GeometryFriends Technical Report

Yanir Buznah[1], Shai Avivi[1], and Amitai Yacobi[1]

[1]Department of Computer Science, Bar-Ilan University

August 2023

Abstract

In this report, we introduce our new circle agent developed which is designed to gather all diamonds by solving two main challenges: figuring out the shortest path to reach all diamonds and deciding how the agent should move along that path. Our findings demonstrate that our agent effectively overcomes significant obstacles that were previously unsolved according to current knowledge.

1 Introduction

This report introduces our circle agent designed for the Geometry Friends competition. We crafted our agent using a method outlined by [1]. This method equips our agent to efficiently gather all diamonds in different levels by solving two connected challenges. The first challenge involves finding the shortest path to collect all diamonds, while the second requires selecting the best actions for the agent to follow along that path. To solve the first challenge, we use a search algorithm on a directed graph that represents each level. For the second challenge, we employ a method based on reinforcement learning.

Our experiments confirm that this method performs better than other approaches used in past competitions. However, we found that the agent described in the work of [1] struggles to achieve high scores on certain levels due to some limitations. In this report, we suggest specific changes to help the agent overcome these limitations and handle more complex environments and situations effectively.

2 Agent Design

The way [1] designed the agent involves two methods to solve the two subproblems we talked about earlier. This section briefly explains these two methods.

2.1 Finding the shortest path

Before we delve into defining a path, let's establish the concept of a "platform." A platform refers to a level, flat obstacle that the agent has the capability to step onto. To transition from one platform to another, the agent is required to jump or descend. This vertical movement enables the agent to travel between platforms. Additionally, the agent can successfully obtain each diamond by executing jumps or descents from these platforms.

A path refers to the sequence of platforms the agent travels across. More precisely, it's like a list of steps in a graph where each platform is a point and the connections between platforms are represented as lines. These lines show where the agent can go from one platform to another. We find the quickest path using a search algorithm to navigate this graph.

In order to generate the directed graph, the following two pieces of information is required:

1. Platforms to which the agent can move from each platform,
2. Diamonds which it can collect from each platform.

The information is gathered by observing how the agent jumps or falls. This is accomplished by predicting the paths the agent takes when it jumps or falls. The paths are predicted for two distinct situations: one without obstacles causing a collision (Case 1), and the other involving an obstacle collision (Case 2). In Case 1, the agent's trajectory is assumed to follow a parabolic pattern, represented by these equations:

$$x = v_x t$$

$$y = v_y t - \frac{1}{2} g t^2$$

Here, t signifies time, originating at zero when the agent initiates its jump or fall. The variables x and y stand for horizontal and vertical positions, respectively, with the agent's starting point considered the origin at $t = 0$. The constants v_x and v_y denote the agent's horizontal and vertical velocities at the outset ($t = 0$), while g signifies the acceleration due to gravity. These constant values are determined through appropriate methodologies.
In Case 2, the trajectory from Case 1 is utilized until the agent encounters an obstacle collision. Following the collision, the agent's movement is assumed to be a straightforward downward fall. Based on these anticipated trajectories, a graph is constructed. This graph establishes connections between platforms depending on whether the agent can leap or descend between them. The distance between the agent's current position and its landing point post-jump or fall defines the length of these connections. Subsequently, the shortest path within this graph is ascertained through a technique termed Subgoal A*.

2.2 Selecting agent's actions

When the shortest path is found, the following information is obtained:

- Position in which the agent jumps or falls from each platform,
- Horizontal velocity with which it jumps or falls from each platform.

By adhering to actions that position the agent to match the acquired positions and velocities, the agent can effectively follow the determined path. Consequently, the task of choosing the agent's actions can be framed as a relatively straightforward problem in reinforcement learning. The primary goal here is to attain the target position on a singular platform with the desired velocity. In order to learn for any target position, states are defined by:

- Relative distance from the agent to the target position.
- Velocity of the agent.

and actions are defined by:

- Rolling in the positive direction.
- Rolling in the negative direction.

The positive direction is defined as the direction of the target velocity and the negative direction as the opposite direction. Rewards are basically defined as follows:

- $+200$ in the target state,
- -1 in the other states.

The agent employs Q-learning as the learning approach. Given that optimal actions hinge on the target velocity, the agent learns autonomously for multiple reinforcement learning scenarios, each involving distinct target velocities.

3 Limitations

We found that the agent in the method proposed by [1] struggles with getting high scores in the following three scenarios:

1. Extremely narrow platforms.
2. Wide chasms.
3. Colliding with the floor.

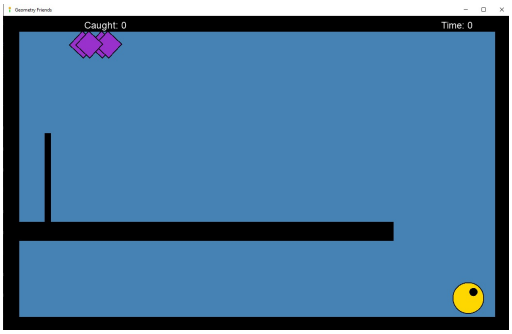

Figure 1: Narrow platform

3.1 Narrow platforms

As it is illustrated in Figure 1, When the platform is extremely narrow, the agent should first stop on the platform and then jump to collect the diamonds. However, the agent in [1] and other agents we explored, often slip down from the narrow platform. One possible reason for this limitation is that the agent is able to jump only when its center is above the platform. If more than half of the circle is out of the platform, it will surely slip down from it.

3.2 Wide chasms

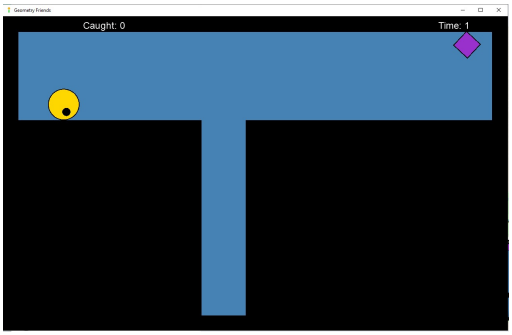

Figure 2: A level where the agent inevitably collides with a ceiling

As it can be shown in Figure 2, the chasm is pretty wide and the agent can move to the right platform only after colliding with the ceiling to collect the diamond. However, the agent in [1] and other agents we explored, cannot move to the right platform and fails to collect the diamond.

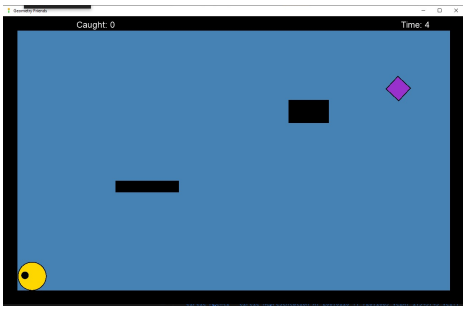

Figure 3: Colliding with the floor

3.3 Colliding with the floor

In the level depicted in Figure 3, the nearest platform to the agent is situated at a considerable height. In this scenario, the effectiveness of the rebound is of great importance. The challenge lies in the agent's disregard for how contact with the ground impacts its velocity. As seen in [1], this issue prevented the agent from reaching said platform and successfully completing the environment.

4 Solving the Limitations

In order to solve the above limitations we made the following modifications:

4.1 Narrow platforms and wide chasms

The main problem with narrow platforms and wide chasms is that the agent cannot jump when more than half of it is out of the platform. We want to allow the agent to jump even when most of it is out of the platform so that in narrow platforms it will not slip down, and in wide chasms, it will be able to jump from the edge and "save" distance by it. To address this problem, we expanded the platforms on the right and left so that the center of the agent will remain above the platform. This makes the jump riskier so we only expand the platforms if the agent don't find a path to goal.

4.2 Colliding with the floor

To address this limitation occurs when colliding with the floor, we added consideration in how colliding with the floor affects the agent's velocity. Before when colliding with the floor the AI zeroed the x and y velocity. Now it reverses the y velocity and divides the result by 3 which resembles the mechanics of the game. Also, it doesn't change the x velocity. The change in velocity is true only if the air rotates to the side of the jump so we force rotation when the only solution the a level must consider what happens after colliding with the floor.

5 Other Improvements

In our efforts to attain better scores across various levels, we made slight adjustments. For instance, we swapped out the usual euclidean distance in Subgoal A^* with a modified version that takes into account the platform locations. This change aimed to provide a more precise heuristic. Moreover, we realized that gathering diamonds positioned above the agent presents more difficulty. This is due to gravity and the challenge posed by the inability to retrieve a missed diamond after a fall. To address this concern, we attempted to make the agent prioritize reaching upward to collect diamonds located in the upper part of the screen.

References

[1] H. Oonishi and H. Iima. Improving generalization ability in a puzzle game using reinforcement learning. In *IEEE Conference on Computational Intelligence and Games, CIG 2017, New York, NY, USA, August 22-25, 2017*, pages 232–239. IEEE, 2017.