Simulation Minus One Makes a Game

Noriyuki Amari and Kazuto Tominaga

School of Computer Science, Tokyo University of Technology
1401-1 Katakura, Hachioji, Tokyo 192-0982 Japan
ronsum@mozart.tomilab.net,tomi@acm.org
http://www.tomilab.net/

Abstract. This paper presents a way to develop a game using an artificial chemistry. An artificial chemistry is an abstract model of chemical system. It is used in the research field of artificial life. We develop a roguelike game using an artificial chemistry with a specific approach, which we propose in this paper: first, we build a system to simulate the world of a roguelike game; then we remove a part of the system to make it a game. A small set of rules in the artificial chemistry is able to define the simulation, and removing a rule makes it a game. This shows the effectiveness of the present approach in developing a certain type of game using the artificial chemistry.

Key words: artificial life, artificial chemistry, roguelike game, simulation

1 Introduction

Artificial life (ALife) is a research area where the simulation and synthesis of living systems are studied. Recently it is getting close relation to games. For example, Spore [1] is a game to create a virtual space ecosystem, and Panic Shooter [2] is a shooting game implemented on cellular automata. Since a main focus of ALife research is to give rise to life-like behaviour in computer simulation, it is suitable to produce unexpected but natural events in games [3], which have been conventionally programmed algorithmically in game software.

Artificial chemistries (AChems) are one of the research methodologies used in ALife studies [4]. An AChem is a formal model that describes an abstract chemical system. We proposed an artificial chemistry that represents molecules with character strings and that deals with compartments separated by membranes [5]. A chemical reaction in this AChem is a recombination of strings defined as a rule similar to a chemical equation. The system of rules is executed on simulation software (a simulator) of the AChem, and its behaviour is observed.

We develop a roguelike game using the AChem. A roguelike is a genre of game; the player's goal in a roguelike game is to reach the deepest part of a dungeon where monsters prowl. The player character's encounter with a monster is regarded as the collision of molecules. In this paper, we present a method to develop a game, in which we take the following two steps. First, we build a

Fig. 1. An example v-molecule.

system to simulate the world of the roguelike game. Then, we remove a part of it to make it a game.

The organisation of the paper is as follows. Section 2 briefly explains the AChem. Section 3 illustrates the construction of a system that simulates the world of a roguelike game. In Section 4, we change it into a game. Section 5 discusses the effectiveness of the approach. And Section 6 concludes the paper.

2 Artificial Chemistry with Compartments

In this section, we briefly explain an AChem with membrane connection [5], which is an extension to an AChem with membrane division and merger [6]. The base AChem [6] is explained through Sec. 2.4; Section 2.5 explains the extension.

2.1 Virtual Atoms and Virtual Molecules

A *virtual atom* (or *v-atom* for short) corresponds to an atom in nature. A v-atom is denoted by a capitalised word such as `Abc1`. A *virtual molecule* (*v-molecule*) is a string of v-atoms or a stack of such strings (called *lines*). Figure 1 illustrates an example v-molecule; it is denoted by `0#AbcDef/1#GhiJkl/`. The slashes delimit the lines, and the number at the beginning of each line is the *displacement* of the line relative to the first line. The displacement 1 of the second line means that it starts at the position to the right by one v-atom from the first line's starting position, as shown in the figure. A displacement can be negative, in which case the line starts at some position to the left from the first line.

2.2 Patterns and Wildcard Expressions

In this AChem, chemical equations are expressed in terms of *patterns*. A pattern matches (or does not match) a v-molecule.

A pattern is composed of *literals* and/or *wildcard expressions*. A literal matches a v-atom, and is denoted by the v-atom's name; for example, the literal `Abc` matches a v-atom `Abc`. There are two types of wildcard expressions. One is an *atomic wildcard expression*, denoted by a number surrounded by angle brackets such as `<1>`. An atomic wildcard expression matches any v-atom. The number is the identifier of the expression. The other is a *sequence wildcard expression*, denoted by a number and an asterisk surrounded by angle brackets such as `<*1>` and `<2*>`. A sequence wildcard expression matches any sequence of v-atoms of any length, including the null sequence. An asterisk represents the direction in which the sequence can extend.

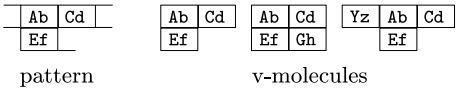

Fig. 2. An example pattern and matching v-molecules.

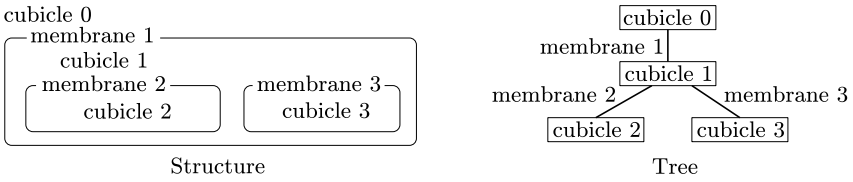

Fig. 3. An example system and the tree structure.

A pattern matches a v-molecule if and only if each line of the pattern matches the corresponding line of the v-molecule with the proper displacement. For example, the pattern `0#<*1>AbCd<2*>/0#Ef<3*>/` matches the v-molecules `0#AbCd/0#Ef/`, `0#AbCd/0#EfGh/` and `0#YzAbCd/1#Ef/` (Fig. 2). Note that, in the last case, the displacement of the second line is 1 since the expression `<*1>` represents `Yz` to make the position of `Ab` (and thus of `Ef`) become 1.

2.3 Recombination Rules

A *recombination rule* is a chemical equation in this AChem. It transforms a group of v-molecules that are matched by the rule's left-hand side patterns into the group of v-molecules expressed by the right-hand side patterns. An example rule is shown below:

$$\texttt{0\#<*0>AbCd/} + \texttt{0\#Ef<1><2*>/} \rightarrow \texttt{0\#<*0>AbCd/1\#Ef<1><2*>/} \quad (1)$$

When this rule is applied to the v-molecules `0#WxYzAbCd/` and `0#EfGh/`, they are recombined to a v-molecule `0#WxYzAbCd/3#EfGh/`.

2.4 Membranes and Cubicles

This AChem has the notion of *membrane*, which models a lipid membrane of living cells. A membrane surrounds a *cubicle*, which models a space surrounded by a lipid membrane, such as cytoplasm. A cubicle has a *reaction pool*, a multiset of v-molecules that can react with each other. A membrane also has a reaction pool; v-molecules in it model protein molecules embedded in the membrane. Cubicles and membranes are called *reaction spaces* in general. A reaction space has its reaction pool and a set of recombination rules. Since membranes can be nested, a *system* of reaction spaces forms a tree (Fig. 3).

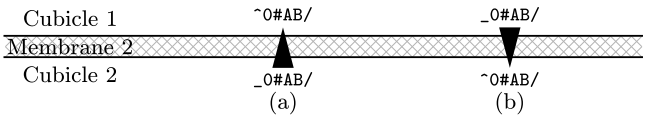

Fig. 4. Directions of membrane v-molecule.

A v-molecule embedded in a membrane, or a *membrane v-molecule*, has its direction when it is looked at from one of the neighbouring cubicles. In order to express reactions in which membrane v-molecules are involved, there are patterns to represent such v-molecules. The direction is *top* (indicated by a hat sign (^)) or *bottom* (by an underscore (_)). Figure 4 shows two membrane v-molecules. Both are `0#AB/` as v-molecules. If a v-molecule is `^0#AB/` when it is viewed from Cubicle 1, it is `_0#AB/` from Cubicle 2 (case (a)). If a v-molecule is `_0#AB/` when it is viewed from Cubicle 1, it is then `^0#AB/` from Cubicle 2 (case (b)). So the direction is relative to a cubicle from which the v-molecule is viewed.

Since membrane v-molecules model membrane protein, reactions between them can also be described. Relative directions of membrane v-molecules are expressed based on the notion of *polarity*: for example, the v-molecules shown in Fig. 4 have the reverse polarities to each other.

Polarity is indicated by an exclamation mark (`!`) in a pattern. An example rule for reaction between membrane v-molecules is shown below:

$$\texttt{0\#AB/} + \texttt{!0\#CD/} \rightarrow \texttt{0\#AB/0\#CD/} \tag{2}$$

In this rule, `!0#CD/` is *negative* to `0#AB/` (i.e., `0#CD/` has the opposite direction to `0#AB/`), and `0#AB/0#CD/` on the right-hand side is *positive* to `0#AB/`. If the first term `0#AB/` matches a membrane v-molecule `^0#AB/` (from the viewpoint of a neighbouring cubicle), the second term `!0#CD/` matches a membrane v-molecule `_0#CD/` (but does not match `^0#CD/`), and the product of the recombination is `^0#AB/0#CD/`.

2.5 Connecting Membranes

In the base AChem [6], the division of membrane is modelled as one-step action: one application of recombination rule to a membrane v-molecule may produce two membranes. The extended AChem [5] introduced an intermediate *connected* state (Fig. 5). One membrane is *divided* into two membranes (by applying a recombination rule to one of the original membrane's membrane v-molecules) but they are still *connected* by a *connecting v-molecule*; if a recombination rule to *separate* them is applied to the connecting v-molecule, they are separated. The merger of membranes goes through a connected state as well.

A connecting v-molecule is denoted by an equal sign (`=`) and membrane identifiers such as `=1(3)0#AB/`. This v-molecule, of the form `0#AB/`, connects two membranes whose IDs are 1 and 3. A connecting v-molecule has its direction,

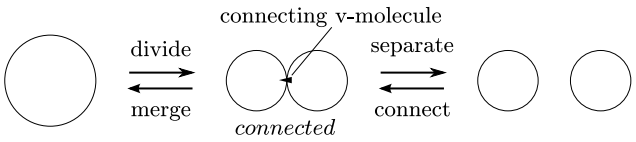

Fig. 5. The dynamics of membranes.

as membrane v-molecules do; the parentheses indicate the membrane on the bottom side of the connecting v-molecule.

A recombination rule that induces change in the membrane structure includes patterns that represent connecting v-molecules. Such patterns are denoted using a hat sign, an underscore and an exclamation mark in a similar way to denoting membrane v-molecules.

Membrane connection and separation are described by recombination rules. Rule (3) below connects two membranes.

$$\texttt{\^{}0\#A/} + \texttt{(\^{}0\#B/)} \rightarrow \texttt{=0\#AB/} \tag{3}$$

The parentheses indicate the membrane that comes to the bottom side of the produced connecting v-molecule. Swapping the left-hand side and the right-hand side makes a rule for separation.

Division of a membrane is described as the following example rule.

$$\texttt{\^{}0\#A/} + \texttt{\^{}0\#B/} \rightarrow \texttt{=\^{}0\#AB/} \tag{4}$$

This rule divides a membrane that has membrane v-molecules `^0#A/` and `^0#B/` into two membranes and leaves them connected with a connecting v-molecule of the form `0#AB/`. The cubicle v-molecules in the original cubicle are distributed to the produced cubicles randomly, and the original membrane v-molecules are also distributed to the two daughter membranes. The new cubicles inherit the recombination rules of the original cubicle, and the new membranes inherit those of the original membrane.

Swapping the left-hand side and the right-hand side of Rule 4 will make a rule for merging. When the reaction spaces are merged, the produced reaction pool is the multiset union of the original two, and the set of recombination rules of the new reaction space is the set union of the original two.

2.6 Dynamics of System

The system is executed on a simulator as follows.

1. Initialise the reaction spaces of the system.
2. Select a reaction space S at random.
3. Select a group of v-molecules in S at random.
4. Recombine them by a randomly-chosen rule R of S, if possible; this may induce structural change in the system.
5. Go to Step 2.

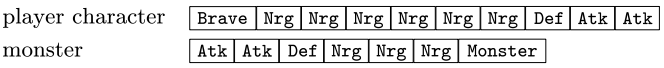

Fig. 6. The v-molecules for a monster and the player character.

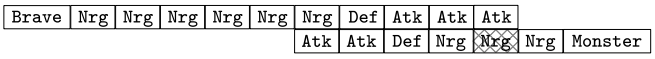

Fig. 7. The encounter of the player character and a monster.

3 Simulating the World of Roguelike Game

In this study, we first construct a system that simulates the world of a roguelike game, then change it into a game. This section illustrates the first part.

A general play of roguelike game is as follows. The human player controls the player character to have it reach the deepest part of the dungeon; the character must overcome obstacles like monsters and traps to accomplish the objective.

In this construction, the player character and monsters are represented as v-molecules, and their behaviour is described by recombination rules. Furthermore, a dungeon, represented as a system of membranes and cubicles, is generated by recombination rules as well.

3.1 Describing Battle

Figure 6 illustrates the forms of v-molecules for a monster and the player character. The v-atoms `Monster` and `Brave` differentiate them. The numbers of v-atoms `Nrg`, `Atk` and `Def` represent the amounts of energy ("hit points"), attacking strength and defence strength, respectively.

The encounter of the player character and a monster is described by (5), which forms a compound v-molecule shown in Fig. 7.

```
0#<*0>DefAtk<1*>/ + 0#<*2>AtkDef<3*>/ →
                          0#<*0>DefAtk<1*>/0#<*2>AtkDef<3*>/   (5)
```

To this v-molecule, one of (6) and (7) can be applied.

```
0#<*0>Atk<1*>/-1#<*2>NrgNrg<3*>/ →
                    0#<*0>Atk<1*>/ + 0#<*2>Nrg<3*>/ + 0#Nrg/   (6)
```

```
0#<*0>NrgNrg<1*>/1#<*2>Atk<3*>/ →
                    0#<*0>Nrg<1*>/ + 0#<*2>Atk<3*>/ + 0#Nrg/   (7)
```

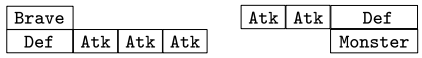

Fig. 8. The dead bodies.

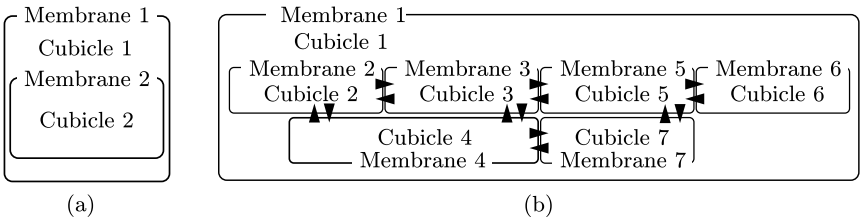

Fig. 9. The initial state of the dungeon and an example dungeon generated.

Rule (6) expresses the monster's damage, and when it is applied, one of the monster's v-atoms `Nrg` below the player's `Atk` (i.e., not defended by the monster's `Def`; the one hatched in Fig. 7) is removed. Rule (7) expresses the player character's damage in a similar way.

Shown in Fig. 8 are the v-molecules that represent the dead bodies of the player character and a monster. Note the positions of `Brave` and `Monster`; these arrangements prevent (5) from being applied to these v-molecules, so they are inactive.

A monster becomes dead when it loses all its energy. This transformation is carried out by (8).

```
0#<*0>Atk<1*>/-1#<*2>DefNrgMonster<3*>/ →
                    0#<*0>Atk<1*>/ + 0#<*2>Def<3*>/0#Monster/   (8)
```

The player becomes dead in a similar way, by the following rule.

```
0#<*0>BraveNrgDef<1*>/1#<*2>Atk<3*>/ →
                      0#Brave/0#<*0>Def<1*>/ + 0#<*2>Atk<3*>/   (9)
```

3.2 Generating a Dungeon

A dungeon in a roguelike game is typically composed of rooms (surrounded by walls) and corridors. We model them as cubicles (surrounded by membranes) and connecting v-molecules, respectively, and define rules to generate a dungeon. Figure 9(a) is the initial state of the system.

Membrane 2 has a number of v-molecules of the form `0#Cnt/`; these are raw materials of corridors in the dungeon.

To this system, the following rules are applied repeatedly. They divide a room (10) and connect the produced rooms (11) with bidirectional corridors (12); the

process generates a dungeon like the one shown in Fig. 9(b). The structure of dungeon continues to change until all the membrane v-molecules of the form `0#Cnt/` transform to connecting v-molecules `=0#Cnt/`.

$$\texttt{\_0\#Cnt/ + \_0\#Cnt/} \rightarrow \texttt{=\^{}0\#CntCnt/} \tag{10}$$

$$\texttt{\^{}0\#Cnt/ + (\^{}0\#Cnt/)} \rightarrow \texttt{=0\#CntCnt/} \tag{11}$$

$$\texttt{=0\#CntCnt/} \rightarrow \texttt{=0\#Cnt/ + =!0\#Cnt/} \tag{12}$$

The following rules enable the player character and monsters to go through corridors.

$$\texttt{=\_0\#Cnt/ + 0\#Brave<1*>/} \rightarrow \texttt{=\_0\#CntBrave<1*>/} \tag{13}$$

$$\texttt{=\^{}0\#CntBrave<1*>/} \rightarrow \texttt{=\^{}0\#Cnt/ + 0\#Brave<1*>/} \tag{14}$$

$$\texttt{=\_0\#Cnt/ + 0\#<*0>Monster/} \rightarrow \texttt{=\_0\#<*0>MonsterCnt/} \tag{15}$$

$$\texttt{=\^{}0\#<*0>MonsterCnt/} \rightarrow \texttt{=\^{}0\#Cnt/ + 0\#<*0>Monster/} \tag{16}$$

3.3 Picking Up and Using Items

In a usual roguelike game, the player character picks up and uses various items. In this game, an item is represented by a v-molecule. The following rules enable the player character to use a bomb.

$$\texttt{0\#Brave<0*>/ + 0\#Item/0\#<1*>/} \rightarrow \texttt{0\#Brave<0*>/0\#Item/0\#<1*>/} \tag{17}$$

$$\texttt{0\#Brave<0*>/0\#Item/0\#<1*>/} \rightarrow \texttt{0\#Brave<0*>/0\#<1*>/ + 0\#Item/} \tag{18}$$

$$\texttt{0\#Brave<0*>/0\#Emat<1*>/} \rightarrow \texttt{0\#Brave<0*>/ + 0\#Emat<1*>/} \tag{19}$$

A v-molecule `0#Item/0#EmatEmatBomb/` represents a bomb, and (17) attaches it to the player character (which means the character picks it up) to make a compound v-molecule `0#BraveNrgNrgDefAtk/0#Item/0#EmatEmatBomb/`. The player character uses the bomb by (18), and then the bomb is released by (19). When it hits a monster, the monster is damaged.

3.4 Creating a Monster

In a usual roguelike game, monsters come out of nowhere. The following rule expresses it.

$$\texttt{0\#Nest/} \rightarrow \texttt{0\#Nest/ + 0\#AtkDefNrgNrgMonster/} \tag{20}$$

4 Changing Simulation into a Game

The previous section described the system to simulate the world of the roguelike game, in which the player character and monsters move autonomously, requiring

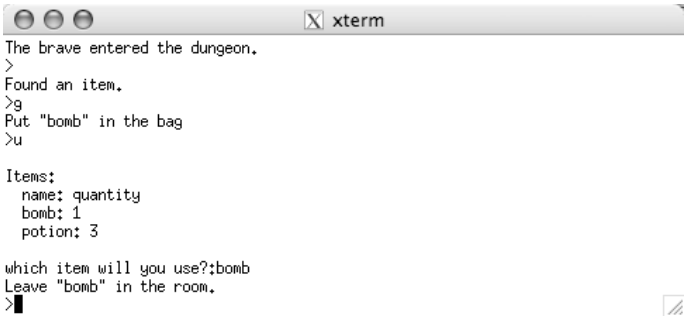

Fig. 10. A snapshot of the developed game.

no human intervention. Now we change the simulation into a game, in which the human player controls the use of items obtained by the player character.

To do this, we remove a rule. The player character picks up an item by (17) and uses it by (18), and the item works by (19). So we remove (18) from the system, and add code outside the AChem to control the use of items. Figure 10 shows a snapshot of the obtained game.

5 Discussion

A significant result of the study is the brevity of description that defines the simulation. A set of 22 rules describes the generation of dungeon, battle between the player character and monsters, the movement of them in the dungeon, the player character's picking up and using items (bombs and healing potions), and creating monsters. When this type of game is implemented with a traditional programming language, the programmer must use conditional statements and iterations. But the AChem implies them: v-molecules that are matched by the left-hand side of a rule are selected and transformed, and this process is iterated. The whole process is automatically performed by the simulator. Moreover, the AChem simulator uses random numbers to choose v-molecules and rules. It may eliminate coding for, say, selecting actions using random numbers, which is typically required in traditional game programming. For example, in our game, choosing a monster to meet the player character is not explicitly described using a random number; neither is deciding which of the player character and the monster making a battle v-molecule wins. These properties seem to be beneficial not only to implementing a roguelike game but also another type of game if an element of the game meets another element probabilistically and the meeting gives rise to a certain event, like chemical reaction.

Using the AChem, we took a specific approach to make the game: first, we built a system to simulate the world of the game; then we removed a rule to change the mere simulation into a game. Specifically, we removed the rule for using items and left the function outside the AChem to have it controlled by

the human player. This method — to design simulation and then to remove a function — seems a general way to design a game based on simulation. Suppose we remove a rule to create monsters, instead of the one for using items, from the simulation and let the human player control the creation. Then a different game is obtained, in which the human player thwarts the hero's adventure. A famous simulation game, SimCity, can be viewed in a similar way. The player plays the role of government of the city to design the city. This game simulates everything about the city but the government. Panic Shooter [2] is a shooter game built on cellular automata. The behaviour of bullets and barriers is defined as rules of cellular automata. The player proceeds toward a designated goal, shooting or avoiding barriers. This is similar to our approach, in which the world of game is defined by rules for simulation, and the only part the player controls is outside the simulation.

6 Concluding Remarks

In this paper, we have illustrated the development of the roguelike game in the artificial chemistry with membranes. We took the two steps to develop it: first, we built an AChem system to simulate the world of the game; then, we removed a rule for using items and left the function outside the AChem simulator. We speculate this "simulation-minus-one" approach may be useful to develop simulation games. Since AChems are tools for simulation, they may be an effective platform for building such kinds of games. We also showed how easy it is to do "minus one" in the AChem.

Because the "minus-one" part is not dealt with by the AChem simulator, its function must be implemented by an outside mechanism, such as a program written in a traditional programming language. A framework to support this implementation will enhance the effectiveness of the present approach.

References

1. Electronic Arts Inc.: The official Spore and Spore creature creator site. `http://www.spore.com/ftl`
2. Tosik: Cell automaton wo motiita shooting game. `http://d.hatena.ne.jp/tosik/20070409/1176081337` In Japanese.
3. Kim, K.J., Cho, S.B.: A comprehensive overview of the applications of artificial life. Artificial Life **12**(1) (2006) 153–182
4. Dittrich, P., Ziegler, J., Banzhaf, W.: Artificial chemistries – a review. Artificial Life **7**(3) (2001) 225–275
5. Watanabe, T.: Extending an artificial chemistry by incorporating membranes. Master's thesis, School of Computer Science, Tokyo University of Technology, Tokyo, Japan (2008) In Japanese.
6. Tominaga, K., Watanabe, T., Suzuki, M.: Formulating membrane dynamics with the reaction of surface objects. In: Advances in Artificial Life: Proceedings of the 9th European Conference on Artificial Life, Springer (2007) 12–21