Rapidly-Exploring Random Tree approach for Geometry Friends

Rui Soares and Francisco Leal
INESC-ID

We divided solving a level of Geometry Friends into: Planning and Control. We first create a plan with fixed steps (with a size big enough to not generate too many unnecessary states) marking the map with relevant points (like where to jump, where to turn etc) and then on the control phase, we follow this points adapting the velocity to them.

Planning

```
Algorithm 1 RRT
1: Input:
2: $d$ : Domain
3: $x_{\text{init}}$ : Initial state
4: $X_{\text{goal}}$ : Set of goal states
5: $Validate$ : Validation function
6: $BetterThan$ : Comparing function
7: $limit$ : max iterations
8: graph ← NewGraph($x_{\text{init}}$)
9: best ← $x_{\text{init}}$
10: for $i \leftarrow 1, limit$ do
11:     $x$ ← GetAvailableState(graph)
12:     $x'$ ← ApplyTactics($x$, graph, $d$)
13:     if $Validate(x', x, graph, d)$ then
14:         graph.addNewNode($x'$)
15:         if BetterThan($x'$,best) then
16:             best ← $x'$
17:         if $x' \in X_{\text{goal}}$ then
18:             return TraceBack($x'$, graph)
19: return TraceBack(best, graph)
```

Figure 1: RRT algorithm.

Figure 1 shows our algorithm for planning, based on the Rapidly-Exploring Random Tree search. We start by creating a graph containing only the initial state and we generate a new state every iteration (line 12) from the expandable states (line 11), until we reach a final state (line 17) or the limit of iterations (line 10), in which case we return the best state so far (line 19).

Our Validate function (line 13) validates a state by checking if there was an obstacle between the initial state and the final one, and if the new state is inside the level borders.
Since we're returning the best state we found in case we reach the iterations' limit we used a function to compare two states (line 15). This function tells that a state is better than another state if it has a higher number of collectibles caught.
The way we get new states is by applying tactics to previous states (line 12), these tactics are:

- For the rectangle: go left, right, morph-up, morph-down and tilt;
- For the circle: go left, right, jump up and jump to a platform.

In other RRT's approaches tactics are made of skills, in our approach the skills are left for the control part, and will be the basic actions of the characters (roll, slide, jump and change size).
The result of all this planning is a path, which is a list of points. A point has: x coordinate, y coordinate, size, and point type. Point types are: turning point (TP), fall (F), morph (M), diamond above (DA), gap (G) and jump (J). In figure 2 we can observe a mapped level of the rectangle track with the points of the path that leads to its resolution. In figure 3 we have an example for the circle.

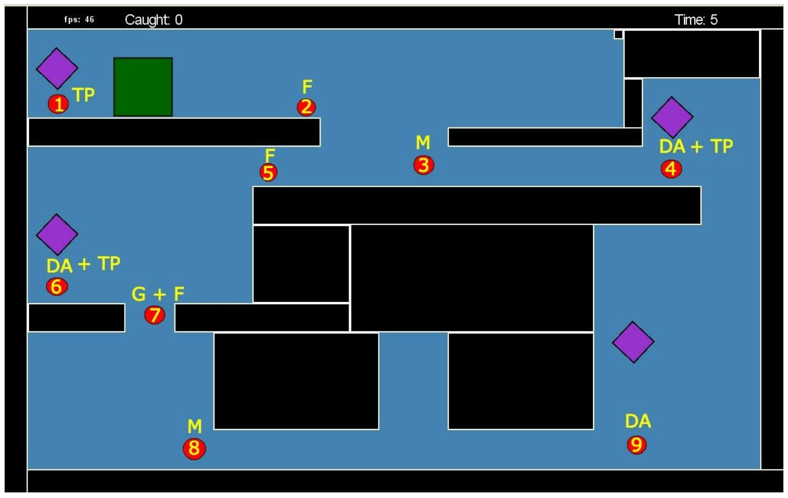


Figure 2: Level of Rectangle track marked with the path points.

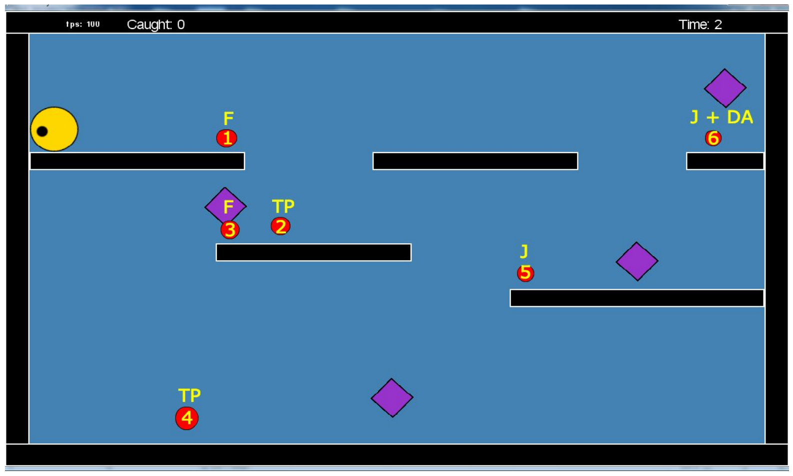
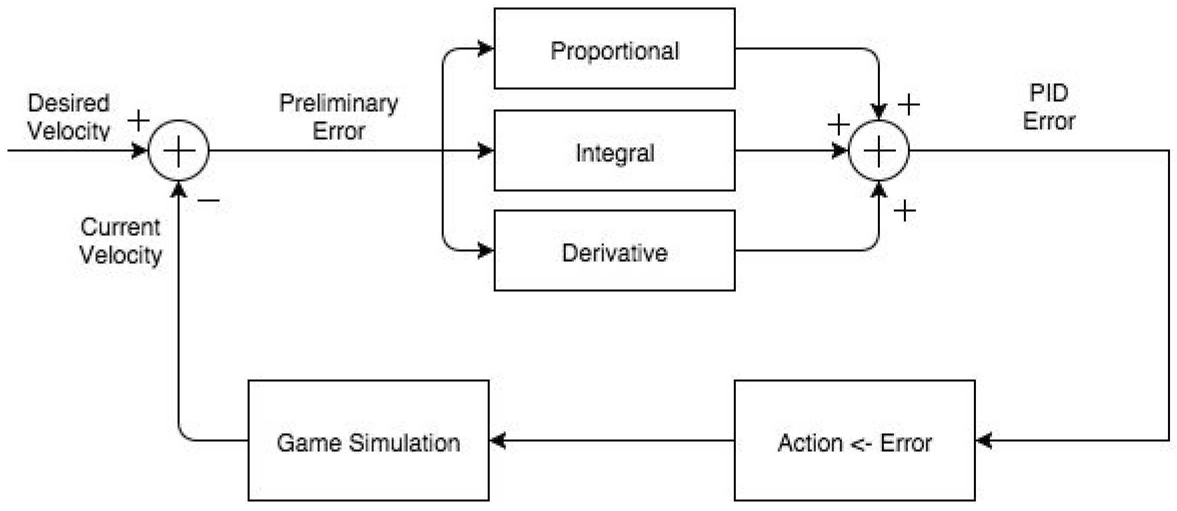


Figure 3: Level of Circle track marked with the path points.

Control

Upon reaching this part we have the full information given by the planning stage, therefore we know all the points required to complete the level.
In real-time (the part where the control stage occurs) we can only control the velocity and make use of the character's skills to perform special moves, therefore, before the game starts we add some extra values to each point in order to be able to have the agent fulfil the requirements to pass each point. The extra values are a required theoretical velocity that depends on the type of each point and a certain threshold when a special move is required to reach that point.
Turning points will have a required velocity of 0, since we want to reach that point and turn back, thus changing the direction. Fall points have either velocity 0, which happens in most cases, or some higher value when it is required to go from one platform to the other, for example in figure 3 if we wanted to go from point 2 straight to point 5 a velocity of 0 would of course result in failing to reach the platform. Morph points have maximum velocitysince this point is required only for when the rectangle needs to squeeze in order to go beneath a platform. DiamondAbove points have 0 velocity since we want to position ourselves beneath it to be able to catch it, however, in real-time the agent will not wait to be perfectly under the point, a threshold is used for when the agent is close enough that it can catch it. Gap point has velocity zero since we want the agent to be centered in the gap to be able to perform a correct fall. Jump point has maximum velocity, however in real-time the landing place is calculated as the agent is moving and if within a certain error the landing place is the point we want to reach we perform the jump even if we haven't reached maximum velocity.



$$P : K_p e(t), K_p = 5 \qquad I : K_i \int_0^t e(\tau) d\tau, K_i = 3 \qquad D : K_d \frac{de}{dt}, K_d = 3$$

Figure 3: PID controller scheme and constants used for Geometry Friends.

For the velocity control in real-time we use a proportional integral and derivative controller (PID controller) as seen in figure 3. We use such a system because it minimizes the error over time,

allows for a fast response to sudden speed changes and will approach the desired speed fast enough for our needs. The values for the constants Kp, Ki and Kd were calculated through trial and error. The best results were obtained using Kp = 5 and Ki=Kd=3.
In figure 4 we have the flowchart for the real-time routine, the PID controller is used in the "correct velocity" step.

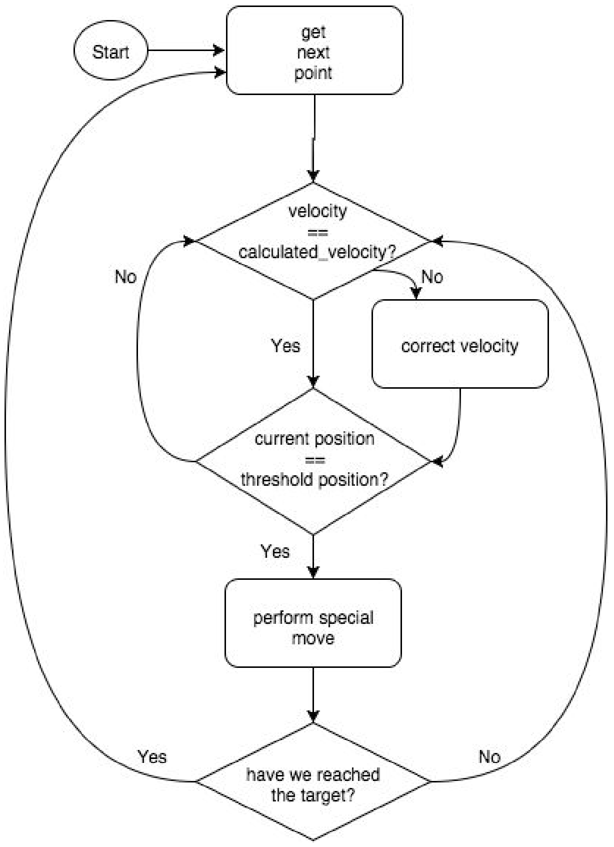

Start
get next point
velocity == calculated_velocity?
No
No
Yes
correct velocity
current position == threshold position?
Yes
perform special move
Yes
have we reached the target?
No

Figure 4: Flowchart of the real-time routine.