UCMAgent Cooperative Report

Alberto Almagro Sánchez
Complutense University of Madrid
Madrid, Spain
alberalm@ucm.es
contact@alberalm.com

Juan Carlos Llamas Núñez
Complutense University of Madrid
Madrid, Spain
jullamas@ucm.es
jcllamas2000@gmail.com

***Abstract*—This report presents our cooperative agents for the Collaborative Track of the Geometry Friends Competition at International Joint Conferences on Artificial Intelligence (IJCAI) and IEEE International Conference on Games (CoG) 2023.**

***Index Terms*—Artificial Intelligence, planning, physical modeling, expert system, physical environment, multi-agent environment, cooperation.**

I. Introduction

This document aims to give an overview of our solution implemented for the collaborative agents of the Geometry Friends game. The strategy followed by the UCMAgents to solve the levels is divided into three phases: representation of the level, action planning and execution of the plan. We will illustrate this strategy mostly using the level shown in Fig. 1 as an example.

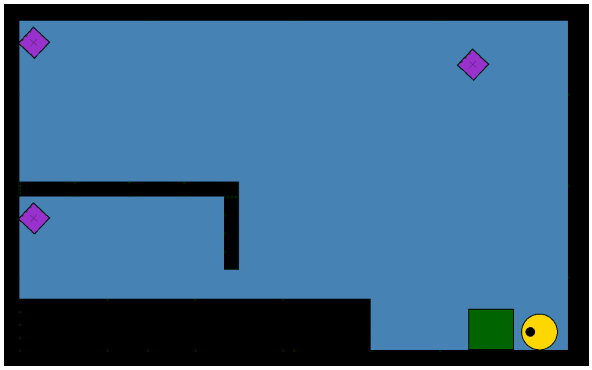

Fig. 1. Cooperative level example.

It is important to mention first that we have used an object, which we call *SetupMaker*, that collects all the information generated by our agents, and that acts as a communication channel for our agents, having many useful variables that change how our agents behave.

II. Representation of the levels

To make an adequate representation of each level, the steps our agents follow can be summarized as the identification of the platforms of the level, the generation of movements that indicate connections between them and a filtering that allows to keep a minimal set composed of the best movements. In the following, we will discuss each of these stages. It is recommended to understand the individual behaviour of each agent by reading the corresponding reports, since here we will focus on the cooperative layer we have added to them.

A. Platform identification

To process each level, each of the agents works independently, following the same steps as in their individual track and thus generating their own platforms. However, to capture the new possibilities offered by cooperation, it is also necessary to include new cooperative platforms.

The most basic form of collaboration is for the circle to land on the rectangle and use it to reach higher heights. Therefore, on each of the rectangle's small platforms, we evaluate if the circle could rest on each of the rectangle shapes admissible on that platform. In Fig. 2, some of the different situations that our agents may encounter during this process are represented.

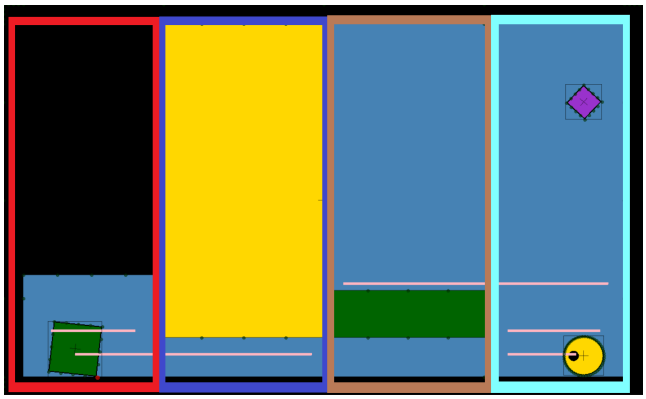

Fig. 2. Cooperative platform identification.

On the left hand side of the picture (red area), the circle can rest on the rectangle if its shape is either square or horizontal, but not vertical, so only two cooperative platforms are generated. In the blue area, the rectangle can only rest in horizontal position and the circle can rest on it because it can go throw yellow obstacles, so only one cooperative platform is generated. Analogously, in the brown area, the rectangle can morph to all three of its basic shapes, but, as the circle collides with green obstacles, the only cooperative platform that is possible is the vertical one. Finally, on the right-hand side of the level (cyan area), the rectangle can rest in all of its shapes and the circle is always able to pose on it.

After this identification, we also identify simplified cooperative platforms, which represent the idea of connection through

the altering of the rectangle's shape. For example, in Fig. 2 all the cooperative platforms in the red and blue areas would be merged into a single simplified cooperative platform, and the same would apply to the ones in the brown and cyan areas. This process is much harder that what it may seem at first glance, and there are particular cases that need to be taken into account when joining these platforms.

B. Move generation

Once platforms have been identified, and in order to reflect the agents' abilities to change platforms or reach diamonds, we present the new cooperative moves, which represent sequences of atomic actions that let both agents achieve one or both of the above high-level objectives. We also have to make some adjustments to the moves generated by our agent's individual abilities.

In the case of the circle move, it is necessary to consider the parabolic trajectories of jumps and falls that start or end on collaborative platforms. In addition, we have to run the simulations until landing on a non-collaborative platform, potentially returning more than one parabola for each velocity and initial position. This is because collaborative platforms may not be obstacles if the rectangle is not in the proper position and therefore we have to consider all the possibilities. For the circle, we have also added a new collaborative move: the (cooperative) ADJACENT move. This movement represents the situation where the rectangle performs an ADJACENT movement while the circle is mounted on it and we want the circle to continue to rest on the rectangle while the last is sliding through the hole.

In the case of the rectangle, we add a brand new move to represent a new type of collaboration. Until now, we have only considered that the agent that provides help is the rectangle, letting the circle rest on it. However, our agents may face other situations that need cooperation. This is the case of what we call CIRCLETILT moves, in which the circle is the one who helps the rectangle. Specifically, the circle helps the rectangle perform a TILT where the landing platform is higher than usual. This is definitely useful, as the rectangle may be the only agent able to get some collectibles due to the color of certain obstacles of the level, like in the level of Fig. 3, and it may need circle's help to climb to a platform that it wouldn't reach on its own.

C. Move filtering

As in the previous phase a large number of movements have been generated, many more than desired, it is necessary to perform a filtering in order to keep a reduced subset, which guarantees the maximum connectivity between platforms. To do this, a similar process to the one carried out for the individual agents is performed. A new situation is a circle move which takes off and lands in different simplified cooperative platforms. These moves are deleted, because we cannot guarantee that the rectangle can reach the circle's landing point in time. The main novelty is that we prioritize landing and taking off in collaborative platforms when the rectangle's

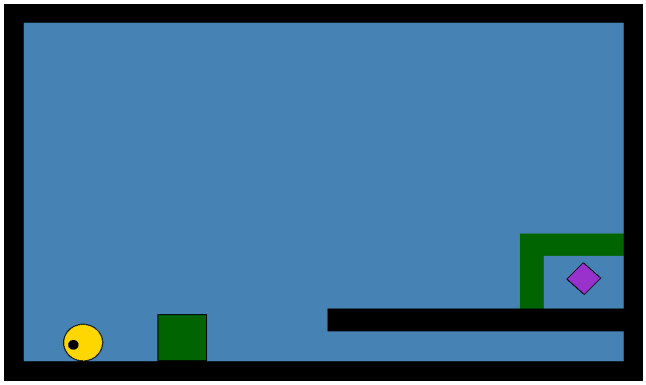

Fig. 3. Level in which a CIRCLETILT move is needed.

shape is horizontal rather than square and vertical due to its stability. Aside from this, the filtering criteria are pretty much the same as for our individual agents.

At the end of this stage, we would have a similar representation as the one shown in Fig. 4 or the simplified graph version in Fig. 5. We can see the different platforms and the connections between them with the most suitable moves left after the filtering. As for our individual agents, each of these moves has characteristic information that identifies it and that is deduced from the simulation. This information includes the move type, the starting point, the landing point, the initial speed, the origin and destination platforms and the diamonds captured, among others.

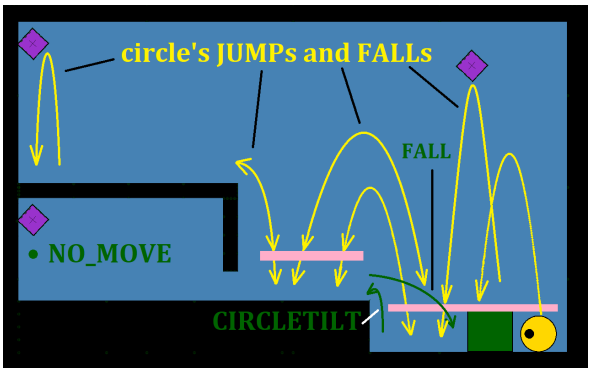

Fig. 4. Move generation and filtering.

III. Action planning

In this section, we are going to explain how to draw up a cooperative plan. It must be kept in mind that we have to make a plan for each of the agents and that, in the execution phase, its interpretation will be analogous to that of the individual plans of the rectangle and the circle. That is, first of all, each character reaches all available diamonds using moves that end on its current platform and, when there are no more left, it uses the first move of the plan to change platform and the process is repeated again. In the case of cooperation, we must take into account that there are moves by one of the characters that require the help of the other character, for which it must

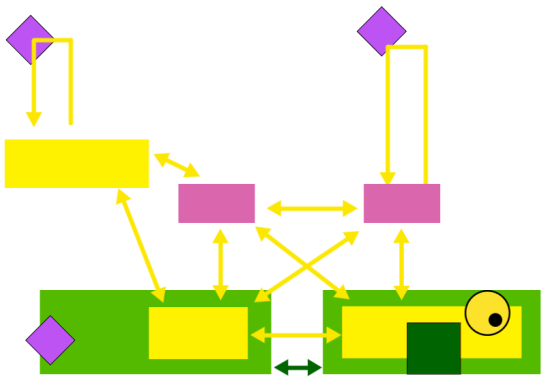

Fig. 5. Graph representation of a cooperative level: the light green (yellow, pink) blocks represent the rectangle (circle, cooperative) platforms and the dark green (yellow) arrows, the rectangle (circle) moves between them.

be on an appropriate platform. Likewise, it has to be possible to include waiting moves within the same platform while the other agent may be reaching a diamond in another area of the level.

The plan is generated by performing a time-limited breadth-first search with repetition control over a graph whose nodes are composed by the two agents' current plans and the diamonds collected so far. If the search time expires, which is unusual because the graph is usually small in size, the best plan so far, i.e. the one that captures the highest number of diamonds, is returned. Our planning algorithm also takes into account real-time replanning and alternative paths (if they exist) to avoid moves that were proved to be difficult to execute and failed.

We will now explain how the nodes are expanded, since it is not as straightforward as in the case of the individual agents. For this purpose, given a node, we first add the diamonds that are directly collected without leaving the platform and then we consider the lists of movements starting from the circle's and rectangle's platforms. To these lists, we add an auxiliary movement to stay on the same platform, which will serve us to model both the situation in which one agent must wait for the other, and the assistance in a hypothetical cooperative action. With these considerations, the expanded nodes are formed considering all possible *compatible* combinations in which each agent takes one movement from its move list. In this way, the new node will be the same as before, except that the current plans will have one more step and the diamonds collected will be updated. It only remains to clarify when a movement of the circle and a movement of the rectangle are compatible and when they are not.

First of all, we do not want to expand nodes in which both agents remain on the same platform, so the simultaneous choice of auxiliary movements is not compatible. Similarly, movements that are neither waiting nor cooperative and that end in the same platform from which they start are discarded, because the auxiliary movements already capture that behavior. The other situations in which a combination of moves may not be compatible are given by the constraints imposed by collaboration. When the rectangle movement has type CIRCLETILT, in order to expand the node, the circle needs to be on the same platform to help the rectangle perform the move. Something similar happens with the jumps or falls of the circle from or to collaborative platforms, i.e., movements of the circle initiated on the rectangle or intended to land on the rectangle. These movements will only be compatible if the rectangle is on the appropriate platform and the movement it chooses is the cooperative auxiliary movement. Finally, the circle can execute an ADJACENT type movement only if the rectangle also performs an ADJACENT type movement and in the same direction.

Although it may seem that the combination of moves may explode, making the search impractical, the truth is that our algorithm has given satisfactory results at all levels where it has been tested. The fact that there are many constraints of incompatibility between moves and that the plans are usually not longer than 4 steps guarantees the completion of the search in the time limit we impose.

To make the argument more visual, let's take a look at Fig. 6. In this case, the possible moves available for the circle are to jump on the rectangle (pink cooperative platform), to jump to the platform on the left or to jump to the pink cooperative platform on the left. To this list, the auxiliary move is appended. As for the rectangle, it can only perform a CIRCLETILT or an auxiliary move. Initially, $4 \times 2 = 8$ combinations should be considered, but there are many restrictions. In the end, from the 8 theoretical combinations of move, only 3 are actually compatible (the rectangle's CIRCLETILT, the circle's JUMP_MOVE to get over the rectangle and the circle's JUMP_MOVE to move to the center platform), making the ramification factor much lower than what it may seem.

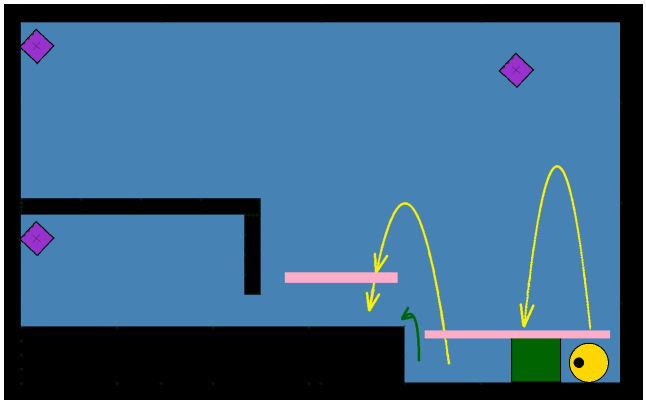

Fig. 6. Possible moves for the rectangle and circle from the initial platform.

IV. Plan Execution

Once our agents have a plan to follow, the last step to solve a level is to perform the atomic actions necessary to complete it. The execution of the actions is very similar to the one performed in the individual modes, so we are only going to focus on the new features of cooperation.

First of all, we start with the CIRCLETILT move, and we assume that the rectangle and the circle are on the same platform. First, the circle is told to go to the edge of the platform where it should be positioned to act as a support for the rectangle, next to the rectangle's target platform and with speed 0. Meanwhile, the rectangle is notified to go to the opposite end of the platform (so that both agents do not get in each others way and it has enough space to accelerate) and wait for the circle to signal that it is ready to make the move. When the circle is in the appropriate position, it sends a signal to the rectangle, which, upon receiving it, moves with maximum speed towards the point where the circle is located and maintains the action until it lands on the target platform. Additionally, when the rectangle is flying over the circle, the latter jumps to give it a small extra boost and accelerate the process. The only inconvenience that arises in these movements is when the rectangle is located between the end of where the CIRCLETILT must be performed and the position of the circle, since, in that case, both agents must exchange their positions. We will deal with these position exchanges in a more general context shortly.

The next cooperative execution challenge arises when the circle has to land on a collaborative platform, i.e. it has to land on top of the rectangle. To do this, before starting the jump or the fall, the circle must wait to receive a signal informing it that the rectangle is ready to serve as a landing platform, i.e. it is in the place where the circle will land. What the rectangle must do is to go to that landing position, which was already calculated during the move generation, and position itself in the appropriate shape to receive the circle (the horizontal was prioritized whenever possible because it was more stable). When it has reached this position with speed 0, it will activate a signal that will tell the circle that it is ready to serve as a platform and that it can perform the jump or fall in the same way as it would do in the individual case. An improvement of the system is that, during the circle's flight time, the rectangle recalculates the landing position with the parabolic trajectory simulator used in the move generator. This way, the predicted landing position of the circle is always more and more accurate. However, even if the rectangle is in the right position, it does not guarantee that the landing will be successful. To multiply the chances of landing consolidation, our rectangle has a system so that, from the first bounce resulting from the landing, it follows the circle for a short period of time so that the circle does not fall out of the rectangle.

V. Further improvements

Aside from the basic behaviour we discussed in the previous section, we have needed to implement several systems to improve our agents' performance.

First and most importantly, we needed a system that let our agents to change positions. This is strictly required in many levels, but it is also specially useful for us, since, given a specific platform, our circle usually saves just one trajectory to get over the rectangle. If this trajectory needs speed, the rectangle must be previously located in the approximate landing point. If our agents could not change positions, this would be frequently impossible.

In fact, this feature of saving a single move to get over the rectangle has also required other additional systems. Due to this trajectory needing most of the times a low speed (or even no speed at all), the rectangle can find itself interfering with the circle trajectory's if located on the jump's landing point. Therefore, we needed a way that let our rectangle locate either to the right or to the left of the landing point, and, once the circle is at a high enough height, get to the correct area while the circle is falling.

However, sometimes there is not enough room for our rectangle agent to move neither to the right, neither to the left. This is the only case in which we perform a dynamic move generation that replaces the one the circle originally planned to execute. This new move must not cross the rectangle if it was to be located on the landing point. To illustrate this strategy, consider the level represented in Fig. 7. Normally, the circle would like to execute the trajectory with null velocity. However, if the rectangle tried to move and let some space for this move to be possible, it would fall from the yellow platform. Therefore, our agents need to generate a new move that lets the circle get over the rectangle.

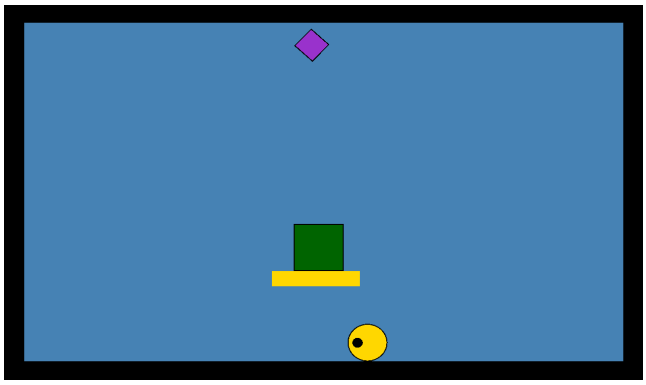

Fig. 7. Level in which our agents need to dynamicly generate a new movement.

Finally, we implemented a recovery system that is used to escape from situations where our agents are both stuck. To do this, they perform random actions for a short period of time and, despite its simplicity, it is a very effective system without which it would be difficult to complete some levels. Our circle now also has a system to help it avoid falling from the rectangle's edges when it is over it, which may otherwise occur when trying to achieve a high speed.

VI. Conclusion

To conclude, we have based our solution for the cooperative agents in our individual solutions, by also identifying and studying different cooperative situations they may face. Our search algorithm, combined with our efficient movement generation and level representation, makes our agents solve almost every level from previous competitions.