Geometry Friends Environment using OpenAI Gym

Bruna Kimura[1] and Raphael Costa[1]
[1] Universidade do Porto, Porto, Portugal.
Up201902504@fe.up.pt, Up201902503@fe.up.pt

Abstract

Geometry Friends is an AI (Artificial Intelligence) competition, that has as its main objective a simple cooperation game, that can be implemented in various platforms. In this article is built a simplified GeoFriends environment using OpenAI Gym and Pygame. The algorithm for the AI was built using Reinforcement Learning techniques, for example PPO (Proximal Policy Optimization), that presented interesting results in specific situations.

1 Introduction

The purpose of this project was to develop an OpenAI Gym environment with a simplified version of Geometry Friends game. The OpenAI Gym is a library for Python, with tools for developing and compare Reinforcement Learning algorithms. In addition, Geometry Friends (or just GeoFriends) is a very simple game, in which two players have the main goal of collecting objects that are allocated all over the map. The players are composed by a circle and a rectangle, that can both move in any direction. The rectangle player can change its height and width as it's needed, and the circle has the ability to jump. In this project was proposed a simplified version where just the circle player plays.

Also, as a base for developing this project, some ideas and functions were used from the user *bluemoon93* of Github (bluemoon93 2019). The target of this work was also to build and environment for Geometry Friends but using both players (Simões, Lau e Reis 8-13 July 2018).

2 Methodology

2.1 Description and rules of the game

Geometry Friends is a cooperative puzzle platformer game, developed and created by GAIPS INESC-ID (GAIPS, INESC-ID s.d.). It works in a 2D domain, with physics applied (gravity and friction). The goal of the players is cooperating to collect the diamonds widespreaded in the map as fast as possible. The game is exemplified in Figure 1.

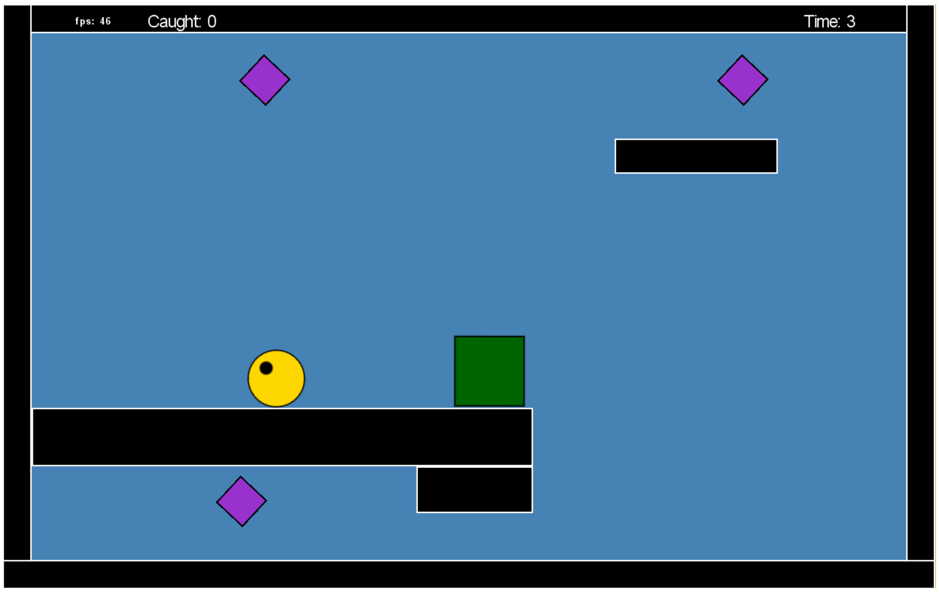

Figure 1: Example level

Each player is a different agent. The first one is a yellow circle, that can roll in any horizontal direction and can also jump, as it seems in the left image in Figure 2. The second one is a green rectangle, that can slide in any horizontal direction, but can't jump. In addition, the rectangle can change its width and height, as long its area is maintained, exemplified in the right image on Figure 2.

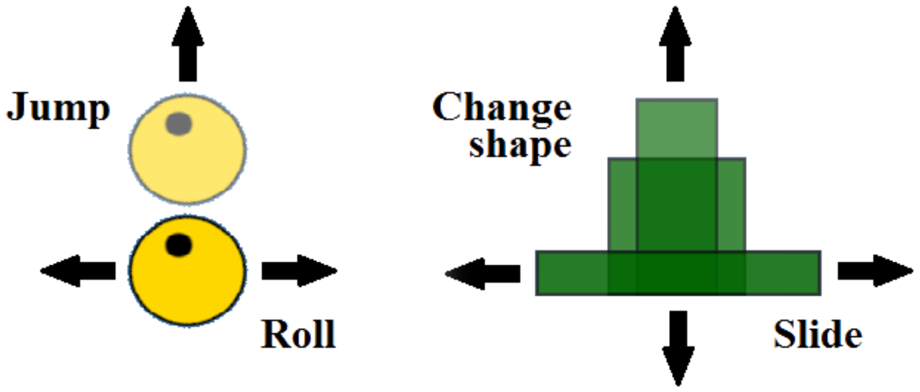

Figure 2: The game players

The goal and the rules of the games are simple and intuitive. The real challenge is to develop an AI code for the players, which means that they have to learn by themselves how to be efficient and how the game works.

In this project, it was proposed a simplified version of the game. Just one player plays it (the circle one), the number of obstacles is limited, and the physics simulated consists just on simulating gravity. Furthermore, the player has to collect a single diamond in the map, showed in Figure 3.

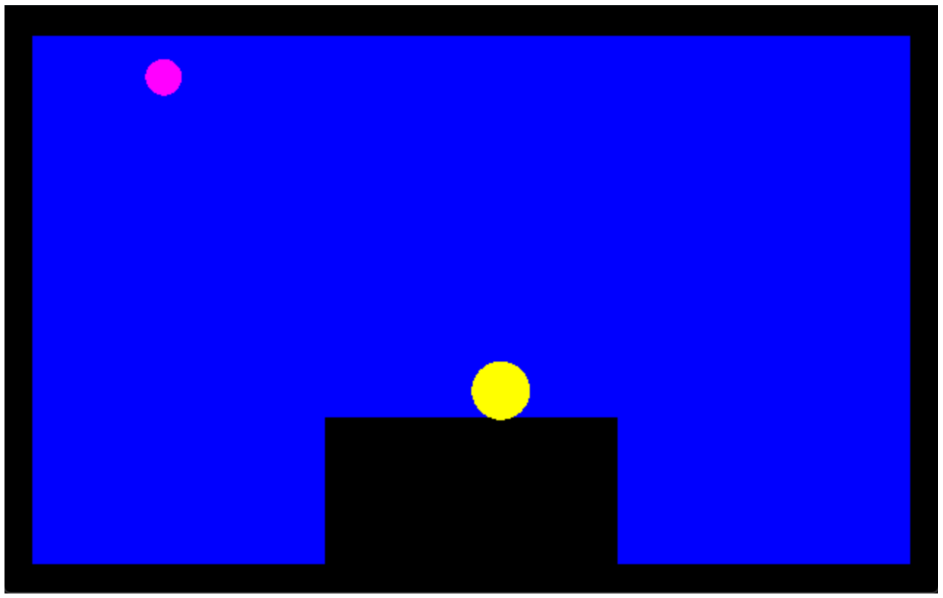
Figure 3: Example level of the proposed game

2.2 Basic definitions for reinforcement learning

To go deeper in how the project works, it's necessary to understand some basic concepts of Reinforcement Learning (Lima 2018). In general, in reinforcement learning, an algorithm has some knowledge of its task and must learn how to do it in the best way.

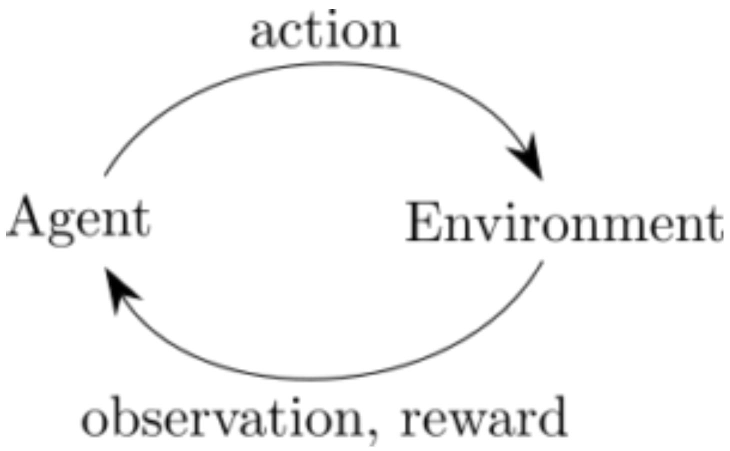

Figure 4: Loop Agent-Environment

Thus, the main actors in a Reinforcement Learning environment are described in Figure 4. The Agent, the one that will perform the specified task, chooses an action, this action will impact in the environment that he is in. The environment, based on the chosen action, gives an observation, that represents the current state of the environment, and gives a reward for that action. The goal of the Agent is to learn on how to get the best rewards. In the GeoFriends domain, the agent will be the circle player and the environment will be the current map that is being played. The observations and the rewards will be discussed in a forward section.

Additionally, to train and build an algorithm for the agent behavior in Reinforcement Learning environments, it is used Neural Networks structures.

2.3 OpenAI Gym Environments

The OpenAI Gym (OpenAI s.d.) is an open source library developed for Python programming language, with the goal of providing tools for developing and compare reinforcement learning algorithms. In this way, the library comes up with working environments for exploring the theme, but also specifies a developing format for custom environments. In this manner, the user provides the algorithm and the library helps with its training and execution.

Each OpenAI Gym environment works basically with three functions: reset, render and step. The reset one, as its name says, is responsible for restating the agent and the environment variables for the initial state in each end of episode. The render is in charge of rendering the game, it can be a game screen or even simple prints in a command terminal. Finally, the step function is in control of implementing the Agent-Environment loop shown in Figure 4. For each action taken, the step function interprets this action and returns four values: an observation, that indicates specifics and wisely selected characteristics of the environment; the reward given for that action; the information if the game is in a terminal state; and, finally, any other information that can be necessary.

More than that, it's important to talk about another two main concepts to construct and OpenAI environment. The first is what is called as observation space. The observation space is the information given to the agent in each step, in other words, is the only information that the agent knows about the environment. Choosing what information to put in observation space is the most sensitive part of developing in OpenAI, since that an environment space with few or too much information, or with inconclusive information directly interferes in the agent performance. The second one is what is called the action space. This variable represents what are the possible actions that the agent can take.

It is also important to say that each iteration of the loop Agent-Environment is called an episode and a collection of episodes is called epoch.

2.4 Observations and rewards

As it was said before, what to observe and how to reward the agent are fundamental in a good environment. For this work, a bunch of attempts were made in search of the best result.

Two formats of observation were tested:

1) Observe just the distance between player and diamond;
2) Observe the player and the diamond position in the map.

Each observation method was tested together with three different reward manners:

1) Reward 1 (one) just when the player collects the diamond;
2) Reward -1 if the player distance himself from the diamond and 1 if he collects it;
3) Reward 1 if the player gets closer to the diamond in that step, and -1 if he distance himself from it. Also, reward 1 if he collects the diamond.

After these tests, new features were proposed. The first one was to detect collisions between the player and walls and obstacles. With this type of detection, three tests were made:

1) Finish an episode if the player collides with any obstacle, including walls;

2) Don't finish the episode if detects a collision, just don't allow the player to move in the direction of the detected collision;

3) Punish the agent with a reward of -1 in case that he collides with an obstacle.

The second feature developed was gravity and a new form of collecting diamond reward. In summary, the agent was punished based on the steps that he made until collection the diamond. When the game started, the diamond worthed 1000, but when the player collected it, the reward given was 1000 - steps taken until there. In this way, the player got less reward as more steps as he needed.

2.5 Initial Position

For the first part, it was used a fixed initial position or a limited region for both player and diamond. For the final part, an entirely random initial position for both of them was added. Evidently, since this position wasn't outside the map or over an obstacle or wall.

2.6 Physics

The first physics simulation implemented was, as said before, the collision detection between player and obstacles (also walls). For it, it was used an external function (Thompson s.d.) that identifies collisions between circle and rectangle objects types of Pygame's library.

Gravity simulation was the second physics added to the game. In this case, instead of the player moves freely in map, there's a vertical constant force that makes it go the ground. In this type of environment, the algorithm would have to learn on how to deal with multiple jumps, instead of just taking the shorter path.

3 Results

3.1 Reinforcement Learning Concepts

The first propose used in this project was the PPO (Proximal Policy Optimization) algorithm. In this case of learning, the agent doesn't rely on previous iterations, consequently, the train took place very efficient and fast. Besides that, the fact that it just depends on the current state of environment, makes it difficult to the agent reach diamonds in specific situations. For example, in case that there is an obstacle between the agent and the diamond, as showed in Figure 5, the first one won't be able to achieve its goal, since that he will need to receive some negative rewards for dodging the obstacle and receive a higher reward when collecting the diamond. In other words, since he doesn't rely on previous actions, he will just try to optimize its path in that episode and don't accept punishments.

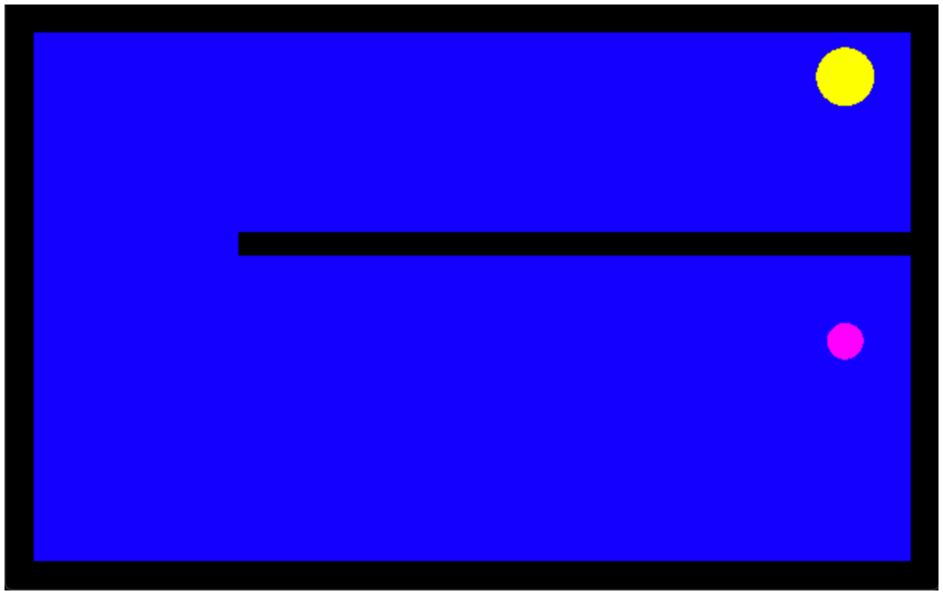

Figure 5: Obstacle between the agent and the diamond

Another attempt of algorithm was the DQN (Deep Q-Learning). In this type of learning, it directly relies on previous movements, which would probably lead to a different type of behavior, the agent would accept negative rewards in order to receive the honey pot at the end, since it knows that there is a bigger reward available. The problem presented by the PPO attempt in dodging some types of obstacles wouldn't exist in this case of training, since it considers previous events and movements. In the other hand, since the algorithm has to update its behavior considering all previous actions, this train turns to be much slower when compared to PPO.

3.2 Training The Agent

3.2.1. Observing Distance Between Player and Diamond

The first approach was to use PPO algorithm together with the following data in observation space: distance between player and diamond.

Initially it was used a reward of 1 just when the player collects the diamond. In this case, it was observed that the player couldn't learn how to collect the diamond at all. Since during the trainment, the agent had few or no contact with positive reinforcement, he doesn't know the best way the highest reward that he can get.

In order to correct this, the way of rewarding the player was changed to a -1 reward when he distances himself from the diamond and 1 when he collects it. With that, it was reached an agent that always tries to leave the map. Since that when he goes out the map the episode finishes, he doesn't get any negative reward, so he just prefers to finish the game as fast as possible.

Finally, one more attempt with another rewarding policy was made. In this one, the agent got reward 1 if he gets closer to the diamond in that step and -1 if he goes farther. Finally, collection the diamond would reward him with 1. In this iteration, it was noticed that the player moves to a horizontal close position of the diamond but can't collect it.

3.2.2. Player and Diamond Position

Since that none of the training until this point were satisfactory, a new observation was proposed, still using PPO. In this case, the player and diamond positions (coordinates for their center position) were observed.

The same was as it happened in the previous approach, the first reward method tested was to reward the player 1 just when he collects the diamond, but the final result was the same, the agent didn't have enough contact with positive reinforcement, so he doesn't learn to collect the diamond.

The second attempt, again, was to punish him with -1 when goes farther to the diamond and 1 when he collects it. Again, the same result. The agent always tried to leave the map as fast as possible.

The final and best result was using the reward policy of giving agent 1 if he goes closer to the diamond or collect it and -1 if he goes further. The result was finally satisfactory since he learns how to reach the diamond. Training with this version was also fast.

3.2.3. Training with obstacles

With a more functional model, the next step was to train the agent with obstacles. In this model, some changes in the reward policy were also made.

Initially, the same iteration that the agent had with walls he would have with obstacles, that is finishing the game when they collide. But the agent couldn't learn how to dodge the obstacles.

The next attempt was to not finish the episode if the agent collides with walls and obstacles. The final result was the same as the previous one, but in this case the agent just got stuck on the obstacle.

The third attempt was to punish the agent with -1 if he collides with the obstacle. But, again, the result was not satisfactory.

In order to try to make him don't stuck at the obstacle, a bigger reward for collecting the diamond was proposed, as it was said in end of the section 2.4. The objective here was to make the agent more tempted to collect the reward, so then he would indirectly accept negative rewards to achieve this big reward. Again, the result was satisfactory.

In the end, it was realized that these errors were more related to the way that PPO works rather than with the type of rewarding. It is directly related with the fact that he always perceives the positive rewards and don't accept punishments, as it was explained in Section 3.1. In cases that an obstacle obligates him to get negative rewards to dodge and then get the diamonds he won't do it, as Figure 5. In the other hand, if there's a path without negative rewards to collect the diamond, as showed in Figure 6, he would do it. This last behavior is not observed in the final implementation, since that the size of trainment wasn't big enough, so during it, the agent doesn't see himself in a situation like that very much. In order to that, a longer trainment is necessary.

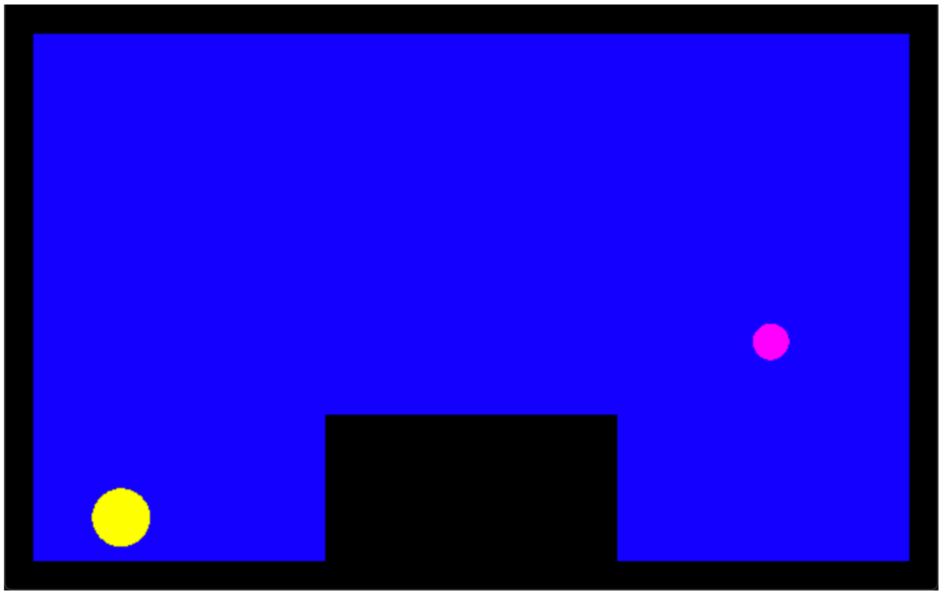

Figure 6: Obstacle between the agent and the diamond without negative rewards

3.2.4. Training with Gravity

Using the best results so far, a new trainment with gravity simulation happening was made. The result was satisfactory, since the player learns how to deal with jumping and with gravity forcing him to go lower.

3.3 Initial Position

In the first attempts, which both agent and diamond started in a fixed position, it was observed that the agent just learned a specific fixed route. In other words, instead of learning to reach the diamond, he agent just learned one single path, since that in all iterations, doing this path was sufficient to find the diamond. In this way, if the diamond was forced to start in a different position, the agent did this single path and didn't find the diamond.

In order to correct this mistaken behavior, it was thought to randomize both player and diamond initial positions. For this case, the train would be more loyal to the real game, making the agent of not just learning a single path, since that it can start in any position of the map. The result for this train has shown as being more efficient when compared to the first one, the agent learned to find the diamond in most of the cases.

4 Conclusion

In order to build an efficient and working environment for GeoFriends game in OpenAI Gym, it was possible to observe that different combinations of observation and rewards produced distinct results. Thus, it's necessary that agent knows the position of the diamond during the trainment, and not just the distance to it. The distance is more useful on rewarding the agent during his search. When these couple of specifications aren't implemented following these descriptions, the agent is unsuccessful in learning how to find the diamond.

Another observed aspect was that the randomness in agent and diamond initial positions is critical in the final result of trainment. When the positions are fixed, the agent just learns how to do one single path, while when they are random, the agent learns how to collect the diamond.

Furthermore, the type of algorithm used in the trainment is fundamental in the observed behavior. When using an algorithm like PPO, the agent won't learn how to dodge some types of obstacles, because of its way of operation. Meanwhile, using an algorithm like DQN, it is expected that the agent learns how to dodge the obstacle, but it demands a much bigger trainment for achieving a satisfactory result, which requests time and computational power.

Finally, the gravity implementation, despite the fact that it worked well, there are several improvements needed, since that the collision with obstacles doesn't work very well when the agent is jumping.

References

bluemoon93. 2019. "GeoFriends2-v2." *GitHub.* Latest commit 9de7316 on November 13, 2019. Accessed January 8, 2020. https://github.com/bluemoon93/GeoFriends2-v2.

GAIPS, INESC-ID. n.d. *Geometry Friends.* Accessed January 8, 2020. http://gaips.inesc-id.pt/geometryfriends/.

Lima , Cezar Augusto Cordeiro de. 2018. "Introdução à Reinforcement Learning com Deep Q-Learning." *Neoronio.ai.* November 18. Accessed January 8, 2020. https://medium.com/neuronio-br/introdu%C3%A7%C3%A3o-%C3%A0-reinforcement-learning-com-deep-q-learning-b3e7baa4e7a6.

OpenAI. n.d. *Gym.* Accessed January 8, 2020. https://gym.openai.com/.

Simões, David, Nuno Lau, and Luís Paulo Reis. 8-13 July 2018. "Guided Deep Reinforcement Learning in the GeoFriends2 Environment." *2018 International Joint Conference on Neural Networks (IJCNN).* Rio de Janeiro, Brazil: IEEE.

Thompson, Jeff. n.d. "Collision Detection: CIRCLE/RECTANGLE." *Jeff Thompson.* Accessed January 8, 2020. http://www.jeffreythompson.org/collision-detection/circle-rect.php.