Solving Tile-Matching Puzzle game using CSP solver

Shuma Kurihara s1190220 Supervised by Maxim Mozgovoy

Abstract

There are many tile-matching games such as Tetris, Puyo-Puyo, Colums, Puzzle and Dragons and so on. To solve tile-matching puzzle games, human players created a lot of strategies. These strategies are also useful for AI. However, in most of cases, applying these strategies to AI is very hard and takes long developing times. Many strategies can be expressed by the set of constraints. If we regard these methods as CSP and use CSP solver, to develop these methods might become easier and be finished in a short period. In this research, we apply this idea to improve the AI of Puyo-Puyo which is made by Tomizawa [1].

1 Introduction

1.1 The rule of Puyo-Puyo

Puyo-Puyo is one of the tile-matching puzzle games. Players can move tiles in the field like tetris. The field has 6 rows and each rows have 13 columns. Tiles have a color. There are four colors of tiles. Colors are red, blue, green, and yellow. In this game, the tile is called a puyo. Puyos are little creatures who fall from the top of the field in a pair. The pair is called hai-puyo. Hai-puyo can be moved left and right and rotated. Hai-puyo falls until it reaches another puyo or the bottom of the field(figure(1)). Hai-puyo is notified beforehand. Players can see a current hai-puyo, next one and after next one. Every time hai-puyo is placed on the bottom of the field or on the top of another puyo, new hai-puyo is chosen in randomly. When field is filled by puyos, players can not select any actions. Then, game is finished.

When four or more puyos of the same color are placed near each other to create a group, they disappear. This is called a Rensa(Chain). Individual puyos can connect horizontally or vertically, but not diagonally. The puyos above those that are cleared fall onto other puyos or the bottom of the field(figure(2)). A Fluent Rensa(step-chain) is made when falling puyos form a new group in a chain reaction. When player make N step-chain, player can get a gain which is proportional to N^2. Because the gain increase exponentially, to make long step-chain is important.

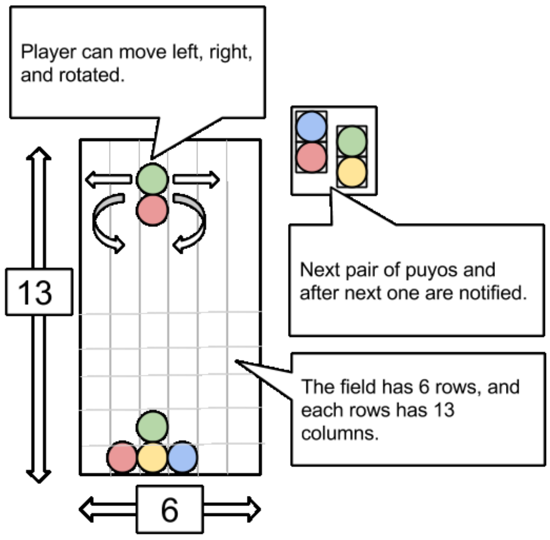

Player can move left, right, and rotated.
13
6
Next pair of puyos and after next one are notified.
The field has 6 rows, and each rows has 13 columns.

Figure 1:

However, since the hai-puyo is chosen in randomly, it is hard to make a large step-chain. One of the strategies to make a long step-chain easily is Teikei.

1.2 Teikei

Teikei is a good pattern of the opining phase. It is similar to chess opening (the book moves) or similar to joseki in the game of Go. Figure(**??**) shows domino-chain which is one of the most famous teikei. This domino-chain has a potential of 5 step-chain, however it use only 20 puyos. Without few exceptions, teikei has characteristics which are the symmetrical arrangement and the compactness. Note that it is no problem even if red puyos is replaced with yellow puyos. Also, it is no problem even if red puyos and blue puyos are swapped.

This idea is very useful for human. and Tomizawa state that it is also useful for AI [1].

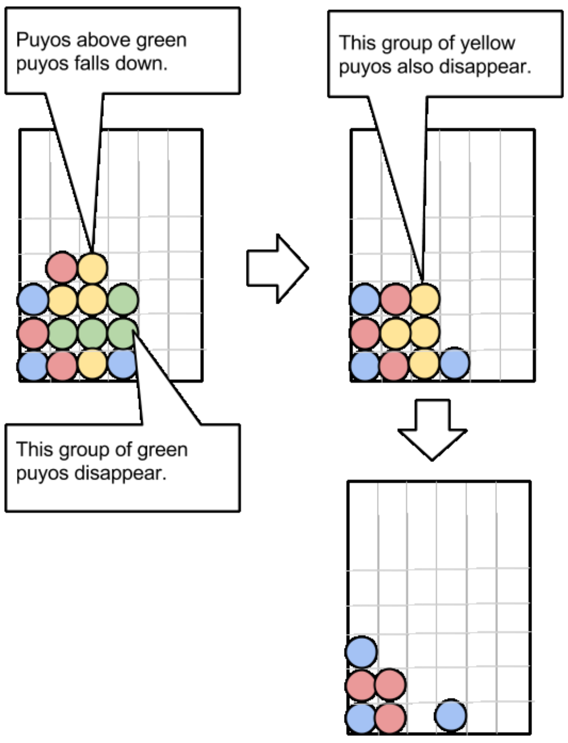

Puyos above green puyos falls down.
This group of yellow puyos also disappear.
This group of green puyos disappear.

Figure 2:

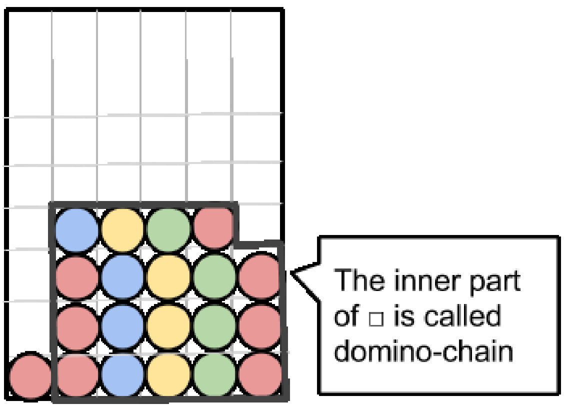


Figure 3:

2 Past research

2.1 tomizawa's method

Tomizawa proposed a method to make large step-chain efficiently. The most important idea of his method is to use teikei.

In his method, we use a evaluation function.This evaluation function returns a positive value if the state of the field corresponds to teikei. Or else, this evaluation function returns the value of $-\infty$.

To implement this function, he use two matrices in this function. One is called state-matrix, another is called template-matrix. When the size of the field is n, both matrices has n rows and each rows has n columns. State-matrix and template-matrix indicate that relations between two cells which are the same color or the different color. The (i, j) element of template-matrix has a positive value if the cell of i and the cell of j must be the same color. This element has a negative value if these cells must be different colors. If these cells have no relation, this element has a zero value. The (i, j) element of state-matrix is 1 if the cell of i and the cell of j is the same color. If the cell of i and the cell of j is different colors, the (i, j) element is -1. Else, the value of the element is 0. We can get matrix M by multiplying template-matrix and state-matrix. If any elements of matrix M is 0, this evaluation function returns the value of $-\infty$. Or else, this evaluation function return the sum of the elements of matrix M.

He compared developed AI which use this method to the previous AI [6]. Results of the test indicated that his AI is better than the previous one.

2.2 The error of teikei

Figure(4) is similar to figure(5). Each figures, tiles in the area of inside of the black frame satisfy arrangements of domino-chain. Figure(4) surely has a potential of 5 step-chain. By contrast, figure(5) has a problem. It seems that figure(5) also has a potential of 5 step-chain. However it will stop at third disappearance. Such errors are called "bouhatsu" or just "error".

2.3 Weakness of Tomizawa's method

Tomizawa's method has two problems:

1. His method doesn't consider the error of teikei.
2. There is no room for extension in his evaluation function.

Therefore, Tomizawa ' s method can ' t prevent errors well.

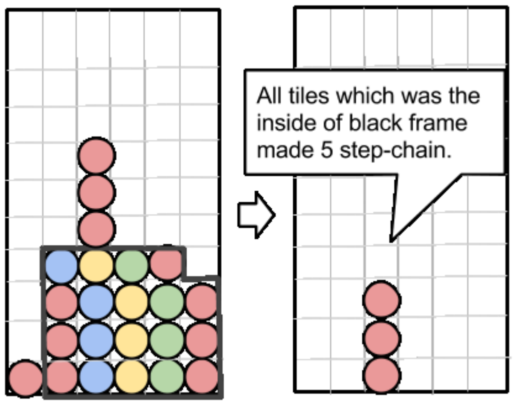

All tiles which was the inside of black frame made 5 step-chain.

Figure 4:

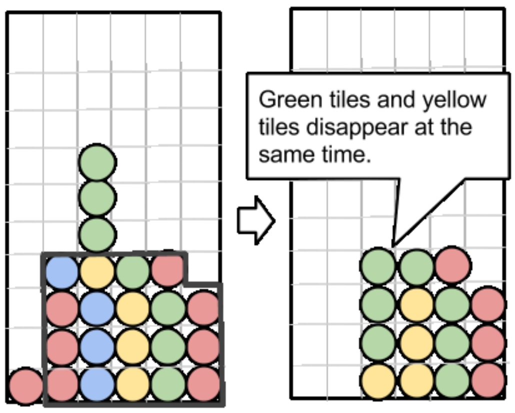


Figure 5:

When we want to prevent errors using original evaluation function with some treatment, I think that we can choose from two ways. One is like that:

1. Search all relations which make the error like a, b, c, d in figure() .
2. Pick out two positions from set of relations which is searched in 1, and add negative value to elements of template-matrix which indicate the relation of these positions.
3. Forward step-chain and Repeat from 1 if there are any group which can disappear.

Another is like that:

1. Pick out two positions which are adjacent by horizontally or vertically like a, b, and add negative value to elements of template-matrix which indicate the relation of these positions.
2. Forward step-chain.
3. Repeat 1 and 2 for new field.

Although we take either way, we can not express actual constraints.

3 Proposal method

3.1 CSP

Constraint satisfaction problems (CSPs) are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods. CSPs are the subject of intense research in both artificial intelligence and operations research, since the regularity in their formulation provides a common basis to analyze and solve problems of many unrelated families. [4]

There are many programing libraries which solve CSP. In this research, We use python-constraint the external library of Python. [5]

3.2 Improvement of Tomizawa's method

It is too hard to prevent errors if we use only Tomizawa's method. Therefore, we created the plugin using CSP solver. This plugin works as evaluation function which checks if the state of the field corresponds to teikei.

CSP consists of three elements. They are variables, domains and constraints. In this method, variables correspond to the position of the field. Variables are named $X_{(i,j)}$ which is regarded as the (i, j) element of the field. Usually, each domains is set of *Red*, *Blue*, *Yellow*, *Green* and *Empty*. Those domains are reduced during the progress of a game. Constraint has three types as bellow:

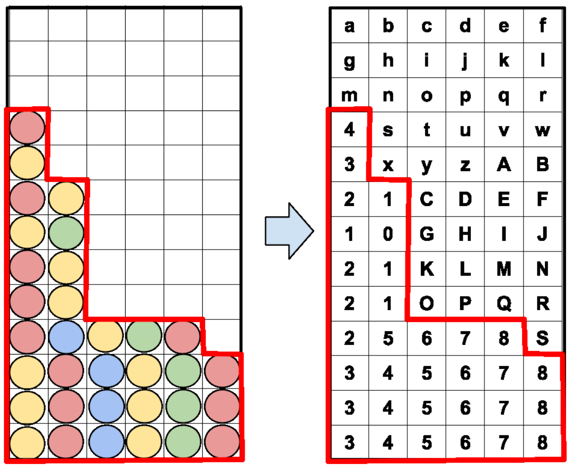
eq2: the relation between two variables which are the same value.

ne2: the relation between two variables which are the different value.

ne4: the relation between four variables all of which are not the same value or which have the value of "Empty".

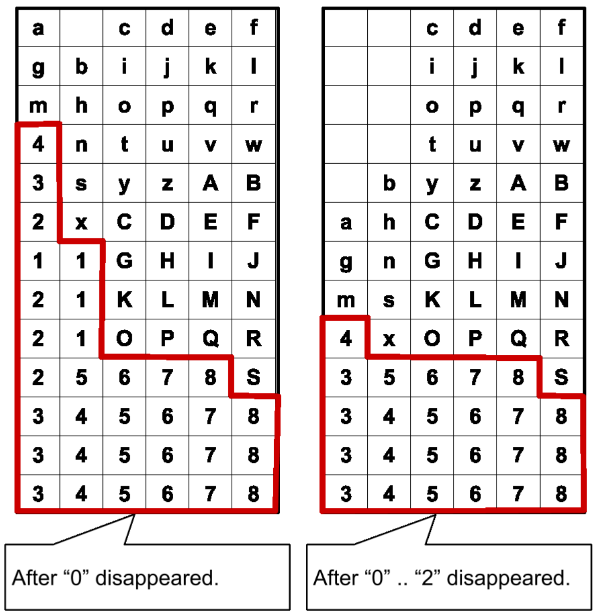
The proposed method is separated into two parts: preparing before the game and selecting a action during the game. Before the game, constraints are defined. During the game, domains are defined and field are evaluated to select the action which seems the best to make the teikei.

Before the game, we make a set of constraints. In this part, we use a template table. In the case of making teikei described figure(6).a , the template table becomes figure(6).b. This table has potential to produce 8 step-chain. Numbers in the table indicate the order of disappearance during the step-chain. Letters indicate that the puyo which is placed on the position which letter is in must not be related to step-chain. Letters would be used to set a constraint to prevent an error. To find a constraint is performed in two steps. First step is for making teikei, second step is for preventing an error. The reason of dividing into two step is that former constraints are useful to reduce the amount of latter constraints. Positions which have the same label in template-table must have the same value. For example, $X_{(3,12)}$, $X_{(3,11)}$, $X_{(3,10)}$ and $X_{(2,9)}$ must have the same value. We add these sets and a flag of eq-constraint to set of constraints together. To find ne-constraint, we have to take some steps. First, we consider positions which have the same number in the template-table as a group. The pair of group which adjoin horizontally or vertically must be the different value, If these group is the same value, it would be disappear in the same time. We add the pair which are pulled out from each groups and a flag of be-constraint to set of constraint together. Next, we make a template-table which is lack of the



a	b	c	d	e	f
g	h	i	j	k	l
m	n	o	p	q	r
4	s	t	u	v	w
3	x	y	z	A	B
2	1	C	D	E	F
1	0	G	H	I	J
2	1	K	L	M	N
2	1	O	P	Q	R
2	5	6	7	8	S
3	4	5	6	7	8
3	4	5	6	7	8
3	4	5	6	7	8

Figure 6:



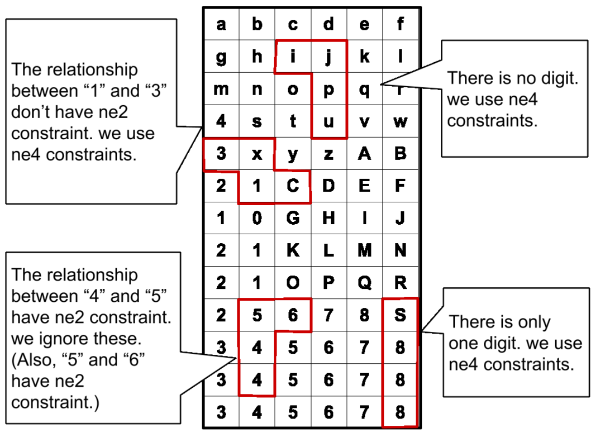
a		c	d	e	f
g	b	i	j	k	l
m	h	o	p	q	r
4	n	t	u	v	w
3	s	y	z	A	B
2	x	C	D	E	F
1	1	G	H	I	J
2	1	K	L	M	N
2	1	O	P	Q	R
2	5	6	7	8	S
3	4	5	6	7	8
3	4	5	6	7	8
3	4	5	6	7	8

After "0" disappeared.

		c	d	e	f
		i	j	k	l
		o	p	q	r
		t	u	v	w
	b	y	z	A	B
a	h	C	D	E	F
g	n	G	H	I	J
m	s	K	L	M	N
4	x	O	P	Q	R
3	5	6	7	8	S
3	4	5	6	7	8
3	4	5	6	7	8
3	4	5	6	7	8

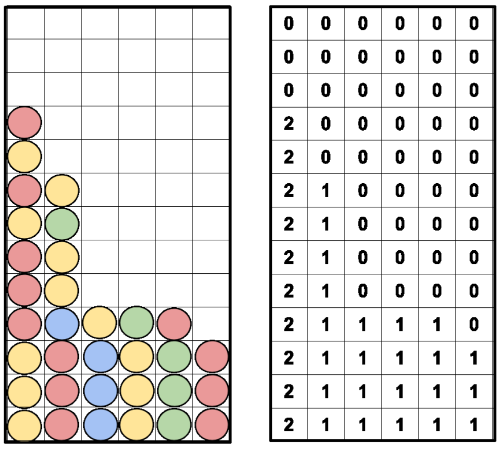
After "0" .. "2" disappeared.

Figure 7: temp figure


The relationship between "1" and "3" don't have ne2 constraint. we use ne4 constraints.
There is no digit. we use ne4 constraints.
The relationship between "4" and "5" have ne2 constraint. we ignore these. (Also, "5" and "6" have ne2 constraint.)
There is only one digit. we use ne4 constraints.

a	b	c	d	e	f
g	h	i	j	k	l
m	n	o	p	q	r
4	s	t	u	v	w
3	x	y	z	A	B
2	1	C	D	E	F
1	0	G	H	I	J
2	1	K	L	M	N
2	1	O	P	Q	R
2	5	6	7	8	S
3	4	5	6	7	8
3	4	5	6	7	8
3	4	5	6	7	8

Figure 8:



0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
2	0	0	0	0	0
2	0	0	0	0	0
2	1	0	0	0	0
2	1	0	0	0	0
2	1	0	0	0	0
2	1	0	0	0	0
2	1	1	1	1	0
2	1	1	1	1	1
2	1	1	1	1	1
2	1	1	1	1	1

Figure 9:

label 0 (figure(7).a). Likewise, we repeats the same operations for this template-table. This repeating is to prevent such situation(figure(7).b). In the figure(7).b, values of $(1, 8)$, $(0, 8)$, $(0, 9)$, $(0, 10)$, $(0, 11)$ are disappeared if these values are the same color, and then this step-chain is stopped in the label 4. We repeat these operations for the template-table which is lack of the label 0 and 1, and also repeat for the template-table which is lack of the label 0, 1 and 2. To find the constraint which is used for preventing an error, we pull out four positions which adjoin horizontally or vertically such as figure(8).a and figure(8).b If all labels which are pulled out are some numbers, we ignore those. Also, if any pair of the members of the group is appeared in the ne2-constraint, we ignore those. In other cases, we add these group and a flag of ne4-constraint to set of constraint together. We repeats the same operations for the template-table which is lack of the label 0, which is lack of the label 0 and 1.

During the game, every time an state of field is changed, we search all possible actions for three turns later. Then, we find the values which are satisfied all constraints for all valuables and evaluate the state of field. To find the answer which is satisfied constraints, we set domains conforming to a current state of the field. If there is a puyo in the position (i, j) of the field, the domain of $X_{(i,j)}$ is only the color of this puyo. For example, if there is the puyo which color is red, the domain of $X_{(i,j)}$ is a set of *Red*. In other case, if the label of template-table is a digit, the domain of $X_{(i,j)}$ is a set of *Red*, *Blue*, *Green* and *Yellow*. If the label of template-table is a letter, the domain of $X_{(i,j)}$ is a set of *Red*, *Blue*, *Green*, *Yellow* and *Empty*. Then, we find the answer by using CSP solver. If there is no answer, we evaluate this state of field as $-\infty$. If there are some answer, we evaluate this state of field by some kind of evaluation function.

4 Performance Test

We took two AI. One used Tomizawa's AI with our plugin (proposal method) and another used Tomizawa's AI without my plugin. To clarify the difference whether to use plugin or not, both AI made the same domino-chain which has a potential of 8 step-chain (figure**??**) with preventing errors, and both AI use the same weighting function. We compared both AI by the number of failures in 100 test cases. The weighting function consists of the following flow:

1. Prepare the weight-table (figure(9)).

2. Calculate the sum of all values on the weight-table

3. Calculate the sum of the value which position is filled by any puyo.

4. Return the value: (2) divide by (3).

The value is the closer to 1.0, it is the better . If the value is 1.0, we see that teikei is completed.

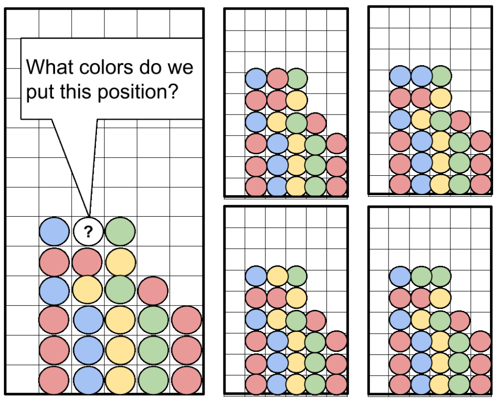


Figure 10:

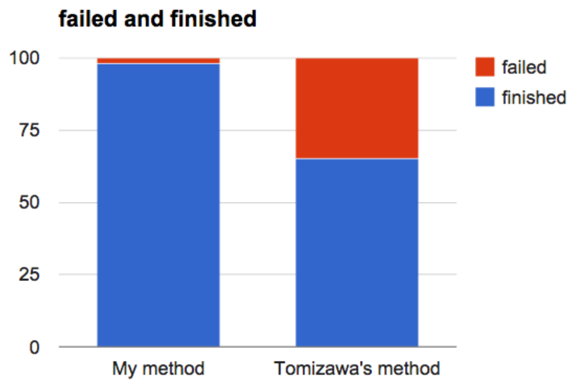


Figure 11:

steps	with my plugin	without my plugin
failed	2	35
finished	98	65

5 Results

The result is shown in the table(5). It indicate that proposal method is better than Tomizawa's method in the parts of composing teikei with preventing error. Proposal method failed only 2 cases. On the other hand, Tomizwa's method is failed in 35 cases. Because of randomness, it is hard to succeed by all test cases. However, 35 cases are too much. In figure(10), if we use proposal method, we can put any puyo in the position $(3, 8)$. However, if we only use relationships between two positions, we cannot put any puyo in the position $(3, 8)$. These situation would occur in many test cases. It is reasonable to suppose that these situation increase the rate of failure.

6 Conclusion and Future Work

This research indicated that some strategies of tile-matching puzzle games can be applied to AI easily by CSP solver.

From examine the findings, the rate of failure depend on not only constraints but also weighting functions. We will try to improve it by machine learning in the future.

Acknowledgement

I would like to thank Prof. Mozgovoy for supervising my works.

References

[1] 富沢大介 and 池田心, "落下型パズルゲームの定石形配置法とぷよぷよへの適用," 2012, 情報処理学会.

[2] Wikipedia, "Puyo Puyo(series)," 05:52, 10 September 2014.

[3] 武永康彦, "一般化ぷよぷよの NP 完全性," 2005, 数理解析研究所講究録, 1426, 147–152.

[4] Wikipedia, "Constraint satisfaction problem," 06:27, 7 February 2015.

[5] "python-constraint," http://labix.org/python-constraint.

[6] Ikeda Laboratory Project, "Poje" http://www.jaist.ac.jp/is/labs/ikeda-lab/poje/index.html.