Rule Base Cooperation in Geometry Friends

João Rafael Soares
joao.rafael.pinto.soares@tecnico.ulisboa.pt

Pedro Rodrigues
pedro.p.rodrigues@tecnico.ulisboa.pt

***Abstract*— This is a report of the work done towards a Rule Based implementation of cooperative agents for the Geometry Friends game, including improvements on previous submissions, a general overview of the problems, the implementation and future work**

I. Improvements upon previews implementations

A. Overview

Our work was based on the RRT Agents 2017 baseline available on the competition website. This baseline comes with working Singleplayer agents for both the Circle Agent and the Rectangle Agent. Although functional, they still have their fair share of problems, mainly regarding the Simulator used to simulate the game physics.

B. Simulators

The first change to the code was to refactor the existing RRT code in order to easily swap out game Simulators. This way, as long as the new Simulator implementation follows the available interface, it can be integrated into the existing code.

1) Circle Simulator: Using a modified version of the Simulator for the Circle Agent, we greatly reduced the path search times compared to the previews version. This was achieved by using a simplified physics simulator, avoiding the use of the simulator integrated into the game at the cost of minor precision loss.

2) Rectangle Agent: As no Simulator was available for the Rectangle Agent, we used the built-in Action Simulator for the planning. Therefore, we have different code for this agent's simulator (as opposed to the refactored one).

C. Removal of Plan Recovery

The simplified physics leads to miss-planning, as some moves (mainly jumps) are not well calculated. This, in combination with Plan Recovery, leads to the agents repeating the same faulty plans multiple times before re-planning and fixing the error. With the addition of the faster Simulator, re-planning became faster than recovering, so the plan recovery aspect was removed.

II. Cooperation Problems

We have identified three categories of problems in which each diamond fits in terms of cooperation:

A. Height Problem

When the Circle Agent cannot jump high enough to reach the diamond, nor is there any platform around to jump to in order to get enough height. The solution is for the Rectangle Agent do act as a platform for the Circle Agent.

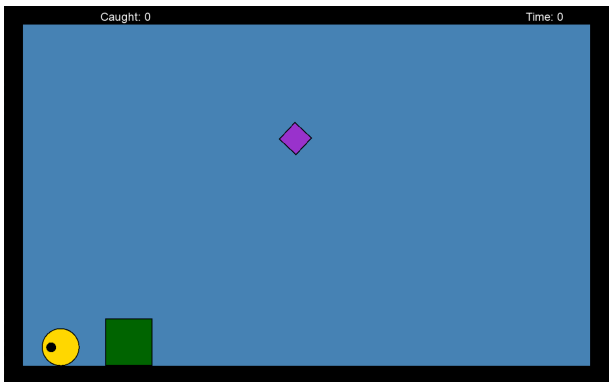

Fig. 1. Example of a Height Problem

B. Tight Space Problem

When a Diamond is in a small gap between platforms, in such manner that the Circle Agent can not fit and the Rectangle Agent does not have the height to reach. The solution is for the Circle Agent to act as a platform for the Rectangle Agent to climb into the gap.

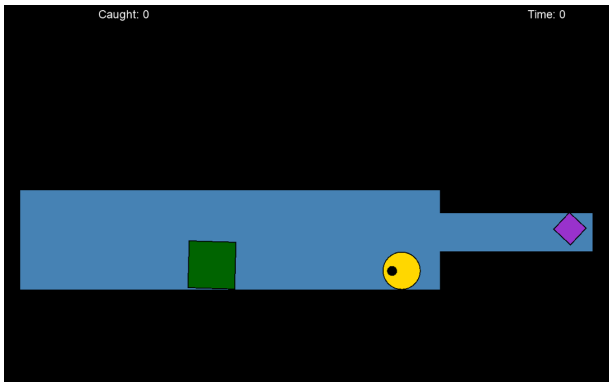

Fig. 2. Example of a Tight Space Problem

C. Agent Specific Platforms Problem

When, to get to a certain diamond (at a height only achievable by the Circle Agent), you have to go through a Circle specific platform. The solution to this problem revolves around the Rectangle Agent carrying the Circle Agent to the required destination.

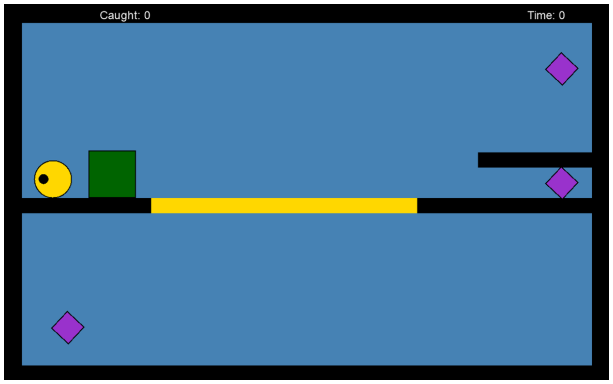

Fig. 3. Example of an Agent Specific Platforms Problem

III. Cooperation Framework

In order to solve the aforementioned Cooperation problems, we developed a Cooperation Framework based on Singleplayer Agents. With this framework, it is possible to use any Singleplayer Agent implementation (as a drop in replacement) to solve cooperative problems. We split the framework in four different sections:

A. Action Rule

For each cooperative problem we create an Action Rule class to represent it. An Action Rule is composed of various Action States which the agent must follow to catch the Diamond, having a different set for the Circle and the Rectangle Agent.

B. Action States

Each Action State represents a section of the problem which must be completed. These can be, for example, to move to a certain place or to perform a certain action. The Action States have the possibility to include an instance of the Singleplayer agent to use for movement and actions.

C. Filter Rules

Filter Rules are used to identify what problem each Diamond present in the level corresponds to, returning the ActionRule which solves that particular problem.

D. Coop Rules

Coop Rules will be the aggregating element of the framework, having all the Filter Rules to be applied, and including a method to return a list of Action Rules given a set of diamonds.

With this implementation, we can easily add new Rules for future Cooperation Problems.

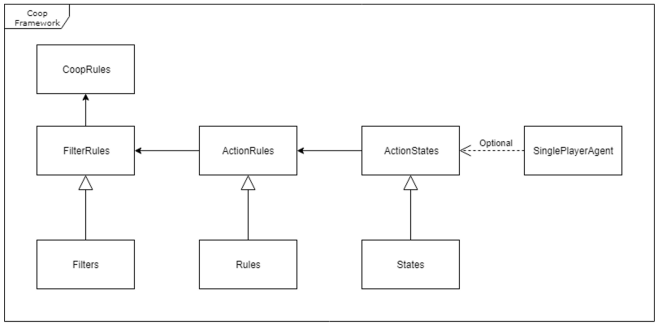

Fig. 4. Schematic of the structure of our framework

IV. Framework Usage

In this section we will expand on how we use the framework within the game.

1) At the start of the level, the Diamonds are filtered by the Filter Rules within Coop Rules, giving us a list of the Action Rules necessary to solve the level;
2) Currently the Action Rules are not ordered and are done sequentially, leading to the

agents not working in parallel and being inefficient;
3) In order to avoid hard-coding movement, we use a technique we call *Fake Diamonds*.

V. Fake Diamonds

Fake Diamonds are Diamonds injected by the Action States into the Singleplayer agents in order to lead the Agents to certain positions. This way, we can use the Singleplayer Agent to move the Agents to the positions required to complete the States. This method does not work with the Rectangle Agent however, due to the fact that the built-in Action Simulator obtains the Diamond information directly from the game, resulting in us being unable to bypass and inject information.

VI. Working with the Cooperation Framework

A. Adding new rules

In order to add new Rules to the existing framework, you have to:

1) Create a new Action Rule, representative of the newly added Rule;
2) Add to the Action Rule the list of desired Action States (or create your own if the state needed currently does not exist);
3) Create a new Filter Rule with the criteria to identify the Diamonds to which the new Action Rule should be applied;
4) Add the newly created Filter Rule to the Coop Rule list;

Warning! Coop Rules chooses the **first** Filter Rule that applies to the diamond, so you must be careful with the Filter Rule order in the list.

B. Switching between Single Player Agents

All that is required is to replace the Agent folder of the framework with your own Agent's code, as long as you have within a CircleSingleplayer and RectangleSingleplayer that implement all the methods in the abstract agents classes (that is, have all the methods required by the game)

C. Switching between Simulators

To switch between Simulators, you need to perform two steps:

- Change the instance of the Simulator in the PlanSolution (and RecoverPlan method) in the Single Player implementation;
- Change the Simulators cast in the RRTUtils class in the ApplyPrediction method, as it needs to know the Simulator type.

VII. Current Problems and Future Improvements

A. Parallel Solving

With our current implementation, Action Rules do not support parallel solving of both Circle and Rectangle Agents. Adding this feature would great reduce the execution time.

B. Rectangle Agent Movement

The Rectangle Agent does not have a fully fledged Simulator for its movement, using the sub-optimal game simulator. Adding a Simulator would greatly improve the Rectangle Agent movement and remove the need for any hard-coded movement.

C. Circle Simulator and Fake Diamonds

Our objective with using Fake Diamonds is to lead the Circle Agent to each position we need to complete the Action State. However, the current implementation of the Circle Simulator only cares about catching the Diamond, running trough it and not stopping on its location, which the Rectangle Agent needs to synchronize with the Circle Agent. Creating or altering a new Simulator for the Circle Agent to stop in the Fake Diamond position would greatly reduce synchronization problems

D. Tight Space and Agent Specific Platforms Problem

This problems do not yet have an implementation, so some courses will not be able to be completed.

E. Recovery Failure

On our current failure recovery technique (for Cooperative Action Rules) we simply add the current Action Rule to the bottom of the list of Action Rules and move on. A more personalized recovery technique for each Action Rule or State could increase the efficiency of the Agents.

F. Diamond Filtering Flexibility

Right now which rule is decided in Filter Rules is dependent in which order they exist in our code. A more flexible approach should be developed for easiness on adding new Filter Rules.