# UCMAgent Circle 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 circle agent solution for the Circle 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.**

## I. Introduction

This document aims to give an overview of our solution implemented for the circle agent of the Geometry Friends game. The strategy followed by the UCMAgent 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 using the level shown in Fig. 1 as an example.

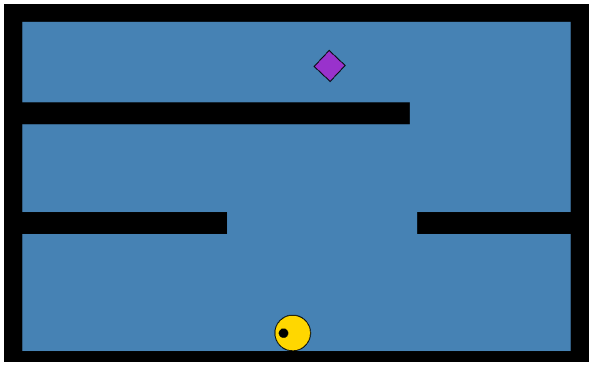

Fig. 1. Circle level example.

## II. Representation of the levels

To make an adequate representation of each level, the steps our agent follows 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.

### A. Platform identification

To approach each level, we first make a discretization of the information that represents the level and that is provided by the game's interface. We perform this simplification by dividing the lengths of the objects by 8 and keeping the integer part. Although it may seem this compromises our agents precision, we concluded after several tests this first step had almost no harmful consequences and thus allowed us to work more comfortably.

Next, we establish the type of element represented in each discretized pixel (either obstacle, diamond or empty) and, from this distinction, we identify the platforms we would use throughout the rest of the process. We refer by platforms to the upper continuous horizontal strips of the obstacles where the circle can rest. By construction, it can be assured that the circle is able to move within the same platform by just taking the ROLL_RIGHT or ROLL_LEFT actions, as can be seen in Fig. 2.

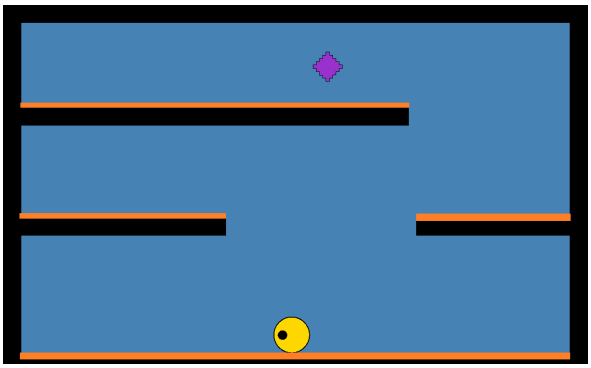

Fig. 2. Platform identification.

### B. Move generation

Once platforms have been identified, and in order to reflect the circle's ability to change platforms or reach diamonds, we introduce the concept of "moves", which represent sequences of atomic actions that achieve one or both of the above high-level objectives. For the circle agent, we consider four types of moves: FALL, JUMP_MOVE, NO_MOVE and CLIMB.

*1) NO_MOVE:* This move represents the possibility of capturing a diamond that is on a platform without the need to jump.

*2) JUMP_MOVE:* It captures the circle's ability to reach a diamond or change platforms by performing a JUMP action. For this purpose, all possible trajectories of the circle, starting from each possible point of each of the platforms with all possible speeds, are simulated. Specifically, by "each possible point", we mean a sufficiently fine discretization of them and by "all possible speeds" we mean a discretization of

them from the minimum velocity (-200) to the maximum velocity (200) with a small step (15). It has been tested that these discretizations are able to maintain a compromise between accuracy and efficiency and are therefore useful. The simulation of each of the trajectories is carried out using the equations of projectile motion

$$\begin{cases} x(\mathtt{t}) = x_0 + v_x \mathtt{t} \\ y(\mathtt{t}) = y_0 + v_y \mathtt{t} - \frac{g\mathtt{t}^2}{2}, \end{cases} \quad (1)$$

where the gravity $g$ and the initial vertical velocity $v_y$ are given by the environment. The parameter $y_0$ is the initial height and varies from one platform to another, the parameter $x_0$ changes within a platform according to the take-off point and the parameter $v_x$ moves in the domain of the initial horizontal velocities. During the trajectory simulations, it is also taken into account both the possible diamonds captured and the collisions with the obstacles scattered throughout the level. A novelty with respect to other agents' implementations from previous years is that we consider multiple rebounds with walls and ceilings, which allows us both to perform more realistic simulations and to execute movements that other agents do not even consider. To evaluate the captured collectibles, we intersect the discretization of the diamonds with an inner discretization of the circle, while to evaluate collisions with obstacles, we intersect the obstacles with an outer discretization, as shown in Fig. 3. This criterion is conservative in the sense that whenever a simulation has no collisions with obstacles, then the real trajectory will not have them. And if the simulation ensures that a diamond is captured, the real trajectory will catch it.

Fig. 3. Outer and inner discretizations of the circle.

*3) FALL:* The FALL movements represent the possibility of the circle falling off the ends of the platforms. Its simulation is very similar to that of the JUMP_MOVE where the initial vertical velocity $v_y$ has to be set to 0 in Equation (1) and consider as $x_0$ just the edges of the platform.

*4) CLIMB:* The last type of move we have implemented is the CLIMB moves. They represent the circle's possibility of climbing small heaps by just rolling over it. An example of these heaps are shown in Fig. 4.

### 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. First, we eliminate those movements that do not change platform and do not reach any diamond. Among the remaining

Fig. 4. Climb move pattern.

ones, when two have the same take-off and landing platforms, we will say that one is better than the other if:

1) Its type is NO_MOVE and the other's type is FALL or JUMP_MOVE.
2) Its type is CLIMB and the other move's type is JUMP_MOVE.
3) It reaches more diamonds.
4) It has a better value in a heuristic function that measures the distance of the trajectory to any obstacle, the space available to take off and land and the horizontal speed, parameters that have a determining influence in the success of the jumps and falls of the circle.

At the end of this stage, we would have a similar representation as the one shown in Fig. 5. We can see the different platforms and the connections between them with the most suitable moves left after the filtering. Each of these moves have characteristic information that identifies them 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.

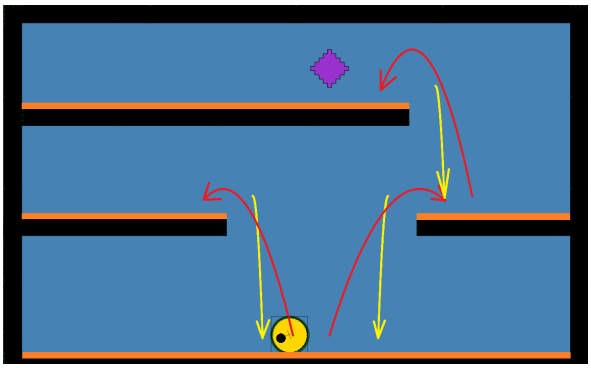

Fig. 5. Move generation and filtering.

## III. Action planning

Once the level representation phase has been completed, we can collect the resulting information in a graph whose nodes are the platforms and whose edges are the moves connecting them. These moves are associated with the diamonds they capture according to the simulation previously performed. The objective of this phase is to generate a high-level plan, i.e., a sequence of moves to change platforms so that all diamonds

are reached. During execution, our strategy involves that, once a platform is reached, all the collectibles of that platform are captured before moving to the next one, and that is why movements within the same platform are omitted.

The plan is generated by performing a time-limited breadth-first search with repetition control over the graph. 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.

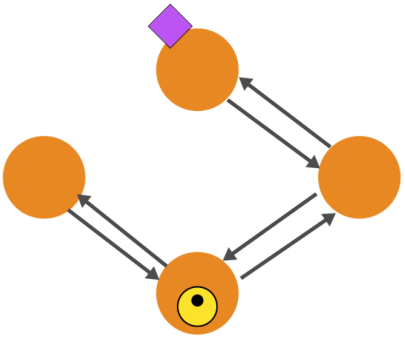

Fig. 6. Graph representation.

## IV. Plan Execution

Once our agent has a plan to follow, the last step to solve a level is to perform the atomic actions necessary to complete it. For this purpose, Algorithm 1 is followed. It is only necessary to clarify what we mean by "best action that leads to a completion of a movement".

When the circle is on a platform and its next move is $m$, it must reach the starting point of $m$ with the appropriate speed. To do this, the circle must know how to reach any point on a platform with a certain speed, given that it is possible, which has been checked during the simulations. That will allow the circle to meet the preconditions of the move which will lead, assuming the simulation was accurate, to catch the diamonds that go through its trajectory and land on the predicted platform.

To solve the problem of reaching a certain point with a certain speed, we first attempted a solution based on reinforcement learning using the Q-Learning technique, similar to that proposed by previous solutions (see MARL-Agent [1]). However, the solution that we finally present is a rule-based system inspired by the motion equations, which allows us to decide optimally the best action to take at each instant. For this purpose, two special points are considered which can be calculated with the data of circle position and velocity and target position and velocity. We call *acceleration point* to the place from which the circle should be positioned, with zero velocity, to reach the take-off point with the appropriate departure velocity if it always takes the action that brings it closer to that take-off point. Also, we call *braking point* to the place where the circle would stop if it always takes the action that opposes the direction of its velocity.

**Algorithm 1** Circle's execution algorithm

```
1:  while Level not finished do
2:      if current platf. != depart. platf. of plan's 1st step then
3:          Replanning
4:      end if
5:      if There are diamonds left on the current platf. then
6:          m ← Move that reaches the nearest diamond
7:      else
8:          if Plan is not empty then
9:              m ← Plan's first move
10:         else
11:             Execute a random action
12:         end if
13:     end if
14:     Execute best action that leads to completion of m
15: end while
```

Considering these two points, an exhaustive analysis of the different cases is carried out according to the relative position of the circle and the target point, the acceleration and braking point and the sign of the circle velocity and the target velocity. To exemplify the rule-based system, let us suppose that we are in a situation like the one shown in the Fig. 7 in which the circle is on the right side of the target point, has a positive velocity and the target velocity is negative. It is not easy to determine at a glance whether the ROLL_RIGHT or ROLL_LEFT action is more appropriate. We should consider the braking and acceleration points and in this case the former is to the left of the latter. If the ROLL_LEFT action was taken at this point, the circle would come to a complete stop at the braking point, but would not have enough space to reach the target point with the required speed. In this case, the most appropriate action is ROLL_RIGHT in order to gain more acceleration space.

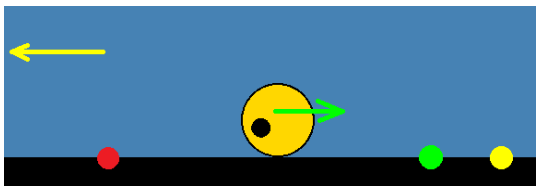

Fig. 7. Representation of the target point (red point), braking point (green point), acceleration point (yellow point), target velocity (yellow arrow) and current velocity (green arrow).

If the braking point was at the right side of the acceleration point (see Fig. 8), the ROLL_RIGHT action would only worsen the situation as the braking point would move even more to the right. So in that case the proper action would be ROLL_LEFT. A thorough study of the possible combinations leads a to a system that covers all situations and solves them the most efficient way.

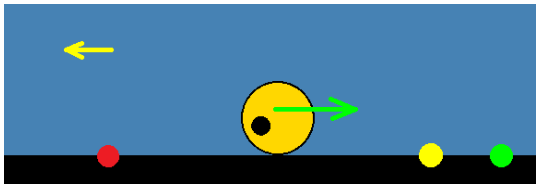

Fig. 8. Representation of the target point (red point), braking point (green point), acceleration point (yellow point), target velocity (yellow arrow) and current velocity (green arrow).

Another of our contributions in the circle execution phase is the jump policy. Although our system of choosing the best action to take is very accurate, sometimes the speed with which the circle reaches the target point is not identical to the target velocity. However, in the vast majority of cases, the trajectory resulting from jumping or falling with small variations in initial position or speed can be just as valid as the simulated one. Therefore, when the circle is very close to the target position, a new runtime simulation is performed to check if said jump or fall would meet the same postconditions in terms of landing platform, captured diamonds and guarantee of not falling over the edges of the landing platform. As we can see in Fig 9, the green parabola, that represents the trajectory of the circle if it jumped now, reaches the same collectibles and lands in the same platform as the red simulated parabola, so it is equally valid.

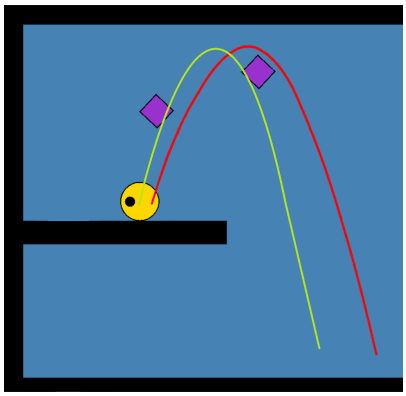

Fig. 9. Alternative trajectory (green) vs. simulated trajectory (red).

This concludes our basic explanation of our agent's behaviour. In the following section, we will point out some minor improvements we have implemented.

## V. Minor Improvements

To complement and refine the above algorithm, we present some of the systems we use to improve the performance of our circle agent.

The first improvement decides the action that the circle takes during its flight. While the circle agent cannot modify its trajectory once it has taken off, it has the ability to change its angular momentum (which is not given by the environment) by taking the ROLL_LEFT or ROLL_RIGHT actions, which can help to make a successful landing. In most cases, the main risk that the circle faces is to slip out of the platform due to a lack of space to brake. In that case, the action preformed during the flight should be to roll in the opposite direction of its velocity to lose momentum during the landing. However, there are some particular cases in which gaining momentum is desired. For instance, if the landing platform is too high and the landing point is predicted to be the closest to the departure point, the agent may need and additional impulse to climb the edge and consolidate the landing. In that case, the circle should roll in the same direction as its velocity to gain momentum, as in Fig. 10.

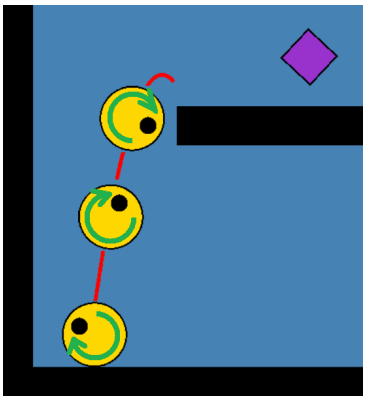

Fig. 10. Action that the circle takes during its flight.

Furthermore, if the circle lands on a platform's edge, it is easy that it rebounds and falls. In those cases, it is also advantageous to gain angular momentum by rolling in its velocity's direction.

Also, a second system was implemented to prevent unwanted falls from the ends of the platforms and represents a significant improvement in the agent's performance, especially on levels with narrow platforms.

Finally, we included a small check so that, if, at any moment, the circle can achieve the last diamonds by jumping, and independently on where it will land, it jumps, even if that was not the intended movement. This is a minor improvement in very particular situations.

## VI. Conclusion

To conclude, we have based our solution for the circle agent in expert knowledge, by identifying and studying different situations the circle may face. Our search algorithm, combined with our efficient movement generation and level representation, makes our circle solve almost every level from previous competitions.

## References

[1] Sequeira, R.A. *Building a Multi-Agent Learning System for Geometry Friends.* Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, 2019.