioned above, your program used by the website has to recognize the first name entered by the user and know what to do with this particular piece of information so that the user can log in. If a different first name is entered, your program must log the user in a different account.

- **You might require more than one input to produce required output.**

  Sometimes you need two different atoms coming from two different input areas to produce a required molecule. In other words, you need a username and a password entered by a user in two different fields so that the user can be logged in by your program. You won't achieve your goal if you don't get either of them or if you accidentally apply an algorithm that matches a different input than actually present.

- **Tasks are complex but can be divided into sections.**

  A task of creating a molecule (e.g. logging a user in) is complex: it consists of several sections. Handle one section, test if it does what it is supposed to and then proceed to another section. Try to combine all sections such that the required output is created.

- **Tasks can be similar and can be re-used.**

  If you know how to bind an oxygen atom with a carbon atom (for example, a username with a password), you can use this template to create an algorithm for binding an hydrogen atom with a nitrogen atom (for example, a different username with a different password).

- **Programs are made of loops.**

  You get two atoms and you have to create a molecule. You recognize them, apply the sequence of instructions, produce the output molecule and move back to the beginning. Your program is ready to get new inputs and create new outputs.

- **Programs can contain branches.**

  Sometimes your inputs contain atoms that you don't need or that appear in a wrong sequence – this happens in production assignments. In this case, you can create branches and direct those wrong atoms straight to the recycler. The same can happen in our real-life example: if your program expects a particular input after the first one (a password after a username), it must be prepared for a situation where, for instance, a username is entered again instead of a password. A programmer must expect the unexpected and prepare branches to handle such incorrect inputs.

- **Downstream reactors must wait for input from upstream reactors.**

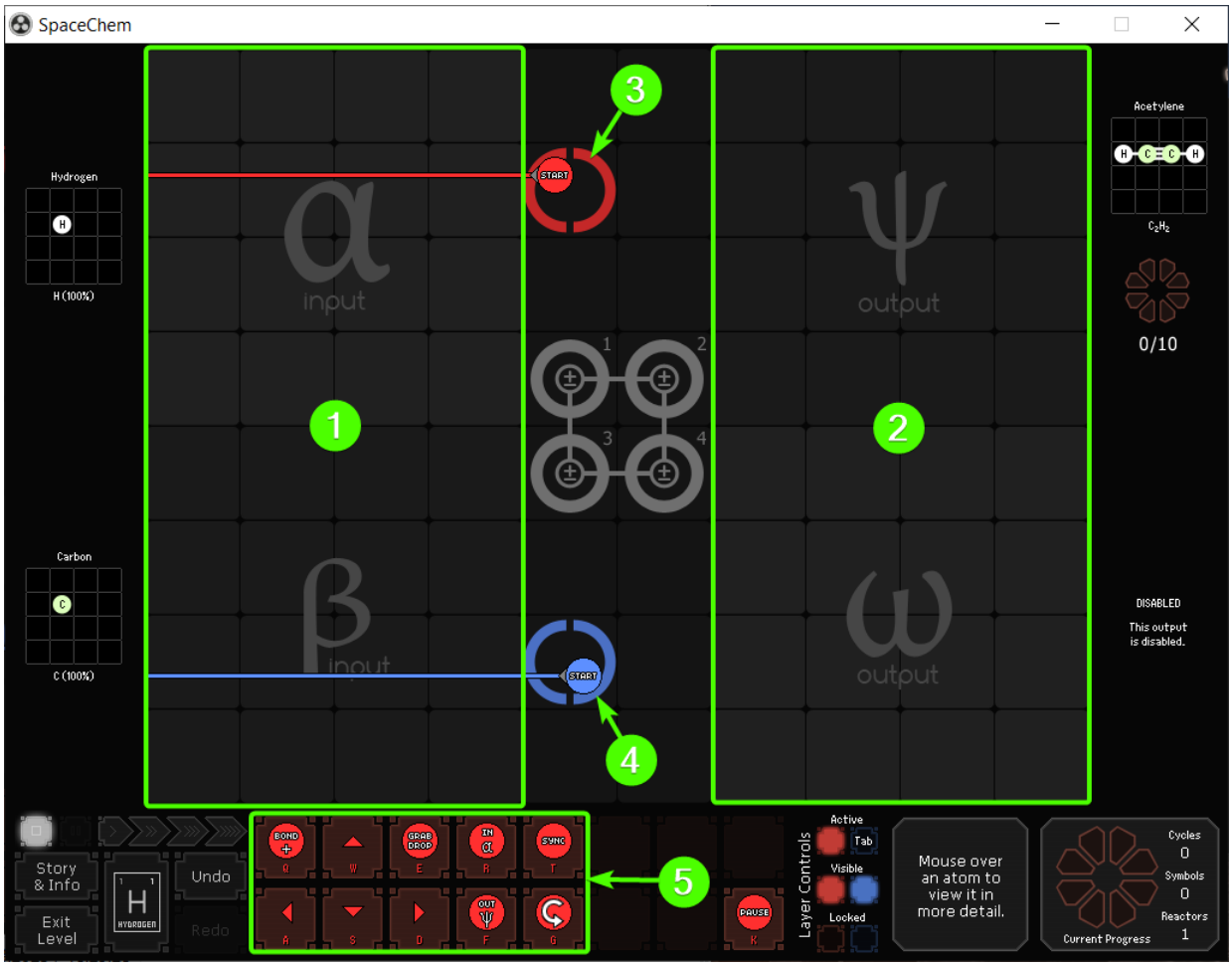
  In production assignments, the last reactor must receive inputs from downstream reactors to execute the final reaction and produce the final output. It can happen that one input is received much earlier than the other, so the reactor cannot proceed with its algorithm. This situation can be compared to logging in a bank's website with user authentication. Let's say that the user entered the username and the password but a security code is still required. The program will not log the user in until the security code is entered.

Game mechanics

This chapter describes the mechanics of SpaceChem. You will become familiar with the structure of the reactor and the special objects it contains, and learn how the basic and advanced instructions work.

- Reactor structure
- Reactor interface
- Basic instructions
- Advanced instructions
- Special objects

Reactor structure



Reactor structure with annotations

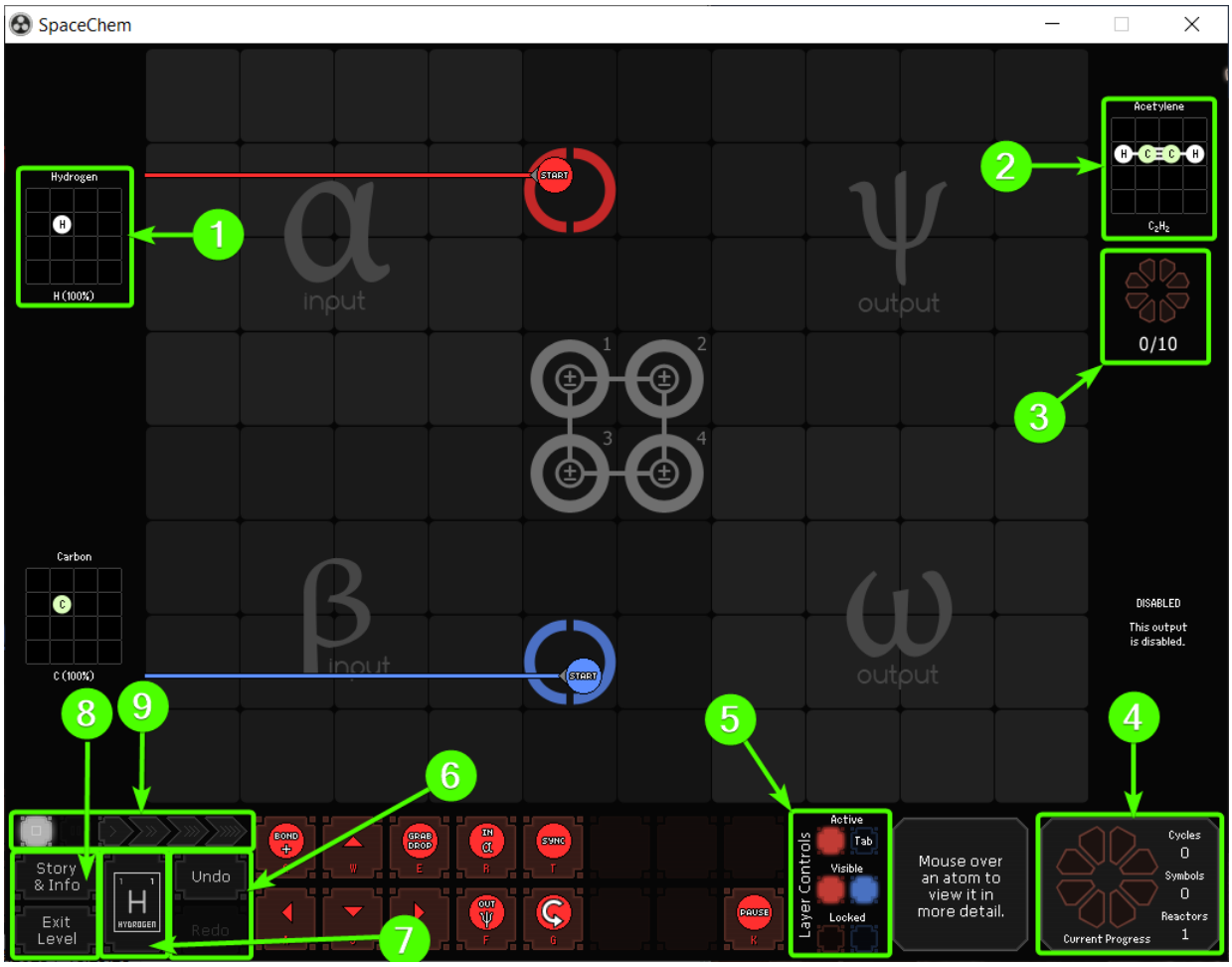
The reactor is composed of three parts: the input area to the left (1), the output area to the right (2) and the area between them. In most levels the input and output areas are sub-divided into α and β input areas and ψ and ω output areas, respectively. Atoms/molecules appear in the input areas and must be delivered to the output areas.

Your task is to carry out various reactions with the use of waldos. There is a red (3) and a blue (4) one, so they can execute two different (or the same) algorithms at the same time. The goal is to create paths for the **waldos** with the use of the available instructions so that the required chemical compound can be formed and output in the correct output area.

Below the reactor are buttons with various instructions (5) that you are supposed to use. They are described in detail in Basic instructions. Remember that Advanced instructions (which are not visible in the image above) will be introduced later in the game. The other elements of the interface are described in detail in Reactor interface.

The reactor most often contains also special objects, which are required for carrying out the reaction or executing the algorithm. These are bonders, sensors and fusion laser targets. They are described in detail in Special objects.

Reactor interface



Reactor interface with annotations

Apart from the instructions, the reactor interface contains the following components:

1. **Input note**

   This is a small representation of the input area grid, which shows what atom/molecule will be input and in which grid cell.

2. **Output note**

   In research assignments this is a small representation of the output area grid, which shows what atom/molecule must be output. It doesn't need to be output in the presented grid cells and doesn't need to have exactly the presented shape.

   In production and defense assignments you have to devise and make an output note yourself because it represents an intermediate molecule that you prepare on your own.

3. **Output counter**

   Shows how many output atoms/molecules you have successfully created out of the total required number.

4. **Progress counter**

   Shows the number of cycles, symbols and reactors used in your solution, as well as the overall progress of your solution.

5. **Layer controls**

   Allow you to change the color of your instructions from red to blue and the other way round, lock the desired color or show only the desired color.

6. **Undo/Redo**

   You can undo or redo your actions using the **Undo** or **Redo** buttons, respectively.

7. **Periodic table of elements**

   Click this button to open the periodic table of elements.

8. **Story & Info / Exit Level**

   Click **Story & Info** to view the **Story** and **Training** tabs.

   Click **Exit Level** to exit the current level.

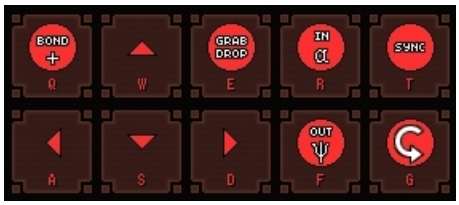
9. **Play/Pause/Stop**

   Click **Play** to start playing your solution. The more arrows, the faster your solution is played.

   Click **Pause** to pause the playing.

   Click **Stop** to stop the playing.

Basic instructions



Basic instructions

Instruction	Icon	Description
START		The first and the most important instruction of all. It marks the place where the **waldo** begins executing the algorithm.
INPUT		This instruction makes an atom/molecule appear in the input area, in the grid cell that is indicated in the input note presented to the left of the reactor grid.
OUTPUT		This instruction makes an atom/molecule disappear in the output area.
GRAB/DROP		There are three variants of the **GRAB DROP** button. The default one makes the **waldo** grab or drop an atom/molecule located exactly at the place where the instruction is located. The button can be switched to either **GRAB** or **DROP** so that the **waldo** can perform only one operation (grabbing or dropping) instead of both of them.
ROTATE MOLECULE		This instruction is responsible for rotating the molecule grabbed by the waldo 90 degrees clockwise or counter-clockwise. It does not rotate single atoms.
BOND+/BOND-		These instructions are variants of the **BOND+** instruction button. **BOND+** creates a bond between atoms that are present on adjacent **bonders**, whereas **BOND-** destroys a bond between atoms that are present on adjacent **bonders**. The instruction does not have to be placed directly on **bonders**.
ARROW INSTRUCTIONS		The arrows force the **waldos** to change the direction of their paths in the reactor.
SYNC		**SYNC** is useful for avoiding collisions and making sure that both **waldos** execute their respective algorithms in the proper sequence. You have to place one **SYNC** for the red **waldo** and one for the blue **waldo** to achieve this effect. The first **waldo** reaching its **SYNC** will stop there and wait until the other **waldo** reaches its **SYNC**. Once this happens, the first **waldo** starts running again.

Advanced instructions

Branches

Instruction	Icon	Description
CONTROL		This instruction allows you to redirect the **waldo** manually to respond to the situation outside the reactor. **BRANCH-CONTROL** is available only in defense assignments.
SENSE ATOM		This instruction allows you to redirect the **waldo** automatically when a **sensor** detects a specified element. **BRANCH-SENSE ATOM** is useful when multiple atoms/molecules are input in a single input area and need to be recognized and sorted correctly.
FLIP-FLOP		This instruction allows you to redirect the **waldo** automatically when it passes through a **FLIP-FLOP** the second time.

Other

Instruction	Icon	Description
FUSE		This instruction triggers atomic fusion on **fusion laser targets**. You don't have to place it directly on the **fusion laser target**.

Special objects

Object	Icon	Description
Bonder		There can be two, four (default) or eight **bonders** in a reactor. Their task is to create or remove a bond between atoms placed on adjacent bonders. When the **BOND** instruction is executed, it affects all **bonders** at the same time.
Sensor		**Sensors** are tasked with detecting the element atom passing over them. When the specified element passes over the **sensor**, the corresponding **BRANCH-SENSE ATOM** instruction is activated and enables a **waldo** to be redirected.
Fusion laser target		They allow atomic fusion: the atom located on the left side of the object is pushed by the laser onto the atom located on the right side of the object. The result is a completely new atom, the atomic number of which is the sum of the atomic numbers of the original atoms.

Gameplay

This chapter discusses the gameplay of SpaceChem. You will get to know the types of assignments and learn how to create an algorithm in the reactor and to combine multiple reactors or other structures into a program.

- Assignments
- Programming a reactor – part 1
- Programming a reactor – part 2
- Production assignment interface
- Defense assignment interface
- Setting up production and defense

Assignments

Assignment	Description
Research assignment	These are the basic assignments in the game. Their aim is to program a single reactor so that atoms/molecules can be input and modified in such a manner by the waldos that the required atoms/molecules can be output.
Production assignment	These assignments require several structures, such as reactors, storage tanks, cargo freighters, etc. These structures are connected by pipelines. The basic principle is that atoms/molecules are supplied to reactors via pipelines. Reactions are executed in the reactors to obtain the molecules, which are finally sent via pipelines to their respective destinations (for example, harvesters).
Defense assignment	These assignments require defending the control center from various threats, from rogue machinery to monsters. The basic principle is similar to production assignments. However, unique **BRANCH-CONTROL** instructions are available in defense assignments, which allow you to manually redirect the **waldos** to respond to the situation outside.

Programming a reactor – part 1

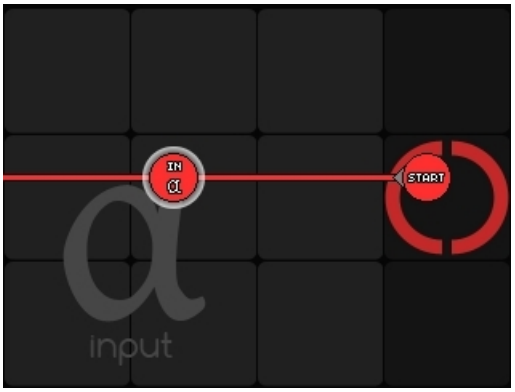
Although each level needs to be solved in a different manner in terms of planning, the physical steps you are required to take are the same in terms of procedure. Regardless of the assignment, you always have to input atoms/molecules, grab them, change the direction of waldos, drop and output the required atom/molecule. Apart from that, you may need to bond atoms, remove bonds from molecules, rotate molecules, create branches, sense atoms and fuse atoms – depending on assignment.

The first, second and fourth assignment in the game are simple tutorials, which introduce a few basic instructions and some initial knowledge about reactions. This section summarizes everything you have learned during the initial four assignments and helps you solve the fifth assignment. Create an algorithm on your own or make use of the following procedure if you can't remember what to do next at a certain point. All steps are accompanied by screenshots, which will help you with the visual arrangement of instructions.

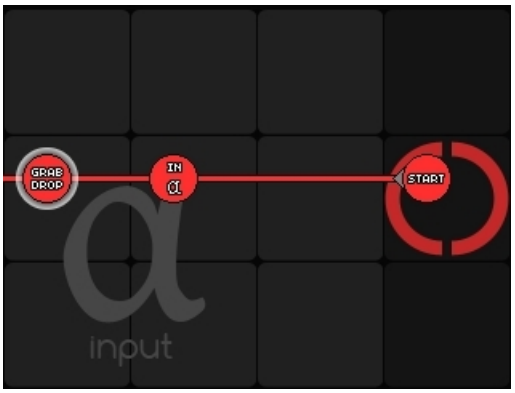
After opening any assignment you will see an empty reactor with the default starting setup: two Start instructions in the middle of the reactor grid. In this particular assignment, you will also see four bonders in the middle of the reactor grid, two input notes showing the atoms which will be input, and one output note showing the required molecule which must be output.

Inputting and grabbing atoms, redirecting waldos

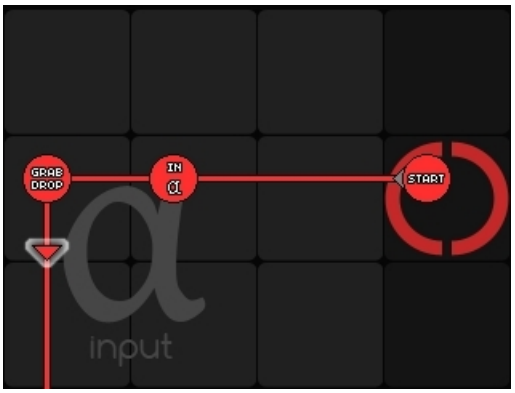
1. Click the **INPUT α** instruction, drag it to a grid cell located along the red waldo path before the place where the atom will be input, and drop it there.



2. Click the **GRAB DROP** instruction, drag it to the grid cell where the atom will be input, and drop it there.

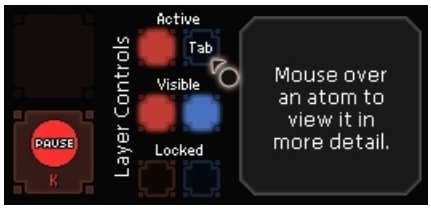


3. Click the **ARROW DOWN** instruction, drag it to the grid cell where you have placed the **GRAB DROP** instruction in the previous step, and drop it there.



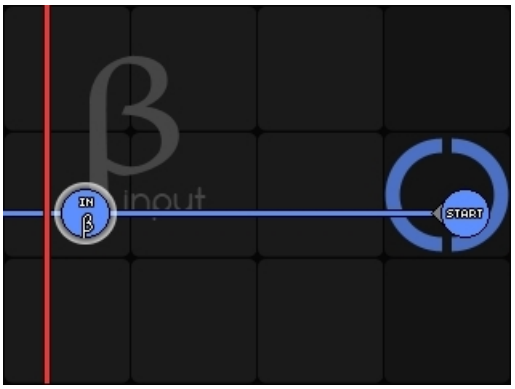
Now you have to start creating an algorithm for the other atom. To do that, you should use the blue waldo.

4. Click the **Tab** toggler in the **Layer Controls** section to the right of the available instructions.

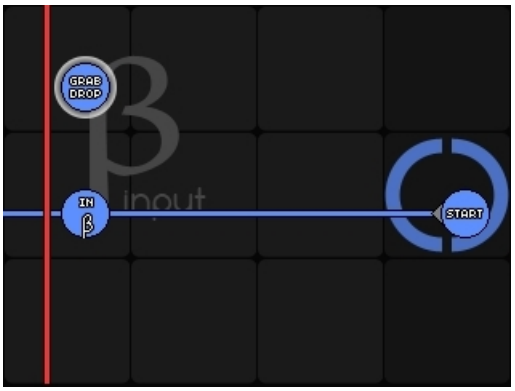


As a result, the available instructions will now be blue.

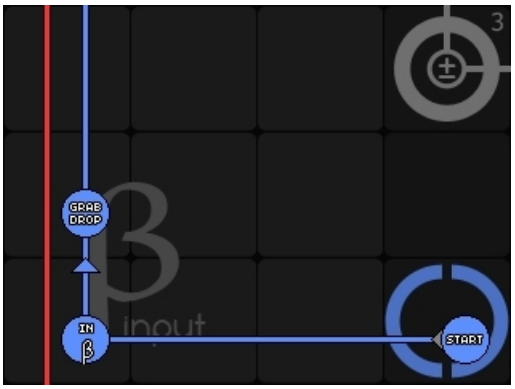
5. Click the **INPUT β** instruction, drag it to the grid cell located along the red waldo path before the place where the atom will be input, and drop it there.



6. Click the **GRAB DROP** instruction, drag it to the grid cell where the atom will be input, and drop it there.



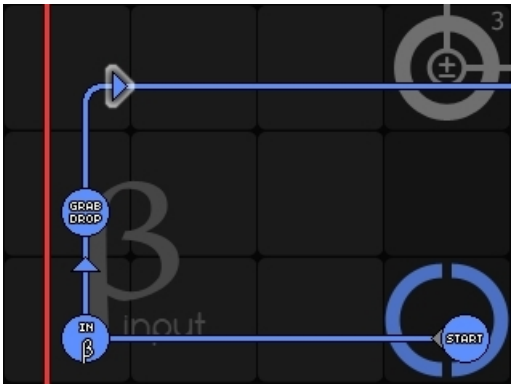
7. Click the **ARROW UP** instruction, drag it to the grid cell where you have placed the **INPUT β** instruction in the previous step, and drop it there.



Adding bonds

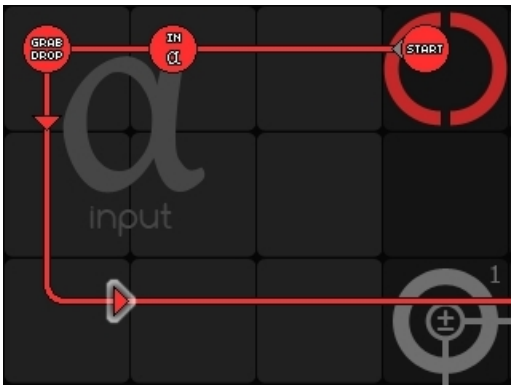
Now your waldos will input both atoms and grab them. Next you have to create a bond between both atoms. You have two options: either one atom will be placed above the other, or one atom will be placed next to the other. There is no difference between these options in terms of output - the orientation of the molecule doesn't matter. In this example, you will place one atom above the other.

8. Click the **ARROW RIGHT** instruction, drag it to the grid cell above the blue **GRAB DROP** instruction, and drop it there.

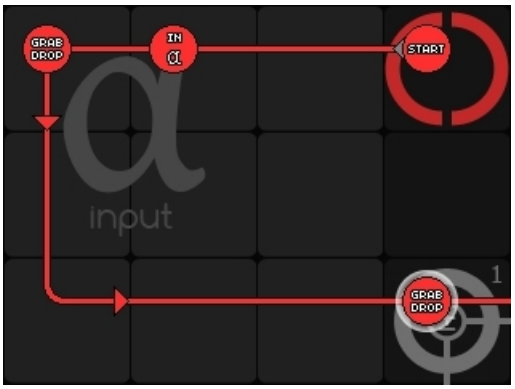


9. Click the **Tab** toggler in the **Layer Controls** section to the right of the available instructions to switch to the red layer.

10. Click the **ARROW RIGHT** instruction, drag it to the grid cell located two grid cells below the red **GRAB DROP** instruction, and drop it there.

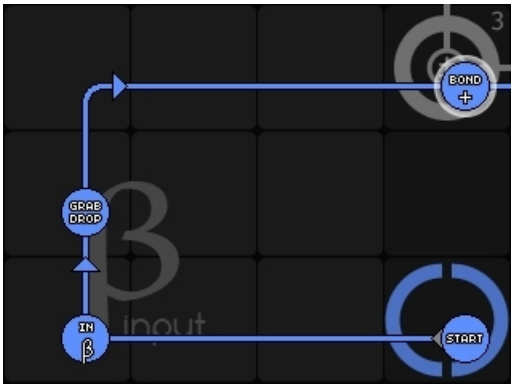


11. Click the **GRAB DROP** instruction, drag it to the grid cell with **Bonder 1**, and drop it there.



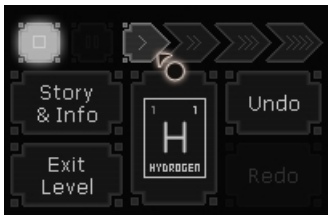
12. Click the **Tab** toggler in the **Layer Controls** section to the right of the available instructions to switch to the blue layer.

13. Click the **BOND+** instruction, drag it to the grid cell with **Bonder 3**, and drop it there.



Now you have placed enough instructions to create a bond between the atoms. To find out if the algorithm is successful:

14. Click the **Play** button and watch your partial solution.

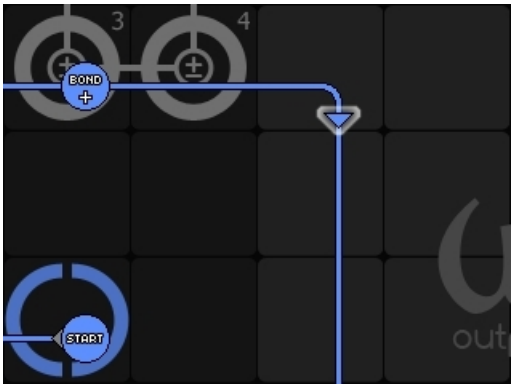


Success! Both atoms were input and grabbed by the waldos and there was no collision. The red waldo dropped the atom on the **bonder**, while the blue waldo created a bond between the atoms, thanks to which the required molecule was created. Moreover, creation of the bond prevented the blue waldo from dropping the molecule. Now you can click the **Stop** button to stop your solution.

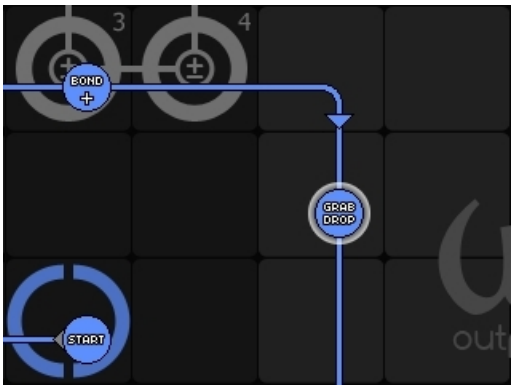
Outputting molecules

The blue waldo still holds the molecule and now it has to output it in the correct output area. The red waldo, in turn, has to return to the beginning of the algorithm.

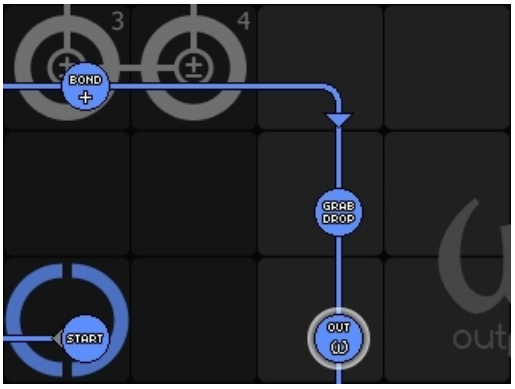
15. Click the **ARROW DOWN** instruction, drag it to the first grid cell in the **Output ω** area along the blue waldo path, and drop it there.



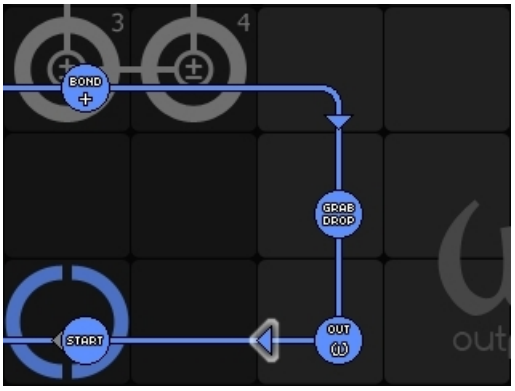
16. Click the **GRAB DROP** instruction, drag it to the grid cell below the **ARROW DOWN** instruction you have placed in the previous step, and drop it there.



17. Click the **OUTPUT ω** instruction, drag it to the grid cell below the **GRAB DROP** instruction you have placed in the previous step, and drop it there.

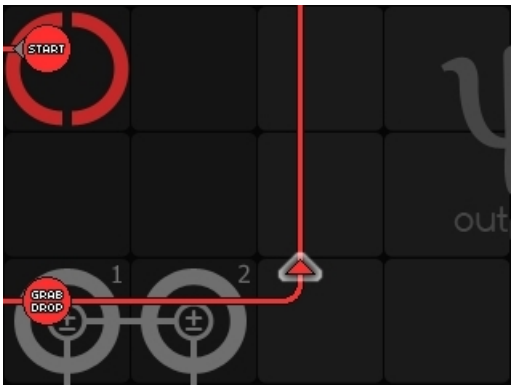


18. Click the **ARROW LEFT** instruction, drag it to the grid cell where you have placed the **OUTPUT ω** instruction in the previous step, and drop it there.

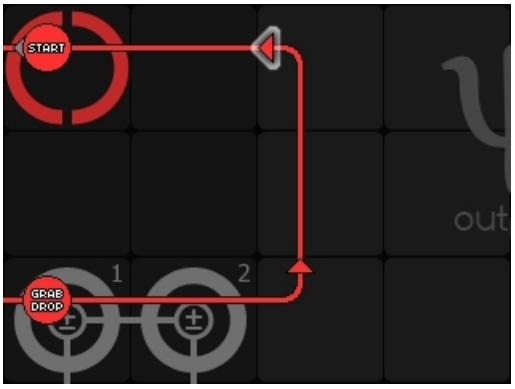


19. Click the **Tab** toggler in the **Layer Controls** section to the right of the available instructions to switch to the red layer.

20. Click the **ARROW UP** instruction, drag it to the first grid cell in the **Output ψ** area along the red waldo path, and drop it there.

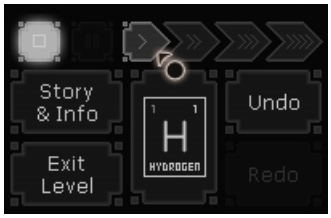


21. Click the **ARROW LEFT** instruction, drag it to the grid cell located two cells above the **ARROW UP** instruction you have placed in the previous step, and drop it there.



Theoretically, your solution is complete. Feel free to check if that actually is the case. To do this:

22. Click the **Play** button and watch your solution.



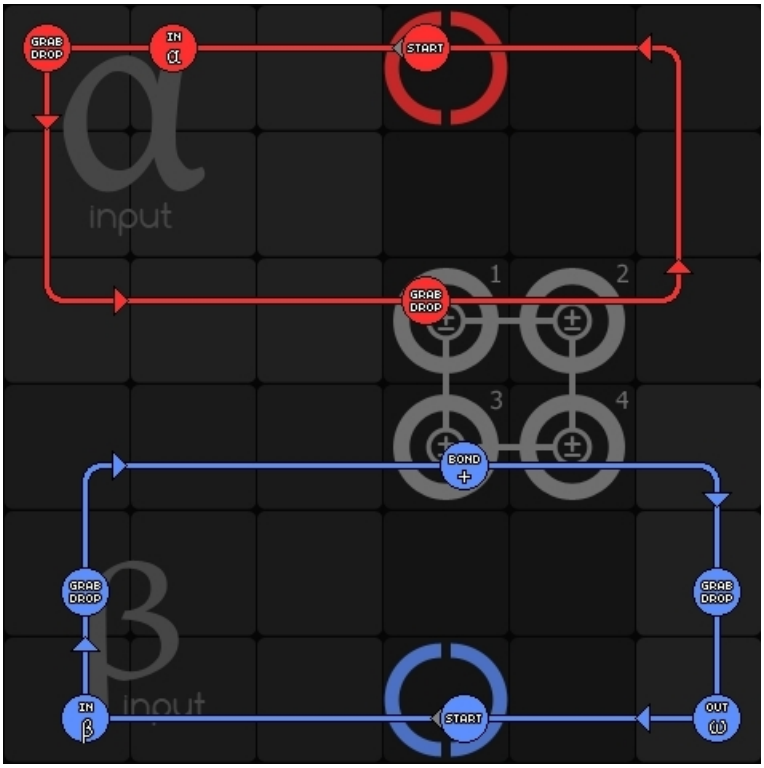
Success! The molecule was dropped in the correct output area and output, and both waldos returned to the beginning of the algorithm.

Programming a reactor – part 2

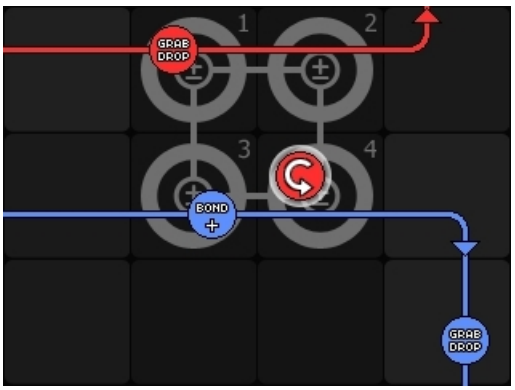
In the previous section you have successfully input atoms, created a bond between them, and output the molecule. However, that particular solution does not apply all the available basic instructions. You can learn how to use other basic instructions and to do some more basic actions by modifying the solution you have created in Part 1. At the same time you will learn how to make your solution faster.

Rotating molecules, modifying instructions

1. Open your solution, which should look like this:

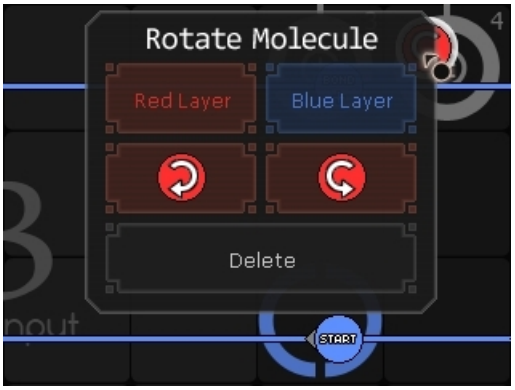


2. Click the **ROTATE** instruction located under the reactor, drag it to the closest grid cell on the right side of the blue **BOND+** instruction.

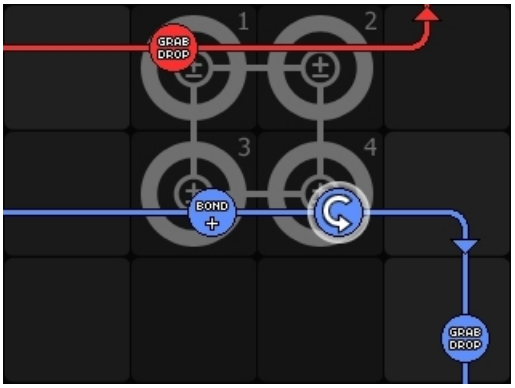


Notice that you added a red instruction to a blue path and the instruction lies outside the path. The reason is that the blue waldo is unable to execute red instructions and vice versa. However, this situation can be fixed.

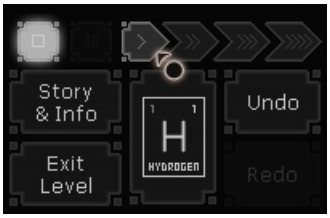
3. Right-click the **ROTATE** instruction you have just placed.



4. Click the **Blue Layer** button.

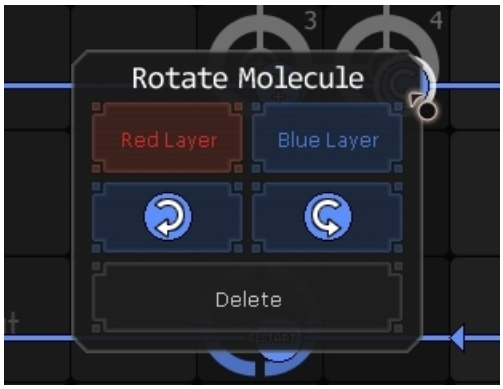


Now you can see that the color of the instruction switched to blue. The blue waldo can execute this instruction. You should check if something changed. To do this, click the **Play** button.

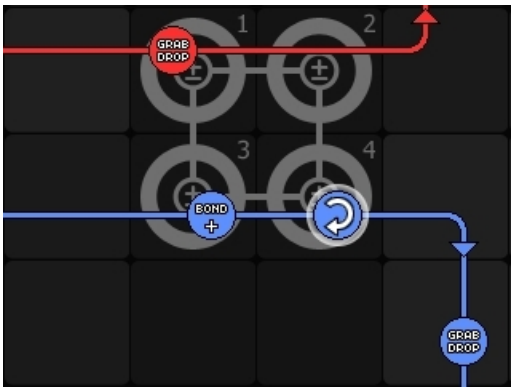


By default the **ROTATE** instruction rotates your molecule counter-clockwise. This is why the molecule did not fit correctly in the output area and was not output, which led to a collision. Click **Okay** so that you can proceed to fix this situation:

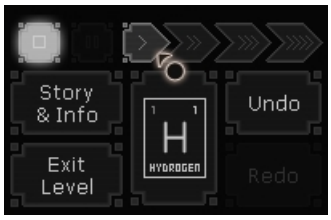
5. Right-click the **ROTATE** instruction you have just placed.



6. Click the symbol with an arrow rotating clockwise.



7. Click the **Play** button.

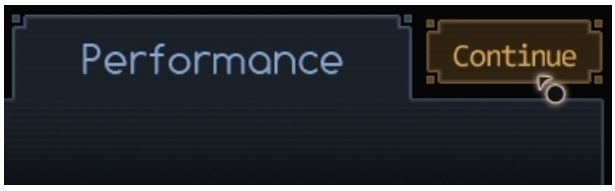


The molecule fits the output area and the solution is correct. However, as you can see in the **Performance** tab, your solution uses 148 cycles now. This is the effect of the **ROTATE** instruction, which is responsible for one additional cycle per loop. Fortunately, this situation can be fixed as well.

Making your solution faster

The simplest way to make your solution faster is to reduce the length of the path along which your waldos run. The shorter their paths, the faster they can return to the beginning of the algorithm. Let's find out if this rule can be applied to your solution.

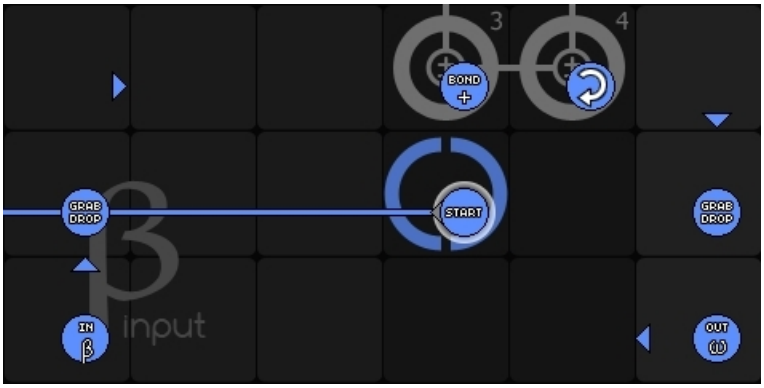
8. Click **Continue**...



... and then click the SpaceChem symbol shown in the image below to reopen the assignment.

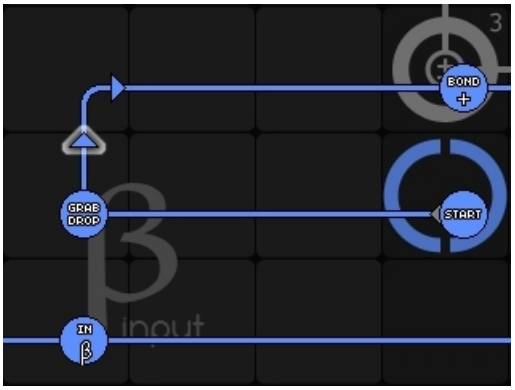
It looks like both the red and blue loop can be reduced. Let's start with the blue loop. The atom will be input in the third row counted from the bottom of the reactor, while the **START** instruction is placed in the second row counted from the bottom of the rector. You should change it.

9. Click the blue **START** instruction, drag it to the grid cell located directly above, and drop it there.

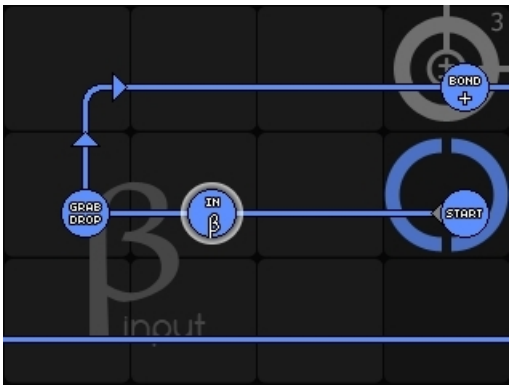


The loop fell apart and now you have to relocate other instructions to recreate it.

10. Click the blue **ARROW UP** instruction located in the reactor, drag it to the grid cell with the blue **GRAB DROP** instruction, and drop it there.

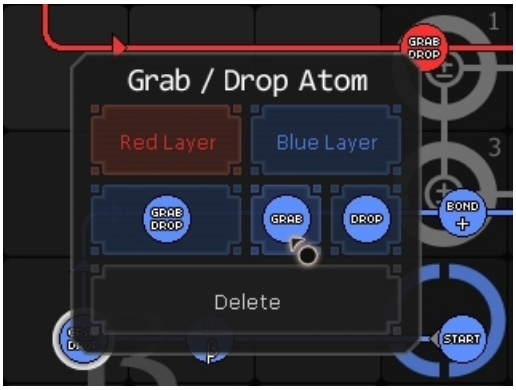


11. Click the blue **INPUT β** instruction located in the reactor, drag it to the grid cell located immediately to the right of the **GRAB DROP** instruction, and drop it there.



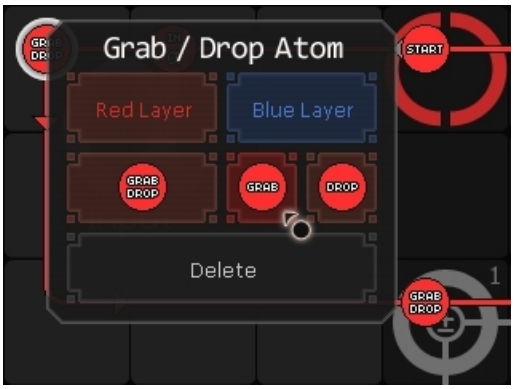
Notice that you have two blue and two red **GRAB DROP** instructions. To avoid confusion, they should be modified:

12. Right-click the left blue **GRAB DROP** instruction and click the **GRAB** symbol.

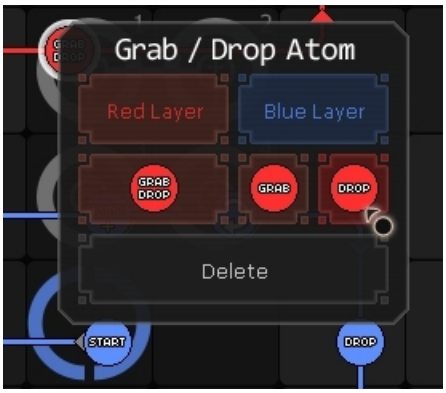


13. Right-click the right blue **GRAB DROP** instruction and click the **DROP** symbol.

14. Right-click the left red **GRAB DROP** instruction and click the **GRAB** symbol.

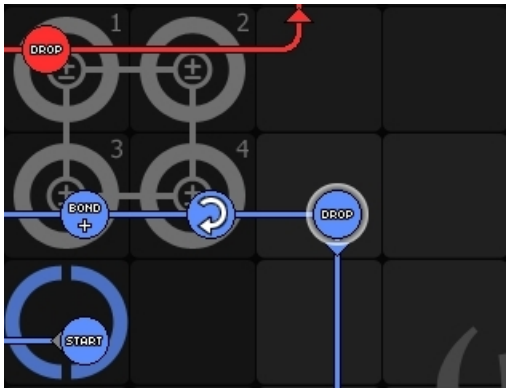


15. Right-click the right red **GRAB DROP** instruction and click the **DROP** symbol.

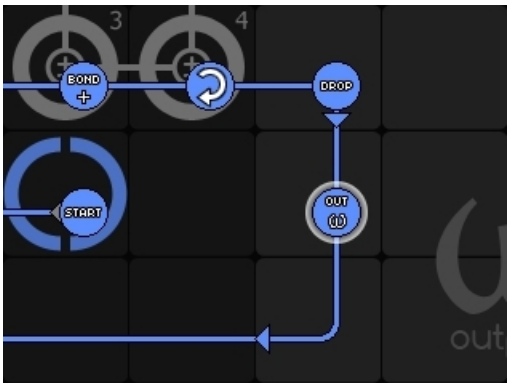


Now the instructions are much more legible. And because you placed the **ROTATE** instruction, you can drop the molecule earlier.

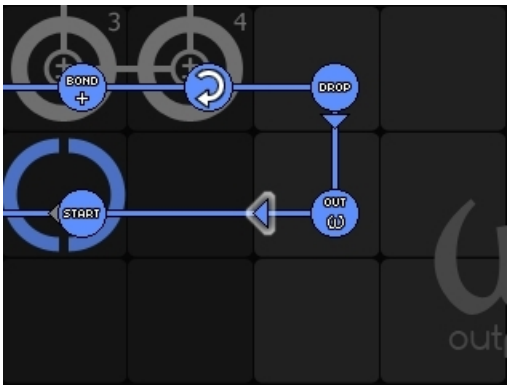
16. Click the blue **DROP** instruction located in the reactor, drag it to the grid cell located directly above, and drop it there.



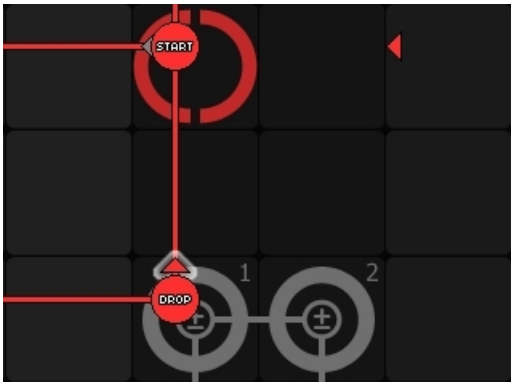
17. Click the blue **OUTPUT ω** instruction located in the reactor, drag it to the grid cell located directly above, and drop it there.



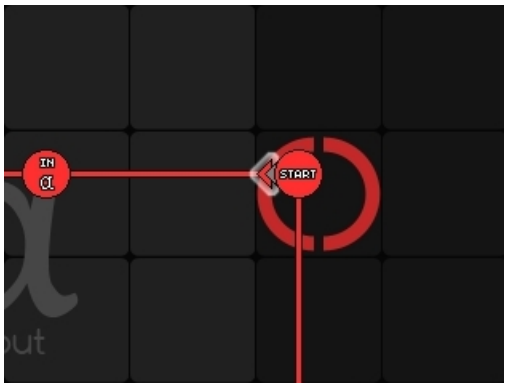
18. Click the blue **ARROW LEFT** instruction located in the reactor, drag it to the grid cell located directly above, and drop it there.



19. Click the red **ARROW UP** instruction located in the reactor, drag it to the grid cell with the red **DROP** instruction, and drop it there.

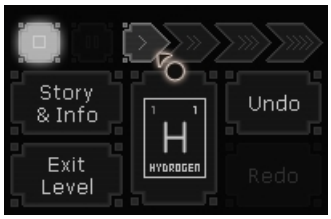


20. Click the red **ARROW LEFT** instruction located in the reactor, drag it to the grid cell with the red **START** instruction, and drop it there.



The loops are shorter so now you should check your solution.

21. Click the **Play** button.

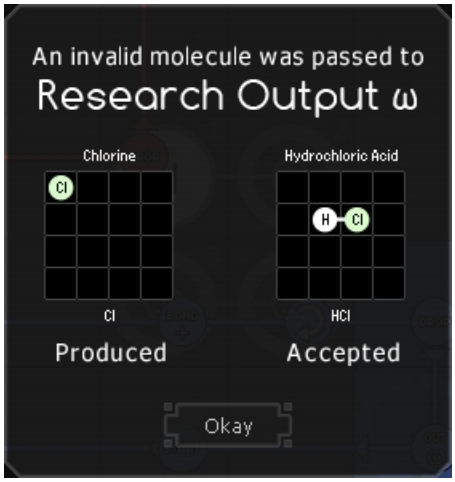


Unfortunately, the reaction could not be completed and there is an error. After shortening the loops, both waldos are no longer synchronized – the blue waldo reached the **bonder** faster than the red waldo and the bond was not added. This situation can be fixed as well.

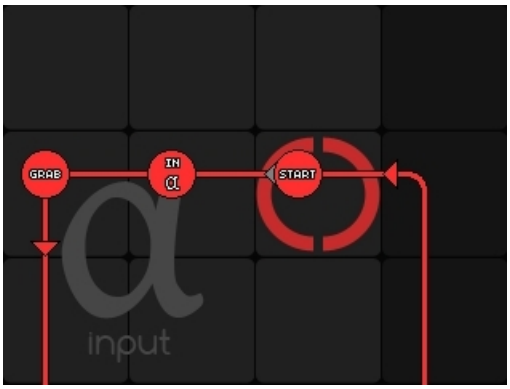
Synchronizing the waldos

For a bond to be formed, both atoms must be present on adjacent **bonders** at the same time. You can try to synchronize the waldos in two ways. In this solution, the red waldo has to run along 8 grid cells to reach its **bonder**, while the blue waldo has to run only 7 grid cells to reach its **bonder**. So you should try moving the red waldo one cell closer to the **bonder**.

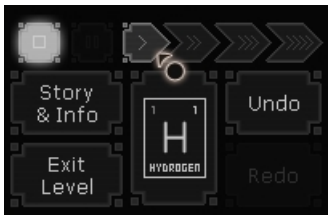
22. Click **Okay** to return to the reactor view.



23. Click the red **START** instruction, drag it to the grid cell immediately to the left, and drop it there.



24. Click the **Play** button.

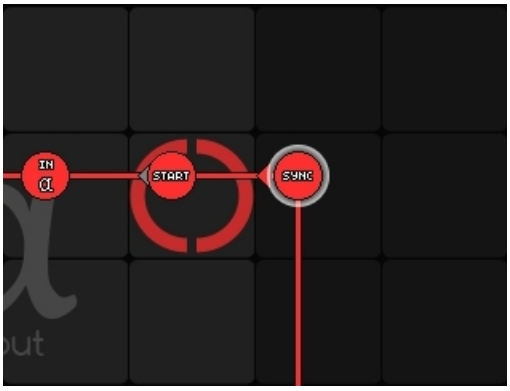


Relocating the red **START** instruction synchronized both waldos at the beginning, thanks to which a molecule was created and output. However, because the red loop is much shorter than the blue loop, the red waldo input and dropped its atom more often than the blue waldo was able to form a bond and output the molecule. The result was a collision.

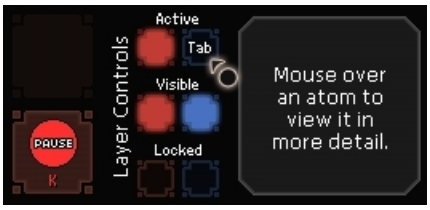
You will fix this situation by adding **SYNC** instructions.

25. Click **Okay** to return to the reactor view.

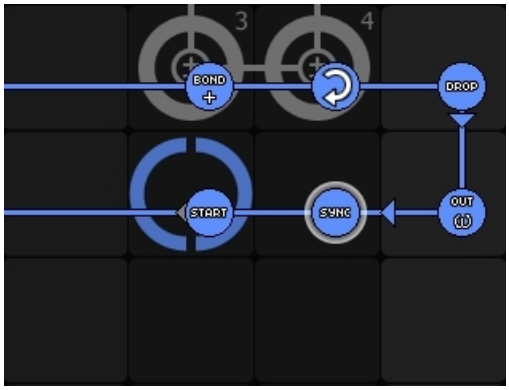
26. Click the **SYNC** instruction located under the reactor, drag it to the grid cell immediately to the right of the red **START** instruction.



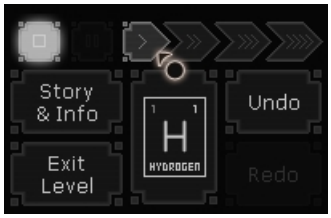
27. Click the **Tab** toggler in the **Layer Controls** section to the right of the available instructions to switch to the blue layer.



28. Click the **SYNC** instruction located under the reactor, drag it to the grid cell immediately to the right of the blue **START** instruction.

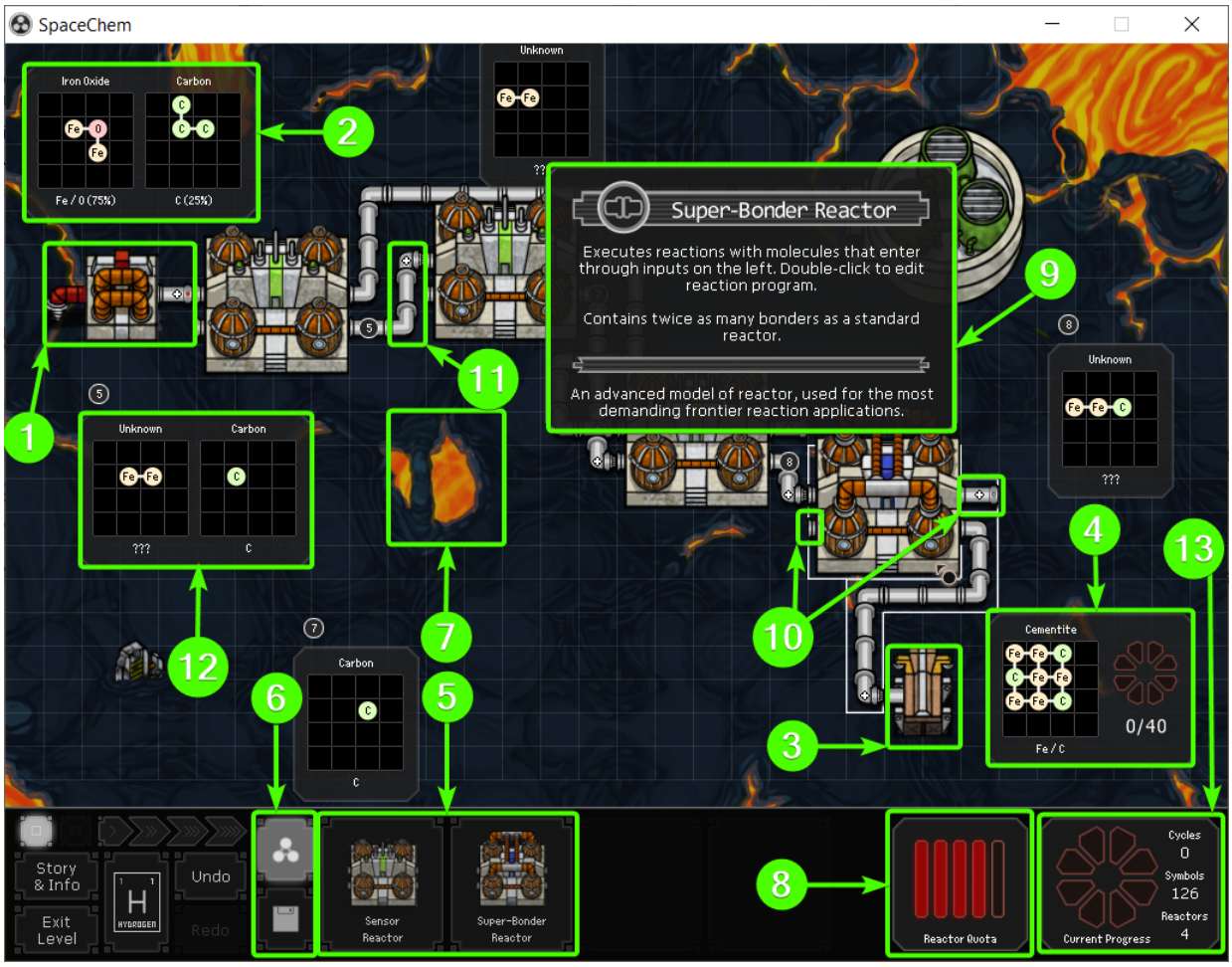


29. Click the **Play** button.



Success! There were no collisions and your solution used 128 cycles.

Production assignment interface



Production assignment interface with annotations

1. **Input structure**

   It produces input atoms/molecules which you must use to obtain the required molecule.

2. **Input note**

   This is a small representation of the input area grid, which shows what atom/molecule will be input and in which grid cell.

3. **Output structure**

   It accepts the required molecules.

4. **Required molecule**

   This is your target molecule in a production assignment. It is accompanied by a molecule counter, which shows the progress of the assignment.

5. **Available structures**

   This section includes structures which you can place in a production assignment.

6. **Reactor toggle**

   You can choose between empty reactors and any pre-programmed reactors you saved during previous assignments.

7. **Obstacle**

   No structures or pipelines can be placed on obstacles. You can recognize them by the absence of grid cells.

8. **Reactor quota**

   Shows the number of structures you are allowed to place in a production assignment. Filled bars show the number of structures you are using now.

9. **Info box**

   Mouse over a structure to see an info box containing additional information and tips.

10. **Stub pipes**

    The one on the left side is for the input, while the one on the right side is for the output.

11. **Pipeline**

    Used for connecting input structures with reactors and output structures.

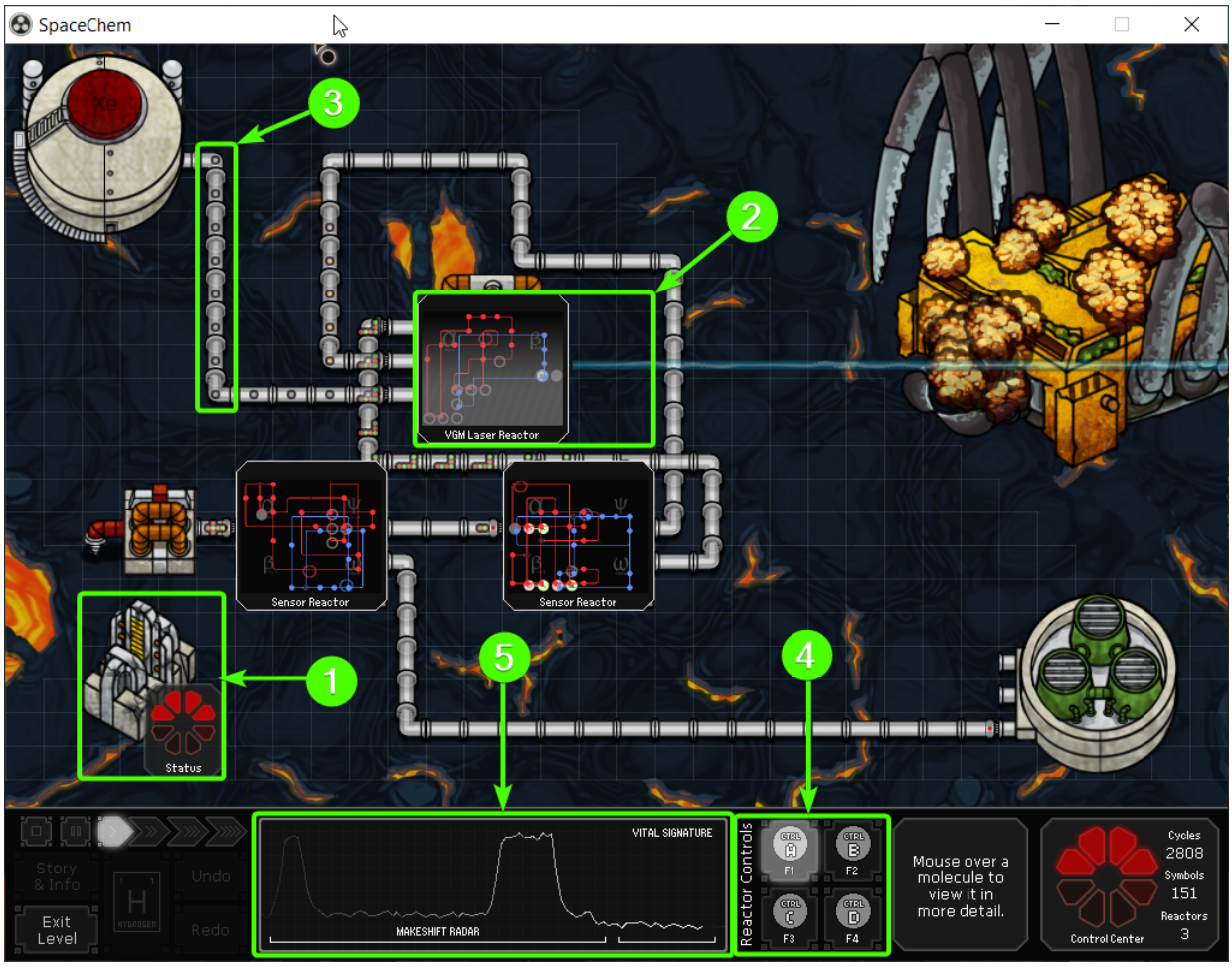
12. **Output note**

    Helps you remember what intermediate molecules are output at which reactor. You should create it yourself.

13. **Progress counter**

    Shows the number of cycles, symbols and reactors used in your solution, as well as the overall progress of your solution.

Defense assignment interface



Defense assignment interface with annotations

The interface of defense assignments is similar to the interface of production assignments, which can be viewed in Production assignment interface. However, there are some differences, which are highlighted in the above image. It presents a running defense program.

1. **Control center**

   The most important structure in a defense assignment. It must be defended at all costs. Contains a status meter, which shows the integrity of the control center.

2. **Output structure**

   In defense assignments it serves as a makeshift weapon. There is always a different one in each defense assignment: methane storage tank, particle accelerator, rocket launch pad, variable gain medium laser reactor, hydrogen laser, chemical laser.

3. **Pipeline with atoms**

   When a defense or production assignment is in progress, you can see atoms/molecules inside pipelines. This is very useful for detecting possible jams, which can lead to collisions.

4. **Reactor controls**

   Click them to activate **BRANCH-CONTROL** instructions you placed in your reactors without actually doing that inside the reactor .

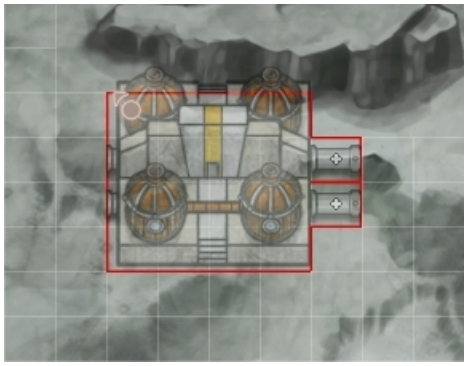
5. **EMF detector**

   Measures electromagnetic signals outside the control center. Helps predict what is going to happen next during the defense assignment so that you are ready and can respond at the right time.

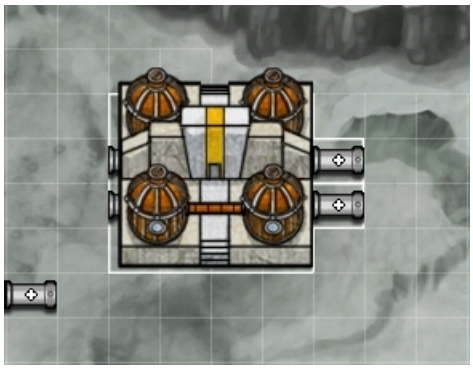
Setting up production and defense

Production and defense assignments are similar in terms of steps you have to execute to create a system of necessary structures. The steps and images presented below refer to the assignment *There's Something in the Fishcake* on the second planet.

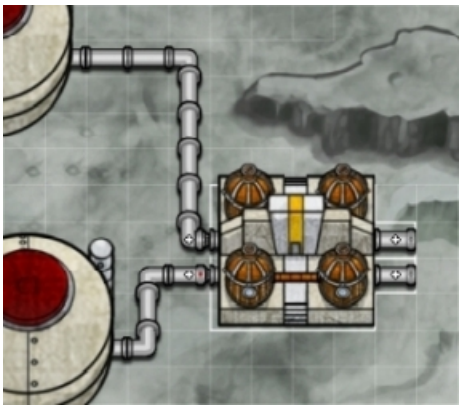
1. Click the **Standard Reactor**, drag it to the desired place on the map, and drop it there. Note that there are obstacles on the map, where you are not allowed to place any structures. When you are trying to drop a structure on an obstacle, the structure will receive a red frame.



The desired result should look like this:

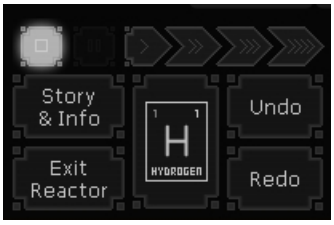


2. Click the stub pipe protruding from one **Storage Tank**, drag it to the stub pipe on the left side of the **Reactor**, and drop it there. Repeat this step to connect the other **Storage Tank** with the **Reactor**.

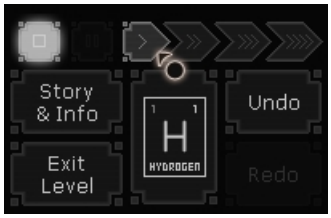


3. Click one of the stub pipes on the right side of the **Reactor**, drag it to the stub pipe on the left side of the **Cargo Freighter**, and drop it there.

4. Double-click the **Reactor** and program it to create the required molecule. When you are ready, click **Exit Reactor**.



5. Click the **Play** button to check if your solution is correct.



Help

This section is supposed to help you solve common problems encountered while playing SpaceChem and give you some tips so that you become more adept at the game.

- Rules and tips
- Troubleshooting
- How to improve performance
- Bibliography

Rules and tips

As a visual programming puzzle game, SpaceChem is based on simple mathematics. It is helpful to remember basic mathematical rules as well as other rules, which will help you create your solutions more efficiently.

Mathematics

- There are always two **waldos** and two input areas. There can be either one or two output areas and either two, four or eight **bonders**.
- The reactor is composed of a grid divided into cells. The size of the reactor is 10 cells wide and 8 cells high.
- You can place exactly one instruction that triggers an operation and exactly one arrow instruction in a single grid cell.
- You can count the number of grid cells between the input atom/molecule and the beginning of the output area.
- In assignments without branches, the waldo can enter a grid cell from exactly two directions.
- Each element has a specified exact number of possible bonds.
- One **BOND** instruction creates/removes exactly one bond between all adjacent **bonders** at the same time.

General

- You can move the **START** instruction wherever you want. You can change the direction of the **START** instruction. You don't have to end your loop on the **START** instruction.
- You don't have to use both waldos – sometimes one is able to execute the algorithm.
- Workload can be divided in different ways between both waldos. For example, one waldo can be responsible only for inputting and the other only for outputting. One waldo can be responsible for inputting and outputting one atom/molecule and the other for inputting and outputting the other atom/molecule.
- It is a good idea to place **bonders** on the input.
- The actual shape of the molecule to be output is not important: only the bonds between particular atoms matter.
- **Bonders** have numbers, which give them priority. This means that, for instance, bonds will be created first between **bonder 1** and **bonder 2** than between **bonder 1** and **bonder 3** if all of them are adjacent to one another. This is very important because sometimes you will want to create a bond on a specific side of an atom rather than on a different side.

Sensing

- You don't have to place **sensors** exactly where atoms/molecules will be input.
- When planning your sensing algorithm, first think of a "default" loop and only then think where sensing branches can be started and where they should end.

Synchronizing

- You can synchronize both **waldos** either by adjusting the length of their loops or by placing **SYNC** instructions.
- The most popular location of **SYNC** instructions is one grid cell before the **START** instruction.

Production assignments

- The molecule output in a reactor will appear in the next reactor in the exact position on the grid in which it was output. Clever planning of the position of output molecules can make it easier to create new molecules in the downstream reactors.
- Molecules need some cycles to reach the reactor after leaving the previous structure. Sometimes you have to adjust the length of the pipelines to avoid jams.
- Plan in advance the intermediate molecules you want to create. This will help you decide how many reactors and in what configuration should be used.

Defense assignments

- Although they are designed to apply **BRANCH-CONTROL** instructions, sometimes it is possible to solve such an assignment without these instructions.
- Often you don't know at the beginning of the assignment what to expect later. Don't worry. Simply create your programs to obtain the required molecules and switch to the map view to find out what and when is happening.

Troubleshooting

This section presents the most commonly occurring problems with gameplay, the most probable causes and suggested solutions.

Bonding

Problem	Possible cause	Solution
Bonds are still present	Too few **BOND-** instructions	Add more **BOND-** instructions
	Bonders do not support bond removal	Replace reactor with disassembly reactor
Bonds are not added	**Bonders** do not support bond addition	Replace reactor with assembly reactor
	No more bonds can be added to this particular atom	Remove some bonds earlier or get a different atom
Bonds are added not between desired atoms when all bonders are placed adjacent to one another	**Bonder** priority was triggered	Check **bonder** numbers and switch their places to achieve required bonding

Outputting

Problem	Possible cause	Solution
Molecule is not output	Molecule was not dropped	Drop molecule
	Molecule was not dropped entirely in output area	Drop molecule entirely in output area
	Molecule was dropped in inactive output area	Drop molecule in active output area
	Wrong instruction was placed (for example, **OUT Ψ** instead of **OUT Ω**)	Place correct instruction

Collisions

Problem	Possible cause	Solution
Atoms collide inside reactor	**Waldos** are on collision course	Re-route **waldos**
		If re-routing is impossible, add **SYNC** instructions to change timing of each **waldo**
	Atoms/molecules were not dropped	Drop atoms/molecules at required places
	Jam in downstream reactor	Remove jam by extending relevant pipeline

Sensing

Problem	Possible cause	Solution
Waldo is not redirected by the Branch-Sense Atom instruction	Instruction senses incorrect element	Change element in instruction
	Sensor placed in wrong grid cell	Relocate **sensor**
	BRANCH-SENSE ATOM instruction placed too early/late along **waldo** path	Place instruction in correct place along **waldo** path

Fusion

Problem	Possible cause	Solution
Atom is not fused at the Fusion Laser Target	No atom present on right part of **fusion laser target**	Drop required atom on right part of **fusion laser target**
	No **FUSE** instruction along **waldo** path	Place **FUSE** instruction along **waldo** path
	FUSE instruction placed too early/late along **waldo** path	Place instruction in correct place along **waldo** path

How to improve performance

Solving a level is only the first step. You will realize quickly that your solutions can be improved, or even *must* be improved so that you can beat some more difficult levels later on at all.

Reducing the number of cycles

The number of cycles is the measure of how quick your solution is. It can be reduced in a few ways. These tips don't apply to all specific cases, but it's worthwhile to check if they work:

- Algorithms are based on loops. Try to reduce their size so that the **waldo** can complete them along a shorter path.
- Try to create loops for both **waldos** such that they are completed in the same number of cycles. If one **waldo** does its part faster than the other, try to share the workload between them more equally.
- Try to route your loops such that the **waldo** changes its direction as rarely as possible. At the same time you are reducing the number of symbols.
- If an assignment can be completed with the use of only one **waldo**, create the same algorithm with the other **waldo**, while remembering that both **waldos** must not collide with each other.
- Place the **START** instruction as close to the grid cell where the atom/molecule will be input as possible.
- In production assignments, try to place successive reactors as close to each other as possible, while remembering to avoid traffic jams in pipelines.
- In production assignments, try to create as simple and short reactions in successive reactors as possible.

Reducing the number of symbols

The number of symbols is the measure of how efficient your solution is. It can be reduced in a few ways. These tips don't apply to all specific cases, but it's worthwhile to check if they work:

- Try to route your loops such that the **waldo** changes its direction as rarely as possible. At the same time you are reducing the number of cycles.
- If multiple identical instructions are necessary in an algorithm, try to place only one instruction and route your **waldo** such that it moves through this instruction more than once.
- Drop more than one atom/molecule in the output area such that they don't collide with each other and output them at once with a single instruction.
- Drop all atoms necessary for creating a molecule on all available **bonders** and bond them all at once with a single instruction.

Bibliography

This section contains links to sources used during the preparation of SpaceChem Beginner Guide as well as to other reference materials, thanks to which you can expand your knowledge.

- SpaceChem page on the game developer's website
- SpaceChem page on Steam
- SpaceChem review by a computer scientist
- More on visual programming
- More on waldos
- More on Turing-complete machines and languages

Glossary

Here you can find definitions of the key terms used in the SpaceChem Beginner Guide

Assignment

A specialised term used in the game for a game level. There are research, production, and defense assigments.

Bonder

A special object inside the reactor. Used for adding or removing bonds between atoms.

Cycle

A unit of measurement of execution of an assignment. A waldo requires at least one cycle to move to the next grid cell located along its path. The lower the number of cycles used in an assignment, the faster your solution is.

Fusion Laser Target

A special object inside the reactor. Used for performing atomic fusion of two atoms to obtain a completely new one.

Grid cell

The smallest section of the grid inside the reactor. Has the shape of a square.

Instruction

A building block for algorithms in reactors. Must be placed inside the reactor along the path of a waldo so that it can be executed.

Object

A component of a reactor, which is required for executing particular operations with the use of a corresponding instruction. It can be a bonder, a sensor or a fusion laser target.

Reactor

The primary workspace in SpaceChem. This is where you use instructions to transform input atoms/molecules into output atoms/molecules. It is divided into input area, output area and the area between them.

Sensor

A special object inside the reactor. Used for detecting which atom is present above it at a given moment.

Structure

A component of production and defense assignments. It can be placed on the map and connected with another one via a pipeline. It can be an input structure, a reactor or an output structure.

Symbol

A unit of measurement of execution of an assignment. Every instruction placed in the reactor counts as a symbol. The number of symbols used in an assignment does not reveal if your solution is fast or not.

Waldo

In other words, a remote manipulator. Used inside the reactor for executing the instructions placed along a path. There are two of them: a red and a blue one.