

# Building a Multi-Agent Learning System for Geometry Friends

Ricardo Ari Sequeira  
ricardo.ari.sequeira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2019

## Abstract

In this paper, we propose a consistent and cooperative multi-agent learning system for the Geometry Friends game, inspired by the single agent success using a weighted directed graph and Reinforcement Learning. The motivation is to build a solid foundation for future cooperation solutions that want to expand and exploit other Machine Learning knowledge. The final results will demonstrate that our system outperforms all proposals submitted to date, based on a relative simple structure giving the complex demands of a multi-agent environment.

**Keywords:** Multi-Agent System, Reinforcement Learning, Cooperative Agents, Geometry Friends

## 1. Introduction

Artificial Intelligence (AI) is a Computer Science field that has recently gained a notorious fame in more traditional and popular media. Games have a fundamental role in encouraging AI research, particularly Machine Learning (ML) techniques. Not only that, but they are teaching tools with dedicated disciplines and books [8] [5]. Classic board games like checkers and chess were the first games, seen as the most natural challenge due to their determinism and finite state/rule set. It was like that until artificial agents learned to play backgammon [13] in 1995 and the chess world champion was defeated by machine Deep Blue (IBM) in 1997 [2]. The focus then shifted to 2D games and later 3D real-time ones.

Famous Atari games, such as Super Mario Bros., became the next goal for new ML strategies. However, even these are now trivial given new knowledge and newer computers [11][6][3][14]. People began to study real-world problems in which they came across a majority of multi-agent systems (MAS). These represented a big leap in problems' complexity, but once again, games took over as the best playground to make that transition from single-agent problems to multi-agent ones.

Geometry Friends (GF) [7] is one good example, with its own competition aiming to encourage the development of cooperative and non-cooperative agents. Currently, an adaptation of A\* algorithm called Subgoal A\* [4], is the best shortest-path search algorithm applied to GF. As for the single-player mode, Reinforcement Learning [12] and Rapid-Exploring Random Trees (RTT) [10] [9]

are the most successful strategies, achieving the best performance on the respective competition. The multi-player component of GF has not taken so many approaches (RTT is also the best submission in this aspect) but it is expected to be addressed by more future proposals.

We intend to match those expectations by building a multi-agent system inspired by the good single-player results achieved by RL. Our idea is to design an architecture with a solid mapping, planning and control phase that supports the necessary cooperation mechanisms. We will achieve this through strategies that have already been successful in this game, such as weighted directed graphs and  $Q$ -Learning [15] [16]. Additionally, we will implement new communication and synchronization processes between agents.

### 1.1. Objectives

Our first objective is to develop a rectangle agent with the same approach as the KIT circle agent (Section 2.2.2), achieving the same quality in performance but on the Rectangle Track (ideally winning it). However, our main goal is to successfully join those two agents into a learning system equipped with the necessary cooperation mechanisms. We intend to compete in the Cooperation Track and significantly improve the results obtained by previous proposals. Attached to these objectives, we also assume responsibility for implementing a modular and organized system architecture to contribute to future Geometry Friends research.

## 2. Background

### 2.1. Q-Learning

The best way to model a RL problem is as a Markov Decision Process (MDP) [1], which implies defining states, actions and rewards. The goal is to train the agent to choose actions that not only reach the target but also maximize the reward (optimal policy). We are going to use one of the most famous RL techniques: *Q*-Learning [15] [16]. The result of this algorithm is a *Q*-Table in which each entry represents the quality of choosing an action *a* in state *s*. During training, these entries are updated according to Equation 1.

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a)) \quad (1)$$

Regarding Equation 1,  $\alpha$  is the learning rate,  $\gamma$  is the discount rate,  $r$  is the action reward,  $s'$  and  $a'$  correspond to the next state and the best next state's action, respectively. Choosing actions during training depends on the learning policy. The  $\epsilon$ -greedy strategy is one of the most common because it is simple and ensures convergence to the optimal policy. The idea is to choose the best action of the current state (greedy action) with probability  $\epsilon$  or a random one with probability  $1 - \epsilon$ .

### 2.2. Geometry Friends

Geometry Friends (GF) is a two-dimensional physic-based puzzle game with one and two player mode. There is a yellow circle that rolls and jumps, while a green rectangle slides and morphs without changing its area (see Fig. 1).

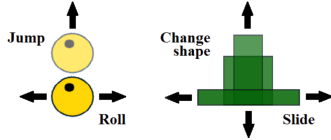


Fig. 1: GF Characters and Actions

The game objective is to collect all diamonds, which vary in number and position depending on the level (exemplified in Fig. 2). The remaining elements of the game are obstacles that prevent characters from moving unless they share the same colour. This means that the yellow circle can't get through black and green obstacles while the green rectangle is obstructed by black and yellow ones.

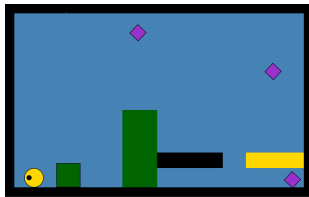


Fig. 2: Geometry Friends Level Example

Since 2013, a competition is held every year at the IEEE Conference on Games, challenging AI players to play GF. The main goal is to promote the study of solutions for agent-agent cooperation problems, from action coordination to action planning. There are three distinct tracks: Circle, Rectangle and Co-operation Track. The first two encourage each agent development in single-player levels while the latter evaluates the performance of multi-agent cooperative systems throughout more complex levels. All participants are tested ten times in ten different levels (only half public before submissions). The ranking is based on the average score of the ten runs, whose formula is given by Equation 2.

$$V_{Completed} \times \frac{(maxTime - agentTime)}{maxTime} + (V_{Collect} \times N_{Collect}) \quad (2)$$

The more diamonds collected,  $N_{Collect}$ , in less time,  $agentTime$ , the higher the score. Completing a level is rewarded with  $V_{Completed} = 1000$  and each collected diamond with  $V_{Collect} = 300$ .

#### 2.2.1. Subgoal A\* Search Algorithm (2015)

In 2015, Daniel Fischer turned the game levels into weighted directed graphs and tested different versions of A\* search. The result of his research was a Subgoal A\* algorithm that still remains the best pathfinding strategy for GF.

The original algorithm processes a weighted graph and works on a closed and an open list of nodes, according on whether they have been expanded or not. Starting from the initial node, the strategy is to find the minimal-cost path to one goal node. Each iteration, the node from the open list with lowest sum of cost and heuristic is expanded before being placed in the closed list. This means that its unexpanded neighboring nodes are inserted into the open list.

The Subgoal A\* proposal uses Euclidean distance as cost measure and goes without heuristics (no estimation of future cost) but the main characteristic is the node property associated to the path diamond collecting order.

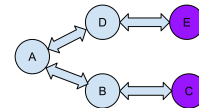


Fig. 3: Graph example in which Subgoal A\* returns the path with nodes A-D-E (all without diamonds), then E-D-A-B-C (all with diamond collected in E)

#### 2.2.2. KIT Circle Agent (2017)

The best circle agent (KIT) splits GF in subproblems. The first one is modeled in Fig. 4 and is solved using *Q*-Learning. The goal is to reach a certain position and horizontal velocity, using only

ROLL LEFT and ROLL RIGHT as possible actions. States are defined by discrete values of relative velocity and distance regarding target. The state search space is reduced given the existing symmetry in states like those shown in Fig. 4.

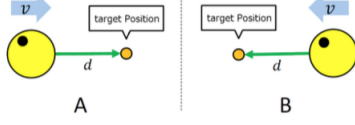


Fig. 4: KIT Learning Problem (Symmetric States)

The rest of the GF problem is divided into Planning and Control. Planning consists in using Subgoal A\* to find the shortest path to collect all diamonds. It includes building a weighted directed graph with platforms as nodes and moves as edges. For falling and jumping trajectories simulation, KIT uses the Motion Equations 3 and 4.

$$x = v_x t \quad (3) \quad y = v_y t - \frac{1}{2} g t^2 \quad (4)$$

To perform the path obtained in Planning, the Control phase selects the current state's best action, according to the agent training results ( $Q$ -Table).

### 3. System Overview

Our approach to GF subproblems and system architecture is organized in 3 phases: Training (on training levels), Mapping (pre-level) and Control (iterating until level completion or timeout).

#### 3.1. Circle & Rectangle Approach

Phase	Rectangle Approach (KIT Adaptation)
<b>Tra.</b>	Solving the subproblem of reaching a particular position with a certain horizontal velocity and height, using $Q$ -Learning.
<b>Map.</b>	Representing the level as a weighted directed graph with Platforms as nodes and Moves as edges, adapted to rectangle. Fall trajectories simulated using Motion Equations 3 and 4.
<b>Con.</b>	Running Subgoal A* on graph to obtain the next move. The action is selected according to the move and the training records.

Table 1: Rectangle Approach (KIT Adaptation)

Inspired by the KIT agent individual success, we built a system with a similar structure in order to incorporate it as our circle agent (without changing it). This is evident in Table 1, where we describe our rectangle approach. The complexity of the rectangle problem, in particular during Training and Mapping, is greater the more forms/heights we consider. After analyzing different levels and studying which

forms did not increase the character's efficiency, we narrowed them down to the ones shown in Fig. 5.

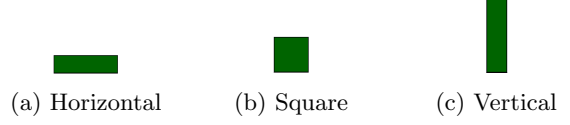


Fig. 5: Rectangle Forms/Heights Considered

#### 3.2. Cooperation Approach

The cooperative component of GF significantly increases the problem complexity. Similar to the rectangle, we found it necessary to simplify and reduce the coverage of cooperative movements by our system. We noticed that most of these could be viewed as a single riding ability whose variants we show in Figure 6. Thus, the Ride Position (Fig. 6a) became the basis of all our cooperation approach.

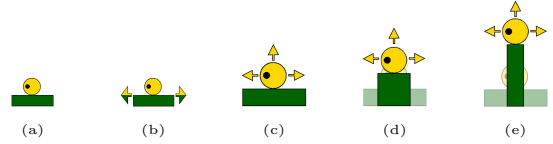


Fig. 6: Riding Ability

Besides a multi-agent training, implementing this type of moves has two main requirements: identification and incorporation in the agents' graphs (Mapping) as well as synchronization mechanisms between rectangle and circle (Control).

The latter would be solved if both agents built an equal graph and selected simultaneously the same cooperative move. However this solution is heavy, repetitive and ultimately unnecessary given the existence of a communication channel between agents in GF. Therefore, we decided to use a leader-follower strategy in which the leader makes the cooperative decisions and communicates to the follower. The circle was chosen as system leader since it is easier for this agent to check possible trajectories involving the Ride Position.

Our cooperation approach is detailed in Table 2.

Phase	Cooperation Approach	
	Circle(Leader)	Rectangle(Follower)
<b>Tra.</b>	Joint training for riding ability. Additional $Q$ -Table obtained with Team $Q$ -Learning.	
<b>Map.</b>	Identification and graph incorporation of cooperative moves based on the riding ability.	-
<b>Con.</b>	Sends cooperative moves.	Receives cooperative moves.
	Consult individual or team $Q$ -table records according to Move type.	

Table 2: Cooperation Approach

### 3.3. System Architecture

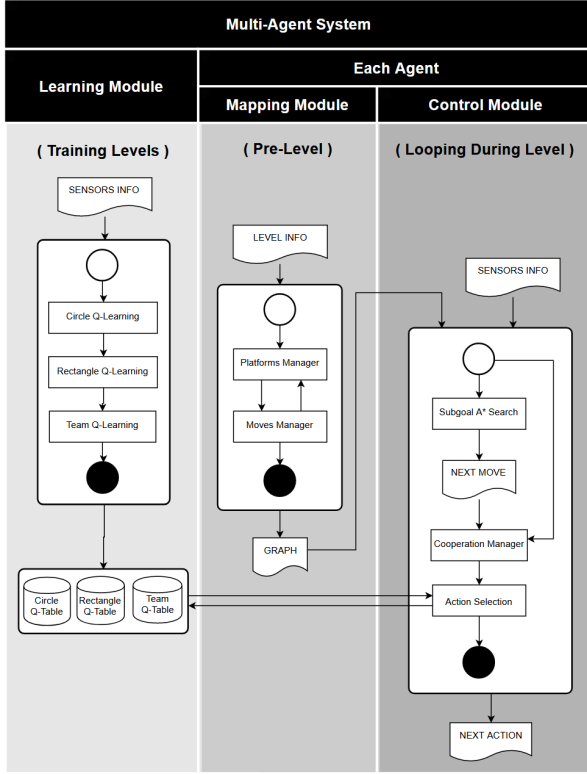


Fig. 7: System Architecture

Given our approach, the system implementation followed the architecture shown in Fig. 7. There is a Learning Module responsible for three training sessions and for keeping records for future use. After that, the Mapping Module receives information about a level, identifies platforms and moves, and returns a graph for each agent. Finally, the Control Module uses all this information to plan the best current solution, NEXT MOVE, and to realize it, NEXT ACTION.

## 4. Mapping Module

The purpose of this module is to condense all pre-level data on obstacles and collectibles in a single graph with Platforms as nodes and Moves as edges. They are identified by the Platform Manager and the Move Manager, respectively.

### 4.1. Platforms Manager

#### 4.1.1. BASIC Platforms

A basic platform is any game obstacle, or a set of obstacles, in which the character can move from one edge to the other without colliding with any obstacle and maintaining the same set of allowed heights. Obstacles of the same color as the character are ignored while the rest are seen as black ones (they are all obstacles in the same way).

Unlike the circle, where BASIC platform identification is straightforward, this process is more complicated for the rectangle due to its morphing ability (as shown in Fig. 8). Whenever a change in allowed height influences the possible rectangle heights (those in Fig. 5 plus the Ride Position - Fig. 6a), a new platform is created.

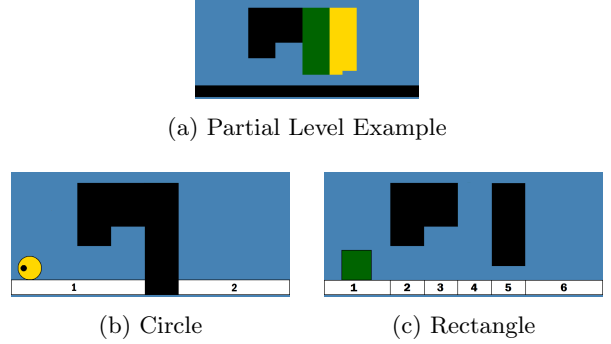


Fig. 8: BASIC Platforms Identification

#### 4.1.2. GAP Platforms

The rectangle has the peculiarity of being able to fall between platforms, morphing upwards with zero horizontal velocity. This is possible when there is a sufficiently large gap for it to fall vertically but also small enough for it to pass horizontally (as platform 2 in Fig. 9). Those are rectangle platforms abstractions, called GAP. Although they are not obstacles, they are useful for Moves Manager.

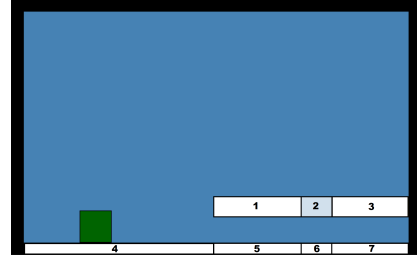


Fig. 9: Rectangle Platforms Identification for level of Fig. 2

#### 4.1.3. COOPERATIVE Platforms

As system leader, the circle is responsible for mapping cooperative moves. Although this is a Moves Manager task, this process begins with identifying COOPERATIVE platforms (circle exclusive abstractions). They represent the possible positions of the rectangle at which the circle can land (allowing the Ride Position).

To identify them, the Platform Manager verifies which rectangle BASIC platforms allow the Ride Position and are reachable from the rectangle starting position. If those conditions are true, platforms, like 3 and 4 in Fig. 10, are inserted as COOPERATIVE in the circle graph, adding the horizontal rectangle height to the original).

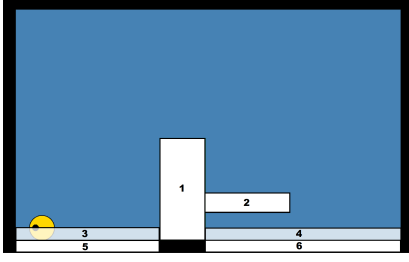


Fig. 10: Circle Platforms Identification for level of Fig. 2

## 4.2. Moves Manager

One Move represents a trajectory that allows the agent to change platforms and/or collect one or more diamonds. For the move to be successful and the trajectory calculated by Moves Manager to be verified, the agent must reach certain coordinates, horizontal velocity and height. These set of values constitute an agent State - a data structure designed to replace both CircleRepresentation and RectangleRepresentation (GF primitives) in order to equalize all processes running within the system.

### 4.2.1. COLLECT/TRANSITION Moves

A COLLECT type move consists only of an agent positioning itself to collect a diamond in the current platform. On the other hand, a TRANSITION move results in a current platform change, collecting or not diamonds along the way. The trajectories of these types of moves are simple to simulate because the Moves Manager only need to check if there are no obstacles and, for the TRANSITION ones, if the gap or stair between platforms is not too big. For the rectangle, these moves are simulated for each of the three possible heights of Fig. 5 (as shown in Fig. 11 and 12).

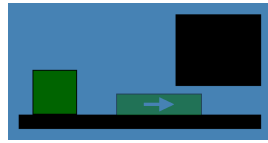
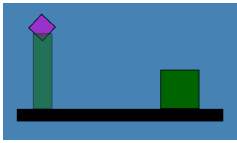


Fig. 11: COLLECT Move Fig. 12: TRANSITION Move

### 4.2.2. FALL/JUMP Moves

For FALL and JUMP moves, we need to use Motion Equations 3 and 4 to predict the complete trajectory and therefore being able to check for obstacles and reachable collectibles. Additionally, we also identify the landing point and target platform (number 5 in the example of Fig. 13). The initial positions of FALL moves are both platform edges while for the JUMP ones, the full discretized length of the platform must be considered.

Whether falling or jumping, the process is the same for each possible initial position: simulate the move with different initial horizontal velocities.

These values are not only discretized but also limited by the platform length available for the agent to accelerate. More about values discretization and limits in Section 6.

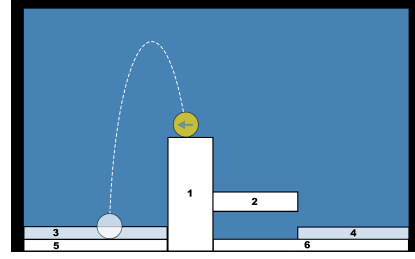


Fig. 13: JUMP Move

### 4.2.3. COOPERATIVE Moves

These moves are identified by the Moves Manager for the circle graph, using the COOPERATIVE platforms that represent the rectangle's possible positions. In fact, it does not require any new trajectory simulation because this process is included in the identification of FALL and JUMP circle moves.

There are two reasons for a FALL or JUMP move to be classified as COOPERATIVE: either the circle is falling/jumping from a COOPERATIVE platform or it is landing on one. This means that, as shown in Fig. 14, the simulation of a jump trajectory can not only result in a JUMP move (Fig. 13) but also a COOPERATIVE one. The latter stores the required rectangle position as partner\_state.

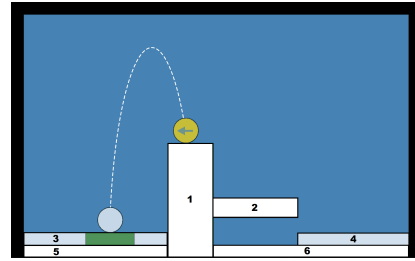


Fig. 14: COOPERATIVE Move

Although the rectangle does not identify these moves, it receives them and incorporates them into its graph with just one adaptation: the state property is replaced with that of partner\_state and vice versa to match the correct player. From its perspective, performing this move is very similar to a COLLECT one, giving that, not considering the required synchronization, it only has to assume a certain position.

Finally, COOPERATIVE moves can start from COOPERATIVE platforms. These platforms have Dynamic Height, meaning that it can be seen as three different BASIC platforms for each rectangle height and for which the Moves Manager simulates every possible movement. This process results in COOPERATIVE moves like the one from Fig. 15

(note that the partner\_state includes information about the rectangle height).

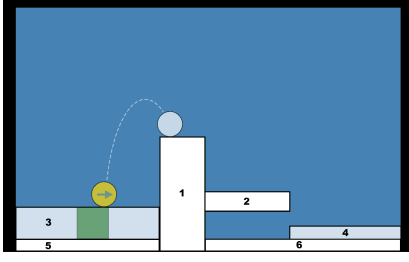


Fig. 15: COOPERATIVE Move with Dynamic Height

## 5. Control Module

### 5.1. Cooperation Manager

Agents in our system can play both individually and as a team, depending on the type of move they select to perform. This is expressed by their Cooperation Status. We show in Table 3 how these values are associated with the agents' synchronization process.

Cooperation Status	NEXT MOVE	Agent State
<b>SINGLE</b>	$\neq$ COOPERATIVE	Playing individually.
<b>UNSYNCHRONIZED</b>	COOPERATIVE	NEXT MOVE state not reached.
<b>SYNCHRONIZED</b>	COOPERATIVE	NEXT MOVE state reached.
<b>RIDING</b>	COOPERATIVE	Moving with partner while assuming the Ride Position.

Table 3: Cooperation Status Values

The Cooperation Status value of one agent often depends on the partner value. This dependency is the result of message exchange between them, handled by the Cooperation Manager. These messages serve as an update to the circle and rectangle relative to the synchronization step they are in. In fact, the Cooperation Status of an agent can go through four types of transition, as modeled in Fig. 16.

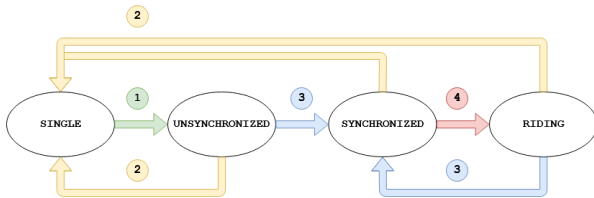


Fig. 16: Cooperation Status Diagram

Transitions between Cooperation Status involve agent-to-agent communication (except transition type 3), respecting our leader-follower strategy.

In transition type 1, the circle, previously playing individually, selects a new NEXT MOVE of type

COOPERATIVE. This move is attached to a message sent to the rectangle. The rectangle does not make this transition immediately, unlike the circle.

In transition type 2, a cooperative move finishes successfully or unsuccessfully, leaving the circle free to play individually. A message is sent to the rectangle in order to free him too.

In transition type 3, the agents reached the states required by a COOPERATIVE move. They both receive that information from their own sensors so no communication is needed.

The only difference between transition type 1 and type 4 is that both agents start SYNCHRONIZED and the Ride Position must be maintained until they reach the states required by the COOPERATIVE move.

### 5.2. Control Module Flow

The Control Module is responsible for the agents' behaviour from the moment they start playing the level, using data from Mapping and Learning Modules. In fact, these data is used in two distinct stages as briefly presented in Table 4.

Stage	Input	Strategy	Output
<b>Planning</b>	SENSORS INFO GRAPH	Subgoal A* Search	NEXT MOVE
<b>Action</b>	SENSORS INFO NEXT MOVE	Training Records (Q-Tables)	NEXT ACTION

Table 4: Control Module Stages

The control module flow iteration for the circle and rectangle is shown in Fig. 17, divided into these two stages.

#### 5.2.1. Circle Control Module Flow

The Planning stage is skipped if NEXT MOVE is defined, which is the most common case since a move is rarely chosen and executed entirely in a single iteration. On the other hand, if NEXT MOVE is not defined, the Control Module stays in Planning and obtains it using Subgoal A\* on the circle graph.

The Cooperation Status is updated frequently during these iterations by checking different conditions. For example, before using the pathfinding algorithm, it checks if the characters are in the Ride Position. The Cooperation Status is then SINGLE or RIDING according to the veracity of this condition. Note that in Fig. 17, values for Cooperation Status do not necessarily mean that there is a transition (the previous status may be the same).

This covers situations like the when the circle comes from a cooperative movement and is no longer in Ride Position. This is a type 2 transition between RIDING and SINGLE, requiring a message to release the rectangle.

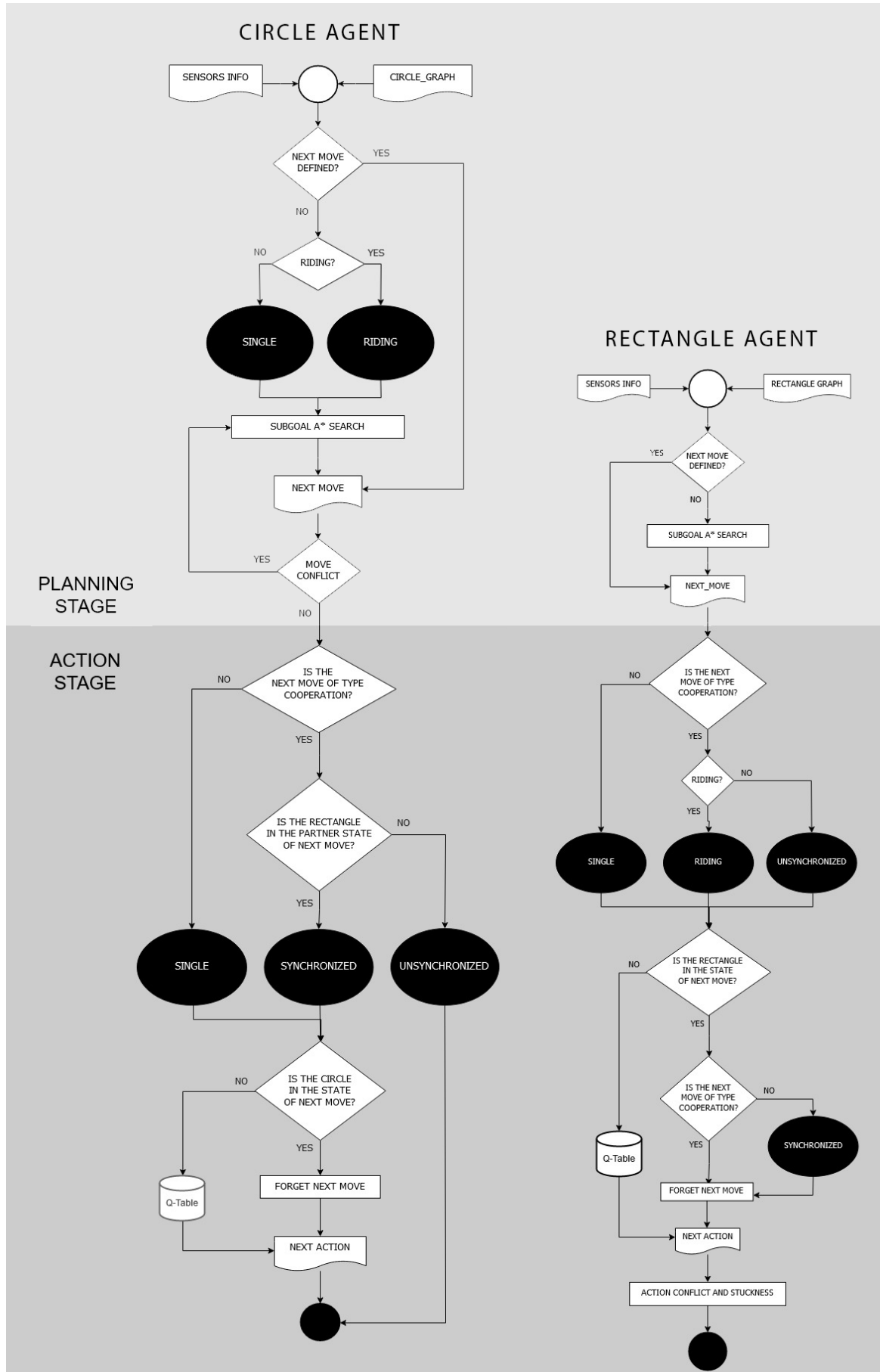


Fig. 17: Control Flow Iteration for the Circle and Rectangle Agents



Between Subgoal A\* Search on the circle graph and the Action Phase, there is one more operation: move conflict identification. Basically, if there is conflict, the search is repeated, otherwise the agent can finally proceed to the Action Stage.

This stage begins by checking if the type of NEXT MOVE is COOPERATIVE. In that case, the circle as system leader, must verify the current synchronization status. It does that by checking if the rectangle is already in the partner state required by the cooperative move. If it is, the synchronization is completed otherwise the agent remain UNSYNCHRONIZED and the circle control flow iteration ends there with no action.

The last step is to use the training results by querying the team  $Q$ -Table or the circle one, depending on whether the next move is cooperative or not. With this, the Control Module gets the best NEXT ACTION in order to reach the required position and horizontal velocity included in the NEXT MOVE state. If the agent already reached it, the NEXT ACTION corresponds to the move type: jumping if JUMP, rolling if TRANSITION or no action. Additionally, the executed NEXT MOVE is forgotten.

### 5.2.2. Rectangle Control Module Flow

The rectangle Planning stage is equal to the circle one but the Cooperation Status is not changed until the Action Stage when it checks the type of NEXT MOVE. If it is COOPERATIVE, the Cooperation Status becomes RIDING if the agents are in Ride Position, otherwise UNSYNCHRONIZED. If it is an individual move, Cooperation Status is SINGLE.

The rest of the algorithm is practically the same compared to the circle. Specifically all the logic regarding the use of individual or team  $Q$ -Tables as well as reaching NEXT MOVE state (completing synchronization if it is cooperative). In the end, there is one last difference: checking for action conflict or stuckness. These were mechanisms we found necessary to implement in the rectangle agent given its particular characteristics.

## 6. Learning Module

The Learning Module is responsible for all the three training sessions: circle, rectangle and Ride Position. The first two use single  $Q$ -Learning while the latter uses the multi-agent version Team  $Q$ -Learning (joint actions). In our system, each  $Q$ -Table entry is updated according to the Equation 1, using  $\alpha = 0.3$ ,  $\gamma = 0.999$  and  $r = 200$  if reached target, otherwise  $r = -1$ . As for learning policy, we chose the  $\epsilon$ -greedy with  $\epsilon = 0.6$ .

The learning problems' states are defined as represented in Fig. 18. In the individual cases, states depend on the relative distance and relative velocity to the target velocity (preserving the symmetric

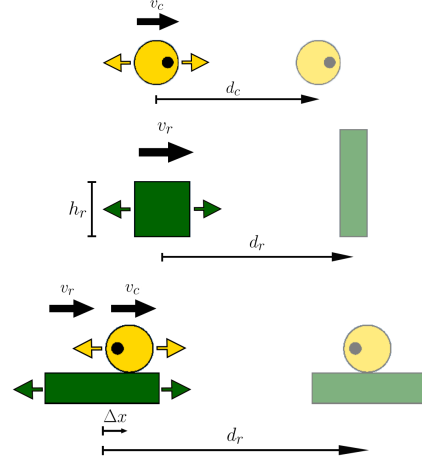


Fig. 18: State Definition for each Learning Problem

property from KIT) while actions only allow the characters to move sideways.

As for the Ride Position case, both agents keep their possible actions but as a system, the training actions are instead joint actions. The state maintains other elements such as the horizontal velocity values for each character and the relative distance between rectangle and target. The Ride Position always assumes that the circle is on top of the rectangle. For this reason, it is unnecessary for the state to consider the relative distance from the circle to the target. Instead of this value, we have the deviation between the rectangle and circle positions, represented by  $\Delta x$ . The goal is not only for each character to reach a certain position and velocity but also for the deviation between them to be zero.

Considering all possible values for the elements enumerated in the previous sections, training would be nearly impossible. The solution is to discretize and limit these values, reducing state and target space, training necessity and  $Q$ -Table size.

In fact, we started doing this right from the definition of rectangle actions. Morphing actions are not considered because the rectangle performs these two actions, as needed, before the movement begins. However, the rectangle learning state includes its height because the agent trains for each one (from Fig. 5). All information about values limitation and discretization for each learning problem is summarized in Table 5.

The maximum horizontal velocity in the game is 200. The KIT also experimentally concluded that the circle only needs 200 units of distance to reach this velocity and therefore  $d_c$  values are only considered below that. The distance values are discretized in intervals of 4 while the velocity ones are of 10.

Regarding the rectangle, the distance to the target  $d_r$  had to be increased to 300 because the character has less acceleration. In the circle, the hor-





	Action	State	Target
		Limits (Discretized)	
	ROLL_LEFT ROLL_RIGHT	$ d_c  \leq 200(4)$ $ v_c  \leq 200(10)$	$d_c = 0$ $v_c = \{0, 20, \dots, 180, 200\}$
	MOVE_LEFT MOVE_RIGHT	$ d_r  \leq 300(4)$ $ v_r  \leq 200(10)$	$d_r = 0$ $v_r = \{0, 50, 100, 150, 200\}$ $h_r = \{50, 100, 200\}$
	[ROLL_LEFT, MOVE_LEFT] [ROLL_LEFT, MOVE_RIGHT] [ROLL_RIGHT, MOVE_LEFT] [ROLL_RIGHT, MOVE_RIGHT]	$ d_r  \leq 100(4)$ $ v_c  \leq 90(10)$ $ v_r  \leq 90(10)$ $ \Delta x  \leq 60(4)$	$d_r = 0$ $v_c = 0$ $v_r = 0$ $\Delta x = 0$

Table 5: Learning Problems Overview

horizontal velocity  $v_c$  can assume 20 different values. The same implementation on the rectangle would be difficult because we already increased relative distance values and added height to the target. For this reason, we considered only five values (0, 50, 100, 150, 200). Moreover, the rectangle has no jump trajectories, so it does not need as much flexibility in target velocity.

For the Ride Position, the priority is to keep the relative position of the characters as stable as possible while moving. In order to achieve this, we reduced each agent maximum velocity to 90. Additionally, the relative distance from the rectangle to the target drops to 100. Still regarding the state definition, the deviation between the characters cannot be greater than 60. All these values were thought considering their consequences in the training process. As for the target, given the large increase in the state space size, and for the reasons already stated in our cooperation approach, we only considered the movements in which both agents end with zero speed and still in Ride Position.

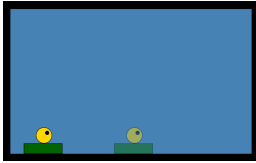


Fig. 19: Training Level (Cooperation Case)

The training conditions are the same as those used by KIT: an empty level as shown in Fig.19 with the respective target position being the middle  $x$ . In conclusion, the circle has 2 actions, 4.000 states and 20 targets, therefore, 160.000  $Q$ -Table entries. The KIT agent trained 5.000 times for each target, meaning 100.000 in total. Following the same reasoning, the rectangle with 2 actions, 6.000 states and 15 targets, has 180.000  $Q$ -Table

entries and it must train 7.500 times for each target (total of 112.500 times). As for the Ride Position with 4 joint actions, 486.000 states and one target, has 1.944.000  $Q$ -Table entries and it must train 1.215.000 times.

Due to time constraints and the mentioned learning problem size, the agents we submitted to the 2019 competition were incomplete. Specifically, the Rectangle and Riding Position were not trained as described, which means that they did not have a  $Q$  Table designed according to their characteristics. The solution was to use the circle  $Q$ -Table as the rectangle one and the Riding Position being individually guaranteed by the agents, with a simple mechanism to keep  $\Delta x$  at a stable value.

## 7. Results

The Rectangle and Cooperation Track results were influenced by a game bug in which the rectangle was blocked if it fell on one of its faces. We ignore bug levels for comparison purposes.

Level	Collectibles	Time(s)	Score
1	1.0 (2)	120.0	300.0
2	2.7 (3)	59.41	1314.9
3	2.1 (3)	104.931	755.58
4	0.9 (3)	120.0	270.0
5	2.0 (2)	16.165	1465.29
6	2.7 (3)	59.127	1317.28
7	2.0 (2)	20.216	1431.53
8	1.5 (2)	67.525	887.29
9	3.0 (3)	29.28	1655.17
10	2.0 (2)	20.531	1428.91
bug level	Total (bug-free)		10825.94(5119.37)

Table 6: RTT Results in 2019 Rectangle Track

Level	Collectibles	Time(s)	Score
1	2.0 (2)	12.685	<b>1494.29</b>
2	0.9 (3)	90.349	517.09
3	3.0 (3)	22.195	1715.04
4	2.8 (3)	31.513	<b>1577.39</b>
5	2.0 (2)	12.572	<b>1495.23</b>
6	3.0 (3)	18.981	1741.82
7	1.9 (2)	26.005	1353.29
8	1.1 (2)	69.577	750.19
9	3.0 (3)	14.413	<b>1779.89</b>
10	2.0 (2)	9.423	<b>1521.47</b>
bug level	Total (bug-free)		13945.72(7868.27)

Table 7: Our Results in 2019 Rectangle Track

In the Rectangle Track, our agent has the best score in 5 out of 5 bug-free levels (in bold), up by 50% in total over the runner-up and baseline, the RTT agent. Our rival had difficulties at levels 1 and 4, otherwise the difference between the two would have been very small. However, levels 5, 9 and 10 that were completed by both agents and are the best comparative factor, show that our solution does it in a shorter time (two times faster in 9 and 10). Note that there was no stuckness verification in this agent version (that is why level 4 does not record all diamonds collected, for example).

Level	Collectibles	Time(s)	Score
1	0.8 (3)	120.0	240.0
2	0 (1)	120.0	0
3	1.5 (3)	120.0	450.0
4	0.8 (3)	120.0	240.0
5	2.6 (3)	67.89	1214.26
6	1 (3)	120.0	300.0
7	1 (2)	111.40	371.65
8	1 (3)	120.0	300.0
9	1.7 (2)	44.77	1074.25
10	1 (3)	120.0	300
bug level	Total (bug-free)		4490.16 (1530.0)

Table 8: RTT Results in 2019 Cooperation Track

Level	Collectibles	Time(s)	Score
1	2.8 (3)	32.55	<b>1568.72</b>
2	1 (1)	15.70	<b>1169.19</b>
3	2.5 (3)	65.77	<b>1201.89</b>
4	2.5 (3)	58.96	<b>1258.68</b>
5	2.8 (3)	32.38	1570.14
6	3 (3)	21.24	<b>1723.02</b>
7	0 (2)	120.0	0
8	2.9 (3)	38.12	1552.34
9	2 (2)	6.89	1542.57
10	1 (3)	120.0	300.0
bug level	Total (bug-free)		11886.56 ( <b>7221.5</b> )

Table 9: Our Results in 2019 Cooperation Track

In the Cooperation Track, our proposal is once again superior to RTT in bug-free levels. This time, much more clearly, getting a total score with an increase of approximately 470% compared to the baseline. Regarding the bug levels, our agent was not affected in any of them. That is why at levels 5, 8 and 9 we see the same excellent performance.

Level	Collectibles	Time(s)	Score
1	2 (2)	12.6	<b>1495</b>
2	3 (3)	18.1	<b>1749</b>
3	3 (3)	20.6	<b>1728</b>
4	3 (3)	23.4	<b>1705</b>
5	2 (2)	10.7	<b>1511</b>
6	3 (3)	21.5	1721
7	2 (2)	9.9	<b>1550</b>
8	2 (2)	18.8	<b>1444</b>
9	3 (3)	12.6	<b>1795</b>
10	2 (2)	9.3	<b>1523</b>
Total			<b>16221</b>

Table 10: Post Bug-Free 2019 Rectangle Track

Level	Collectibles	Time(s)	Score
1	3 (3)	20.1	<b>1732</b>
2	1 (1)	14.7	<b>1178</b>
3	3 (3)	29.9	<b>1655</b>
4	3 (3)	29.7	<b>1653</b>
5	3 (3)	11.5	<b>1804</b>
6	3 (3)	22.5	1701
7	2 (2)	45.5	<b>1220</b>
8	3 (3)	21.9	<b>1718</b>
9	2 (2)	7.8	1535
10	3 (3)	35.6	<b>1603</b>
Total			<b>15799</b>

Table 11: Post Bug-Free 2019 Cooperation Track

In Table 10 and 11, we see how completing the implementation and conflicts resolutions mechanisms improved performance in bug-free post-competition.

## 8. Conclusions

The objectives we proposed when introducing this work were successfully achieved. The victory in the 2019 competition was only a recognition for the good performance and correct implementation of our agents. Taking advantage and reusing some established strategies, we were able to build a complex system consisting of two players capable of playing individually and in cooperation. More than that, our proposal is now motivation for people to continue to experiment and compare new approaches to solve the Geometry Friends problem. Equally interesting will be proposals that build on the solid foundations of our system architecture and expand the cooperation and learning mechanisms of our agents. Conflict resolution and increasing player training variety is the first work task for those who decide to invest in our results.

## References

- [1] R. Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.
- [2] M. Campbell, A. Hoane, and F. hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57 – 83, 2002.
- [3] V. Firoiu, W. F. Whitney, and J. B. Tenenbaum. Beating the world’s best at super smash bros. with deep reinforcement learning. *CoRR*, abs/1702.06230, 2017.
- [4] D. Fischer. Workflow of subgoal a \* agent solving the rectangle track of geometry friends, 2015.
- [5] I. Millington and J. Funge. *Artificial Intelligence for Games, Second Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2009.
- [6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. 12 2013.
- [7] R. Prada, P. Lopes, J. Catarino, J. Quiterio, and F. Melo. The geometry friends game ai competition. pages 431–438, 08 2015.
- [8] S. Rabin. *AI Game Programming Wisdom*. Charles River Media, Inc., Rockland, MA, USA, 2002.
- [9] A. Salta, R. Prada, and F. Melo. Towards a cooperative agent for human-agent interaction for geometry friends. Master’s thesis, Instituto Superior Técnico, Lisboa, Portugal, 2017.
- [10] R. Soares, F. Leal, R. Prada, and F. S. Melo. Rapidly-exploring random tree approach for geometry friends. In *DiGRA/FDG*, 2016.
- [11] S. Sodhi. Ai for classic video games using reinforcement learning. Master’s thesis, 2017.
- [12] R. S. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 1998.
- [13] G. Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, Mar. 1995.
- [14] J. Tsay, C. Chen, and J. Hsu. Evolving intelligent mario controller by reinforcement learning. In *Proceedings of 2011 Conference on Technologies and Applications of Artificial Intelligence*, pages 266–272, 2011.
- [15] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King’s College, Cambridge, UK, 1989.
- [16] C. J. C. H. Watkins and P. Dayan. Q-learning. In *Machine Learning*, volume 8, chapter 3, page 279–292. Springer US, 1992.