A new approach for Geometry Friends' RRT Agents

Ana Salta, Rui Prada, Francisco Melo
INESC-ID and Instituto Superior Técnico, University of Lisbon
Oeiras, Portugal
{anasalta,rui.prada}@tecnico.ulisboa.pt,
fmelo@inesc-id.pt

***Abstract*—In this document, we present single player our solution the Geometry Friends problem. A description of the used algorithm and baseline is presented,as well as the modifications and additions made to improve the agent. We then show the results of our tests that prove that our agenst are capable of solving more levels than the available agents.**

***Index Terms*—Geometry Friends, artificial agent, Rapidly-Exploring Random Trees, motion control, replanning**

I. Introduction

Intelligent agents and Multi-Agent Systems (MAS) are becoming more and more popular as they simplify tasks from our daily activities, the industry world, scientific research and so forth.

Geometry Friends [1] is a problem designed to provide new tools to help the development of cooperation as well as motion planning and control in the Artificial Intelligence (AI) field. Geometry Friends is a cooperative physics-based puzzle and platform game for one and, initially, two players, where the main goal is to control the characters to gather all the diamond collectibles present in each level. A competition is taken annually that tests the submitted agents with public and private levels, to understand whether these agents are capable of solving the problems without overspecialization.

Here we present a new solution for both agents that can solve the single player levels of the game, with the prospect of developing a cooperative version of the agents. We used the described solution developed by Soares et al. [2] as a baseline for it has shown good results in previous competitions.

In section II we start to explain the base algorithm used for our agent as well as the agents used as a baseline. We then present our solution in section III, where we explain all the modifications and additions made to improve the agents. In sections IV it is described how the tests were chosen and the data was collected, and section IV presents the results. We then take our conclusions in section VI.

II. Rapidly-Exploring Random Trees

A. Algorithm

The Rapidly-Exploring Random Trees (RRT) algorithm [3] was introduced to solve the problems of the other randomized approaches, like the nonuniform coverage of the state space and the planning not being suitable for nonholonomic or kino-dynamic problems. The algorithm is presented in Algorithm 1, where x_{init} is an initial state and $\mathcal{T}$ is a RRT with up to K vertices.

Algorithm 1 Rapidly-Exploring Random Trees [4]

```
1:  function BUILD_RRT(x_init, K)
2:      T.INIT(x_init);
3:      for k = 1 to K do
4:          x_rand ← RANDOM_STATE();
5:          EXTEND(T, x_rand);
6:      end for
7:      return T;
8:  end function
9:  function EXTEND(T, x)
10:     x_near ← NEAREST_NEIGHBOUR(x, T);
11:     if NEW_STATE(x, x_near, x_new, u_new) then
12:         T.ADD_VERTEX(x_new);
13:         T.ADD_EDGE(x_near, x_new, u_new);
14:         if x_new = x then
15:             return Reached;
16:         else
17:             return Advanced;
18:         end if
19:     end if
20:     return Trapped;
21: end function
```

If the search is limited by K vertices (iterations), for each value of K, a random state, x_{rand}, is selected and extended. The $EXTEND$ function, as illustrated in Fig. 1, chooses the nearest state to the state x, x_{near}, already in the tree $\mathcal{T}$, with the help of a distance metric.

With an input, u_{new}, that can be chosen randomly or by testing all possibilities, the function NEW_STATE is an attempt to move towards the state x from the state x_{near} by applying the action u_{new} to x_{near} and verifying if x_{new} is generated. If x_{new} is the same as x, that state is considered reached, otherwise it is considered as advanced. If the state x_{new} is not possible due to violations of some constraint, it is then considered as trapped.

The goal of the algorithm is to search high-dimensional spaces with algebraic and differential constraints, by biasing the exploration to the unexplored state space and guide the search towards the randomly selected states [4].

B. Baseline

The RRT agents were developed prior to the availability of the game physics simulator. In order to overcome this obstacle,

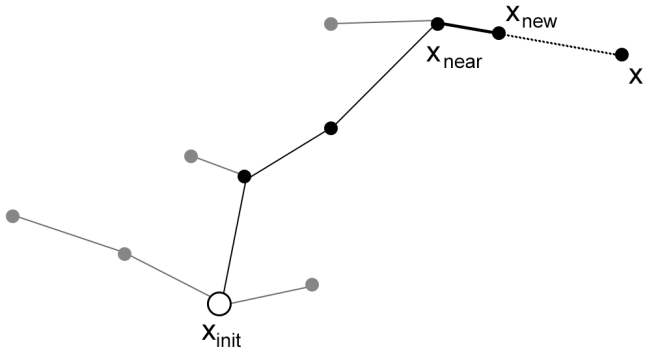

Fig. 1. EXTEND function illustration.

Soares et al. [2] decided to divide the solution in two sub-tasks, the planning and the control.

The planning sub-task uses an adaptation of the RRT algorithm, to search for a possible path. The new states are generated by applying tactics (simple and complex actions) and then checked for their validity. The resulting path is a list of points and their type indicate the actions that should be taken at the given position. There is also a limit of iterations that might produce an incomplete plan, but a plan nonetheless. However, the agent will not attempt to re-plan in order to find a complete plan or correct it when an action fails, and the agent might only end up catching some of the diamonds, or even none, instead of all of them.

The control subtask not only performs the tactics dictated by the plan, but also makes an estimation of variable values like the velocity the agent needs to reach certain points and if it is necessary to perform extra actions in order to complete a tactic. In addition, a standard Proportional-Integral-Derivative (PID) controller is used to correct the velocity in real-time. Certain situations where the agent gets stuck can be detected and resolved with a subroutine that returns the actions needed to revert the situation. These agents have proved to overcome overspecialization and this is the main reason we decided to use this solution as a baseline.

III. Solution

A. Search

1) Improving the algorithm:

Improving the RRT algorithm has already been subject of study. We focused on strategies like biasing the search towards the goal or the region around the goal, the use of Balanced Growth Trees (BGT) [5] when selecting a state, and the use of Skills, Tactics and Plays (STP) when selecting an action.

When searching for the nearest neighbor, the RRT algorithm needs a strong distance metric or it might return not only a state that is not the nearest, but one that is actually far from the selected state. In this case, where physics are applied, this metric is not trivial. The BGT approach tries to avoid this problem by selecting a state which is already in the tree. To do that, it checks the ratio between the average of the leaves depth and the average of the branching factor. If the value is greater than a defined value μ then a random state that is not a leaf is returned, else it returns a random leaf.

Even so, our metric proved to be enough to give better results than using the BGT, tested with different values for μ and using a bias towards the goal, like suggested by LaValle et al. [4]. Defining a valid region around the goal has been difficult but, if achieved, it might give even better results.

The STP approach consists in selecting the action to test using the STP strategy. A *Tactic* is a model of an agent behavior that has a Finite-State Machine (FSM) containing *Skills* which return an action given a state. Since a *Play* is a model of a group of agents, it does not apply for the single player context.

Three simple skills for simple cases were worked on: the agent is on the same platform as a diamond with no obstacles between them; the agent is on a higher platform than a selected diamond and has no walls on both sides; the agent is on a lower platform than the a selected diamond. When selecting a diamond, priority is given to those on the same platform, then the highest ones on the highest platforms.

Adding these simple skills has decreased the search time significantly. Other skills might be added in the future, though a lower probability should be given to the more complex ones to avoid overspecialization.

Since trees in this context can be infinite, we try to avoid the creation of repeated states. We do this by creating a matrix of positions $\times$ diamonds caught, and each time a new state is created, it is only added to the tree if it is not already in the matrix. The size of the matrix is dynamic, and the search starts with a smaller number of possible positions to avoid the creation of very close states. If a search is finished without a solution, a bigger matrix is created and the search continued.

2) Incomplete plans:

Using the game simulator is time-consuming and sometimes an incomplete plan is needed to avoid reaching the time limit without a solution or time to perform it. However, these plans might make the level impossible to solve so they should be used carefully.

Firstly, we find it is safe to catch any diamond that is in the same platform as the agent with no obstacles in between, so, when a plan is found for that diamond, the agent performs it almost right away. The agent is given 20 seconds to find a complete solution. While at it, the agent saves the best incomplete plan so far. To see if an incomplete plan is better than the saved one it has to follow some priorities: have more diamonds caught; if equal number of diamonds, the plan must have higher diamonds on higher platforms; if this number is also equal, the number of nodes in the path must be shorter. If the 20 seconds pass and no complete solution is found, the incomplete plan is followed, have one been found.

3) Simulator:

The RRT agents were developed by Soares et al. [2] prior to the availability of the game physics simulator, and thus, the authors created their own simulator. Since the game simulator is now available, we decided to use it in our approach.

4) Replanning:
The main flaw of the RRT agents developed by Soares et al. [2] is that they do not replan when necessary. Our agents are capable of checking if they have failed an action and, before replanning, they try to check if it is possible to continue the plan from the current state. Sometimes the agents keeps failing the same action, so they only try it twice, to avoid getting stuck, and then replan.

B. Control

As mentioned, Soares et al. developed their own simulator for their agents. During this simulation, extra information is saved to be used by the controller. This controller has tactics implemented and also a PID to help guide the agents.

Since we changed the simulator, this controller became not valid for our solution. We tried to keep the PID to help control the agents but they kept failing a great number of actions. We then decided to create a new controller using a different but similar approach.

Our controller starts by comparing the current velocity with the one we want to reach at the next point: if greater, the agent needs to slow down by moving in the opposite direction; if equal, the agent takes no action to try to maintain the velocity; if less, then it calculates the acceleration during the last time-step to estimate the possibility of reaching the desired velocity in the given distance, moving to the same direction if possible or to the opposite one if not.

This controller has proven to be effective though it still fails some tricky actions since position and velocity error margins are needed when evaluating if the agent reached the desired state.

IV. Tests

The competition public levels, were selected for our tests, and then compared the results with the other available agents. We ran each level three times and took the average time, the number of diamonds caught and the number of times the agent completed the level.

V. Results

In the tables I to IV we can see the results of the 2017 public levels. If an agent could not complete the level once, the mean was not calculated.

TABLE I
OUR AGENT

Circle Levels	**Aprox Time (s)**	**Diamonds**	**Completion**
1	11	2/2	3/3
2	-	2/3	1/3
3	60	3/3	3/3
4	22	3/3	3/3
5	-	2/2	2/3
Rectangle Levels	**Aprox Time (s)**	**Diamonds**	**Completion**
1	48	3/3	3/3
2	17	2/2	3/3
3	15	2/2	3/3
4	25	2/2	3/3
5	-	0/2	0/3

TABLE II
RRT AGENT

Circle Levels	**Aprox Time (s)**	**Diamonds**	**Completion**
1	-	1/2	0/3
2	-	2/3	0/3
3	-	2/3	1/3
4	18	3/3	3/3
5	-	0/2	0/3
Rectangle Levels	**Aprox Time (s)**	**Diamonds**	**Completion**
1	breaks	0/3	0/3
2	breaks	0/2	0/3
3	breaks	0/2	0/3
4	breaks	0/2	0/3
5	23	2/2	3/3

TABLE III
RL AGENT

Circle Levels	**Aprox Time (s)**	**Diamonds**	**Completion**
1	-	1/2	0/3
2	-	2/3	0/3
3	48	3/3	3/3
4	20	3/3	3/3
5	-	1/2	0/3

TABLE IV
SUBGOAL A* AGENT

Rectangle Levels	**Aprox Time (s)**	**Diamonds**	**Completion**
1	-	0/3	0/3
2	-	1/2	0/3
3	-	1/2	0/3
4	-	0/2	0/3
5	16	2/2	3/3

VI. Conclusion

After analysing the results, we can conclude that our agent is capable of solving more levels even if it is not the fastest when other agents are capable of solving the same level. Improvements to the controller should be considered, in order to make our agent faster.

References

[1] R. Prada, P. Lopes, J. Catarino, J. Quitério and F. Melo, "The geometry friends game AI competition," 2015 IEEE Conference on Computational Intelligence and Games, CIG 2015 - Proceedings, 2015, pp. 431–438.

[2] R. Soares, F. Leal, Francisco, R. Prada and F. Melo, "Rapidly-Exploring Random Tree approach for Geometry Friends," Proceedings of 1st International Joint Conference of DiGRA and FDG, 2016.

[3] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," Tech. Rep. TR 98-11, 1998.

[4] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," 4th Workshop on Algorithmic and Computational Robotics: New Directions, pp. 293–308, 2000.

[5] S. Zickler and M. Veloso, "Efficient physics-based planning: sampling search via non-deterministic tactics and skills," The 8th International Conference on Autonomous Agents and Multiagent Systems, pp. 27–34, 2009.