Artificial Player for the Competitive Game Geometry Friends

Joao Afonso Cruz
Instituto Superior Tecnico
May 2017

ABSTRACT

This work has the goal of improving the artificial intelligence of a circle-shaped, whose objective is to collect all diamonds available on each level of the platform game called Geometry Friends. This game is part of the Artificial Intelligence competitions of the IEEE CIG Conference, where participants submit their agents to be tested in both single-player and cooperation levels. The solution for the implementation of these agents is based on a reinforcement learning approach (SARSA algorithm) and divides the problem of solving the game in three sub-problems: solving one platform, deciding the next platform and moving from one platform to another. The first and third sub-problems will be focused throughout this dissertation, where the agent will be tested in a set of simple but very different and symmetrical levels that focus typical challenges that the player faces in a normal level.

Keywords

Artificial Intelligence, Geometry Friends, Reinforcement Learning, Virtual Agent, Platform Game, SARSA Algorithm

1. INTRODUCTION

Geometry Friends is a two-dimensional platform game, where the player must collect all the diamonds in a room, within the least time possible. There are two different types of character which the player may choose to play the game, a circle character and a rectangle character, each one with different traits and skills. The game also includes a cooperative gaming experience where levels can be played by two players, each one taking one of the mentioned characters.

Geometry Friends is also physics-based, taking into consideration variables such as speed and gravity. All these features make for a more complex challenge for the player, which turns the game into a good system to test Artificial Intelligence algorithms. For this reason, Geometry Friends has been featured in the game AI competitions of IEEE CIG Conference in 2013, 2014 and 2015. In this competition, agents controlling the circle and the rectangle characters are tested through various levels, where half of them are public through the submission period, while the rest is unknown to the participants in the competition. Historically, existing algorithms for the Geometry Friends agents operate with much higher success in public levels than in unknown levels. The main reason for this would be the approach that is made to the problem, being over-specialized and not being able to adapt in a general way to the different challenge possibilities each level could impose. Since most of those architectures were based on path-planning algorithms, the solution for a better performance in unknown levels should go through an approach that divides the whole level in smaller problems that can be generalized and solved separately, like a divide-and-conquer strategy.

The approach for this work consists in splitting the level in sub-problems which can be of three different types: solving a single platform, choosing the next platform to move and moving from one platform to the other. To solve any of these sub-problems, the agents will be using Reinforcement Learning algorithms. Both the circle and the rectangle agents will behave individually, as it is important that their actions are independent and not tied with each other in terms of shared internal information.

One of the main problems with this Reinforcement Learning approach, when checking past work, is that the learning process seems uneffective. This could possibly happen due to the failed attempt of the agent to match previous trained scenarios with the new found ones and that could be caused by a faulty representation of the internal state of the system. This problem is related to the granularity level that is applied to the agent, when dividing the various sub-problems and deciding when a new challenge can be solved as a previously trained sub-problem.

Having the above in mind, the implemented agents will have a training period through simple levels, increasing their complexity depending on their results. These levels will explore very distinct and symmetrical situations, with the objective of preparing the agent for various types of different platforms, regarding their features.

2. GEOMETRY FRIENDS

Geometry Friends[1] is a platform computer game in 2D environment for up to two players, with a high cooperative component. It features simulated physics (Farseer Physics Engine), where gravity, mass and attrition affect the characters and, each level, the players have the goal of gathering all the diamond-shaped tokens throughout the room in the least amount of time possible.

There are two available characters that can be played, a yellow circle and a green rectangle. Each players controls one

[1]http://gaips.inesc-id.pt/geometryfriends/

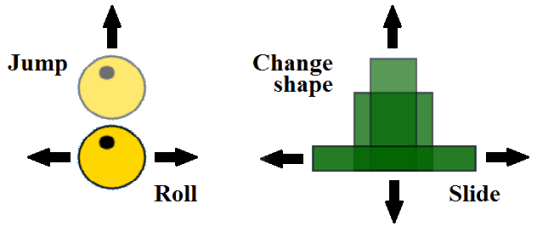

Figure 1: Geometry Friends Character Actions

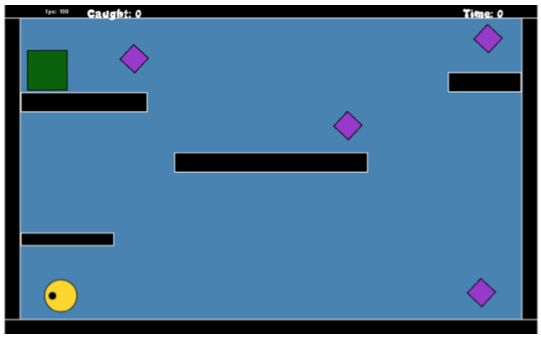

Figure 2: Geometry Friends Sample Level

of these characters, and each of them has its own movement restrictions and actions that it can perform. Due to the physics elements implemented in the game, the outcome of this actions can be complex and collisions with the other character are also a feature.

The circle character can roll both ways, left and right, and also jump. since the characters have acceleration, the jump is influenced by the circle's speed. The rectangle can slide both ways and it can change its form, whether becoming taller or larger, but always maintaining its initial area.

The game can have as many levels as imaginable, through the level editor tools, where the setting of the obstacles and the initial position of each character can be different.

There are also three different types of platforms in the game, with different characteristics. The black platforms block the path of either the circle and rectangle characters. The yellow platforms block the path of the rectangle (green character) and the green platforms block the path of the circle (yellow character). This kind of setting also creates a more complex environment and increases the level difficulty.

Levels consist of a variable number of diamonds which characters must collect. Depending on the type of level, some diamonds may only be available to attain though cooperation[6] and coordination between both the circle and rectangle characters. Also, it happens that some diamonds must be collected before others, as the path to one of those could make it impossible to reach a previously attainable diamond.

3. RELATED WORK

Reinforcement learning has gained attention and extensive study in recent years. As a learning method that does not need a model of its environment and can be used online, reinforcement learning is well suited for multiagent systems, where agents know little about other agents, and the environment changes during learning. When an agent carries out an unknown task for the first time, it does not know exactly whether it is making good or bad decisions. Over time, the agent makes a mixture of optimal, near optimal, or completely suboptimal decisions. By making these decisions and analyzing the results of each action, it can learn the best actions at each state in the environment, and eventually discover what the best action for each state is.

Applications of reinforcement learning in multiagent systems include football pursuit games and coordination games. In most of these systems, single-agent reinforcement learning methods are applied without much modification. Such approach treats other agents in the system as a part of the environment, ignoring the difference between responsive agents and passive environment [3].

Relational representations in reinforcement learning allow for the use of structural information like the presence of objects and relationships between them in the description of value functions. Such representations allow for the inclusion of background knowledge that qualitatively describes a state and can be used to design agents that demonstrate learning behavior in domains with large state and actions spaces such as computer games [5].

A reinforcement learning agent is assumed to select actions following a mapping of each possible environment state to an action [1]. This mapping of states to actions is called a policy, and reinforcement learning algorithms aim to find the optimal policy for an agent, that is, a policy that ensure long term optimal rewards for each state. RL techniques are divided into two types, depending on whether the agent changes acts on the knowledge gained during policy execution. In passive reinforcement learning, the agent simply executes a policy using the rewards obtained to update the value of each state, whereas in active reinforcement learning, the agent uses the new values to change its policy on every iteration of the learning algorithm. A passive agent has fixed policy: at state s, the agent always performs the same action a. Its mission is to learn how good its policy is, to learn the utility of it. An active agent has to decide what actions to take in each state: it uses the information obtained by reinforcement learning to improve its policy. By changing its policy in response to learned values, an agent might start exploring different parts of the environment. Nevertheless, the initial policy still biases the agent to visit certain parts of the environment, so an agent needs to have a policy to balance the use of recently acquired knowledge about visited states with the exploration of unknown states in order to approximate the optimal values [4].

Carlos Fraga was the first person developing artificial intelligent agents for Geometry Friends [2]. His solution used a navigational graph and its development was prior the first edition of the competition and was fully focused in the cooperative environment of Geometry Friends.

The graph has different types of edges depending on whether the edge is traversable by the circle alone, by the rectangle alone, or by both characters. The nodes of the graph are positioned on both agents starting positions and on the positions of the diamonds. These initial nodes are expanded, generating other nodes throughout the level. When the agent is about to perform an action, it determines its path through the A* algorithm. After deciding which path to take, that path is divided into a set of actions, which are a series of movements that will allow the agent to reach specific

nodes of the graph that are on the path. Upon completing each action, the agent checks if it reached the planned action outcome. If it had done so, the agent repeatedly proceeds to next action on the task until that task is fulfilled. Whenever an action is unsuccessful or the task is finished, the agent calculates the next task performed. The main downside of this approach is the processing overhead caused by running the A* algorithm every time the agent has to calculate a task. Another problem is that each agent is only able to cooperate with another agent sharing the same algorithm, which imposes a significant limitation when playing with an arbitrary team-mate.

4. IMPROVING THE RL APPROACH

4.1 The SARSA Algorithm

The algorithm to solve this problem is based on the SARSA (State-Action-Reward-State-Action) algorithm, which is an algorithm for learning a Markov decision process policy, used in the reinforcement learning area of machine learning. With SARSA, the agent starts in state 1, performs action 1, and gets a reward (reward 1).Then, the agent is in state 2 and performs another action (action 2) and gets the reward from this state (reward 2) before updating the value of action 1 performed in state 1. It is different than Q-learning since, in the latter, the agent starts in state 1, performs action 1 and gets a reward (reward 1), and then looks and sees what the maximum possible reward for an action in state 2, and uses that to update the action value of performing action 1 in state 1. So the difference is in the way the future reward is found. In Q-learning it is simply the highest possible action that can be taken from state 2, and in SARSA it is the value of the actual action that was taken.[7]

4.2 The World Features

The world features are responsible for taking the information available from the current state and filter its data into more condensed data that maintains the relevant information that the agent needs to succeed in the current level.

The internal representation of the world uses continuous values for time, distances and speed, and that coincides with the information the game provides to the agents as well. Since having continuous values dramatically decreases the likelihood that two similar situations will be considered the same (because of infinitesimal differences), other methods of filtering the data must be used, in order to keep it relevant to consider two states the same, when the next action is the best action for both states for the same logical reasons.

4.3 Description of the Features

The features for solving a platform that were put up to test were the following:

- Distance to the left and right edge of the platform: These two features are very important, as it is vital that the agent registers his location in the platform that it is solving. It is not that important to know the platform's width, but it is important that the agent senses when he is near the edge of the platform so it can avoid falling. With sufficient training, it can even potentially know how much he can accelerate or even jump without falling. This way, every time the agent is near an edge, it will have experienced that situation before, since the feature is that remaining distance to a potential fall and is not related to the platform's width or height.

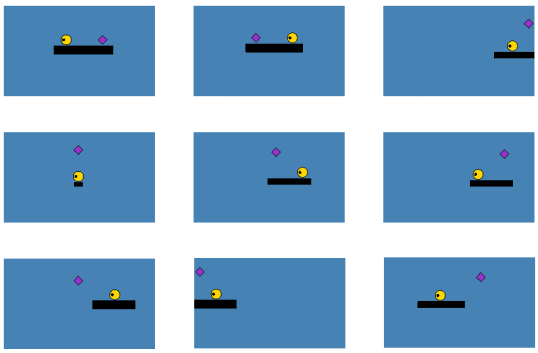

Figure 3: Training Level Set for Sub-Problem 1

- The agent's current horizontal speed: This feature allows the agent to predict in a easier way where its next locations will be. For example, if the agent has current horizontal speed directed to the right, even if not currently moving to the right (but from past actions), for example, with this feature it can perceive that choosing the action to move to the left will not necessarily make it nearer the left edge of the platform, but instead only decelerate its movement to the right. Also, this feature helps when deciding to make a jump, as it influences the jumping trajectory and landing position.

- Number of tokens remaining in the platform: A basic but vital feature which allows the agent to perceive when a platform has been solved.

- Horizontal and vertical distance to the closest token in the platform: This feature allows the agent to perceive if its actions are putting it closer or farther to its main goal: collecting that token.

- Presence of a wall in the platform's left or right edges: with these two features, one for each edge, the agent can distinguish near-edge experiences with and without walls present. Such situations influence the outcome of moving towards an edge or the trajectory a jump takes.

4.4 Training Set

The levels were designed with the specific purpose of teaching the agent what it needs to know to solve most types of platforms. Solving these levels, although seeming basic and easy, would presume the solving of different and more complex platforms because of its connection with the features (that will be described next). Many situations might be similar enough for the agent to consider them the same and therefore simply adding new levels that add a slightly new variation of a known problem probably does not bring any significant improvement to the overall performance of the agent. It is also important to consider the training time. The more levels added to the training set, the more time it will take the agent to train, as reducing the number of times the agent goes through each individual level might affect the quality of the lessons the agent takes from them. It is important that the agent runs each level a significant number of times, so that the learning is not significantly influenced by the non-deterministic nature of both the agent and the environment.

4.5 Exploration versus Exploitation

Finding a good balance between exploration and exploita-

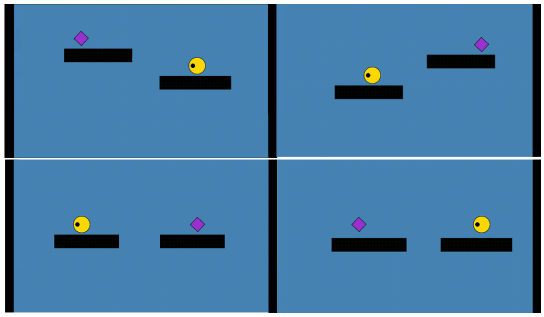

Figure 4: Training Level Set for Sub-Problem 3

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	0	0	0	227	0	227	241	0	242	242	227	227	227	241	227
Avg. Collectibles	0	0	0	1	0	1	1	0	1	1	1	1	1	1	1
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	10	10	10	3,6	10	3,6	2,9	10	2,9	2,9	3,6	3,6	3,6	2,9	3,6

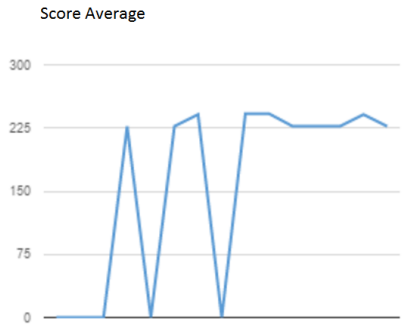

Figure 5: Results for level 1

tion is an important part of any reinforcement learning approach. Each time the agent has to choose the next action to take, it has to decide whether to try an approach it knows or whether it should try something new. Since this work separates training sessions with results testing, it is possible to separate this two distinct approaches in a favorable way. Therefore, when in training the agent calculates the next best action to perform with its Q-function, but has a 40% chance of ignoring this calculation and just randomly choose its next action and exploring new ground. On the other hand, when testing the results of the agent's training, the randomization is decreased to 0% and the agent exploits its full knowledge into solving the given scenarios.

5. RESULTS AND DISCUSSION

The agent's learning was carried out through one thousand and five hundred runs of the training level sets, in batches of one hundred runs at a time at normal speed. In each batch, the agent run consecutively from level 1 to level 9, whereas at the end of each batch, a set of thirty run was carried out with the same levels to check the agent's evolution with the amount of training accomplished. The same was done in parallel for the sub-problem 3 batch, having the agent run consecutively from level 10 to 13, the same amount of times.

For sub-problem 1, the agent seems to adapt to the major-

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	111	202	243	0	241	244	244	242	242	239	244	244	242	242	239
Avg. Collectibles	0,7	0,9	1	0	1	1	1	1	1	1	1	1	1	1	1
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	6,4	3,8	2,8	10	2,9	2,8	2,8	2,9	2,9	3,1	2,8	2,8	2,9	2,9	3,1

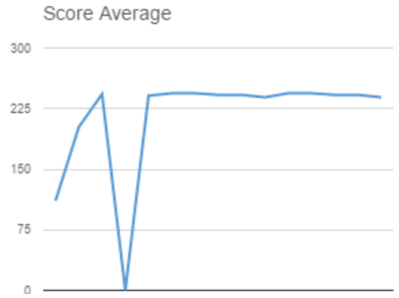

Figure 6: Results for level 2

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	0	254	253	218	0	253	249	254	254	254	254	254	254	249	254
Avg. Collectibles	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	10	2,3	2,3	4,1	10	2,3	2,5	2,3	2,3	2,3	2,3	2,3	2,3	2,5	2,3

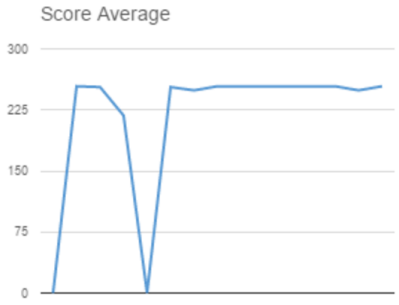

Figure 7: Results for level 3

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	272	272	272	272	272	44	272	272	272	272	272	100	272	272	272
Avg. Collectibles	1	1	1	1	1	0,4	1	1	1	1	1	0,6	1	1	1
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	1,4	1,4	1,4	1,4	1,4	9,8	1,4	1,4	1,4	1,4	1,4	5,2	1,4	1,4	1,4

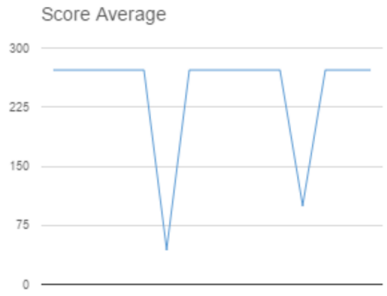

Figure 8: Results for level 4

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	272	272	272	272	272	44	272	272	272	272	272	100	272	272	272
Avg. Collectibles	1	1	1	1	1	0,4	1	1	1	1	1	0,6	1	1	1
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	1,4	1,4	1,4	1,4	1,4	9,8	1,4	1,4	1,4	1,4	1,4	5,2	1,4	1,4	1,4

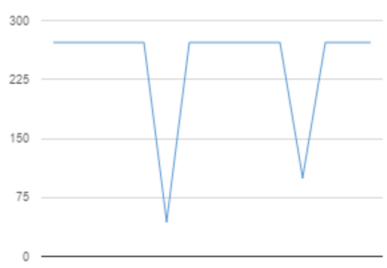

Figure 9: Results for level 5

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	272	272	272	272	272	44	272	272	272	272	272	100	272	272	272
Avg. Collectibles	1	1	1	1	1	0,4	1	1	1	1	1	0,6	1	1	1
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	1,4	1,4	1,4	1,4	1,4	9,8	1,4	1,4	1,4	1,4	1,4	5,2	1,4	1,4	1,4

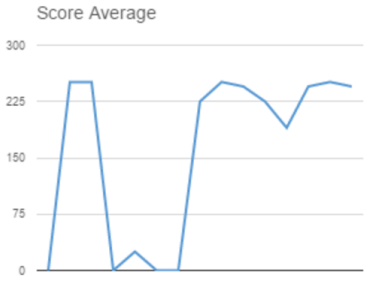

Figure 10: Results for level 6

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	226	0	0	152	0	251	251	251	252	251	251	251	252	252	252
Avg. Collectibles	0,9	0	0	1	0	1	1	1	1	1	1	1	1	1	1
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	3,2	10	10	7,4	10	2,4	2,4	2,4	2,4	2,4	2,4	2,4	2,4	2,4	2,4

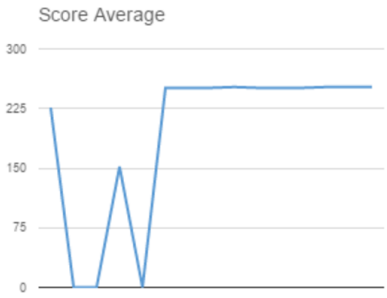

Figure 11: Results for level 7

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	0	0	0	34	93	23	0	0	93	93	150	120	120	130	130
Avg. Collectibles	0	0	0	0,3	0,5	0,1	0	0	0,5	0,5	0,7	0,6	0,6	0,6	0,6
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	10	10	10	9,7	9,3	9,8	10	10	9,3	9,3	8,0	9,0	9,0	8,7	8,7

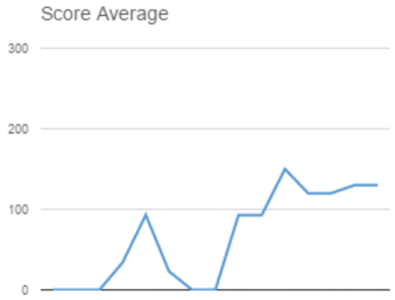

Figure 12: Results for level 8

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	0	0	0	231	0	0	0	0	232	232	231	231	0	232	232
Avg. Collectibles	0	0	0	1	0	0	0	0	1	1	1	1	0	1	1
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	10	10	10	3,5	9,3	9,8	10	10	3,4	3,4	3,5	3,5	10	3,4	3,4

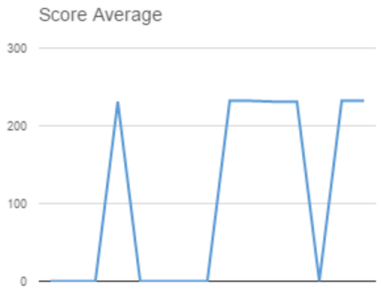

Figure 13: Results for level 9

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	32	40	22	22	40	41	105	104	105	120	76	104	104	121	105
Avg. Collectibles	0,2	0,3	0,1	0,1	0,3	0,3	0,5	0,5	0,5	0,6	0,4	0,5	0,5	0,6	0,5
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	9,8	9,6	9,9	9,9	9,6	9,6	8,8	8,8	8,8	8,4	9,2	8,8	8,8	8,4	8,8

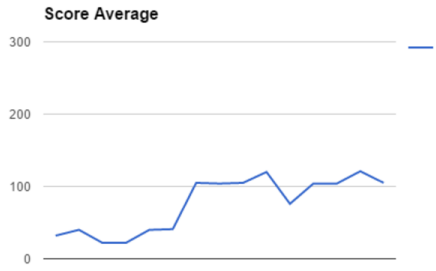

Figure 14: Results for level 10

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	0	0	42	42	25	25	26	42	75	75	32	120	105	120	120
Avg. Collectibles	0	0	0,3	0,3	0,2	0,2	0,2	0,3	0,4	0,4	0,2	0,6	0,5	0,6	0,6
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	10	10	9,5	9,5	9,7	9,7	9,7	9,5	9,2	9,2	9,7	8,8	8,9	8,7	8,7

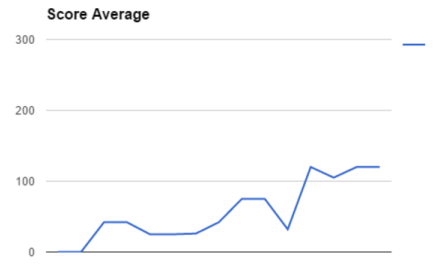

Figure 15: Results for level 11

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	39	39	224	222	222	222	154	186	187	187	224	224	220	221	224
Avg. Collectibles	0,3	0,3	1	1	1	1	0,8	0,9	0,9	0,9	1	1	1	1	1
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	9,6	9,6	4,1	4,2	4,2	4,2	7,6	8,1	8,1	8	4,1	4,1	4,2	4,2	4,1

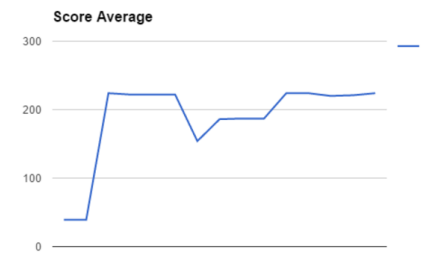

Figure 16: Results for level 12

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Avg. Score	108	109	0	0	0	123	230	225	225	169	170	226	220	226	226
Avg. Collectibles	0,5	0,5	0	0	0	0,6	1	1	1	0,9	0,9	1	1	1	1
# Runs	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
Time Limit	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Avg. Time	8,8	8,9	10	10	10	8,6	4	4,1	4,1	7,8	7,8	4,1	4,3	4,1	4,1

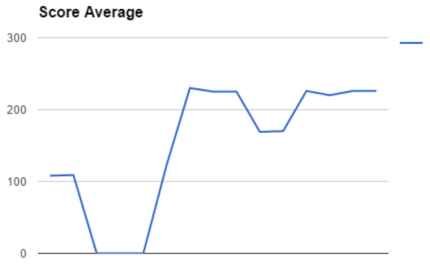

Figure 17: Results for level 13

ity of the levels by the end of the ten batches. The graphics in the figures present good scoring values (the conjunction of the collected tokens with the speed they got caught with) that start to stabilize in general from the batches 7, 8 and so on. It is interesting to verify the impact of the training in the symmetrical levels, namely in the pairs of levels 1/2 and 3/4. In the first case, the results are notorious during the batches 1, 2, 3 and 4, where it is evident that the agent's action of moving to the left overrides, in its decision process, the action of moving to the right in the first three batches, until it reverses abruptly in batch 4 and stabilizes in batch 5. In the second case, the exact same situation is noticed, where the action of moving to the right is preferred by the agent to the action of moving to the left, during the first batch. That tendency reverts during the batches 2 and 3, reversing again in batches 4 and 5 until finally stabilizing from batch 7 and on.

The levels 8 and 9 were the levels where the agent had the worst performances, mainly in the 9th level. In a way, those are also the levels that differ the most from the rest of the training set, since they involve tokens that aren't directly on top of the platforms but, in certain situations (depending on the scenario), it will be only through those platforms that the agent will be able to collect them.

The sub-problem 3 is tested under different circumstances. The number of levels is much lower, which makes it much easier for the agent to learn more specific tasks. On the other hand, it is a more complex sub-problem, so the results actually seem worse than those of solving a platform. To decide whether this task was completed, the agent still required to collect the diamond in the target platform, making it obviously a harder set of levels.

Some reasons could be behind the lesser success of the agent in solving some platforms, namely the last pair of levels, such as:

- The features: even though the results are positive, it is possible that some new features could be used to solve a bigger number of issues. For example, a feature telling the agent if there's another platform or the ceiling blocking it in case of a jump could be useful in some cases. Or a feature that informs the agent if it is in free fall from a jump could help it deciding his next action, since the time step of the agent's decision making has a fixed value between actions and, therefore, a new action decided while the agent being in free fall can jeopardize the expected results and consequently its learning. Also, the scalability of the features could use some adjustments. If all features were just flags with true and false values, for example, its scales would be perfect. But in these case, we have numbers of tokens in a platform, distances and agent's speed values all mixed in the Q-function calculation, which makes it more difficult for the action's weights to be balanced and accurate.

- The values of the algorithm constants, namely the reward values, which only are given when the agent collects a token. A possible improvement could be a policy of negative rewards when, for example, the agent falls from a platform.

- The time steps for the agent's actions also have room for calibration, as Geometry Friends is a continuous game and its states and physics are very unpredictable.

Even with all these options, a key requirement to execute them will always be time. It is difficult to make many tweaks at still expect serious results, as these Q-learning algorithms require time and exhaustive training.

Having the above data in mind, the results seem to confirm a positive feedback from the training set, as the experimented features and SARSA algorithm seem to make for a strong baseline for an agent to be capable of solving levels with bigger complexity and difficulty.

6. CONCLUSIONS AND FUTURE WORK

This work's goal was to improve and continue the work on the existing studies on the agents for the platform game Geometry Friends. This work goes through the basics of the Geometry Friends' game mechanics and objectives, followed up by an analysis on the current state-of-the-art of artificial intelligence in games, more specifically regarding reinforcement learning. It then follows the previous work's theory on solving the problem at hand through a divide-and-conquer approach that splits a Geometry Friends' problem into three different sub-problems (solving one platform, planning the next platform and moving to another platform). The focus is placed on the first and third sub-problems, as it is important to build strong base that allows for the agents to solve platforms in a reliable and methodical way. This base is made from a set of features and a reinforcement learning algorithm (SARSA) that were tested from set of Geometry Friends' scenarios and its results for the circle agent are quite satisfactory and seem to make for a strong baseline to more complex scenarios. There's room for improvement and future work on the sight, but time will always be an important requirement for problems of this nature.

6.1 Extrapolation for the Rectangle

Most of what was done for the circle can also be used for the rectangle with some adaptations, such as the training set, since the rectangle will be presented with different challenges and tasks (instead of jumping like the circle, the rectangle morphs its shape). One important fact to have in mind is that when the rectangle leaves a platform, it won't be able to go back to it, which puts more emphasis on the planning aspect. Other particular challenges presented to the rectangle are gathering enough momentum to go over the gaps between platforms, changing its shape to climb a step and changing its shape to go under an obstacle.

The features for the rectangle agent could essentially be the same as the features for the circle. One obvious difference is that the rectangle needs to keep track of its shape. This can be done simply by keeping track of its height, for example, since the area does not vary when the rectangle changes shape.

6.2 Finishing the Divide-and-Conquer Approach

With the first and third sub-problems tackled, there seem to be the right conditions to approach the remaining sub-problem. Planing the next platform will pass through an implementation of a node search (Depth First Search, for example), compatible with the current updated Geometry Friends' framework, that allows the agent to choose the best path to traverse in the best time possible, without compromising the accomplishment of the whole level with success.

REFERENCES

[1] Y. Björnsson, V. Hafsteinsson, Á. Jóhannsson, and E. Jónsson. Efficient use of reinforcement learning in a computer game. *Proceedings of International Journal of Intelligent Games & Simulation*, 2008.

[2] C. de Sousa Fraga. *Motion Control for an Artificial Character teaming with a Human Player in a Casual Game*. PhD thesis, INSTITUTO SUPERIOR TÉCNICO, 2011.

[3] J. Hu, M. P. Wellman, et al. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML*, volume 98, pages 242–250. Citeseer, 1998.

[4] J. Maissiat and F. Meneguzzi. Adaptive high-level strategy learning in starcraft. In *Proceedings of the SBGames conference on Computing*, 2013.

[5] S. Mohan and J. E. Laird. Relational reinforcement learning in infinite mario. *arXiv preprint arXiv:1202.6386*, 2012.

[6] J. B. Rocha, S. Mascarenhas, and R. Prada. Game mechanics for cooperative games. *ZON Digital Games 2008*, pages 72–80, 2008.

[7] G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.