Geometry Friends

Rectangle Track with an A* approach.

Orfanoudakis Stavros 2015030030

February 2020

1 Introduction

Geometry Friends is a two player cooperative puzzle computer game developed at GAIPS INESC-ID. The competition consists of creating artificial players for the cooperative platform puzzle game Geometry Friends. There are two large sections of the competition. The first consists of the "The Cooperation Track" which focuses on developing two AI players, capable of cooperating with each other in order to solve levels designed for player cooperation. The other one is "The Single AI Track" which consists of two sub-tracks: the Circle Track and the Rectangle Track. Each sub-track will focus exclusively on the development of one of the two characters of the game. These tracks are targeted at participants that want to tackle the problems of the domain (physics-based platform puzzle), without the complexity of cooperation. Typically, the Rectangle Track presents more challenges in puzzle solving and the Circle Track presents more challenges in the actuation in the environment. In this specific project i focused on solving the single rectangle track using A* algorithm and many other physics simulation functions.

2 Rectangle Track

Every rectangle Track consists of the agent, the obstacles and the diamonds that must be collected in order to proceed to the next level (Figure 1). Also, the rectangle agent has available a set of 4 actions. At first ,it can morph up till it creates a slim rectangle or morph down to a flat one. The other two moves are simply moving left or right, while the agent has any possible form , as seen in Figure 2.

The challenges of this competition are to create an AI that can move across the map and capture all diamonds in the correct order. For example if the agent moves straight down through the middle gap in the example map (figure 1), it won't be able to catch the upper left collectible because of the poor planning of the agent. In addition, the state space and action space is very large, so

the problem would not be able to be solved in real time A* without the proper discretization. The last challenge to consider is the gravity of the game, which means that if the agent has no obstacle below it will start falling until it reaches a new (hard to guess) position in the level.

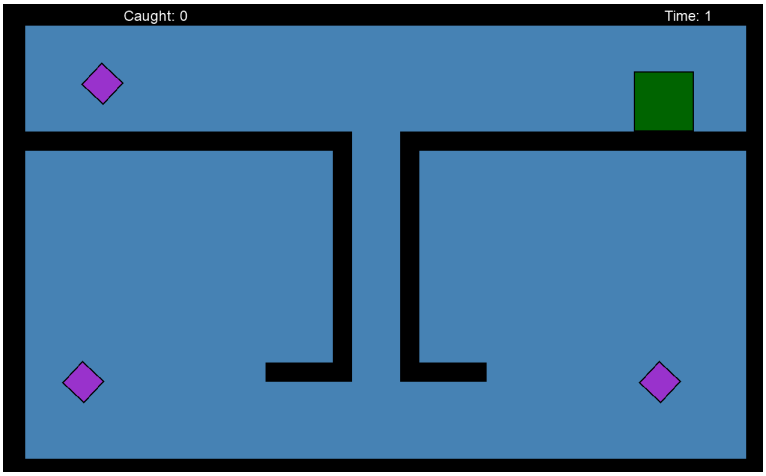

Figure 1: Example Stage

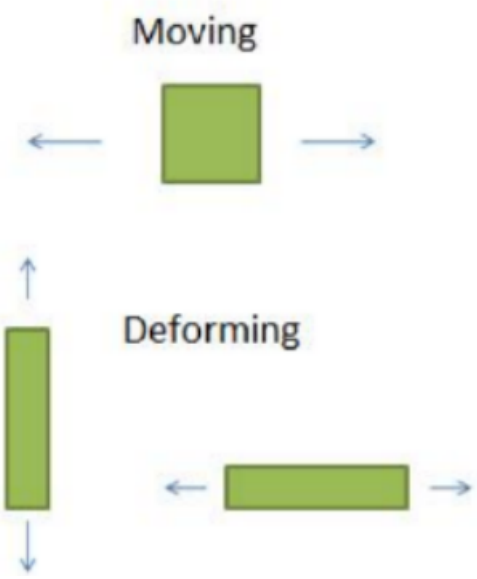

Figure 2: Representation of the rectangle's actions.

3 Agent Implementation

3.1 Discretization

The most important part of the implementation is to make the state space discrete. So i used the sensors info and i created a 1200x720 pixel map which was later used for the navigation. Its important to write here that we have complete information about the map and the only problem we have to solve is the navigation. So there is no need to use SLAM algorithms.

3.2 A*

The next step consists of the implementation of the A* algorithm. Below in Figure 3 we can see the pseudo-code of the A*. The heuristic function that was used $f(n) = g(n) + h(n)$ sums up the the number of steps needed to reach a state $g(n)$ and the Euclidean distance $h(n)$ of the node to the closest collectible. Manhattan distance was also considered but through experimentation it was found that Euclidean distance performed better in this type of problem.

Algorithm 1: A*

Input: start, $goal(n)$, $h(n)$, $expand(n)$
Output: path

```
1  if goal(start) = true then return makePath(start)
2
3  open ← start
4  closed ← ∅
5  while open ≠ ∅ do
6      sort(open)
7      n ← open.pop()
8      kids ← expand(n)
9      forall the kid ∈ kids do
10         kid.f ← (n.g + 1) + h(kid)
11         if goal(kid) = true then return makePath(kid)
12         if kid ∩ closed = ∅ then open ← kid
13     closed ← n
14 return ∅
```

Figure 3: A* algorithm pseudo-code.

3.3 Generating Kid Nodes for A*

One of the most challenging and most time consuming parts of the agent implementation is the generation of new kid nodes. At the simplest scenario the agent is moving on solid terrain without any obstacles and any holes in the ground. In

that case there is no need to use complex functions,so the new node's position would be just some pixels left or right. On the other hand, when there is a hole on the ground like figure 1 the agent would have to consider either to morph down and move the to the other side of the hole or try to fall through the hole . There are many other cases that the agent tries to simulate how he is gonna fall using the pixel map form the first step.

3.4 Final Action Sequence

The game in general can handle continuous action input, so theoretically running the A* only once would be enough to get the whole path to the target. The variation in this problem is that when an action is done for x time(ms) it builds some momentum meaning that it cannot stop instantly in the predetermined position. All of these create a lot of uncertainty during the planning. So to solve this problem i am splitting the time horizon of the game to one second splits. Every one second split consists of the A* search time ,the time(less than 1 second) A* found the key has to be enabled and the rest time is used without any input just so any unnecessary momentum would be eliminated.

Summing up, the agent is calculating its path every 1 second and is only doing the first action of the action sequence generated by the A* algorithm. This way the agent is not affected that much by the uncertainty of the game.

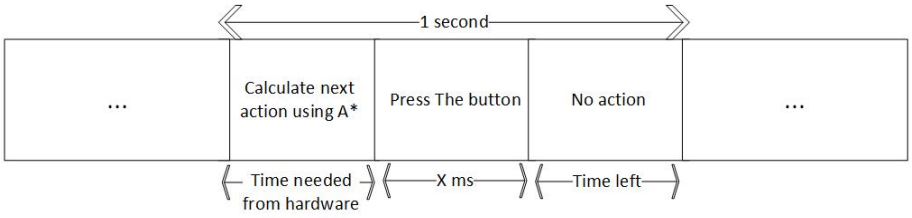

Figure 4: Time splits.

4 Results

In the end, the autonomous agent manages to finish all of the available levels of the 2019 Single Rectangle track competition in a decent amount of time. Some videos from the agent's walk-through are uploaded at this project's web page. Nevertheless , there are some times the agent will fail to finish a level due to the uncertainty and the continuous input of the game i previously described. Some of the reasons the agent might not finish a level could be being trapped in the middle of two obstacles or overturn to a non-returning position.