Improving Generalization Ability in a Puzzle Game Using Reinforcement Learning

Hiroya Oonishi
Kyoto Institute of Technology
Kyoto, Japan
onishi6h@cis.is.kit.ac.jp

Hitoshi Iima
Kyoto Institute of Technology
Kyoto, Japan
Iima@kit.ac.jp

***Abstract*—Nowadays machine learning has attracted much attention. In order to apply it to various problems without relearning, its generalization ability is needed. Geometry Friends is a puzzle game where a player has to collect all targets in a two-dimensional world, and it is used in some artificial intelligence competitions. Although sufficient generalization ability is needed to apply the machine learning to this game, such machine learning methods are not proposed yet. In this paper, we propose a method based on reinforcement learning in which the generalization ability is improved for Geometry Friends.**

I. Introduction

Nowadays machine learning has attracted much attention in many fields such as speech/character recognition, search engine and game artificial intelligence (AI). In the game AI, machine learning such as reinforcement learning [1] is known as an effective method [2] [3] as well as powerful search algorithms [4] [5]. One of the abilities which the remarkable machine learning should have is to be able to learn for various problems without relearning, that is the generalization ability.

A game AI competition is often held, and is used for evaluating the performance of machine learning methods. Geometry Friends [6] is a game used in this competition. It is a puzzle game where a player controls a character or two characters which collect targets in a two-dimensional world, as shown in Fig. 1. Because it has not only the level layout shown in this figure but also other various level layouts, sufficient generalization ability is needed to apply machine learning to it. However, such machine learning methods are not proposed yet.

A method based on reinforcement learning is proposed for Geometry Friends [7]. In this method, an overall problem that a character learns to collect all targets in any level is divided into three sub-problems, and reinforcement learning is used to solve two of them. However, it does not necessarily have the generalization ability because there are too many states in the two reinforcement learning sub-problems to learn optimal actions for all the states.

In this paper, we propose a method based on reinforcement learning whose generalization ability is improved for Geometry Friends. In our proposed method, the overall problem is divided into two sub-problems, which is similar to the existing method, and the reinforcement learning is used to solve one of them. The number of states in the sub-problem is intentionally reduced so that the generalization ability can be acquired by training even a few times. We show from experimental results that our proposed method improves the generalization ability.

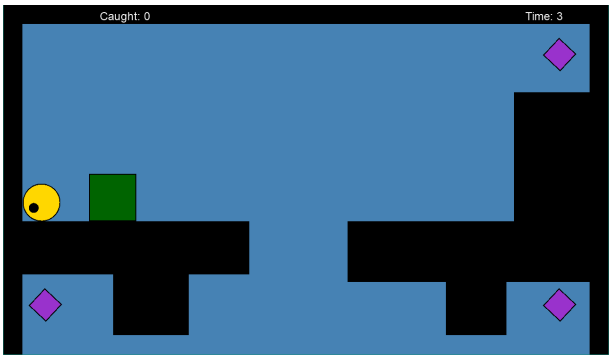

Fig. 1. A screen of Geometry Friends

The structure of this paper is as follows. Section II explains Geometry Friends and its competition. Section III explains the structure of our proposed method in which the overall problem is divided into the two sub-problems. Sections IV and V propose methods to solve the respective sub-problems. Section VI conducts experiments of applying our proposed method and shows from the experimental results that it improves the generalization ability. Section VII concludes this paper.

II. Puzzle game: Geometry Friends

This section explains Geometry Friends and its competition.

A. Details of the game

Geometry Friends is a puzzle game where a player controls a character or two characters whose objective is to collect all targets distributed in a two-dimensional world within a time limit. In Fig. 1, the circle, the rectangle and a diamond are the first character, the second one and a target, respectively. The characters are affected by gravity and friction like the real

world. The player has to make them touch a target to collect it.

There are three kinds of obstacles: a black obstacle, a yellow obstacle, and a green obstacle. The circle character's movement is restricted by the black and green obstacles, and the rectangle character's movement is restricted by the black and yellow ones. In Fig. 1, there are some black obstacles. In addition, there is always an obstacle at the bottom row, although it is not drawn on the screen of Geometry Friends. The characters can get on an obstacle, as shown in Fig. 1.

Geometry Friends has various levels which differ from each other in the positions of the targets and the obstacles and in the initial positions of the characters. Thus, sufficient generalization ability is needed to apply machine learning to this game.

The characters have different possible actions from each other. The possible actions of the circle character are as follows:

- Roll left,
- Roll right,
- Jump,
- Change its size.

The possible actions of the rectangle character are as follows:

- Slide left,
- Slide right,
- Change its shape.

While the circle character's action, "change its size", changes its mass, the rectangle character's action, "change its shape", makes it wider or narrower without changing its mass.

Information which the AI player can obtain from Geometry Friends is as follows:

- Position of each character,
- Horizontal and vertical velocity of each character,
- Position and size of each obstacle,
- Position of each target.

There are three modes in which the player controls only the circle character or only the rectangle character or both. This paper targets the mode in which the player controls only the circle character.

B. Competition

A competition where AI players play Geometry Friends has been held at IEEE Conference on Computational Intelligence and Games since 2013.

In this competition, there are three tracks which correspond to the three modes mentioned above: a track for the circle character only, for the rectangle character only, and for both characters. Each track has ten levels and five of them are made available to the public before the competition, which allows participants to evaluate the performance of their AI players. However, the others are confidential and the participants cannot know them until the beginning of the competition. The AI players are ranked by their scores calculated from the number of targets they collect and from the time they take to collect all targets. Although some methods to design AI players based on reinforcement learning were proposed in a past competition [7] [8], they cannot collect all targets in some levels because their generalization ability is not sufficient.

III. Proposed Method

In Geometry Friends where only the circle character is controlled, even if a method can solve a problem that the circle character learns to collect all the targets in only a specific level, it cannot necessarily do so in other levels and therefore does not have the generalization ability. As explained above, there are proposed no methods which have the sufficient generalization ability. In this section, we discuss problems to be solved in this game and then explain the structure of our proposed method, which has the sufficient generalization ability, based on solving each of the problems.

A. Problems to be solved in Geometry Friends

Problems that must be solved to collect all targets in Geometry Friends are as follows:

- Finding the shortest path to collect all the targets
- Selecting the character's actions to follow the path

Why the problems must be solved is explained using an example of a level, as shown in Fig. 2. In this level, the circle character cannot get out of any hole once it falls into the hole. Hence, it must collect the left target without falling into the hole. For the same reason, the target in the hole must be collected after collecting the left target. Therefore, the path which should be found is to collect the left target and the right one in that order. In addition, the path should be shortest because a high score is obtained in the case where the character collects all the targets in a short time. Therefore, the first problem of finding the shortest path to collect all the targets is addressed.

The second problem of selecting the character's actions must also be solved. This is because the character becomes unable to collect all the targets unless it can select the actions to follow the found path.

B. Structure of the proposed method

In order to solve the overall problem that the circle character learns to collect all the targets, the two problems explained in

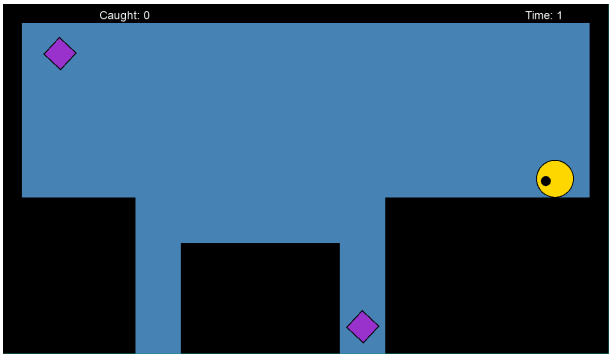

Fig. 2. An example of a level

the previous subsection are defined as the sub-problems of the overall problem and they are solved. In order to be able to collect all the targets in particularly any level, for the first sub-problem, we propose a method which finds a superior path in any level.

In the existing method [7], the second sub-problem of selecting the character's actions is further divided into two sub-problems, which are solved by using reinforcement learning. However, there are too many states in the two reinforcement learning sub-problems to learn optimal actions for all the states. In our proposed method, some information to follow the shortest path is obtained when it is found. By using the information, the sub-problem of selecting the character's actions can be formulated as an easy problem of reaching a target position with a target horizontal velocity on a flat obstacle and the easy problem is solved by using reinforcement learning. In order to be able to follow the found path in particularly any level, we propose a method which acquires optimal actions to any target position with any target horizontal velocity.

Reinforcement learning is performed before the AI player moves the character in unknown levels. In moving it in the unknown levels, the sub-problem of finding the shortest path to collect all the targets from the initial character's position is generated and solved, and actions to follow the found path are determined using the result of the learning. However, there may be the case where the character cannot follow the found path. In the case, the sub-problem of finding the shortest path from the position at which it cannot follow is generated and solved again, and actions to follow the found path are determined.

We propose concrete methods to solve each of the two sub-problems in the next two sections.

IV. Method to solve the sub-problem of finding the shortest path

This section defines the sub-problem of finding the shortest path to collect all the targets from a given position of the character and proposes a method to solve it.

A. Idea of the proposed method

As shown in Fig. 3, a flat obstacle or its part which the character can get on is named a platform. The character can move between the left edge of any platform and the right edge without jumping or falling. In order to move to another platform, it must jump or fall. Whether it can move from a platform to another or not depends on the positions of the two platforms. It can collect each target by moving on a platform or by jumping or falling from it.

A path is defined as the order of the platforms the character must move to. Concretely, we generate a directed graph whose node corresponds to a platform and whose edge is given if the character can move from a platform to another platform. Then, the shortest path in the graph is found by applying a search algorithm. Because many search algorithms have been proposed for graphs, one of them is adopted.

To generate the graph, the following two pieces of information is required:

- Platforms to which the character can move from each platform,
- Targets which it can collect from each platform.

To obtain the information, its movement must be grasped. This grasp is achieved by predicting the trajectories of the jumping or falling character.

In summary, the procedures to solve this sub-problem are as follows:

1) Predict the trajectories of the jumping or falling character,
2) Obtain the platforms the character can move to and the targets it can collect from each platform,
3) Generate the directed graph,
4) Apply a search algorithm to the directed graph.

If the procedures 1) and 2) work effectively for any level, the sub-problem can be solved for the level. The succeeding subsections describe each of the four procedures.

In the existing method [7], although a similar graph is generated and solved, the individual procedures differ from our proposed method. These differences are also explained in the succeeding subsections.

B. Prediction of the trajectories of the jumping or falling character

Since the positions of obstacles differ among different levels, the trajectories of the jumping or falling character also differ among different levels considering collision with the obstacles, which makes it difficult to predict the trajectories. Therefore, first of all, the case with no obstacles is treated, and the trajectories for the case are predicted. Then, for the case with some obstacles placed at any position, the trajectories are predicted by using the ones predicted for the first case.

It is assumed that a trajectory of the jumping or falling character is a parabola in the first case. Hence, for any jump and any fall, the trajectory is formulated by

$$x = v_x t \tag{1}$$

$$y = v_y t - \frac{1}{2} g t^2 \tag{2}$$

where t is the time and is set as zero when the character begins to jump or fall. x and y are the ordinate and the abscissa respectively and the origin is set as the coordinates of the

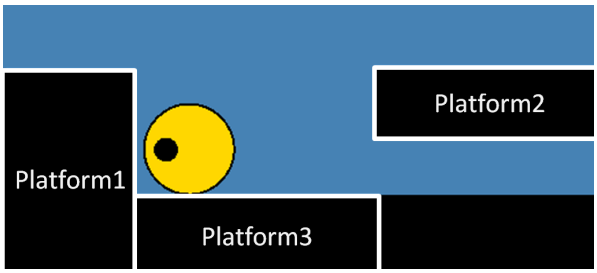

Fig. 3. The obstacles called platforms

character at $t = 0$. v_x and v_y are the horizontal velocity and the vertical velocity of the character at $t = 0$ respectively. g is the acceleration of gravity, and its value is different from that used in the real world. We assume that the horizontal velocity at each time is constant and is the same value as v_x.

By (1) and (2), if v_x, v_y and g are obtained from Geometry Friends, x and y, which mean the trajectory of the character, can be predicted for any jump or any fall. However, it is difficult to obtain them directly because of limitation by Geometry Friends. Because the horizontal velocity at a delayed time can be obtained and is assumed to be equal to v_x, v_x is set as the horizontal velocity at the delayed time. Because it is confirmed empirically that v_y in the case where the character falls is 0, v_y is set as 0.

v_y in the case where it jumps and g are estimated by solving an optimization problem. The estimated values of v_y and g are called $\hat{v}_y$ and $\hat{g}$ respectively, and y calculated by substituting $\hat{v}_y$ and $\hat{g}$ into (2) is called $\hat{y}$. In the optimization problem, the objective function f to be minimized is as follows:

$$f = \sum_t \bar{f}_t, \tag{3}$$

$$\bar{f}_t = \frac{1}{2}(\hat{y}(t) - y(t))^2. \tag{4}$$

The decision variables are $\hat{v}_y$ and $\hat{g}$. To solve the optimization problem, we apply stochastic gradient descent which is widely used in machine learning. In one iteration of the method, the character is made to jump or fall, and $\hat{v}_y$ and $\hat{g}$ are updated by

$$\begin{bmatrix} \hat{v}_y \\ \hat{g} \end{bmatrix} = \begin{bmatrix} \hat{v}_y \\ \hat{g} \end{bmatrix} + \alpha \nabla \bar{f}_t(\hat{v}_y, \hat{g}) \tag{5}$$

where α and $\nabla \bar{f}_t$ are the learning rate and the gradient of $\bar{f}_t$ respectively. In order to update the estimated values by (5), t must be selected and $y(t)$ must be obtained from Geometry Friends. However, $y(t)$ can be obtained only for a limited number of times. Due to this limitation, t is selected at random from among the times, and y for the time is obtained from Geometry Friends. Since v_x, v_y and g are obtained by the procedures mentioned above, the trajectories of the character can be predicted for the first case, where there are no obstacles.

Next, the second case where obstacles exist is addressed. We assume that once colliding with an obstacle, the character simply falls straight down. Until the collision, a trajectory for the first case is used. It is confirmed empirically that this simple movement is similar to the real movement.

C. Obtaining the platforms the character can move to and the targets it can collect from each platform

In the existing method [7], whether the character can move from a platform to another platform is decided by the distance between the two platforms and by the placement of obstacles between them. If there is an obstacle between the two platforms, it is decided that the character cannot move between them because the character should collide with it. However, because it does not necessarily collide with such an obstacle in fact, the errors of the decision occur frequently. Targets which the character can collect from each platform are decided depending to their positions. Concretely, it is decided that a target just above a platform can be collected by the character which is on the platform. Although it is decided that targets between two platforms cannot be collected from any platform, the decisions are frequently incorrect.

In our proposed method, such incorrect decisions are avoided by using the character's trajectories predicted in the previous subsection. If no obstacles are on at least one of all trajectories between the two platforms, that is, the trajectories of the character jumping or falling from all the positions with all the horizontal velocities, then it is decided that the character collides with no obstacles. Platforms and targets the character can move to and collect, respectively, are also decided in the same way. Since any trajectory can be predicted by the method in the previous subsection, these decisions can be made in any level. However, it takes so much time to use all the trajectories. This problem is resolved by using trajectories only for a limited number of positions and velocities. Concretely, the character is limited to jumping or falling from positions at regular intervals. Let V_{max} be the maximum velocity of the character, and the velocities are limited to $-V_{max}, -V_{max}+C, -V_{max}+2C, \cdots, V_{max}$ where C is a positive constant.

As the result of the decisions, the following information is obtained.

- Platform from which the character jumps or falls
- Position in which it jumps or falls
- Its movement (jumping or falling)
- Horizontal velocity with which it jumps or falls
- Targets which it can collect
- Platform on which it lands
- Position in which it lands

The information is used to generate a directed graph.

D. Generation of the directed graph

The directed graph is generated by regarding a platform as a node and by drawing edges based on the information obtained in the previous subsection. Concretely, an edge is drawn from the node corresponding to a "platform from which the character jumps or falls" to the node corresponding to a "platform on which it lands". The length of the edge is defined as the distance from the current position to the "position in which it lands" via the "position in which it jumps or falls". In addition, "its movement", "horizontal velocity with which it jumps or falls" and "targets which it can collect" are set as the properties of the edge.

E. Application of the search algorithm to the directed graph

In order to find the shortest path in the directed graph, Subgoal A* [9] is adopted. Subgoal A* is an improved A* search algorithm and can find the shortest path in any graph even if there is more than one goal node. As the result of applying Subgoal A*, the platforms which the character must get on can be obtained. Moreover, the position in which it must

jump or fall on each of the platforms, the horizontal velocity with which it must jump or fall, and its movement can be obtained from the properties of the edges on the found path. Hence, by making it reach the position with the horizontal velocity and jump or fall in the position, it can move onto the next platform or collect the targets. By doing it repeatedly, the character becomes able to follow the found path.

V. Method to Solve the Sub-Problem of Selecting the Character's Actions

This section defines the sub-problem of selecting the character's actions to follow the path which is found by the method proposed in the previous section, and proposes a method to solve it.

A. Learning problem

As explained in Section IV, the following information is obtained:

- Position in which the character jumps or falls,
- Horizontal velocity with which it jumps or falls.

If the character takes its actions which reach the position and velocity, it can follow the path. Therefore, the sub-problem of selecting the character's actions can be defined as an easy problem whose objective is to reach the target position with the target velocity on a single platform. This problem does not depend on layouts of levels because things to be considered are only the target position and the target velocity.

B. Application of reinforcement learning

This subsection proposes a reinforcement learning method based on Q-learning [1], which is a typical reinforcement learning method, to solve the problem defined in the previous subsection. We explain Q-learning briefly and then the proposed method.

1) Q-learning: In Q-learning, optimal actions are found by using an action-value function $Q(s, a)$ for all pairs of state s and action a. When the character takes action a in state s, it updates $Q(s, a)$ by

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \quad (6)$$

where s' and a' are the next state and the action in s' respectively, r is the reward which it gains for action a, and α and γ are the learning rate parameter and discount rate parameter respectively. In order to apply Q-learning, the states, the actions, and the rewards must be defined appropriately.

2) Proposed reinforcement learning method: Here we propose a method based on Q-learning to solve the problem of selecting the character's actions, given a target position and a target horizontal velocity. Because the optimal actions depend on the target velocity, the character learns independently for multiple problems with various target velocities.

The actions, the states and the rewards are defined by using the fact that the movements of the character are symmetrical. Let us consider two cases A and B shown in Fig. 4, where v is the character's velocity and d is the relative distance from the character to the target position. The two velocities in these

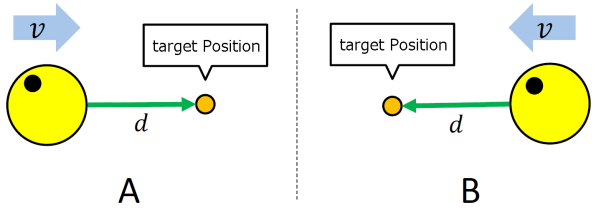

Fig. 4. Symmetrical character's movements

cases have the same absolute value and opposite directions, and so do the two relative distances. Since the character's movements are symmetrical, the action-value of rolling in a direction in the case A is the same as the action-value of rolling in the opposite direction in the case B. Hence, the learning speed is accelerated by defining the actions and the states in such a way that the optimal actions in the two cases can be learned simultaneously by the training for only one of the cases. In addition, for any target position, if the relative distance is the same, the action-value for each action is also the same. Hence, by defining the states as the relative distance instead of the character's position, the optimal actions for any target position can be learned by the learning only for a specific target position. Following this idea, the states are defined as follows:

- Relative distance d from the character to the target position,
- Velocity v of the character.

These can be obtained from Geometry Friends.

The actions are defined as follows:

- Roll in the positive direction,
- Roll in the negative direction.

The positive direction is defined as the direction of the target velocity and the negative direction as the opposite direction. For example, if the direction of the target velocity is right, "Roll in the positive direction" is "Roll to the right".

Because d and v have various values, defining the states straightforwardly from these values may reduce the learning speed. To avoid this, d and v are discretized, and the discretized d and v are used as the states. They are denoted as d' and v' and $d' = 0$, $v' = 0$ when $d = 0$, $v = 0$. The discretized target velocity is denoted as v_t.

It is confirmed empirically that if the character is within a distance D from the target position, it can reach the target position with the target velocity without going D away from the target position. Hence, we make it roll to the target position when $|d| \geq D$. By this, the learning speed is accelerated because there is no need to learn the optimal actions for the states where $|d| \geq D$.

Finally, the rewards are defined as follows:

- $+200$ when $d' = 0$ and $v' = v_t$,
- -1 in the other states.

VI. Experimental Results and Discussion

In this section, experiments of applying our proposed method are conducted. It is shown from the experimental

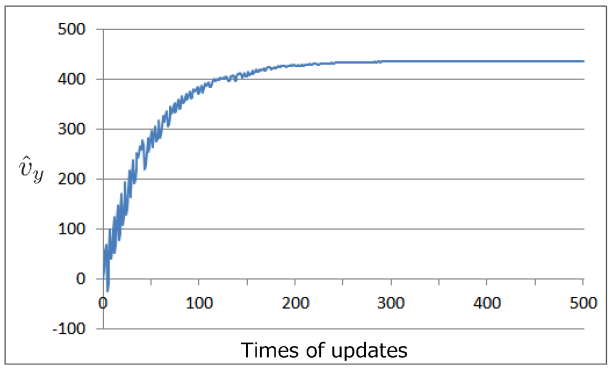

Fig. 5. The transition of $\hat{v}_y$

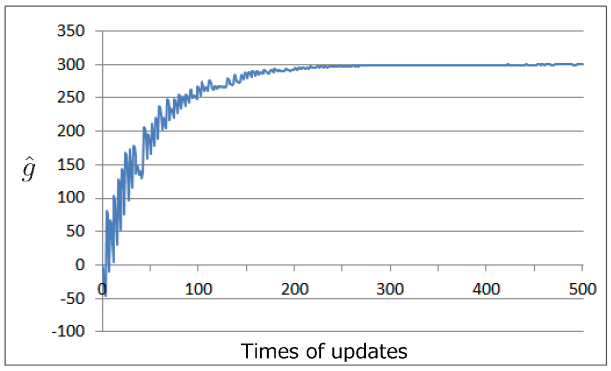

Fig. 6. The transition of $\hat{g}$

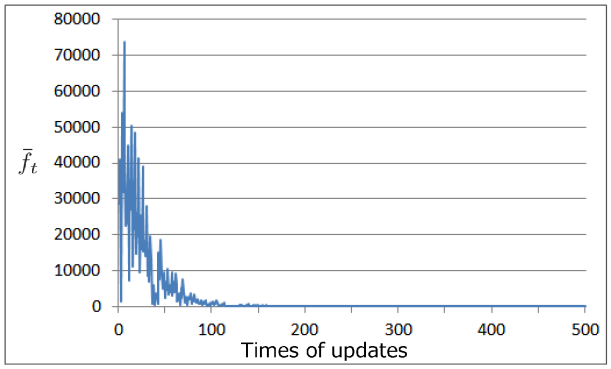

Fig. 7. The transition of $\bar{f}_t$

results that it can solve the two sub-problems and realize effective movements of the character in levels used in a past competition.

A. Performance evaluation of the method to find the shortest path in the directed graph

In this subsection, the method proposed in Section IV is applied to levels, and its performance is evaluated.

First, the performance of the method to predict trajectories of the jumping or falling character is evaluated. In the stochastic gradient descent, we set the initial value of $\hat{v}_y$ and $\hat{g}$ as 0, the learning rate parameter α as 0.1 and the number of times of updates as 500. Figs. 5, 6 and 7 show the transition of $\hat{v}_y$, $\hat{g}$ and $\bar{f}_t$ respectively. In the figures, the y-axis represents $\hat{v}_y$, $\hat{g}$ or $\bar{f}_t$ and the x-axis represents the times of the updates. Because $\bar{f}_t$ in (4) converges on 0 in Fig. 7, the objective function f in (3) becomes 0. In addition, v_y and g also converge in Figs. 5 and 6. Hence, it is confirmed that the trajectories of the character are predicted correctly.

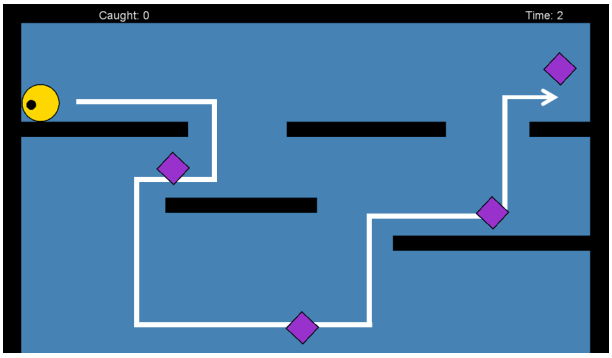

Fig. 8. Found path

TABLE I
THE SETTING OF THE PARAMETERS

Parameter	α	γ	ϵ	D
Value	0.3	0.999	0.6	200

Secondly, platforms the character can move from each platform to and targets it can collect are determined. This determination is made by using the predicted trajectories of the character which jumps or falls from each of several positions with each of several horizontal velocities, as explained in Subsection IV-C. In this experiment, the character jumps or falls from positions at intervals of length 16. The parameters V_{max} and C on the velocities are respectively set to 180 and 20. Hence the velocities are given as $-180, -160, \cdots, 0, \cdots, 160, 180$.

Finally, the directed graph for each level is generated, and Subgoal A* is applied to the graph. Fig. 8 shows a typical path found. In the figure, the white line shows the found path, and it seems to be shortest. Similar paths are found also in other levels. However, in a particular level, there is a problem that some edges of the graph generated are not drawn although they are needed to collect all the targets. This is because we assume that once colliding with an obstacle, the character simply falls straight down. Except for the problem, our proposed method can find the shortest path.

B. Performance evaluation of the proposed reinforcement learning method to select the character's actions

The method proposed in Section V is applied to learn the optimal actions to reach a target position with a target horizontal velocity. While training, the character selects an action according to the ϵ-greedy method [1], that is, the action having the largest action-value with the possibility $1 - \epsilon$ and a random action with the possibility ϵ. The parameters used in the learning method are set as shown in Table I. The level used for the training is shown in Fig. 9. In the level, the width

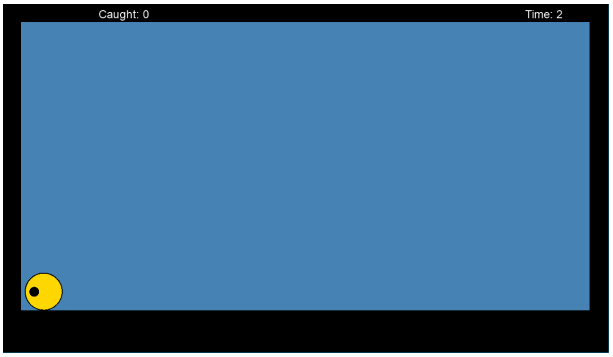

Fig. 9. The level used for the training and the evaluation

is 1200 and the target position is the middle of the platform at the bottom row. The training is performed 5000 times for each target velocity.

After the proposed learning method is performed, in order to evaluate its performance, new 25 levels are generated by giving the initial position of the character, the target position and the target velocity randomly for the level shown in Fig. 9. The character moves according to the learned actions for each level, and the time which is needed to reach the target position with the target velocity is measured.

The character's actions obtained by our proposed method are compared with those by two other methods. In one of them, the character's actions are selected randomly. In the other, actions of rolling to the target position are selected, which makes the character reach the target position in the shortest time. Note that the velocity of the character by the latter method cannot reach the target velocity for most levels, and therefore the method cannot achieve the original objective of learning. The time by this method is used as the lower bound.

The results by the three methods are shown in Fig. 10. In this figure, the x-axis represents each level and the y-axis represents the time which is needed to reach the target position with the target velocity. The blue line and the red line represent the time by our proposed method and random

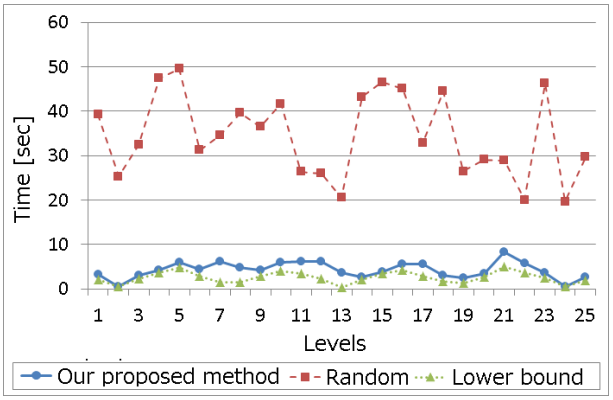

Fig. 10. The time needed to reach the target position with the target velocity

actions respectively. The green line represents the time by rolling to the target position, and means the lower bound. In any level, the time by our proposed method is shorter than that by selecting actions at random, and is almost the same as the lower bound. Therefore, our proposed method can find the optimal actions to reach any target position with any target velocity.

C. Performance evaluation of the whole proposed method

In this subsection, an AI player is designed by our proposed method, and the AI player is made to play Geometry Friends for ten levels used in a past competition. Then, the result of playing is compared with the results by two other existing AI players, Agent α [7] and CIBot [10], to show that our proposed method improves the generalization ability. Agent α is designed by the existing method based on reinforcement learning, and CIBot won first place in the competition held at IEEE Conference on Computational Intelligence and Games in 2014. In the experiments, the levels in the competition are used.

Tables II, III and IV show the results for Agent α, CIBot and our AI player respectively, and the data in Tables II and III are cited from web pages of Geometry Friends [6]. Each AI player plays each level ten times, and "Runs Completed" represents the number of levels for which the AI player successfully collected all the targets. "Diamonds" represents the number of targets the AI player collected, and the number is the mean for the ten trials. The figure in parentheses is the number of targets distributed in the level. "Time[s]" represents the average time the AI player spent collecting all the targets, and the figure in parenthesis represents the time limit of the level. "Score" is used to rank AI players participating in the competition and is the average value of $ScoreRun$ calculated by

$$ScoreRun = V_{Completed} \times \frac{maxTime - agentTime}{maxTime} + (V_{Collect} \times N_{Collect}) \quad (7)$$

where $V_{Collect}$ is the score for one target collected, $N_{Collect}$ is the number of targets collected, $V_{Completed}$ is the score for the completion of collecting all the targets, $agentTime$ is the time the AI player spent collecting the targets and $maxTime$ is the time limit. $V_{Collect}$ and $V_{Completed}$ are set as 100 and 1000 respectively. The winner in the competition is the AI player achieving the highest total score.

Because our AI player achieves a higher score than the other AI players in the tables, we conclude that our proposed method improves the generalization ability. However, the character cannot collect all the targets for only the level shown in Fig. 11. Although it must move from the upper right platform to the upper left one in this level, our proposed method cannot plan the movement because of the problem explained in Subsection VI-A.

VII. Conclusion

In this paper, we have proposed a method based on reinforcement learning to improve the generalization ability

TABLE II
RESULT BY AGENT α

Level	Runs Completed	Diamonds	Time[s]	Score
1	8	1.5(2)	13.0(20)	495
2	0	1.6(3)	45.0(45)	160
3	0	1.0(3)	60.0(60)	100
4	0	2.0(4)	80.0(80)	200
5	2	1.2(2)	65.7(70)	182
6	6	1.4(2)	22.8(40)	574
7	1	0.8(3)	56.5(60)	138
8	3	2.3(3)	38.9(40)	257
9	2	2.2(3)	75.6(80)	275
10	0	0.6(3)	100.0(100)	60
Total Score				2441

TABLE III
RESULT BY CIBOT

Level	Runs Completed	Diamonds	Time[s]	Score
1	10	2(2)	12.7(20)	567
2	10	3(3)	19.9(45)	858
3	10	3(3)	14.8(60)	1053
4	0	1.2(4)	80.0(80)	120
5	0	1(2)	70.0(70)	100
6	0	1(2)	40.0(40)	100
7	10	3(3)	26.2(60)	864
8	0	0(3)	40.0(40)	0
9	10	3(3)	50.0(80)	675
10	0	0(3)	100.0(100)	0
Total Score				4337

TABLE IV
RESULT BY OUR AI PLAYER

Level	Runs Completed	Diamonds	Time[s]	Score
1	10	2(2)	11.4(20)	630
2	10	3(3)	21.4(45)	824
3	10	3(3)	25.4(60)	877
4	10	4(4)	21.4(80)	1133
5	10	2(2)	24.8(70)	846
6	10	2(2)	13(40)	875
7	10	3(3)	19.4(60)	977
8	10	3(3)	28.6(40)	585
9	10	3(3)	31.4(80)	908
10	0	2(3)	100.0(100)	200
Total Score				7853

for Geometry Friends. In our proposed method, the overall problem that the character can collect all the targets in any level is divided into two sub-problems: one of finding the shortest path to collect all the targets and the other of selecting the character's actions to follow the found path. Then, we have proposed a method to solve each sub-problem. It is confirmed from the experimental results for a past competition that it improves the generalization ability.

Our proposed method might be developed to improve the generalization ability for levels which have the same features as the levels used in the past competitions. Therefore, it possibly fails to collect all the targets in levels which have other features. For example, in the level shown in Fig. 12, the character must move to the small platform, stop on it, and then jump to collect the target. However, because our proposed method does not take into account the fact that the character slips in landing on a platform, the character possibly slips down from the small platform and cannot collect the target. Our proposed method has such a limitation in the generalization ability.

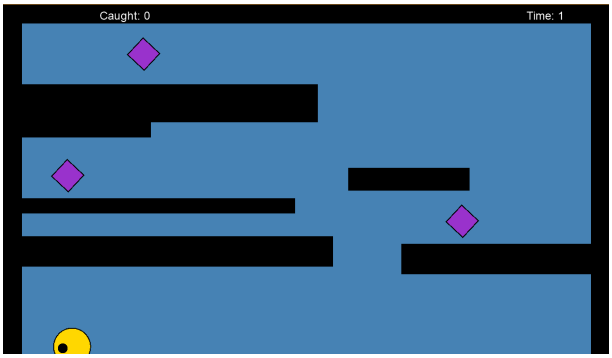

Fig. 11. The level where the character cannot collect all the targets

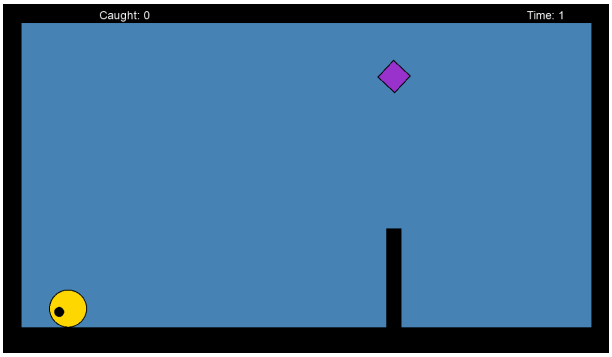

Fig. 12. A level with a small platform

REFERENCES

[1] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, A Bradford Book, 1998.

[2] J. Tsay, C. Chen, and J. Hsu, "Evolving Intelligent Mario Controller by Reinforcement Learning," *Proceedings of 2011 Conference on Technologies and Applications of Artificial Intelligence*, pp. 266-272, 2011.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529-533, 2015.

[4] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen, "Checkers is solved," *Science*, vol. 317, no. 5844, pp. 1518-1522, 2007.

[5] M. Bowling, N. Burch, M. Johanson, O. Tammelin, "Headsup limit hold'em poker is solved," *Science*, vol. 347, no. 6218, pp. 145-149, 2015.

[6] GAIPS INESC-ID laboratory, Geometry Friends — Cooperation Puzzle Game — The Cooperative Agent Competition, <http://gaips.inesc-id.pt/geometryfriends/> (Apr/10/2017).

[7] J. Quitério, R. Prada, F. S. Melo, "A Reinforcement Learning Approach for the Circle Agent of Geometry Friends," *Proceedings of IEEE Conference on Computational Intelligence and Games*, pp. 423-430, 2015.

[8] Y. Lin, L. Wei and T. Chang, "KUAS-IS Lab Technical Report", Geometry Friends Game AI Competition, 2014.

[9] D. Fischer, "Development of Search-Based for the Physics-Based Simulation Game Geometry Friends," Geometry Friends Game AI Competition, 2015.

[10] D. Yoon, K. Kim, "CI Bot Circle Technical Report," Geometry Friends Game AI Competition, 2014.