Rapidly-Exploring Random Tree approach for Geometry Friends

Rui Soares, Francisco Leal, Rui Prada, Francisco Melo
INESC-ID and Instituto Superior Técnico, Universidade de Lisboa
TagusPark, Edifício IST
Porto Salvo, Portugal
{rui.soares,francisco.leal, rui.prada}@tecnico.ulisboa.pt,
fmelo@inesc-id.pt

ABSTRACT

Geometry Friends (GF) is a physics-based platform game, where players control one of two characters (a circle and a rectangle) through a series of both individual and cooperative levels. Each level is solved by retrieving a set of collectibles. This paper proposes an approach using Rapidly-exploring Random Trees (RRTs) to find a solution for the individual levels of Geometry Friends. Solving a level of GF is divided into two subtasks: (1) planning the level; and (2) executing in real time the sequence of moves required to fulfill the plan. We use our RRT approach to solve (1) and a Proportional Integral Derivative (PID) controller to guide (2). The quality of the agent implemented was measured in the 2015 GF Game AI Competition. Results show that our agents are able to plan both public and private levels and are able to control their motion in order to finish most of them.

Keywords

Geometry Friends, Rapidly-exploring Random Tree, PID control, Planning

INTRODUCTION

Geometry Friends[1] is a physics-based platform game developed at the Intelligent Agents and Synthetic Characters Group [2] of INESC-ID. In this game, players have control over two characters, a circle and a rectangle, and the goal is to retrieve a set of collectibles in a 2D world within a time limit in order to complete a level (see Fig. 1). There are two different types of levels: *cooperative levels*, where both characters must join forces to reach some collectibles that would be otherwise impossible to reach; and *individual ones*, focused only in one of the characters.

Solving a level involves performing the right actions at the right time in order to reach all the collectibles and thus fulfill the goal of that level. In order to do this, players must not only plan ahead which actions to do, but also to properly time when to do them. For the cooperative levels, players also have to worry about coordinating the movement of both characters.

All the aforementioned challenges make Geometry Friends an excellent environment for artificial intelligent agents to test their skills of planning, control and cooperation. For this reason, it has been one of the AI competitions in the IEEE CIG Conference[3] since 2013. The

Proceedings of 1st International Joint Conference of DiGRA and FDG

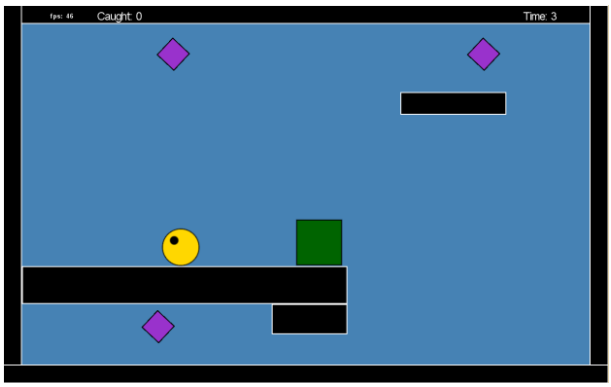

Figure 1: Example of a GF level. The two characters correspond to the green square and the yellow circle, and the collectibles correspond to the purple diamonds.

competition features three tracks: two individual tracks (one for each of the two characters in the game), and a cooperative track, where both characters must play cooperatively to reach the common goal of completing the level. Contestants are asked to create AI agents able to control the characters to solve 10 levels for each track, 5 of which are made public when the submissions open.

However, upon analyzing the performance of existing approaches—namely those submitted to the 2013 and 2014 competitions—they either (i) were "fine-tuned" to the public levels, performing poorly on the private ones (only disclosed after the submissions' deadline); or (ii) involved complex learning algorithms that required significant amounts of data and learning time.

In this work, we propose a planning-based approach to solve a generic level of the Geometry Friends individual track. Our approach is divided in two phases, dubbed *planning* and *control*. Our approach, although designed for and illustrated in the context of the GF domain, rests on first principles and can, therefore, be easily adapted to other domains involving the challenges identified before—namely in terms of planning and control.

The paper starts by providing a more detailed overview of the game and a discussion of existing works that are somehow related to ours. We discuss the competition environment and the results that our approach obtained. Finally, we conclude with some final remarks on our agent's limitations and future avenues for improvement.

GEOMETRY FRIENDS

Geometry Friends is a two-player cooperative game in a 2D platform environment with simulated physics. The game is played with two characters: a green rectangle and a yellow circle. The objective of the game is to retrieve a number of collectibles, represented as purple diamonds and spread throughout the environment, in the least amount of time possible (see Fig. 1 for an example).

Each level has a different layout, depending on the start position of each agent, the location of the collectibles and the location of the obstacles. The latter can be of different types: black platforms obstruct the motion of both agents; green platforms obstruct the motion of

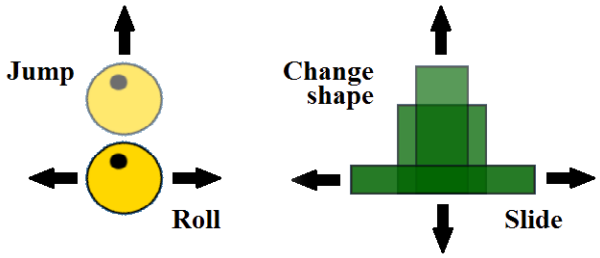

Figure 2: Characters motions.

the circle agent only; yellow platforms obstruct the motion of the rectangle agent only. All obstacles used our experiments were static, although the game allows dynamic obstacles.

Figure 2 depicts the particular actions available to each of the two agents. Both agents can move left and right, either by rolling (circle) or sliding (rectangle). The circle has a lower maximum velocity, while the rectangle has a higher maximum velocity as well as a larger acceleration. The circle can jump to reach other platforms or catch collectibles, while the rectangle has the ability to morph up and down (changing its height and weight without changing its total area), in order to pass under a platform, reach a collectible or tilt over lower platforms (stair-like) in order to climb them.

Both characters are subject to the game physics, including gravity, collisions and friction. Because of the particular characteristics of the agents and the different platform types, some levels can only be solved by one of the agents and some require the cooperation of both agents to be completed.

RELATED WORK

There were few submissions to the previous editions of the Geometry Friends AI competition: 3 in the 2013 edition and 6 in the 2014 edition. Such solutions were either over-specialized in the public levels, or involved learning algorithms that required extensive training.

The CIBot approach (Kim et al. 2014), winner of both the 2013 and 2014 editions of individual GF AI competition (both circle and rectangle), consisted in marking the platforms on the maps and link their edges to form a graph, endowing the agents with the ability to detect impossible connections. This graph is then to find a path, using Dijkstra in the case of the circle agent and Monte Carlo Tree Search in the case of the rectangle. The results on the public levels[4] were good, in the sense that the agents managed to solve and complete five out of five levels. However, the performance on the private levels was not so good, as they were able to solve and complete only two of the five levels, suggesting that this approach may not generalize to arbitrary levels of GF.

Another approach that attained good results on the rectangle track of the 2014 competition was the OPU-SCOM agent (Benôıt and Nakashima n.d.). This approach comprises two layers. The first layer is responsible for planning the motion between the obstacles necessary to reach all the objective points (the position of the collectibles). To do so, it uses two modules: a path finding module, which converts the map into a graph and—much like the

circle agent of the CIBot—applies the Dijkstra algorithm to find the best way to join two objective points; and a second module, based on a particle swarm optimization algorithm, that looks for the best ordering between the points. This first layer is executed beforehand, on an initial setup phase. The result of the first layer is a plan that is then used by the second layer, which generates actions the necessary movement and/or morphing actions in real time to execute that plan. Once again, their results were significantly better on the public levels than on the private ones.

To avoid this excessive specialization to the public levels, Quitério et al proposed a solution for the circle agent based on reinforcement learning (Quiterio et al. 2015). Their approach broke down a level into three sub-tasks: (1) solving one platform; (2) deciding the next platform to solve; and (3) moving from one platform to another. They used reinforcement learning to solve both (1) and (3), and a simple depth-first to solve (2). The results suggest that their approach ismore general: even if their results were not as good as those of CIBot on the public levels, they were superior on the private ones, showing more balanced results.

Another approach appearing in the 2015 edition is the Subgoal A* Agent (Fischer 2015). This agent also marks important points and creates a graph out of them, using A* search to find a route between the points. On the control phase it uses a rule-based "driver" that determines which action should be executed next.

All aforementioned approaches rely on particular map features and, as such, tend to fall into "local optima", over-exploiting some of these featureAs will soon become apparent, our approach does not make use of specific map characteristics. Instead, it uses a path planning approach (Rapidly-exploring Random Trees) which uses random sampling to traverse the level while checking for possible moves and determining whether they contribute to achieving the goal.

SOLUTION

For our initial approach we tried to use physics to plan, in order to have a list of input (like, left, left, jump, left, etc) to execute in real time. We faced some problems that made this approach unfeasible, mainly the fact that Farseer, GF's physics engine, didn't offer us a way to simulate our actions nor did we have access to values for acceleration, collisions and other constants to do our own calculations. So we had to find a way to plan without the physics and leave it for real time, in order to do this we divided the approach into two phases: Planning and Control.

Planning

In order to solve a level of GF, we need to find a way to retrieve all collectibles given the level layout and the characters abilities (henceforth referred as the character's "skills"). Our main focus is a generic solver that is able to complete any given level, and not so much on optimizing the process by which the level is solved. As such, we adapted the Rapidly-Exploring Random Tree algorithm (LaValle and Kuffner Jr 2001; Zickler 2010): if there is at least one solution, the algorithm will find it with high probability.

Algorithm 1 shows the skeleton of this search. The algorithm receives as input the level description, a limit of iterations and the initial state, and initializes a graph with that state.

Algorithm 1 RRT

```
1:  Input:
2:  D: Domain description
3:  x_init: Initial state
4:  MAXITER: maximum number of iterations
5:  G ← NewGraph(x_init)
6:  best ← x_init
7:  for i ← 1 to MAXITER do
8:      x ← GetAvailableState(G)
9:      x' ← ApplyTactics(x, G, D)
10:     if Validate(x', x, G, D) then
11:         G.addNewNode(x')
12:         if BetterThan(x', best) then
13:             best ← x'
14:         if IsGoal(x') then
15:             return TraceBack(x', G)
16: return TraceBack(best, G)
```

The use of graphs instead of trees is a matter of preference, since both structures would be fit for use in our approach. We create a variable to keep track of the best state found, so that if the program reaches the limit of iterations, it will still output a plan, even if incomplete.

A state $\mathbf{x}$ has a set of coordinates, x and y, a direction value, the action that generated it, the size of the agent (used only by the rectangle agent), a reference to the platform the agent is on, and the list of collectibles caught. A collectible is represented by its coordinates and a platform is represented by the coordinates of its center along with its height and width.

In order to generate new states, the algorithm takes an available state (line 8) from the graph and applies *tactics* to it (line 9). A tactic can be a simple action, such as move left or right, or a more complex action, like jump, morph or tilt. The resulting state is then checked by a validation function (line 10); in case the new state is valid, it is added to the graph and compared with the best state so far, replacing it if better. The new state is also tested to see if it is a goal state. The algorithm repeats these steps until it finds a goal state or reaches the limit of iterations, whichever occurs first. In both cases, the algorithm returns a path obtained by tracing back from that state (goal or best state).

The function ApplyTactics receives the current state, graph and domain information, selects a tactic/action (specific for the particular agent) and returns in a new state, $\mathbf{x}'$, with the position of the agent after performing the action. For example, if the circle is on a platform and the tactic "jump to platform" is applied, the generated state will be a position at the edge of the target platform. This way, we need only to worry about the jump on the control phase—during planning it is enough to know that the agent can jump there.

When a state is generated it is always compared to all other states in order to avoid repeated states. A state stops being available when all possible tactics were applied to it. The Validate function checks whether the new state coordinates are inbound according to the level limits,

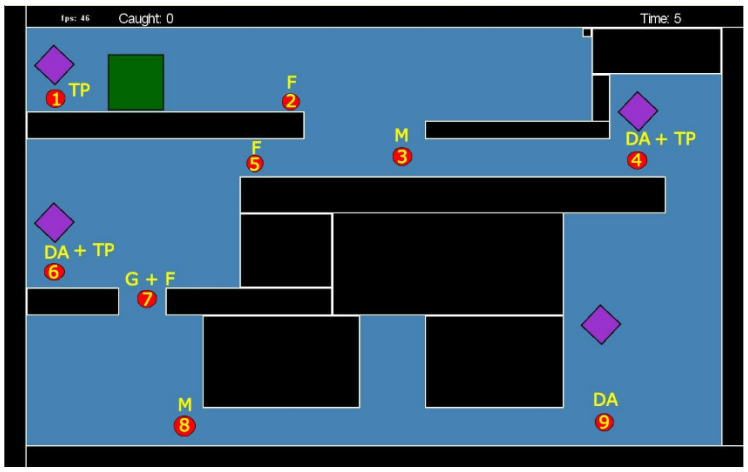

Figure 3: Example of points in a level. TP - Turning Point, F - Fall, DA - Diamond Above, M - Morph, G+F - Gap and Fall

and if it the agent did not go through an obstacle to get there.

Both the BetterThan and IsGoal functions analyze the number of collectibles caught at a given state. A state $\mathbf{x}_1$ is better than a state $\mathbf{x}_2$ if the number of collectibles caught on $\mathbf{x}_1$ is greater than the one on $\mathbf{x}_2$. A state $\mathbf{x}$ is goal state if the number of remaining collectibles on that state equals zero.

The path returned by Algorithm 1 consists of a sequence of *points*, where a point consists of a coordinate (x, y) and a *type*. There are seven point types:

1. *Turning*, where the agent must change direction;
2. *Fall*, where it needs to fall
3. *Diamond above*, when there is a diamond right above it;
4. *Gap and fall*, when the agent needs to fall through a gap;
5. *Morph*, when the agent needs to morph down in order to pass under an obstacle;
6. *Tilt*, when the agent needs to tilt in order to climb to a new platform;
7. *Jump*, when the agent needs to jump to reach a new platform. S

Points can be seen as *essential states*, i.e., states where the agent needs to perform an action or change direction. Some types of points are agent-specific, since they refer to actions that are agent specific. For example, the circle does not *Tilt* or *Morph*, while the rectangle does not *Jump*. The different types of points were created as a minimum set of types required to solve any level of Geometry Friends with our approach, but can also be used by other approaches. Figure 3 illustrates the points identified by the algorithm on one of the levels from the competition. These points will be used in the control phase to guide the agent.

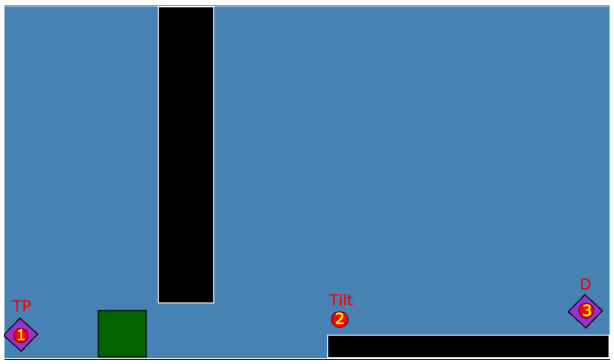

Figure 4: Example of a simple level for the single player using the Rectangle agent.

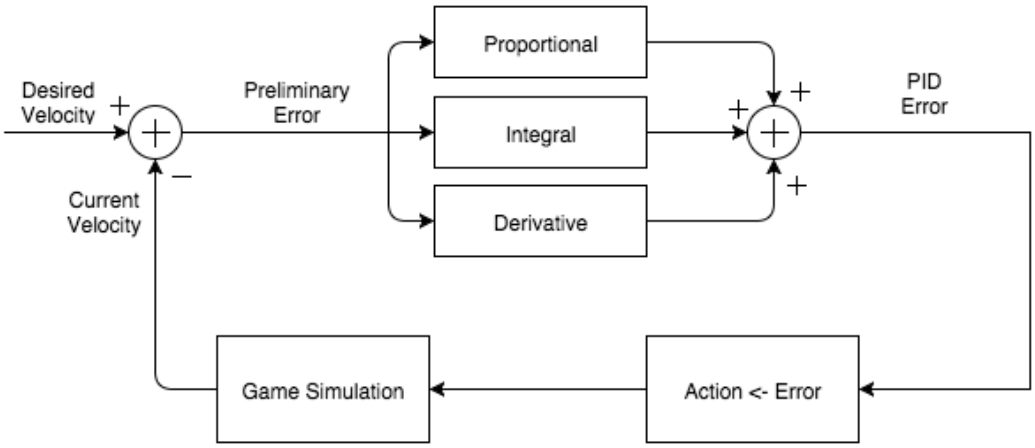

Figure 5: PID System of our controller.

Control

During the control phase, which takes place in real-time, the full plan for the level has already been computed. Using this information, and before the agent starts moving, we compute an estimate of the velocity that the agent should have upon reaching the different points, which are used as a reference by the controller. As an example let us analyze the simple map depicted in Figure 4. In this example, the turning point (on the left) should have velocity zero, in order for the agent to change direction faster. On the other hand, the diamond point on the right should have maximum velocity.[5]

After computing the reference velocity for the different target points, we determine when "special actions" are required to reach certain points. "Special actions" include the jump action for the circle agent and the tilt and morph actions for the rectangle agent. For example, in Figure 4 the tilt point (2) requires a tilt action for the rectangle to be able to reach it. The tilt action requires the rectangle to increase its height, which should not be done before the rectangle passes under the vertical platform. Therefore, we compute a "limit position" to the right of the vertical platform and leave the height of the rectangle unchanged until it reaches such limit position.

To correct the velocity in real-time we use a standard *proportional-integral-derivative* (PID)

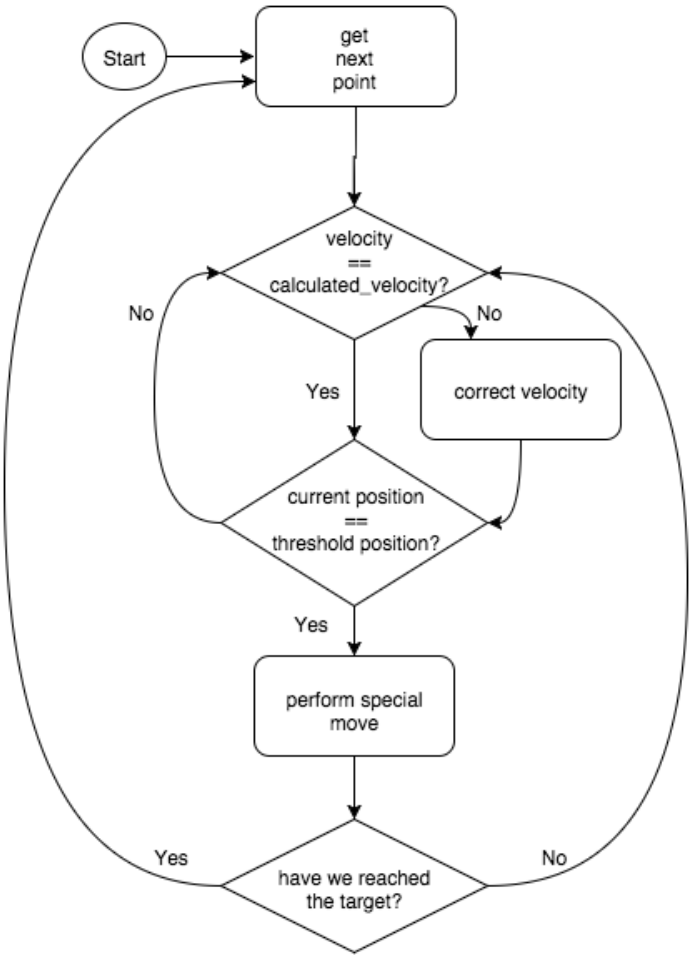

Figure 6: Flowchart for the control phase of an agent.

controller (Araki 2002). A PID controller is a simple controller which determines a control signal by comparing the system's behavior with a reference behavior. In our case, the controller compares the current velocity of the agent with the previously computed reference velocity and uses the difference between the two (referred as the error) to determine whether to accelerate or decelerate the agent. In particular, the control signal is constructed as the combination of three components (see Figure 5):

- *The proportional component*, which is proportional to the error;
- *The integral component*, which is proportional to the accumulated error throughout time;
- *The derivative component*, which is proportional to the rate at which the error changes.

PID control offers a simple approach to minimize the error while allowing a fast response to sudden changes in the velocity (for example, resulting from collisions). Additionally, the PID controller ensures that the agent approaches the reference velocity sufficiently fast

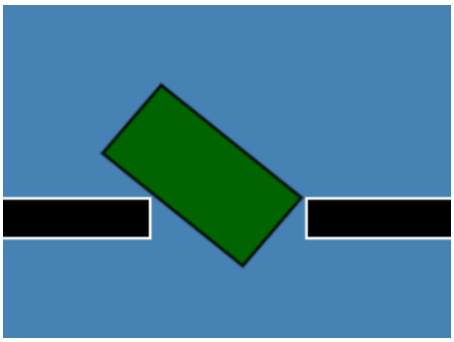

Figure 7: Example of a situation where the agent is stuck.

for our needs. The values for the constants K_p, K_i and K_d used in our experiments were determined empirically to $K_p = 5$ and $K_i = K_d = 3$. The control phase is summarized in Figure 6.

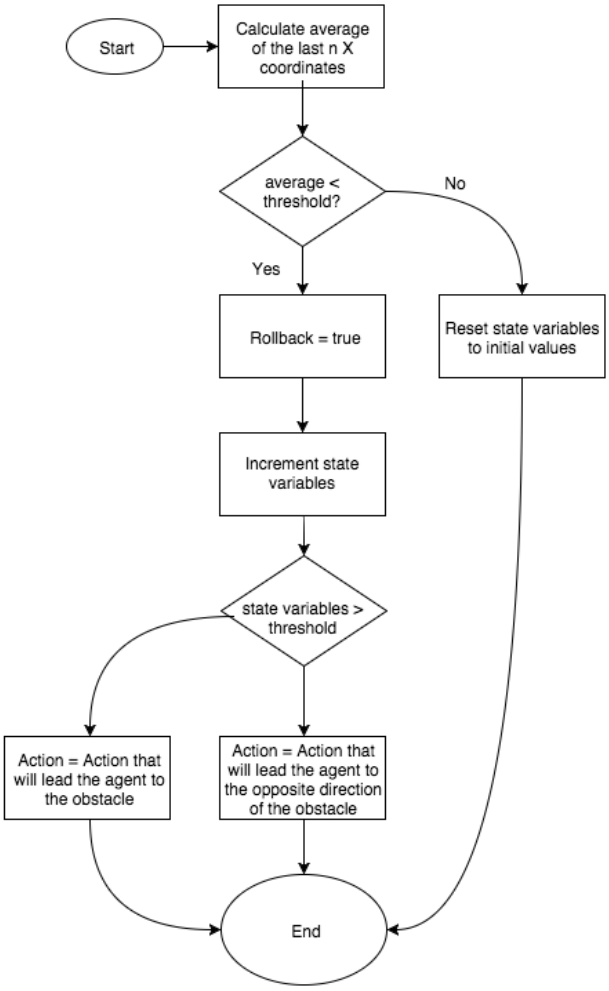

Figure 8: Flowchart of the routine that attempts to remove the agent from an unfavorable or impossible position.

We also note that, using the approach as just described, the agent will often get stuck in certain situations requiring finer control (see, for example, Figure 7). In order to recover from such situations, a specific "release" subroutine was created, which is summarized in Figure 8. To detect when the agent is stuck, we keep track of the past 1000 (x, y) coordinates and velocities and, if they all fall within a specified range of the current position/velocity, the agent initiates the release routine. The routine selects the necessary actions to gain enough velocity to revert and move away from the critical position before trying again. Successive calls to the release routine will push the agent to get further away from the critical position, because most situations can be solved by acquiring a higher speed before attempting to pass an obstacle. The state variables referred in Figure 8 correspond to the number of times that the agent has tried to perform the rollback and the speed required before attempting to attempt the platform.

TEST ENVIRONMENT

To test our approach we submitted our agents to the 2015 Geometry Friends Game AI Competition (Prada et al. 2015), for the individual tracks.

The public and private levels (figure 9 and 10) for the 2015 competition were designed in a way that allows to evaluate the performance of the agents both in planning and control. There are pitfalls embedded into the levels that require the planning stage to plan ahead so that the agent does not find himself in a state that will render the level impossible. There are also several combinations of moves that allow the agents to take advantage of their characteristics in order to solve the level, some of these combinations are very specific and need to be performed with a very good control capability.

In the circle track the levels 1, 3 and 8 are meant to evaluate if the agent recognizes that certain sequence of actions will render the level impossible to complete. For solving levels 1, 2, 3, 7 and 9 very precise control over jump action and speed of the agent is required. Levels 6 and 10 evaluate if the agent is capable of recognizing different types of platforms and knowing with precision its dimensions. Level 8 is meant to judge the ability of recognizing the amount of collectibles the agent can capture with one action, if he does not recognize that several collectibles can be caught at once the planning stage will fail.

In the rectangle track the levels 1, 2, 3, 4, 5, 6 and 8 are meant to evaluate the precision of the control capabilities of the agent and that some actions and poor control will lead to impossible to recover situations. Level 6 is meant to test if the agent is capable to plan ahead enough to realize that he must acquire a certain speed in order to solve the level. Level 3 judges the ability to perform actions in mid-air falls. Level 9 is meant to check if the agent recognizes different types of platforms. Level 10 checks if after performing an action that requires some speed, the agent is able to stand in a small platform in order to catch the collectible.

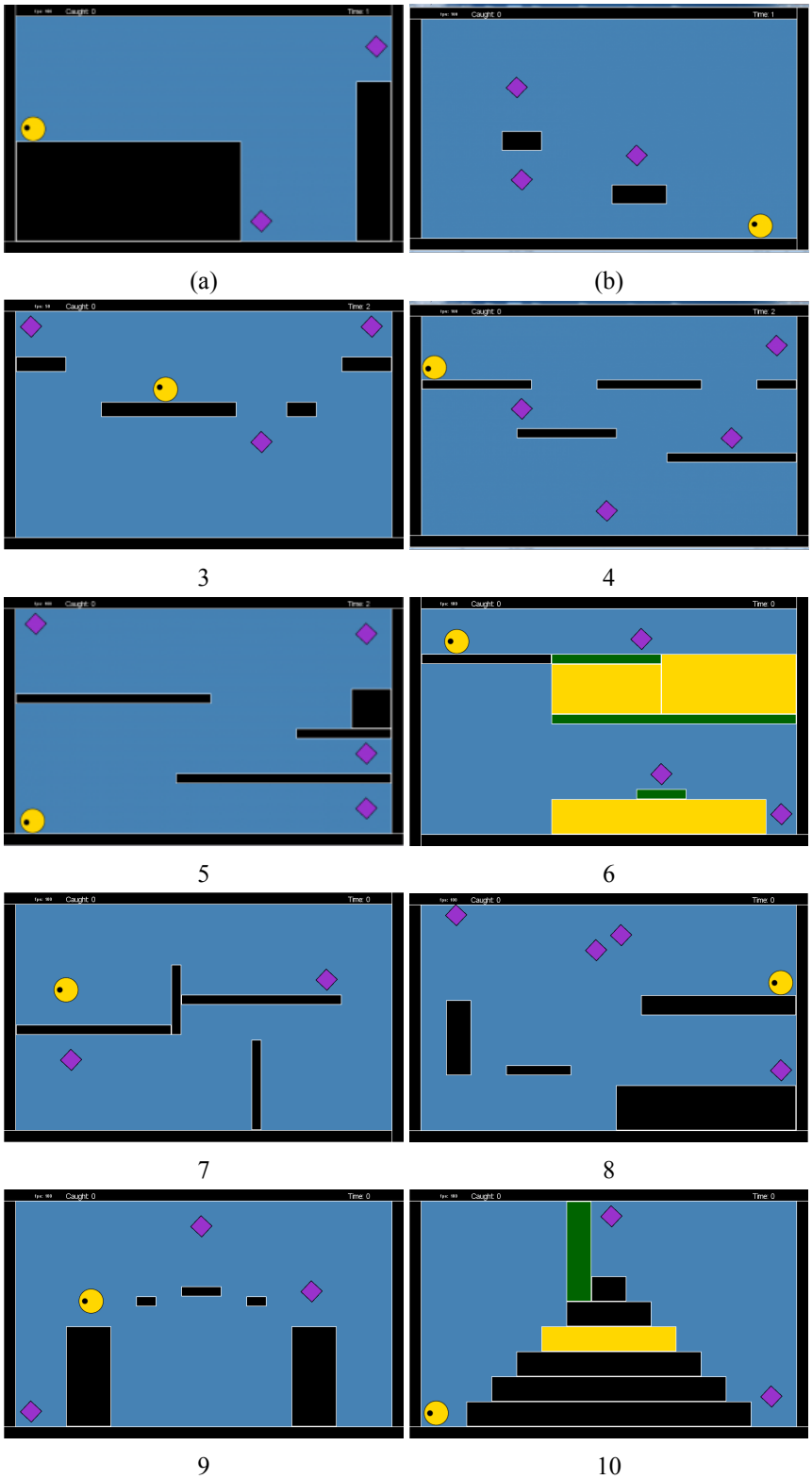

Figure 9: 2015 GF AI Competition Circle Track levels. Levels [1,5] are public and levels [6,10] are private.

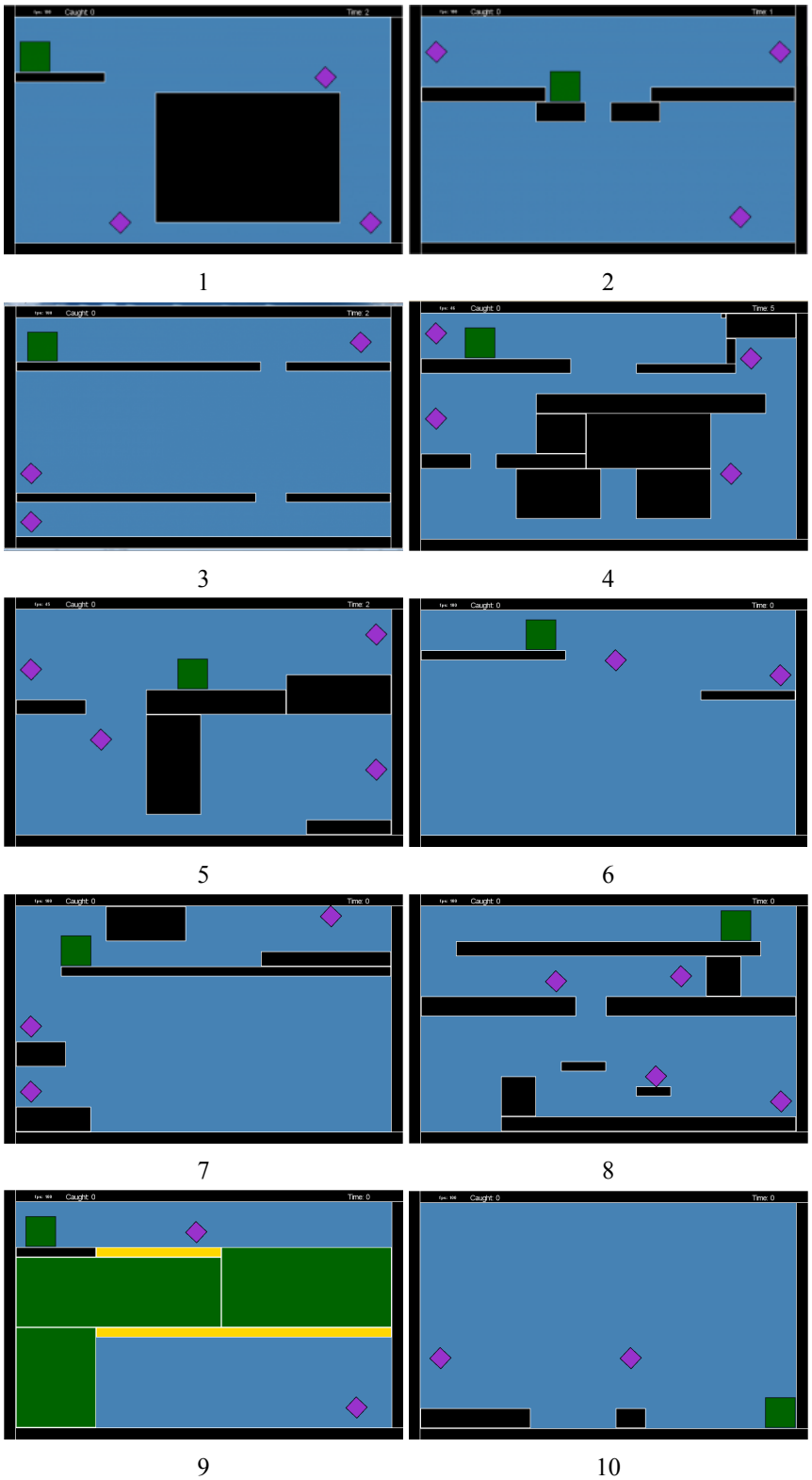

Figure 10: 2015 GF AI Competition Rectangle Track levels. Levels [1-5] are public, levels [6-10] are private.

RESULTS

Due to a late start, we had to submit an incomplete version of our agents, lacking the ability to reach platforms that were of different height and not immediately adjacent. This and some other issues on control reflect on both our agents' results but mainly on the rectangle. Even though our results were a little bit amiss, our circle agent was able to get first place in the circle track and the rectangle agent got a satisfactory third place [6]. The results of both the rectangle and circle tracks can be seen in Table 1 and 2 and detailed tables of both agents' results can be seen in Table 3 and 4.

Team	Score
Subgoal A* Agent	2395
CIBot	1974
RRT Agent	1487
PG-RL Agent	660
OPU-SCOM	200

Table 1: Rank of the Rectangle Agent in the 2015 AI competition of the IEEE CIG Conference.

Team	Score
RRT Agent	2096
PG-RL Agent	1889
CIBot	1443

Table 2: Rank of the Circle Agent in the 2015 AI competition of the IEEE CIG Conference.

Most of the levels where our agents failed were due to planning issues (level 1, 2, 3, 8, 9 and 10 of the circle and 1, 3, 5, 6 and 8 of the rectangle), mainly that inability of reaching non-adjacent platforms at a different height. The control problems are more noticeable on the rectangle, leaving level 2 and 10 unfinished. The fact that the algorithm is random influenced two of the circle levels, 5 and 7, since it took too much time completing them some of the times. Finally, due to some bugs of the game, level 4 and 7 of the rectangle track couldn't be completed. As we have stated, we believe that our agents did not suffer from overspecialization and it is ready to solve any level of Geometry Friends. We can confirm this by analyzing our results in the private and the public levels. For the circle agent, it solved 16 out of 50 attempts and acquired 9.5 out of 16 possible diamonds, on average for the public levels, in the private levels it solved 19 out of 50 attempts and acquired 8.3 out of 14 possible diamonds, on average. Therefor we had a success rate of 0.32 in the public levels, and 0.38 in the private levels.

Level	Runs Completed	Diamonds	Time (Limit) s	Score
1	0	1(2)	30(30)	100
2	0	0(3)	60(60)	0
3	0	1(3)	90(90)	100
4	10	4(4)	32.6(90)	528
5	6	3.5(4)	85.5(90)	359
6	10	3(3)	20.3(40)	399
7	9	1.9(2)	33(60)	271
8	0	2(4)	60(60)	200
9	0	1(3)	40(40)	100
10	0	0.4(2)	50(50)	40
Total				2096

Table 3: Results of the Circle Agent in the 2015 AI competition of the IEEE CIG Conference.

Level	Runs Completed	Diamonds	Time (Limit) s	Score
1	0	2(3)	30(30)	200
2	0	2(3)	60(60)	200
3	0	1(3)	60(60)	100
4	0	1(4)	90(90)	100
5	0	2(4)	90(90)	200
6	0	1(2)	20(20)	100
7	1	0.6(3)	56,1(60)	61
8	0	2(4)	50(50)	200
9	10	2(2)	10(30)	227
10	0	1(2)	45(45)	100
Total				1487

Table 4: Results of the Rectangle Agent in the 2015 AI competition of the IEEE CIG Conference.

FUTURE WORK

Our agents still have problems with some situations, mostly failing to navigate due to control issues. Some of this issues come from the game itself, for example there are cases in which the rectangle gets stuck trying to fall through a hole. An option for solving this issue is to add more information to the points of our plan. For the given example the size of the hole between the platform and the wall may be add, and then change the agent behavior on the control phase accordingly to that new information.

The PID variables where calculated through trial and error, leaving space for improvement in the motion control. A tuning algorithm can be used, in order to perfect those values (Nagaraj et al. 2008).

Other problems could be overcame by training a set of variables with a big number of scenarios, in order to have more adequate values. Like the speed the rectangle needs at the end of the platform to fall to another platform, according to the distance between them.

In order to enter the cooperative track, several additions to our solution are possible. In the planning phase both agents must know that cooperation is required to reach certain goals, such as the circle using the rectangle height in order to jump higher, and have into consideration that now, one agent can carry the other through some specific color obstacle, this will lead to new states, and therefore new level solutions. In the control phase, the agents must know how to avoid each other and how to perform collaborative moves, this will require both control stages to have information about the characteristics of the other agent and its limitations.

Another possible change is to not create a one-shot planning stage, and use a radius of planning that is aware of both individual and cooperative tactics such that, the plan can change throughout the course of the game, when both agents are in range of one other and cooperation is possible. A more different approach to the cooperative track can be the use of utility heuristics, such that, each tactic can be assigned a certain reward or penalty, making the algorithm less random but encouraging that certain actions will benefit both agents leading to better results.

CONCLUSION

With this work we proposed a simpler approach to solve any level of the Geometry Friends individual tracks, divided in two stages, planning and control. We used a Rapidly-Exploring Random Tree based algorithm for the first and a Proportional, Integrative and Derivative controller for the second avoiding dealing with the physics of the game in the planning stage.

Through the results we have obtained by participating in the 2015 Competition of Geometry Friends we conclude our solution is able to compete with more advanced algorithms, whilst using a general approach on resolving the problem the game presents.

Space for improvement still exists leading us to believe that our approach can also be applied to the cooperative competition and perform successfully. We pretend to submit an improved version for 2016's competition, with the current issues fixed and capable of competing on the cooperative track.

Given our approach, adaptation from our work is simple. For the RRT algorithm to be used for other problems, one just needs new ApplyTactics, Validate, BetterThan and IsGoal functions considering the context of the new problem. Same goes for the PID controller, given any two points on the same axis, a initial velocity, a final velocity and the starting and ending points it will be able to guide any agent from one point to the other.

Endnotes

1. http://gaips.inesc-id.pt/geometryfriends/
2. http://gaips.inesc-id.pt/
3. http://www.ieee-cig.org/
4. 2014 Results of GF Competition - http://gaips.inesc-id.pt/geometryfriends/?page_id=529

5. This velocity is agent-dependent, since the circle and the rectangle have different maximum speeds.

6. 2015 Results of GF Competition - http://gaips.inesc-id.pt/geometryfriends/?page id=817

ACKNOWLEDGMENTS

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

BIBLIOGRAPHY

Araki, Mituhiko. 2002. "PID control." *Control systems, robotics and automation* 2:1–23.

Benôıt, D. A. Vallade, and T. Nakashima. n.d. "Opu-scom technical report."

Fischer, Daniel. 2015. "Development of Search-based Agents for the Physics-based Simulation game Geometry Friends." PhD diss., Maastricht University.

Kim, Hyun-Tae, Du-Mim Yoon, and Kyung-Joong Kim. 2014. "Solving Geometry Friends using Monte-Carlo Tree Search with directed graph representation." In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on,* 1–2. August.

LaValle, Steven M, and James J Kuffner Jr. 2001. "Rapidly-exploring random trees: Progress and prospects." In *Algorithmic and Computational Robotics: New Directions,* edited by B Donald, K Lynch, and D Rus, 293–308. Wellesley.

Nagaraj, B, S Subha, and B Rampriya. 2008. "Tuning algorithms for PID controller using soft computing techniques." *International Journal of Computer Science and Network Security* 8 (4): 278–281.

Prada, R., P. Lopes, J. Catarino, J. Quiterio, and F.S. Melo. 2015. "The geometry friends game AI competition." In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on,* 431–438. August.

Quiterio, J., R. Prada, and F.S. Melo. 2015. "A reinforcement learning approach for the circle agent of geometry friends." In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on,* 423–430. August.

Zickler, Stefan. 2010. "Physics-Based Robot Motion Planning in Dynamic Multi-Body Environments." PhD diss., Carnegie Mellon University, Thesis Number: CMU-CS-10-115.