

Building a Multi-Agent Learning System for Geometry Friends

Ricardo Ari Sequeira

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor(s): Prof. Rui Filipe Fernandes Prada
Prof. Francisco António Chaves Saraiva de Melo

Chairperson: TBA
Supervisor: TBA
Member of the Committee: TBA

October 2019

Acknowledgments

First, I would like to thank my supervisors, Professors Francisco Melo and Rui Prada, for their guidance, feedback, patience and mostly, for trusting me. It could not have been easier to work with them.

Secondly, thanks to the best father, best mother and best brother that I could ever ask for. I have never lacked anything in my life and this is because of them. They have always been by my side, so they know all the efforts I made throughout this journey.

Lastly, I would like to thank my friends for being the worst and best influence I need. They know who they are, they know that I love them, and they know that I value every moment I spend with them because it gives meaning to this life.

Resumo

Machine Learning é, hoje em dia, uma área excitante devido ao maior poder de computação disponível atualmente. A descoberta de aplicações para o mundo real aumenta, mas também o desenvolvimento de novos algoritmos e estratégias nesta área. Os jogos sempre foram um excelente laboratório para o estudo destas novas ideias e o Geometry Friends é um excelente exemplo disso. Trata-se de um desafio de estratégia multi-jogador que possui a sua própria competição com o objetivo de distinguir os melhores agentes cooperativos e não cooperativos. O problema do jogador único já teve várias diferentes abordagens ao longo dos anos, com resultados muito satisfatórios. A atenção vira-se agora para a resolução da componente de cooperação associada à jogabilidade multi-jogador. Neste trabalho, nós propomos uma arquitetura consistente de um sistema de aprendizagem multi-agente, inspirado no sucesso alcançado por agentes únicos na utilização de grafos direcionados de peso e Reinforcement Learning. A motivação é construir uma base sólida para futuras soluções de cooperação que procurem expandir e explorar conhecimentos de Machine Learning. Os resultados finais demonstram que o nosso sistema superioriza-se às propostas submetidas até hoje, assente numa estrutura simples apesar das exigências complexas inerentes a um ambiente multi-agente.

Palavras-chave: Geometry Friends, Sistema Multi-Agente, Agentes Cooperativos, Machine Learning, Reinforcement Learning

Abstract

Machine Learning is now an exciting field due to the increased computing power available today. Real-world application discoveries are increasing, but also new algorithms and strategies developed in this area. Games have always been a great playground for studying these new ideas and Geometry Friends is a great example. It is a multi-player puzzle game with its own competition that aims to distinguish the best cooperative and non-cooperative agents. The single player problem has had many different approaches over the years with very satisfactory results. Attention now turns to the resolution of the cooperation component associated with multi-player gameplay. In this paper, we propose a consistent multi-agent learning system architecture, inspired by the single agent success of using a weighted directed graph and Reinforcement Learning. The motivation is to build a solid foundation for future co-operation solutions that want to expand and exploit 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: Geometry Friends, Multi-Agent System, Cooperative Agents, Machine Learning, Reinforcement Learning

Contents

Acknowledgments	iii
Resumo	v
Abstract	vii
List of Tables	xiii
List of Figures	xv
List of Equations	xvii
Nomenclature	xix
Glossary	xix
1 Introduction	1
1.1 Objectives	2
1.2 Thesis Outline	2
2 Background	3
2.1 Geometry Friends	3
2.1.1 Competition	4
2.2 Weighted Directed Graphs	5
2.2.1 A* Search Algorithm	5
2.3 Reinforcement Learning	6
2.3.1 Single Q-Learning	7
2.3.2 Multi-Agent Reinforcement Learning	7
2.3.3 Team Q-Learning	9
3 Related Work	11
3.1 Geometry Friends Proposals	11
3.1.1 Subgoal A* (2015)	12
3.1.2 PG-RL (2015)	13
3.1.3 RTT (2015, 2017)	15
3.1.4 KIT (2017)	18
3.2 Coordinated Multi-Agent Object Transportation	20
4 System Overview	23
4.1 System Execution Phases	23
4.2 Circle Approach	24

4.3	Rectangle Approach	24
4.4	Cooperation Approach	25
4.5	System Architecture	27
4.5.1	Training	28
4.5.2	Pre-Level	28
4.5.3	During Level	28
5	Mapping Module	31
5.1	Platforms Manager	31
5.1.1	BASIC Platforms	31
5.1.2	GAP Platforms	33
5.1.3	COOPERATIVE Platforms	33
5.2	Moves Manager	35
5.2.1	COLLECT Moves	36
5.2.2	TRANSITION Moves	36
5.2.3	FALL Moves	37
5.2.4	JUMP Moves	38
5.2.5	COOPERATIVE Moves	39
6	Control Module	43
6.1	Cooperation Status	43
6.2	Cooperation Manager	44
6.3	Circle Control Flow	46
6.3.1	Circle Planning Stage	47
6.3.2	Circle Action Stage	47
6.3.3	Move Conflict Verification	48
6.4	Rectangle Control Flow	49
6.4.1	Rectangle Planning Stage	50
6.4.2	Rectangle Action Stage	50
6.4.3	Action Conflict Verification	50
6.4.4	Stuckness Verification	51
7	Learning Module	53
7.1	Circle Learning Problem	54
7.2	Rectangle Learning Problem	54
7.3	Ride Position Learning Problem	55
7.4	Values Limitation and Discretization	55
7.4.1	Learning Symmetry	57
7.5	Training	57
7.6	Implementation	58

8 Results And Discussion	61
8.1 2019 Competition Analysis	61
8.1.1 2019 Rectangle Track	61
8.1.2 2019 Cooperation Track	62
8.2 Post-Competition Analysis	64
8.3 Limitations	65
8.3.1 Rectangle Agent	65
8.3.2 Multi-Agent System	66
9 Conclusions	69
9.1 Future Work	70
Bibliography	71

List of Tables

2.1	Q -table entries of a 3-player coordination game state	9
3.1	Geometry Friends State-of-the-Art (before September 2019)	12
3.2	PG-RL Implementation of Reinforcement Learning	15
3.3	KIT Q -Learning Implementation	19
3.4	Example Team Q -Learning Implementation	21
4.1	System Execution Phases	23
4.2	Circle Approach (KIT Approach)	24
4.3	Rectangle Approach (vs Circle Approach)	24
4.4	Cooperation Approach	26
6.1	Control Module Stages	43
6.2	Cooperation Status Values	44
7.1	Learning Problems Overview	56
7.2	Q -Table Size and Training Runs	58
8.1	RTT Results in 2019 Rectangle Track (Runner-Up / Baseline)	62
8.2	Our Results in 2019 Rectangle Track (Winner)	62
8.3	RTT Results in 2019 Cooperation Track (Runner-Up / Baseline)	63
8.4	Our Results in 2019 Cooperation Track (Winner)	63
8.5	Post-Competition Results for 2019 Rectangle Track (Bug-Free Game Version)	64
8.6	Post-Competition Results for 2019 Cooperation Track (Bug-Free Game Version)	65
9.1	Geometry Friends State-of-the-Art (before and after September 2019)	69

List of Figures

2.1	Geometry Friends Characters and Actions	3
2.2	Geometry Friends level example with all elements present	4
2.3	Weighted Directed Graph Example	5
2.4	Reinforcement Learning Overall Model	6
2.5	Classification of MARL algorithms by type of task	8
3.1	Example of Fischer's Geometry Friends Mapping	12
3.2	Simpler adapted example of a Fischer Graph	13
3.3	PG-RL Platform Identification	14
3.4	Geometry Friends level and corresponding example of PG-RL Graph	14
3.5	State properties when using RL for the subproblem Moving to Next Platform	14
3.6	RTT EXTENT function	16
3.7	Difficulty of using euclidean distance as a metric distance for the RTT	17
3.8	KIT Platform Identification	18
3.9	Use of symmetric property by KIT to reduce state space	19
3.10	KIT Training Level	20
3.11	Action selection by KIT agent is practically optimal	20
3.12	Object Transportation Problem	21
3.13	Solution obtained by Team Q -Learning	21
4.1	Rectangle Possible Forms/Heights	25
4.2	Riding Ability	25
4.3	System Architecture	27
5.1	BASIC Platforms Identification	32
5.2	Platforms Identification	34
5.3	Graph Model	35
5.4	COLLECT Moves	36
5.5	Circle TRANSITION Moves	36
5.6	Rectangle TRANSITION Moves	37
5.7	Circle FALL Moves	38
5.8	Rectangle FALL Moves	38
5.9	Possible Trajectory Initial Positions	38

5.10	Example of a JUMP Move (level from Figure ??)	39
5.11	Example of a COOPERATIVE Move (level from Figure ??)	40
5.12	Example of COOPERATIVE Moves from COOPERATIVE platforms (Dynamic Height)	41
6.1	Cooperation Status Diagram	44
6.2	One Iteration of the Circle Control Flow	46
6.3	Move Conflict Verification	48
6.4	One Iteration of the Rectangle Control Flow	49
6.5	Example of a solvable Action Conflict	50
6.6	Solved Action Conflict	51
6.7	Stucked Rectangle	52
7.1	Circle Learning Problem	54
7.2	Rectangle Learning	54
7.3	Ride Position Learning	55
7.4	Learning Symmetry: example of two equal rectangle learning states	57
7.5	Training Levels	57
7.6	Rectangle Agent using the Circle Q-Table	59
8.1	Unsolved Rectangle Level: Morphing While Falling	65
8.2	Uncovered Cooperative Move: Breaking Circle Fall	66
8.3	Uncovered Cooperative Move: Changing Positions	66
8.4	Unsolved Cooperation Level Example 1	67
8.5	Unsolved Cooperation Level Example 2	67
9.1	Future Work Examples	70

List of Equations

2.1 Competition Score Run Formula	4
2.2 Q-Table Update Formula	7
2.3 Policy in Nash Equilibrium	9
2.4 Team Q-Table Update Formula	9
3.1 Horizontal Motion Equation	19
3.2 Vertical Motion Equation	19

Glossary

AI	Artificial Intelligence
GF	Geometry Friends
GLIE	Greedy in the Limit with Infinite Exploration
MARL	Multi-Agent Reinforcement Learning
MAS	Multi-Agent Systems
MDP	Markov Decision Process
ML	Machine Learning
RL	Reinforcement Learning
RTT	Rapid-Exploring Random Trees
TMG	Team Markov Game

Chapter 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 [1] [2]. 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 [3] in 1995 and the chess world champion was defeated by machine Deep Blue (IBM) in 1997 [4] 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 [5][6][7][8]. 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) [9] 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* [10], is the best shortest-path search algorithm applied to GF. As for the single-player mode, Reinforcement Learning [11] and Rapid-Exploring Random Trees (RTT) [12] [13] 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 [14] [15]. Additionally, we will implement new communication and synchronization processes between agents.

1.1 Objectives

Our objectives are related to the agents' performance evaluation but not only. We list them below:

- develop a rectangle agent that wins the Rectangle Track, sharing the structure and using Reinforcement Learning for the same sub-problems as the best circle agent (KIT agent);
- combine the developed rectangle with KIT circle agent into a single system equipped with the cooperation mechanisms in order to win the Cooperation Track;
- achieve the two previous goals with a solid and modular system architecture for future enhancements and explorations of Machine Learning techniques.

If our proposal meets all three objectives, it means that we make an important contribution to completely solve the cooperation problem. It will also be the first Geometry Friends multi-agent system based on the use of Reinforcement Learning. More importantly, the system will allow future work to increase the use of RL to other subproblems or to explore other techniques, taking advantage of the mapping and planning phases as well as other structures.

1.2 Thesis Outline

This dissertation is divided into nine chapters. The Chapter 2 serves to present the subject of this work, the Geometry Friends game. In addition, it also introduces concepts and strategies mentioned and used throughout this paper. In Chapter 3, we describe the proposed solutions to the Geometry Friends problem and its state of the art. Chapter 4 explains our approach to problems and provides a first look at the entire system architecture and module organization. The following chapters describe each system module, starting with Chapter 5 about the Mapping Module in which we detail the platform and move identification process in order to build each agent graph. In Chapter 6, we present all the mechanisms present in the Control Module to ensure the move and action selection as well as the synchronization and communication in the system. Chapter 7 addresses the Learning Module, which is responsible for training agents to perform the necessary movements. This ends the description of our proposal and its implementation. In Chapter 8, we analyze the results of our multi-agent system in the 2019 Geometry Friends competition but also the post-competition development and its limitations. Finally, our concluding and future work comments are included in Chapter 9.

Chapter 2

Background

Before moving on to related work and implementation, it is necessary to explain the Geometry Friends game and review the concepts of weighted graphs and reinforcement learning. This is important to understand why and how we applied a search algorithm adapted from A* and Q-Learning to train agents (individually and as a team).

2.1 Geometry Friends

Geometry Friends [9] is a two-dimensional puzzle game in which all the laws of physics are applied (the laws of motion and the law of gravity, for example) as well as forces such as friction. There is a single-player and a multi-player mode with two available characters:

- Yellow Circle: it can roll left, roll right and jump to a certain height (no less or more than this value);
- Green Rectangle: it can slide left, slide right, morph up and morph down. It always keeps the same area, whatever it does.

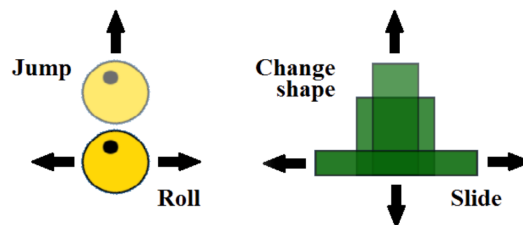


Figure 2.1: Geometry Friends Characters and Actions

The goal is to collect all diamonds that are distributed across the level, within the time limit (may or may not exist). Each level has its own number of diamonds and their respective position. The remaining elements of the game are obstacles that prevent players 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.

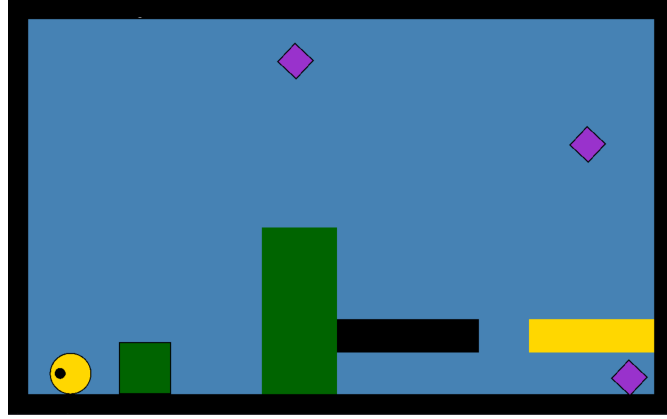


Figure 2.2: Geometry Friends level example with all elements present

Geometry Friends has three playable game modes: controlling only the circle, only the rectangle or controlling both. This means that you can play alone, accompanied by a person or agent, or even just watch intelligent agents playing. This feature is one of the reasons for the existence of a competition.

These artificial players have access to various level and character information:

- collectibles' coordinates;
- obstacles' coordinates and size;
- characters' coordinates, size and velocity (horizontal and vertical).

In addition, they are able to establish communication between them by sending and receiving messages (in any format).

2.1.1 Competition

Since 2013, a competition is held every year at the IEEE Conference on Games, challenging AI players to play Geometry Friends. The main objective 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 Cooperation Track. The first two encourage each agent development in single-player levels. The cooperation one evaluates the performance of multi-agent cooperative systems throughout more complex levels.

All participants are tested ten times at each of five public levels (during submissions) and five more private levels (published after submissions). The ranking is based on the average score of the ten runs, whose formula is given by Equation 2.1.

$$ScoreRun = V_{Completed} \times \frac{(maxTime - agentTime)}{maxTime} + (V_{Collect} \times N_{Collect}) \quad (2.1)$$

The score is higher the more diamonds it collects ($N_{Collect}$) and the less time it takes ($agentTime$). Completing a level is rewarded with $V_{Completed} = 1000$ while each collected diamond means a bonus value of $V_{Collect} = 300$.

2.2 Weighted Directed Graphs

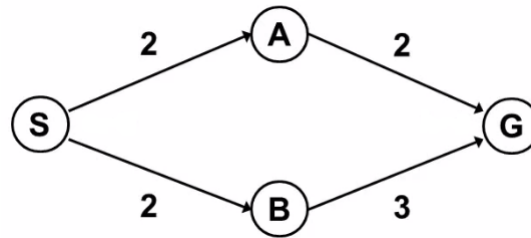


Figure 2.3: Weighted Directed Graph Example

A graph is a structure that organizes objects (represented by nodes) and the relationships between them (represented to edges). The type of graphs we are interested in are the directed weighted graphs where the edges have a direction and weight associated with them.

2.2.1 A* Search Algorithm

Finding the best path between two nodes of a graph is a very common task, for which there are several search algorithms. One of the best and most studied is A* Search [16]. The main elements used in this technique is:

- s as the starting node
- t as the terminal node(s);
- $g(n)$ as the path cost between s and any node n ;
- $h(n)$ as a heuristic function that estimates the cheapest path cost from n to t
- $f(n)$ as the lowest sum of $g(n)$ and $h(n)$ for any neighboring node n .

A path is a set of edges that connect two nodes, each edge having a respective cost (the aforementioned weight). In addition to these elements, the algorithm has two fundamental structures: a list of nodes already visited (closed set) and one with the non-visited (open set).

Every time A* visits a node n (also called expansion), it calculates the value of $f(m)$ for every neighboring node m , puts n in the closed set and expands m with the smallest $f(m)$ value (as long it is in the open set). This cycle begins with s and ends when t is selected from the open set.

This algorithm has excellent results, although dependent on the heuristic function used, which varies greatly according to the problem in question. For example, for problems such as finding the best road between two cities, the most common is to use the Euclidean distance, but it may not be best suited for all cases and adaptations may be required.

2.3 Reinforcement Learning

Reinforcement Learning is a Machine Learning technique in which agents are placed to train in an environment receiving certain input, choosing one of several possible actions and being rewarded for their choices given their goal.

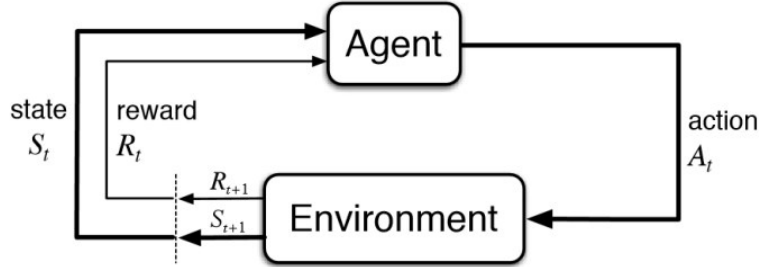


Figure 2.4: Reinforcement Learning Overall Model

The best way to model a single-agent Reinforcement Learning problem is as Markov Decision Process (MDP) [17]. A MDP is represented by a tuple $\langle S, A, R, T \rangle$ where:

- S is a finite state space;
- A is the set of actions the agent can take;
- $R : S \times A \rightarrow \mathbb{R}$ is a reward function for taking certain action a in state s ;
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition function that returns the probability of reaching a destination state given an origin state and an executed action.

The solution for a MDP is a policy $\pi : S \rightarrow A$ that makes the agent choose the best action for each state, known as optimal policy (π^*). This means that π^* is the sequence of actions that maximizes the agent discounted future reward: $\sum_{t=0}^{\infty} \gamma^t E(r_t | \pi)$, where r_t is the reward at time t and $\gamma \in [0, 1]$ is the discount factor.

The main idea of RL is that the achievement of this policy is accomplished through agent training, which consists of agent-environment interactions (such as the one in Fig. 2.4) at fixed time intervals.

Every training iteration, the agent has to choose one action (from the ones available) to execute on that environment. This selection can be totally random but it is often more practical to follow a probability distribution. That way, the requirements of a large state space are reduced.

The most common method is the ϵ -greedy, where $\epsilon \in [0, 1]$ is the parameter used to control the trade-off exploration-exploitation. Exploration means gathering new information that might lead us to better decisions in the future while exploitation reveals the potential of the best decision given the current information.

In ϵ -greedy method, the agent opts for exploration (selects a random action) with probability ϵ . Otherwise, with probability $1 - \epsilon$, exploitation is chosen (selects the best action according to what it has trained in this environment). The ϵ value (and consequently the trade-off approach) depends a lot on the MDP characteristics (size of state space and number of actions, for example).

2.3.1 Single Q-Learning

There are several RL algorithms available but the Q -Learning [14] [15] is the one we are interested in for our proposal. We chose this technique since it is the most used (even in previous attempts to solve Geometry Friends) and, consequently, also the most studied.

The goal of Q -Learning is to obtain the optimal policy π^* , using only the elements of RL. The policy achieved by the algorithm is in table form (named Q -Table) where each entry is associated to a combination state-action and its value represents the quality of choosing that action in that state. Getting the optimal policy (Q^*) means that the highest entry value for a given state corresponds to the best action selection the agent can make, ensuring the best reward.

During the training (and after initializing the Q -Table), the entry associated with the current state and the action chosen in that learning episode is updated according to Equation 2.2.

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a)) \quad (2.2)$$

From which we can say that:

- α is the learning rate (override rate of old information with new one)
- γ is the discount rate (importance of future rewards);
- r is the action reward;
- s' and a' correspond respectively to the next/resulting state and the action of that state with the best current Q value.

The training is then a simple value iteration update, using the weighted average of the old value and the new information. Generally, the more the agent trains, the more he learns from the environment. However, the results also depend on the learning policy adopted (not related to the Q policy). The learning policy is responsible for choosing the next action every training episode by assigning a selection probability to each action.

The convergence to Q^* is obtained if the learning policy verifies the "Greedy in the Limit with Infinite Exploration" (GLIE) property: every state-action pair $\langle s, a \rangle$ is visited infinitely often. In the infinite limit, the action selection is considered greedy because it always selected the best action, ensuring that the resulting policy for the agent is optimal. An example of a policy that checks the GLIE property is the aforementioned ε -greedy strategy.

2.3.2 Multi-Agent Reinforcement Learning

The main goal of Machine Learning is to apply the knowledge to real world situations where most of the things we see are systems composed by multiple elements and even things seemingly individual belong in fact to a bigger and more complex conglomerate. That is why, in the last two decades, we saw an

increase of research papers regarding Reinforcement Learning aimed to Multi-Agent Systems (MAS). The most common challenges associated to learning in MAS are:

- Exponential increase in problem complexity due to the number of agents and their respective states and actions;
- Agents can have different goals and tasks, making it impossible to specify a common general objective to all MAS;
- The environment is non-static because agents are acting and learning simultaneously in the same place;
- Exploration-exploitation trade-off implicates not only the environment but also the remaining agents activity;
- Need for coordination strategies to facilitate the overall learning of the system.

The possible divergence or convergence in agents' tasks, as well as the other obstacles, led to the stipulation of different classifications [18] for Multi-Agent Reinforcement Learning (MARL) algorithms.

Fully competitive	Fully cooperative		Mixed	
	Static	Dynamic	Static	Dynamic
<i>Minimax-Q</i>	<i>JAL</i> <i>FMQ</i>	<i>Team-Q</i> <i>Distributed-Q</i> <i>OAL</i>	<i>Fictitious Play</i> <i>MetaStrategy</i> <i>IGA</i> <i>WoLF-IGA</i> <i>GIGA</i> <i>GIGA-WoLF</i> <i>AWESOME</i> <i>Hyper-Q</i>	<i>Single-agent RL</i> <i>Nash-Q</i> <i>CE-Q</i> <i>Asymmetric-Q</i> <i>NSCP</i> <i>WoLF-PHC</i> <i>PD-WoLF</i> <i>EXORL</i>

Figure 2.5: Classification of MARL algorithms by type of task

In Fig.2.5, it is shown how MARL algorithms are classified based the dynamic between agents (fully cooperative, fully competitive or a mix of both) and on the environment properties (static/stateless/"stage games" or dynamic).

Inside the family of multi-agent problems, Geometry Friends can be described as two agents fully cooperating to achieve a common goal, without any kind of competition between them.

This description fits perfectly the concept of a Team Markov Game (TMG), also known as identical-interest stochastic game. A TMG is a tuple $\langle \alpha, S, A, R, T \rangle$ in which:

- α is a set of n agents; S is a finite state space;
- $A = \times A_{i=1\dots n}$ is a joint action space of n agents;
- $R : S \times A \rightarrow \mathbb{R}$ is the common expected reward function;
- $T : S \times A \times S \rightarrow [0, 1]$ is a transition function.

The goal is to find a joint policy $\pi = \{\pi_{i=1\dots n}\}$, where $\pi = S \rightarrow A$ and $\pi_i = S \rightarrow A_i$ that, similar to the single-agent case, maximizes the sum of their discounted rewards. However, in MAS problems, we may have more than one joint policy that verify this condition and some are better than others. Because of this, the only interesting joint policies are the ones in Nash equilibrium, meaning that each individual policy is the best response to the others' policy (see Equation 2.3).

$$Q^*(s, \{\pi_i(s)\} \cup \pi_{-i}(s)) \geq Q^*(s, \{\pi'_i(s)\} \cup \pi_{-i}(s)) \quad (2.3)$$

for all $i \in \alpha$, $s \in S$ and any individual policy π'_i (where π_{-i} is the joint policy of all agents except agent i). Knowing that this is a team game, each optimal Nash equilibrium is an optimal joint policy. This is not very different from the single-agent case because we consider a joint action a optimal in state s if $Q^*(s, a) \geq Q^*(s, a')$ for all $a' \in A$. Nevertheless, there is still the possibility that there is more than one joint policy in Nash equilibrium, a problem that will be addressed in 2.3.3.

In recent years, there have been several research papers [19] [20] [21] [22] [23] [24] that address cooperative systems and what are the best approaches, for example between joint, individual or hierarchical training. In the next section, we will introduce one of the most simple MARL technique but the truth is that in this area there is still no consensus on which algorithms and strategies are best.

2.3.3 Team Q-Learning

The Team Q -learning algorithm [25] is the application of the Q -Learning technique in MAS most similar to single-agent problems. In particular, it is designed for TMG in which agents have precisely the same goal. This allows a very important simplification since there is only a single reward function that all agents try to maximize. Consequently, there is also only one Q -function for which the formula to update its entries is Equation 2.4.

$$Q[s, a_1, \dots, a_n] = Q[s, a_1, \dots, a_n] + \alpha(r + \gamma \max_{a_1, \dots, a_n} Q[s, a_1, \dots, a_n] - Q[s, a_1, \dots, a_n]) \quad (2.4)$$

Apart from considering the joint actions, the formula is almost identical to the single-agent Q -Learning one and ensures the same convergence. However, when working with a MAS, there may be multiple coordination equilibria. There is no consensual solution to this, one can choose to use tie-breaking schemes like lexicographic order to pick an equilibrium to play.

	$b1c1$	$b1c2$	$b1c3$	$b2c1$	$b2c2$	$b2c3$	$b3c1$	$b3c2$	$b3c3$
$a1$	10	-20	-20	-20	-20	5	-20	5	-20
$a2$	-20	-20	5	-20	10	-20	5	-20	-20
$a3$	-20	5	-20	5	-20	-20	-20	-20	10

Table 2.1: Q -table entries of a 3-player coordination game state

For exemple, in Table 2.1, we can see the entries for all the possible joint action in one single state of a 3-player team game. The table highlights the problem with multiple Nash equilibria since there are three optimal Nash equilibria (a1b1c1, a2b2c2, a3b3c3) and six sub-optimal (a3b1c2, a2b1c3, a3b2c1, a1b2c3, a2b3c1, a1b3c2). In order to achieve an optimal joint policy, MARL algorithms need to adress equilibrium selection and players' coordination. Depending on the problem, it can be solved for example with communication and negotiation between agents (when possible) or tie-breaking schemes as already mentioned.

Chapter 3

Related Work

In this chapter, we start by describing the most interesting solutions that have ever been proposed for Geometry Friends. This introduction is very useful considering that our system uses some of the techniques and its architecture is partly inspired by the existing agents. On the other hand, they are also our competitors and will serve as a comparison when we discuss our own results.

In the final part, we present an example of applying Team Q-Learning in a similar context to Geometry Friends, proving the success of the algorithm in solving such problems.

3.1 Geometry Friends Proposals

The research work on Geometry Friends has already gone through different phases and suffered advances and setbacks over time.

One aspect they all seem to agree is that the overall problem can be split into two main subproblems: Planning (the agent path) and Control (the agent). It was also in this order that the best strategies adapted to this game began to appear. The first one was the CIBot [26] [27] [28] that used Dijkstra algorithm and Monte Carlo Tree Search for pathfinding in circle and rectangle, respectively. However, the best performance regarding the planning subproblem was achieved by Fischer [10] in 2015 with a study on the best search techniques adapted to Geometry Friends. The result of that paper was the Subgoal A* strategy, an adaptation from the A* search algorithm, still used by recent submissions.

At this time, there were several proposals to explore the application of Machine Learning in Geometry Friends agents. Solutions like OPU-SCOM [29] [30] used neural networks while others like KUAS-IS Lab [31] and PG-RL [32] opted for *Q*-Learning technique. None achieved particular success because they underestimated the learning problem or there was overspecialization. Not even deep learning [33] got better results. The potential of Machine Learning in Geometry Friends would only be confirmed with circle agent KIT [34] in 2017.

At the same time, a complete multi-agent system was proposed in 2015 [12] (and then updated in 2017 [13]), based on the Rapid-Exploring Random Trees strategy. This proposal participated in all competition tracks, only losing to the KIT agent in the circle one.

In conclusion, the current state-of-the-art for Geometry Friends problems is shown in Table 3.1.

Subproblem	Proposal
Pathfinding	Subgoal A*
Circle Agent	KIT Agent (using Subgoal A*)
Rectangle Agent	RTT Rectangle Agent
Cooperative Agents	RTT Circle Agent & RTT Rectangle Agent

Table 3.1: Geometry Friends State-of-the-Art (before September 2019)

3.1.1 Subgoal A* (2015)

In 2015, Daniel Fischer conducted a study [10] on the representation of the Geometry Friends problem but, more importantly, the feasibility of various strategies available to solve it. In particular, he turned the game levels into weighted directed graphs and tested different versions of A* search (explained in 2.2 and 2.2.1, respectively), even combined with other algorithms like Monte Carlo Tree Search (MCTS). The result of this work was a strategy called Subgoal A* which remains today the best searching technique for agents with similar Planning phase processes.

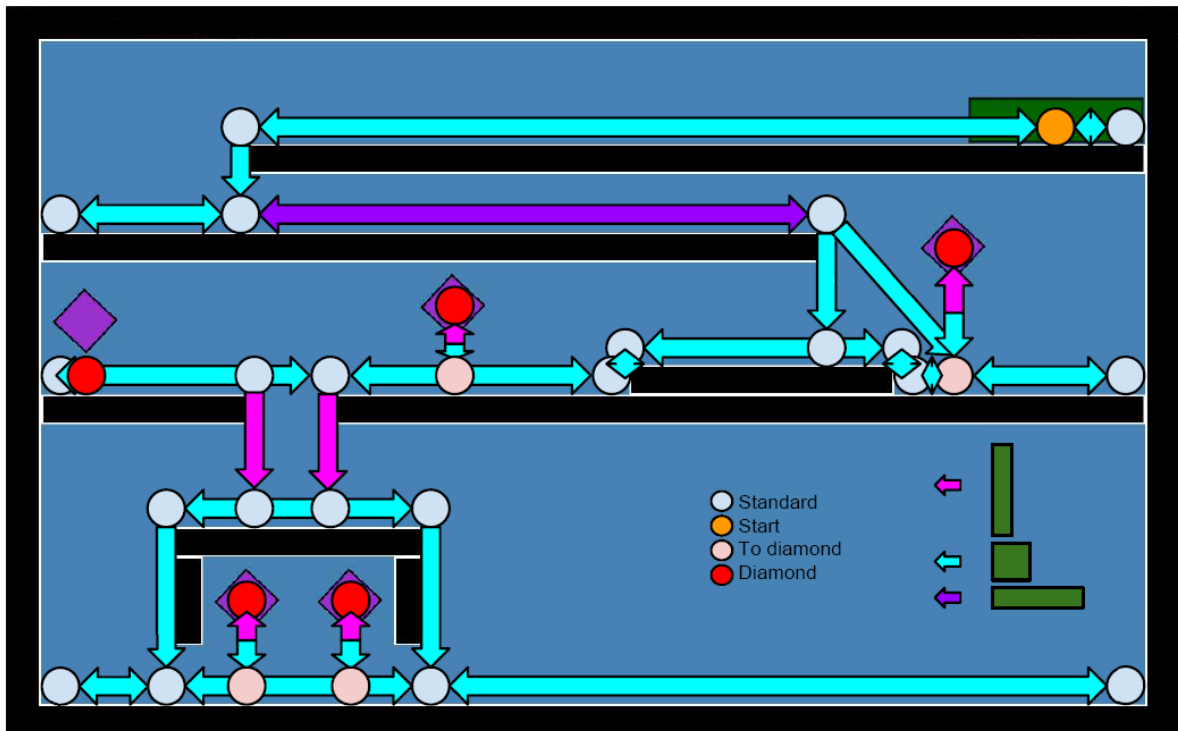


Figure 3.1: Example of Fischer's Geometry Friends Mapping

Fig.3.1 shows how a Geometry Friends level is represented internally in Fischer's proposal. There is one node for the start position, one for every diamond, one for every obstacle edge and one for every possible player landing. Nodes are connected by edges, calculated through a rudimentary process that considers only eight directions and ignore more complex physical properties regarding the agent's trajectory. These edges have the Euclidean distance as cost and they also include information about the rectangle height needed to perform the movement.

The weighted directed graphs built as mentioned were then subjected to several search strategies and the results were evaluated and compared. Subgoal A* search, an A* adaptation, came out as a clear winner. The difference to the original algorithm is a new node property associated to the path diamond collecting order. This allows Subgoal A* to differentiate nodes according to that new property. In addition, no heuristic function is used (its value is zero).

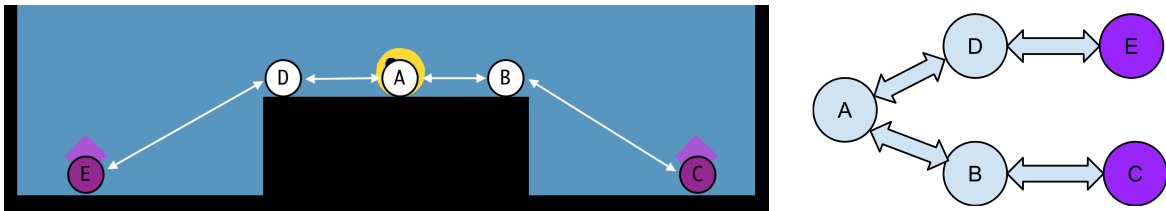


Figure 3.2: Simpler adapted example of a Fischer Graph

In Fig.3.2, we have a more understandable Fischer's graph with five nodes and two of them are diamond nodes (C and E). Without the adaptation, A* would return only the route A-B-C or A-D-E. However, in Subgoal A*, there are different states of B according to the new property: B without a diamond, B with diamond collected at C and B with diamond collected at E. So, if the search is going A-B-C, it does not stop at C. Instead, it returns to B because the first B state did not contain the diamond collected at C. That condition is now true in the current state for all nodes, meaning that the algorithm will eventually arrive at E and return a complete route from A-B-C-B-A-D-E.

3.1.2 PG-RL (2015)

In PG-RL[32] proposal, Quitério summarized Geometry Friends in 3 subproblems:

- Solving Current Platform (collect diamonds of current platform);
- Selection of Next Platform;
- Moving to Next Platform.

However, even before the resolution cycle runs, a module called Platform Manager comes into play at an early setup stage. Its responsibility is to generate and store all platform information as well as the navigation graph corresponding to the current level. A platform is created when the circle can move from the leftmost to the rightmost coordinate of an obstacle without any collision (as shown in 3.3).

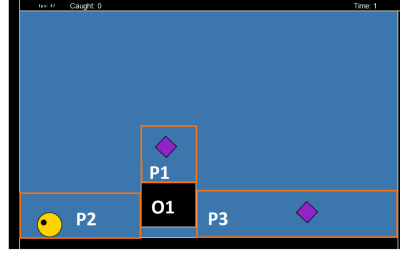


Figure 3.3: PG-RL Platform Identification

Here, unlike Fischer graphs (see 3.1.1), nodes correspond to platforms (not action points). However, the edges and the way they are identified are similar. In this case, it is checked (in a very primitive way) whether there are impeding obstacles within an acceptable distance between platforms. With this, the PG-RL graph is obtained and can be subjected to a Depth First Search (DFS) in order to solve the second subproblem - choosing the next platform.

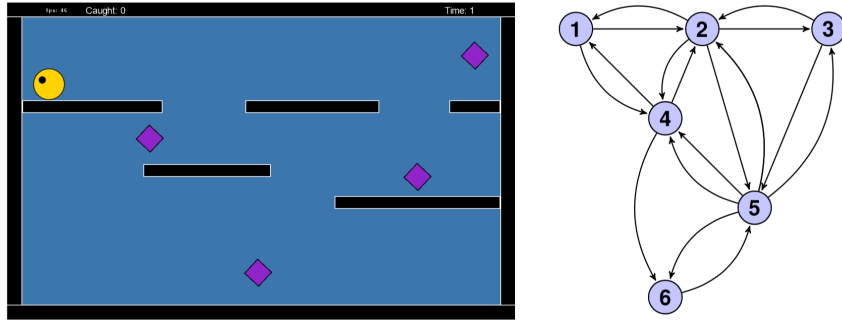


Figure 3.4: Geometry Friends level and corresponding example of PG-RL Graph

Meanwhile, the first and third subproblems are solved with Reinforcement Learning. The states considered and reward parameters used in each solution are shown in Table 3.2.

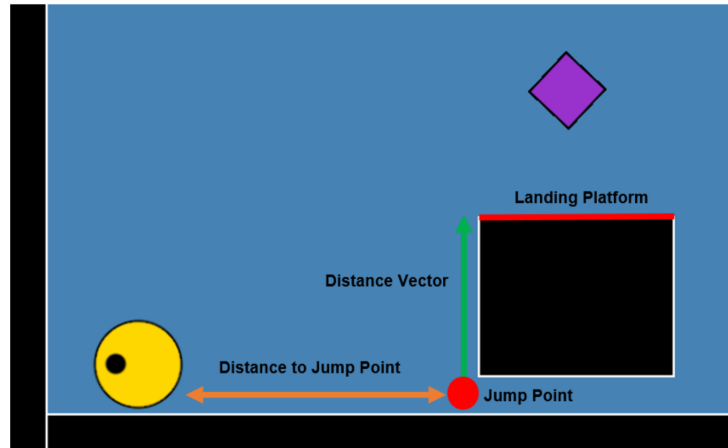


Figure 3.5: State properties when using RL for the subproblem Moving to Next Platform

Due to the number and variety of state properties (and using Figure 3.5 as visual aid), the authors were forced to conclude that the state space size was too big. An implementation such as that specified in the Table 3.2 requires a huge amount of agent training (for each of the subproblems) and has a weak generalization ability.

PG-RL Q-Learning Implementation		
	Solving Current Platform	Moving to Next Platform
Action	ROLL_LEFT ROLL_RIGHT JUMP	ROLL_LEFT ROLL_RIGHT JUMP
State	Distance to platform's left edge; Distance to platform's right edge; Horizontal velocity; Number of diamonds remaining on the platform; Distance vector do closest diamond; Presence of a wall preventing a fall.	Agent velocity; Distance to jump-point; Distance vector; Landing platform width; Graph edge comitted.
Reward	Number of diamonds collected on current platform; Number of diamonds already collected; Distance to closest diamond; Percentage of time available.	Time spent moving; Time spent since committed to edge.

Table 3.2: PG-RL Implementation of Reinforcement Learning

The reason behind the use of such ML technique was to overcome the overspecialisation associated to the first GF submissions. The authors achieved that goal with success since the agent had an equal performance in the private and public levels of the competition. Reinforcement Learning was proven to be a good solution but needing a significant improvement.

3.1.3 RTT (2015, 2017)

In 2015, the strategy Rapid-Exploring Random Trees (RTT) was used on a Geometry Friends solution for the first time [12]. The algorithm calls a function EXTENT (shown in Figure 3.6) to a random state every iteration (limited to the number of tree vertices). This function chooses the next state to the given state x , x_{near} already in the tree, with the help of a distance metric to compare states. The authors objective was to reduce the time spent on the search process even at the cost of the solution optimality. Despite the fact that the submitted version was incomplete, the agent not only overcame overspecialisation but also won the circle track (achieving a third place at the rectangle one).

Two years later, this approach was picked up again and improved towards a cooperative multi-agent.[13]. In this new solution, a state is defined by the position, velocity and size (discrete values) of the agent as well as a list of uncaught collectibles. However, it became evident that using only the RTT algorithm was not enough due to the high dimension of the search space for an almost complete random strategy.

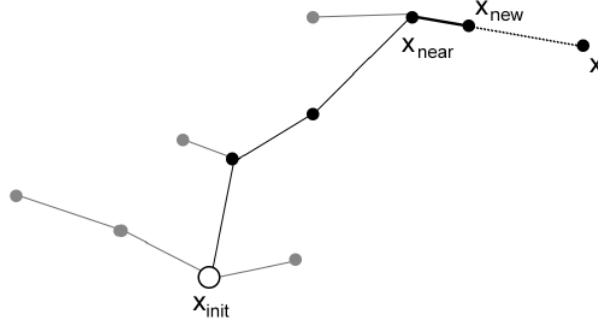


Figure 3.6: RTT EXTENT function

The randomness aspect of action selection was solved with the STP (Skills, Tactics and Plays) strategy. This consists in organising the possible actions of the agent into bigger abstractions like skills, tactics and plays. One Play corresponds to a multiplayer joint action and because of that, was not used in this case. Meanwhile, a Tactic is an agent behaviour model represented by a non-deterministic Finite State Machine consisting of Skills (each one returning an action to execute). If no Skill is suitable, a random action is returned.

For the STP to be used, the Planning phase was itself divided in 3 tasks in a similar way to the PG-RL agent:

- Select Diamond to Collect: no need to implement skills but priority is given to higher diamonds and to the ones reachable at the current platform.
- Go To Diamond Platform: two different set of skills according to the platform position relative to the player - higher (shown in Algorithm 1) or lower (platforms with same height or with obstacles between them are not explored).
- Collect Diamond: set of skills to collect the diamond, covering all the situations where the player is under it or by its sides.

Algorithm 1 Go to Higher Platform

```

1: function SKILL_HIGHER_PLATFORM(node, diamond)
2:   if HEIGHT_DIFFERENCE(node.Position, diamond.Platform.Position)  $\leq$  circle_max_reach then
3:     if UNDER_PLATFORM(node.Position, diamond.Platform) then
4:       return ROLL;
5:     else
6:       if AT_PLATFORM_LEFT(node.Position, diamond.Platform) then
7:         return ROLL_LEFT_OR_JUMP;
8:       end if
9:       if AT_PLATFORM_RIGHT(node.Position, diamond.Platform) then
10:        return ROLL_RIGHT_OR_JUMP;
11:      end if
12:    end if
13:  end if
14:  return RANDOM_ACTION;
15: end function

```

Another obstacle of RTT is the difficulty in using the metric distance by the EXTENT function, giving that the laws of physics are being strictly applied (see Figure 3.7). Instead, the authors decide to use Balanced Growth Trees (BK-BGT) to find the nearest node, x_{near} . This strategy recognises leaf and non-leaf nodes of the tree. When selecting a new state to extend, a *depthaverage/branchingaverage* ratio is calculated and compared to a constant (obtained through testing). If greater, a random non-leaf node is selected. Otherwise, it returns a random leaf node. This means that the search direction is controlled due to a forced boundary on the tree depth (simulation of the future) and the tree width (number of possibilities on each level).

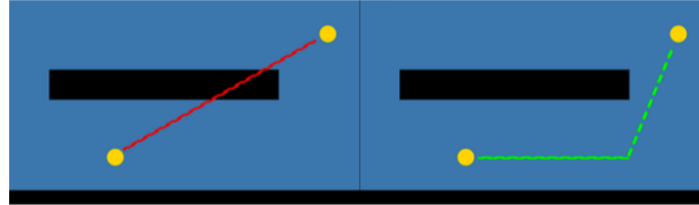


Figure 3.7: Difficulty of using euclidean distance as a metric distance for the RTT

The BK-BGT strategy was tested along with STP, producing good results on more complex levels. On the negative side, the agent struggled to find a complete solution. A time limit was then imposed, leading to the return of a partial solution in case of timeout. Partial solutions with more collected diamonds or reaching higher ones, have priority. The final results were also improved by favouring simple actions (slide left and slide right) instead of complex ones (jump and morph) and with path cleaning (elimination of redundant and unnecessary actions).

Regarding the Control phase, the authors relied heavily on agent acceleration values to achieve the desired velocity and position. Another big difference to the 2015 proposal is that a re-planning is now executed in this phase when needed, through path failure identification processes.

Algorithm 2 Circle - Catching a High Diamond

```

1: function CATCH_FROM_RECTANGLE(diamond)
2:   if circle_on_rectangle then
3:     if circle_below_diamond && HEIGHT_DIFFERENCE(circle, diamond) ≤ circle_max_reach
4:       then
5:         return JUMP;
6:       else
7:         return MATCH_RECTANGLE_VELOCITY();
8:       end if
9:     end if
10:    if rectangle_below_diamond then
11:      return JUMP_TO_RECTANGLE();
12:    else
13:      return WAIT;
14:    end if
15:    INVALID_NODES(plan, invalid_list);
16:    if invalid_list.Count != 0 then
17:       $\mathcal{T}$ .REMOVE_INVALID_NODES();
18:       $\mathcal{T}$ .NEW_ROOT(plan.Points[current_point]);
19:    end if
20:  end function

```

From this work also resulted a complete cooperative agent that was submitted to the competition's Cooperation Track. New skills were added not only because the search space is now exponentially bigger but also because coordinated actions are needed. An example is shown by Algorithm 2 as the circle must wait for the rectangle to position and use it to collect a high diamond. The re-planning methods were also reviewed due to the fact that the players can interrupt each other movement.

The RTT proposal had a good performance on 2017 GF competition, winning the Rectangle and Cooperation Track and only being surpassed in the Circle Track by the KIT agent (see Section 3.1.4). The authors even evaluated the RTT agents in cooperation with human players, achieving a reasonable success. All these results show how this is the best (except Circle Track) and most complete GF solution to date.

3.1.4 KIT (2017)

One of the last submitted GF proposals and the one that inspired us most, is the KIT [34] circle agent, by Hiroya Oonishi and Hitoshi Iima. Their objective was to perfect the application of Reinforcement Learning and the results achieved by the PG-RL agent in 2015. Improving generalization ability was the authors' focus since the agent training must prepare it to play on any type of level and consequent situations.

The main drawback of the PG-RL agent was the search space size caused by the excessive use of RL to solve most of their tri-divided problem. This new proposal divides the overall GF problem in two steps:

- Planning: find the shortest path to collect all diamonds.
- Control: action selection to follow the planned path.

The first step includes building a weighted directed graph with platforms as nodes and moves as edges. As shown in Figure 3.8, the platform identification process is very similar to the PG-RL one: the character is able to move from the left edge to the right edge of the obstacle without any type of collision.

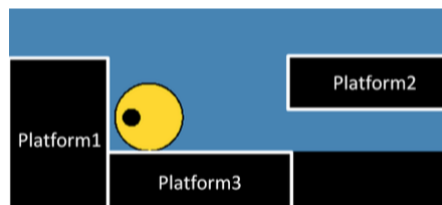


Figure 3.8: KIT Platform Identification

As for the edges identification, jumping and falling trajectories (with or without obstacles) are calculated using the equations of motion (Equations 3.1 and 3.2). Values such vertical jump velocity and gravity acceleration were obtained with stochastic gradient descent since they are not public. Making these trajectory predictions allows the verification of platforms to which the character can move (graph edges) and the diamonds it can collect.

$$x = v_x t \quad (3.1)$$

$$y = v_y t - \frac{1}{2} g t^2 \quad (3.2)$$

KIT graph edges also have information about the movement type, horizontal velocity and reachable diamonds. The final Planning task is to apply a search algorithm. Subgoal A* [10] strategy was chosen by the authors, given the previously success achieved by the algorithm.

It is in the Control phase that Reinforcement Learning, in particular the Q -Learning technique, is used to get the best actions given a target coordinate x and horizontal velocity v_x . The Table 3.3 shows how actions, states and rewards were defined in order to reduce the training required and improve the agent generalization ability.

KIT Reinforcement Learning Implementation	
Action	ROLL_LEFT ROLL_RIGHT
State	Relative Distance Relative Velocity
Reward	Target Reached

Table 3.3: KIT Q -Learning Implementation

In comparison with Table 3.2 in Section 3.1.2, we have a much more simple and lighter implementation. The actions considered are enough to reach the target position with the target velocity (the JUMP action is ignored because it is a punctual action and, most of the times, the target itself). A state is characterized by a discretized and limited value for relative distance and velocity in relation to the target, making use of the symmetric property to reduce even more the state space (concept shown in Figure 3.9). For reward, a high positive value is given if the target is reached, otherwise, it receives a negative one.

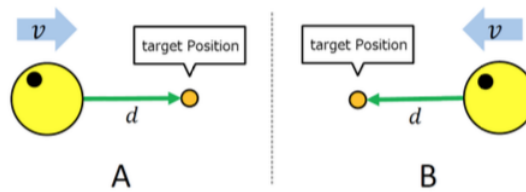


Figure 3.9: Use of symmetric property by KIT to reduce state space

The agent training is done in the level of Figure 3.10 with the target x as the middle position. The authors performed 5000 trainings for each possible value of target horizontal velocity (there are ten discretized values).



Figure 3.10: KIT Training Level

Following training, new 25 levels were generated with target positions distributed randomly across a simple level without platforms (using the same level layout, the target velocity was also chosen randomly). Then, the KIT agent, a random agent and an agent that always slides to the target direction played all three individually those 25 levels. In fact, the third agent never reaches the target velocity but the time it takes to get to the target position can be used as the lower bound. The comparison between the time taken to get the target position and velocity (with the aforementioned particularity of the third agent) can be seen in Figure 3.11. The minimum margin between KIT time and lower bound (third agent time) proves that the process of action selection by KIT is practically optimal.

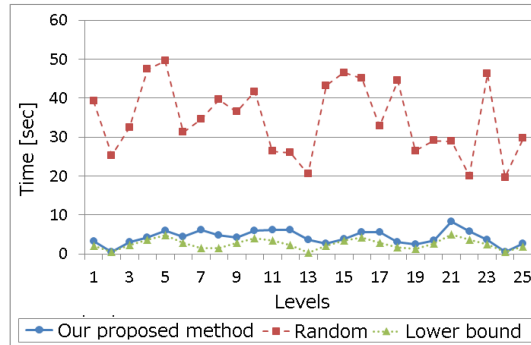


Figure 3.11: Action selection by KIT agent is practically optimal

The KIT agent not only improved the implementation of Reinforcement Learning in Geometry Friends (previously done by PG-RL) but it also won the 2017 Circle Track with near perfect performance. That is why this proposal is our motivation to expand the use of RL to the rectangle agent and, most importantly, to a multi-agent system.

3.2 Coordinated Multi-Agent Object Transportation

We now make a brief reference to a successful use of the Team Q -Learning strategy. In this case, the problem comes down to the transport of an object on a two-dimensional discrete grid with 7x6 cells. Two agents have to transport the object (pushing it simultaneously in the same direction) to the home base (delimited by a dashed black line) in minimum time, while avoiding the obstacles (gray blocks). This is taken from a general study [18] regarding Multi-Agent Reinforcement Learning (MARL).

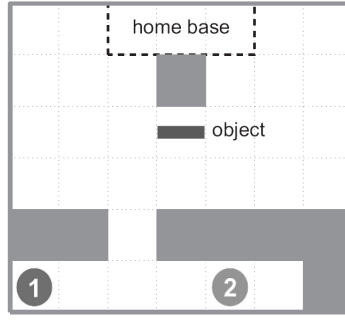


Figure 3.12: Object Transportation Problem

Example Team Q -Learning Implementation	
Action	$\{[a_1, a_2], a_1, a_2 \in \{LEFT, RIGHT, UP, DOWN, STAND-STILL\}\}$
State	$p_{x,1} \in \{1, 2, \dots, 7\}$ $p_{x,2} \in \{1, 2, \dots, 7\}$ $p_{y,1} \in \{1, 2, \dots, 6\}$ $p_{y,2} \in \{1, 2, \dots, 6\}$ $g_1 \in \{FREE, GRASPING-LEFT, GRASPING-RIGHT\}$ $p_2 \in \{FREE, GRASPING-LEFT, GRASPING-RIGHT\}$
Reward	<p>1, if an agent has just grasped the object</p> <p>10, if the object has reached the home base</p> <p>0, in all other conditions</p>

Table 3.4: Example Team Q -Learning Implementation

The authors used several MARL algorithms, one of them being Team Q -Learning. To implement it (see Table 3.4), the actions considered are all the possible joint actions, the states include information about each agent position and transportation. The reward is maximum when the object is at the home base but it is also positive if an agent is transporting the object.

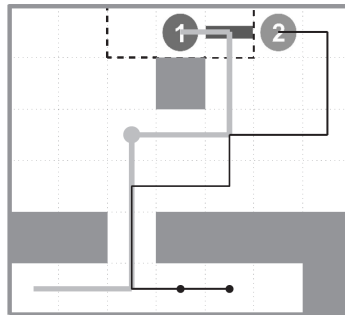


Figure 3.13: Solution obtained by Team Q -Learning

The algorithm had a good performance, obtaining the solution presented in Figure 3.13: agent 1 travels along the thick gray line while agent 2 along the thin black one. The dots mark the position where agents stood still and their sizes represent the time spent waiting for the partner.

We included this multi-agent system in the introductory part of our paper because the problem, while simpler, resembles the Geometry Friends one and the result obtained is also similar to what we hoped to achieve with our proposal.

Chapter 4

System Overview

Our proposal is the third to apply Reinforcement Learning (in particular the Q -Learning technique) to the Geometry Friends solution. The previous two had opposite results. The circle agent PG-RL relied on learning to solve most of the game and eventually the need for training was too much. On the other hand, the KIT circle agent divided the overall problem more efficiently and achieved excellent results using Q -Learning in part of the solution. The latter turned out to be our biggest inspiration in developing our proposal.

In this chapter, we present our approach to the existing subproblems and the first decisions regarding the system building. For easier understanding, we refer to these decisions by organizing them according to its execution phase. We explain how we took circle agent KIT, created our rectangle agent and put it all together in a system capable of learning and performing cooperative movements. In the end, we show the entire solution architecture whose components are detailed in their own chapters.

4.1 System Execution Phases

Phase	Description	Occurrence
Training	The agent/system plays in a controlled environment (training levels) numerous times, subjected to Reinforcement Learning.	Training Levels (only occurs once in the agent/system lifetime)
Mapping	Given the information about a certain level to be played, the agent maps all elements in a data structure to be used for that level resolution.	Before Level
Control	During Planning Stage, the agent obtains the best level solution and current next move, using data from its sensors and mapping. During Action Stage, the agent decides the next action according to next move, current state and training records.	During Level (occurs iteratively until completion/timeout)

Table 4.1: System Execution Phases

4.2 Circle Approach

The KIT circle agent is not perfect, especially regarding obstacles and ceiling collisions. Despite that, it is the best individual player ever proposed to Geometry Friends, solving most of the levels with high score. Thus, from an early stage, we decided to incorporate this agent into our system without changing its individual gameplay. Nevertheless, the cooperative aspect forced us to add various multi-agent features to the circle (without changing individual processes). More about that in Section 4.4.

In Table 4.2, we present this circle approach organized by the execution phases.

Phase	Circle Approach (KIT Approach)
Training	Solving the subproblem of reaching a certain position and horizontal velocity, using Q -Learning.
Mapping	Level as a weighted directed graph with platforms as nodes and moves as edges. Simulation of fall and jump trajectories based on the motion equations.
Control	Selection of the best next move from graph using Subgoal A* Search. Selection of the best next action by consulting the training data, in order to reach the position and velocity of the next move.

Table 4.2: Circle Approach (KIT Approach)

4.3 Rectangle Approach

Our goal is to join the two agents in a system, having the same type of structures and processes. Knowing this and believing that it could outperform the best current rectangle (the RTT agent), our rectangle approach is very similar to the circle one. In Table 4.3, we point out the differences compared to the one presented in Table 4.2.

Phase	Rectangle Approach (vs Circle Approach)
Training	Same training but for each form/height.
Mapping	Identification of platforms and moves (except jump) considering every form/height.
Control	Same individual processes but with own graph, actions and training records.

Table 4.3: Rectangle Approach (vs Circle Approach)

Note that, contrary to the circle, the complexity of the rectangle problem depends on the possible forms that we decide to consider. This is particularly evident when identifying platforms and movements (during mapping) with consequences on the amount of training required. Therefore, we proceed to an analysis of several different levels in order to understand how to minimize the considered forms, without affecting the efficiency of the character. Our conclusion was the three rectangle forms/heights shown in Figure 4.1. Those are the only ones considered throughout this work.

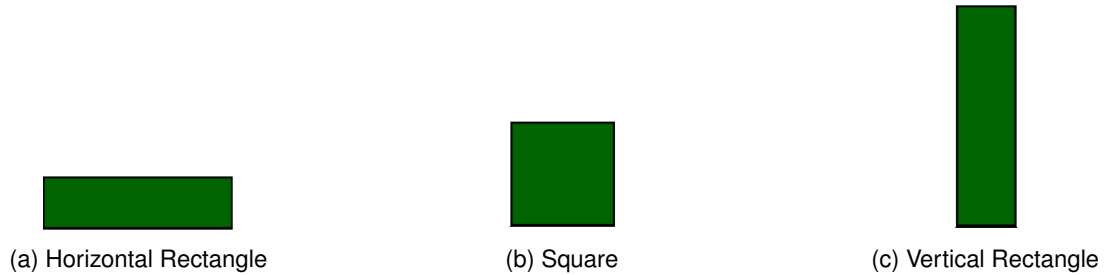


Figure 4.1: Rectangle Possible Forms/Heights

4.4 Cooperation Approach

Cooperation greatly increases the complexity of the Geometry Friends problem. For this reason, we found it necessary to simplify and reduce the coverage of cooperative movements by our system. To achieve this, we analyzed which were the best cooperative moves, specifically those that are most common and effective in solving multiplayer levels. The conclusion was that most levels included in the Cooperation Track (and also out of competition) are solved with moves that involve the "Riding" ability.

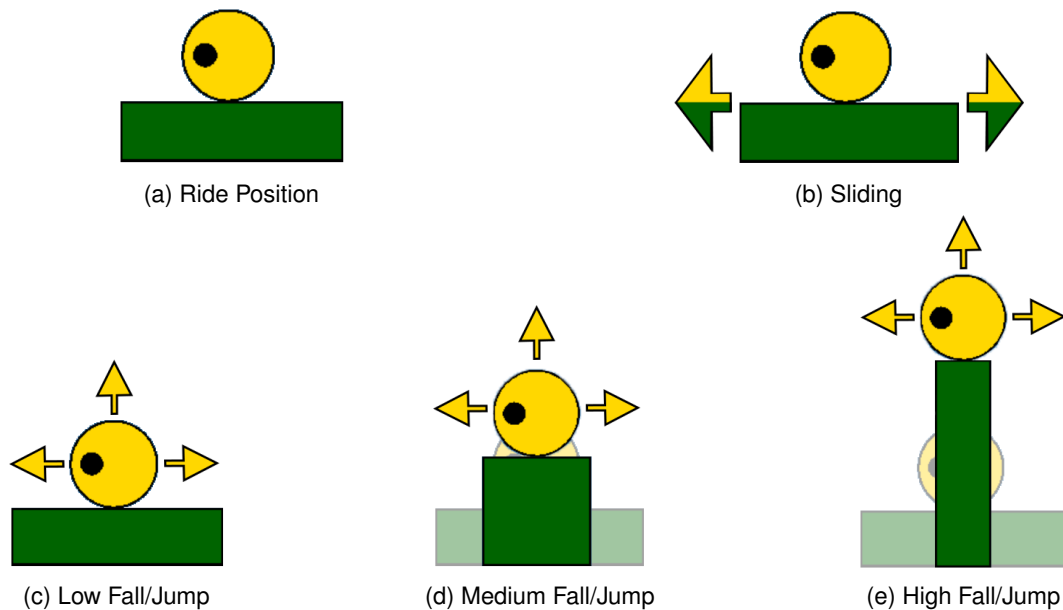


Figure 4.2: Riding Ability

As we can see in Figure 4.2, the Riding ability can be very versatile. To make it possible, agents start in the Ride Position (Figure 4.2a), which means that the circle must be able to land on the horizontal

rectangle. After that, the two characters can move sideways together without breaking the Ride Position rule: the circle stays on top of the horizontal rectangle.

In addition, all possible circle jump and fall trajectories must be simulated from the three rectangular forms (as shown in Figures 4.2c, 4.2d, 4.2e). Due to the poor stability of the Ride Position and because it has little influence on the results, we only considered these movements with the rectangle stationary.

Implementing this new type of movement in the system has the following requirements:

- multi-agent learning (Training phase);
- identification and incorporation in the agents' graphs (Mapping phase);
- agents' synchronization mechanisms (Control phase).

The synchronization problem would be solved if both agents built an equal graph or at least one with the same cooperative movements. In addition, they would have to select the same cooperative movement at the same time. However this solution is heavy, repetitive and ultimately unnecessary knowing that there is communication channel between agents. Therefore, we decided to use a leader-follower strategy in which the leader makes the cooperative decisions and communicates to the follower.

Our system leader is the circle because it is the best candidate. The reason for this is that it is easier for this agent to check if and where the characters can achieve the Ride Position. Specifically, if the circle can land on the horizontal rectangle at the end of a fall or jump trajectory. We present in Table 4.4 the consequences of this cooperation approach for each agent in the three phases of the system.

Phase	Cooperation Approach	
	Circle Agent (Leader)	Rectangle Agent (Follower)
Training	Joint agent training in order to learn move sideways while maintaining Ride Position. Use of Team Q -Learning as MARL technique, obtaining a new team Q -Table. Individual training and corresponding Q -table are not changed.	
Mapping	Identification and graph incorporation of platforms where Ride Position is possible as well as moves that allow it and cooperative moves starting from it.	-
Control	New Cooperation Manager to send cooperative moves and achieve synchronization.	New Cooperation Manager to receive cooperative moves, incorporate them in the graph and achieve synchronization.
	Consult individual or team Q -table records according to Move type.	

Table 4.4: Cooperation Approach

4.5 System Architecture

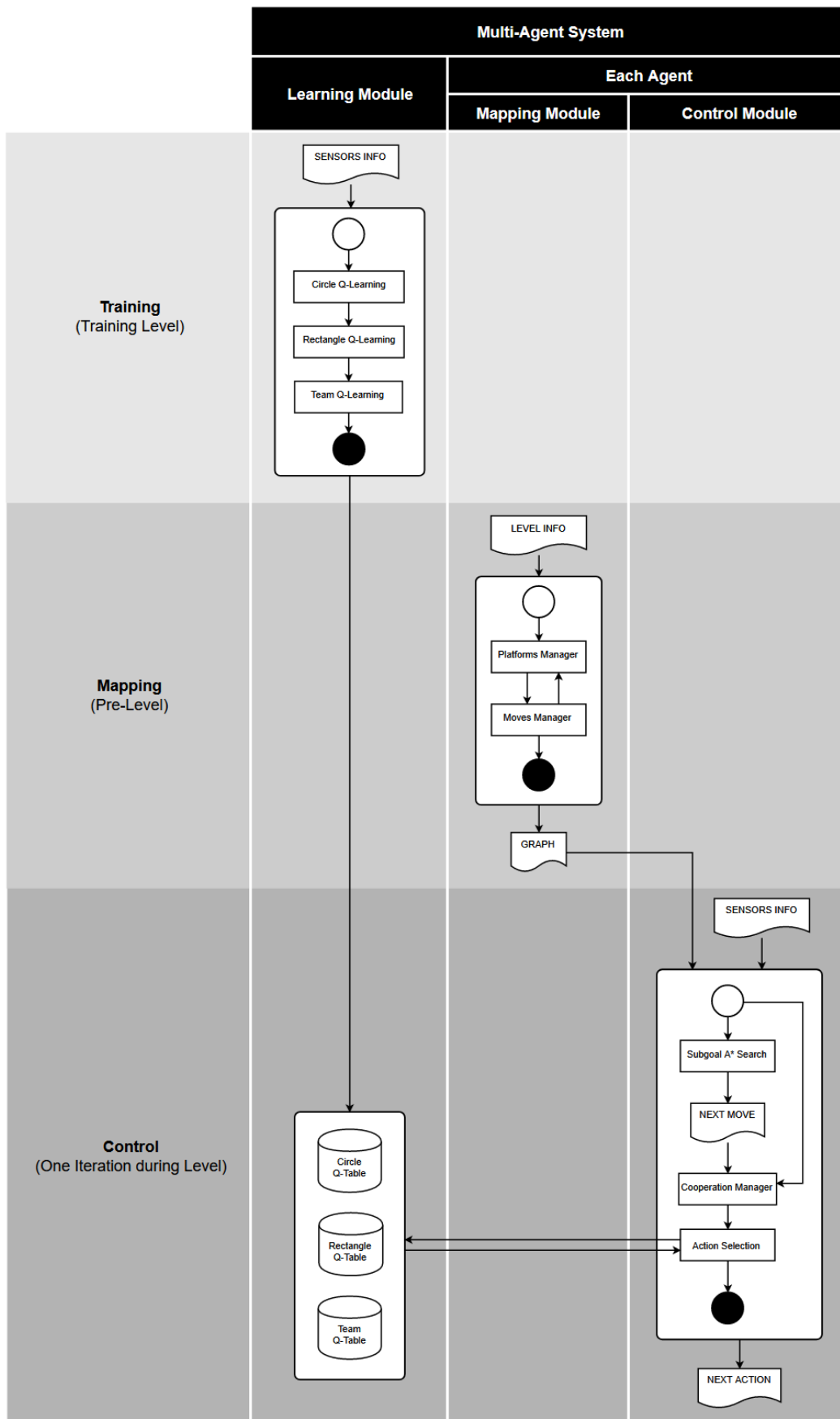


Figure 4.3: System Architecture

Before going into chapters exploring each individual component, we provide a first look at the entire system architecture through Figure 4.3. Specifically, we enumerate the modules and key data structures, organized by the execution phase in which they participate.

4.5.1 Training

Unlike the next two phases, training is performed in a controlled environment (training levels), where the agent plays numerous times. Not only that, but it is also the only one that does not repeat for each game level (not training ones).

The goal of the Learning Module is to make the agents learn to perform the required movements. In our system, this means reaching the position and horizontal velocity required by the selected move. The circle and rectangle training is individually guaranteed by the Single Q -Learning technique. In addition, there is a third training stage for both the circle and rectangle to learn together how to move sideways while maintaining the Ride Position. The latter uses Team Q -learning.

The input required for all the three training sessions is simply the sensors information received by each agent. In the end, the three Q -tables are retrieved and stored permanently to be used by the Control Module during level gameplay.

4.5.2 Pre-Level

When a level is selected, before time counter begins, the system receives as input the information about it. This includes characters' initial positions as well as the coordinates and size of all obstacles and collectibles. The Mapping Module is responsible for building the circle and rectangle graph as output from this phase. These graphs have platforms as nodes and moves as edges, respectively identified by the Platforms Manager and Moves Manager.

4.5.3 During Level

During a game level, every move and action is selected and performed by the Control Module. For this reason, this module can be separated into Planning Stage and Action Stage, from which result NEXT MOVE and NEXT ACTION, respectively. The latter includes the participation of the Learning Module.

Starting at Planning Stage, the initial input is the graph built by the Mapping Module and the agent's sensors information. Therefore, it is easy to identify the current platform that serves as the starting node for Subgoal A* Search. The result from this algorithm is a sequence of moves that is the shortest path to solve the level. From that, the NEXT MOVE is retrieved since it is the first path edge. Note that, this is only necessary if NEXT MOVE is not chosen yet (otherwise, this step is skipped).

We then enter the Action Stage. In case this move is a cooperative one, the Cooperation Manager handles the messages and the associated synchronization between the agents. Knowing the NEXT MOVE (associated to the current platform), the agent only needs to reach the position and horizontal velocity required. If this is already true, actions like jumping, morphing or falling are performed. Otherwise,

the agent consults the correct Q -Table in order to choose the best NEXT ACTION (single agent training records if individual move or team training records if cooperative).

This completes a full iteration. The Control Module continues on cycle, searching and performing the best NEXT MOVE and NEXT ACTION until it successfully solves the level, or otherwise, until time runs out.

Chapter 5

Mapping Module

The Mapping Module is responsible for mapping a game level before the agent begins to play. Specifically, the purpose of this unit is to condense all data on obstacles (black, green and/or yellow) and collectibles in a single graph for each agent. Ideally, we would want an equal graph for both characters but given the difference in the nature of obstacles and movement abilities, this task is impossible.

In Figure 5.3, we present the structures that support our graphs: Platforms as nodes, identified by the Platforms Manager, and Moves as edges, identified by the Moves Manager. The state of an agent and its partner is received by its sensors natively in the form of CircleRepresentation and RectangleRepresentation. We work with this information about character coordinates, velocity and size through a single State structure, standardizing and facilitating many of the mechanisms not only during the Mapping phase, but also in the Control phase.

In the rest of this chapter, we will detail the construction of a graph and the differences between circle and rectangle from the perspective of platforms and movements.

5.1 Platforms Manager

The definition of a platform and its identification process by the KIT agent (see Section 3.1.4) had excellent results but not only is it not suitable for the rectangle but it does not consider the cooperative aspect as well. Since the general problem is now more complex, the planning module has to cover all new situations. This means that we had to expand the use given to graph elements. In the case of platforms, we now have the three types listed in the model in Figure 5.3: BASIC, GAP, COOPERATIVE. For example, as we will see in this section, platforms may not even be obstacles but serve as abstractions for the study of character movements.

5.1.1 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.

The difference of this definition to the ones we saw earlier is the last part regarding the allowed height. We added this property due to the rectangle morphology and we limited its possible values to four. As we introduced in Section 4.4, our system works with only three rectangle heights (Figure 4.1). To these we add the Ride Position, resulting in the four possible values for a platform's allowed height:

- Horizontal Rectangle (50 units);
- Square (100 units);
- Ride Position (130 units) = Horizontal Rectangle (50 units) + Circle Diameter (80 units);
- Vertical Rectangle (200 units).

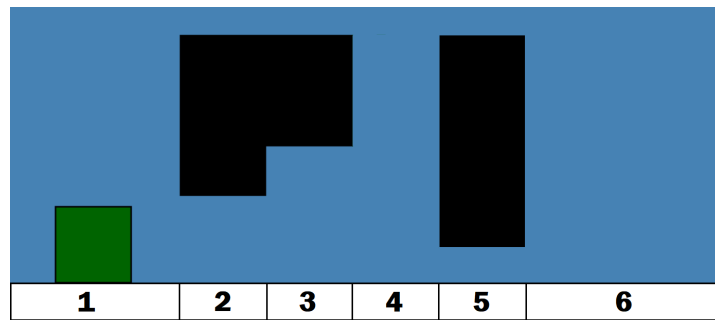
In Figure 5.1c we have a partial level map with a seemingly unique and continuous platform before being submitted to Platforms Manager. The platform identification outcome for the circle and rectangle graph are, respectively, in Figure 5.1b and Figure 5.1c.



(a) Partial Level Example



(b) Circle



(c) Rectangle

Figure 5.1: BASIC Platforms Identification

First of all, obstacles of the same color as the character are completely ignored. The rest are seen as black because there is no reason to differentiate them knowing they are obstacles.

Distinguishing BASIC circle platforms is quite straightforward: if there is space for the circle to be on top of an obstacle then it can be considered a platform. That is why in Figure 5.1b, we have two BASIC platforms separated by a space where there was not enough height for its passage.

Due to the rectangle's ability to morph, this whole process is more complicated. Whenever a change in allowed height influences the forms the rectangle can assume, a new platform is created.

For example, in Figure 5.1c:

- From platform 1 to 2, the ride position and vertical rectangle are not possible.
- From platform 2 to 3, the ride position is now allowed (vertical rectangle still not).
- Platform 4 allows every height.
- Platform 5 is only possible with horizontal rectangle.
- Platform 6 allows every height.

The platform 5 is a good example of how variations in obstacles' allowed height do not always result in new platforms. Considering only the four possible allowed heights, if there is no difference between the allowed heights of one obstacle and another, then the BASIC platform remains the same.

5.1.2 GAP Platforms

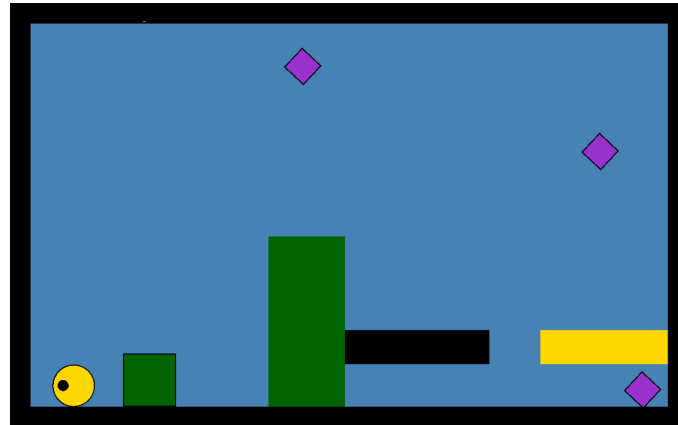
The rectangle has another particularity: the ability to fall at zero velocity between platforms through morphing. This is possible when there is a sufficiently large gap for the rectangle to fall vertically (larger than 50 units) but at the same time small enough for the rectangle to pass horizontally (smaller than 200 units). These are the GAP platforms, unique to the rectangle. They are not obstacles but they represent possible transitions between platforms and vertical falls (useful abstractions for the Moves Manager).

5.1.3 COOPERATIVE Platforms

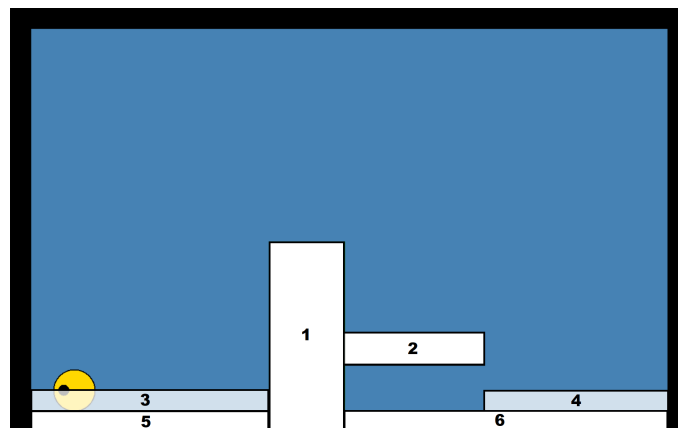
Our cooperation approach (Section 4.4) assumes that the agents' graphs, as the Mapping main tool, contain all the information and possibilities regarding multiplayer movements (task of the Moves Manager). Knowing the existence of a communication channel between the agents, it would be very expensive and unnecessary to make these calculations for both graphs.

That is why we use a leader-follower strategy in which the circle is the system leader. This means that the circle character is responsible for mapping and planning the cooperative moves as well as sending them to the rectangle (more on this in Section 6). Theoretically, this development decision would only have consequences later in moves identification. However, the Moves Manager's task is made easier by the existence of a new type of abstract platforms called COOPERATIVE. These platforms represent the possible rectangle positions that allow circle landing on top of it, to assume the Ride Position.

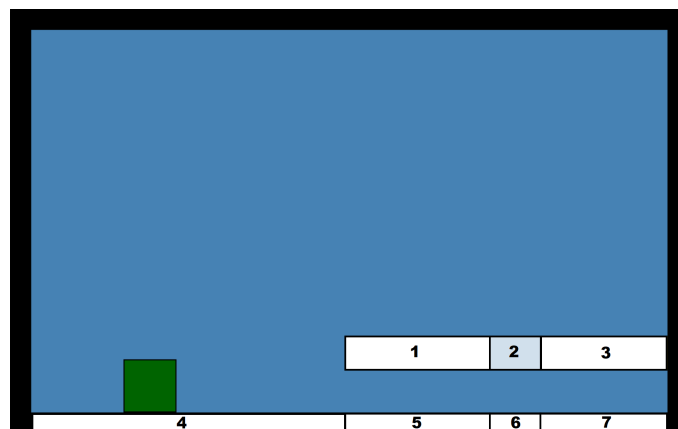
Identifying them implies using the rectangle graph to get its BASIC platforms. Before being considered as circle's COOPERATIVE platforms, the Platform Manager verifies if each one allows the Ride Position and if it is reachable from the rectangle starting position. After checking those conditions, the platform is inserted as a node in the circle graph (with the horizontal rectangle height added to the original platform height). In Figure 5.2b, we see an example of this circle platforms identification process, in which platforms 3 and 4 are COOPERATIVE ones.



(a) Example



(b) Circle



(c) Rectangle

Figure 5.2: Platforms Identification

5.2 Moves Manager

Moves are one of the main structures of our system because they are the connection between platforms and the agents' state (as shown in Figure 5.3).

One Move is defined as a trajectory that allows the character to change platforms, collect diamonds or both. For it to be successful, the agent must match the state indicated in the Move. This includes reaching a particular position, horizontal velocity and height. When all these conditions are met, the action required by the Move type, is performed: jump, morph, roll sideways, move sideways or none.

Throughout this section, we will detail the different types of Move in our system.

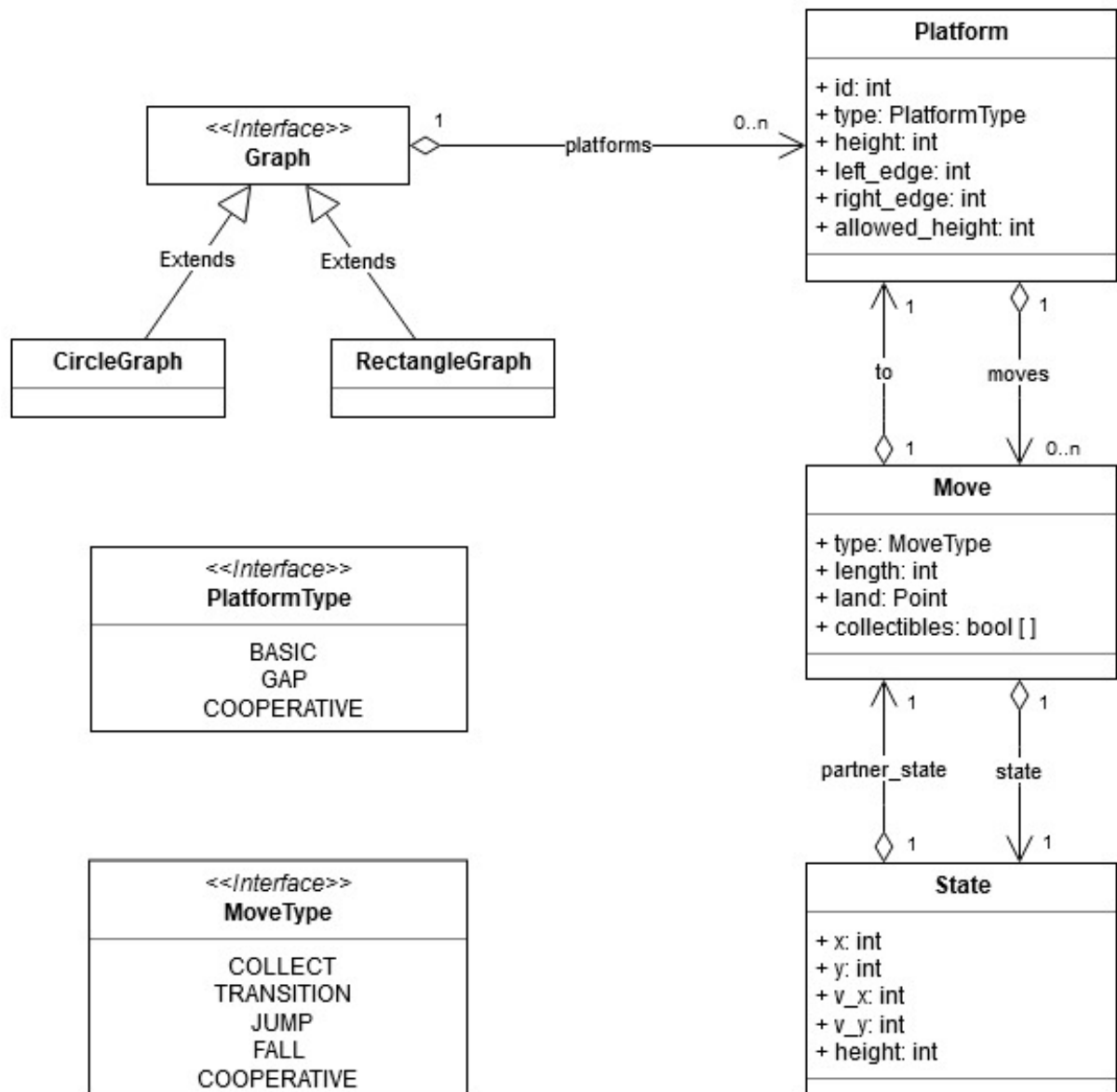


Figure 5.3: Graph Model

5.2.1 COLLECT Moves

The COLLECT move is the most simple one. It only requires the agent to position itself in order to guarantee a collectible without changing the current platform.

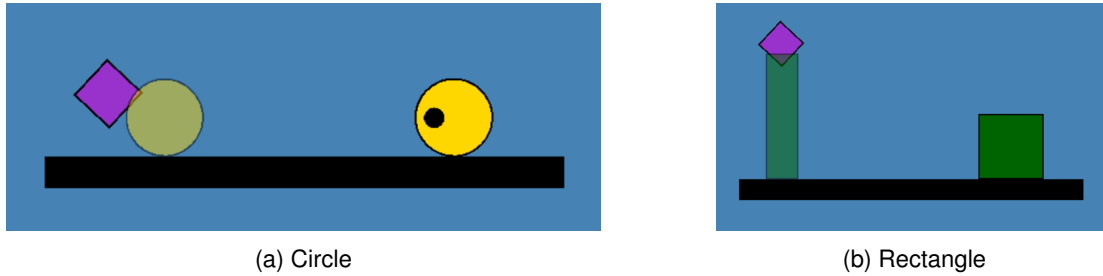


Figure 5.4: COLLECT Moves

For the circle, Move Manager just has to check if the character overlaps any diamond (as in Figure 5.4a). For the rectangle, this check must be done for each possible form/height (horizontal, square and vertical) as the example of Figure 5.4b. The character height required by the Move is included in the state property.

5.2.2 TRANSITION Moves

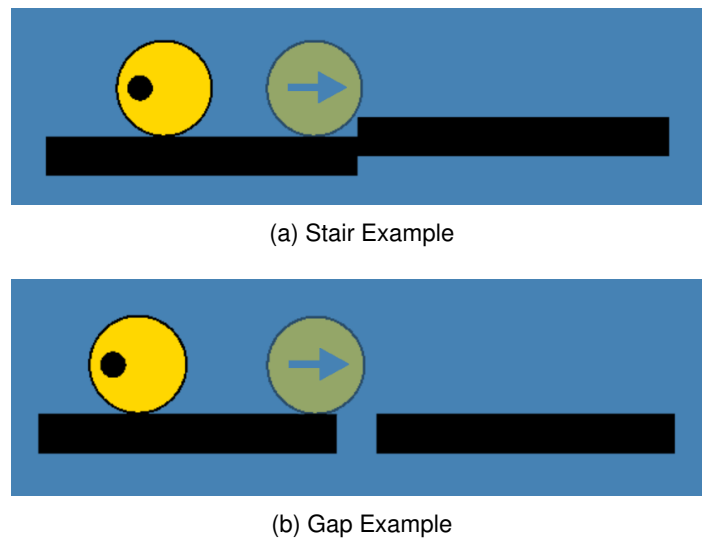


Figure 5.5: Circle TRANSITION Moves

One TRANSITION move is a change on the current platform, collecting diamonds, or not, along the way. They are possible when the distance between platforms and the difference in their heights is not impeding (nor are there any obstacles between them). In the circle case, this definition is enough to identify such movements.

The rectangle case is once again more complicated due to its morph ability. The transition between platforms may force it to take a different form, otherwise the Move is not possible. This also means that this character has greater flexibility in situations involving steps and gaps (see Figure 5.6). These can have bigger dimensions than the ones for the circle (exemplified in Figure 5.7).

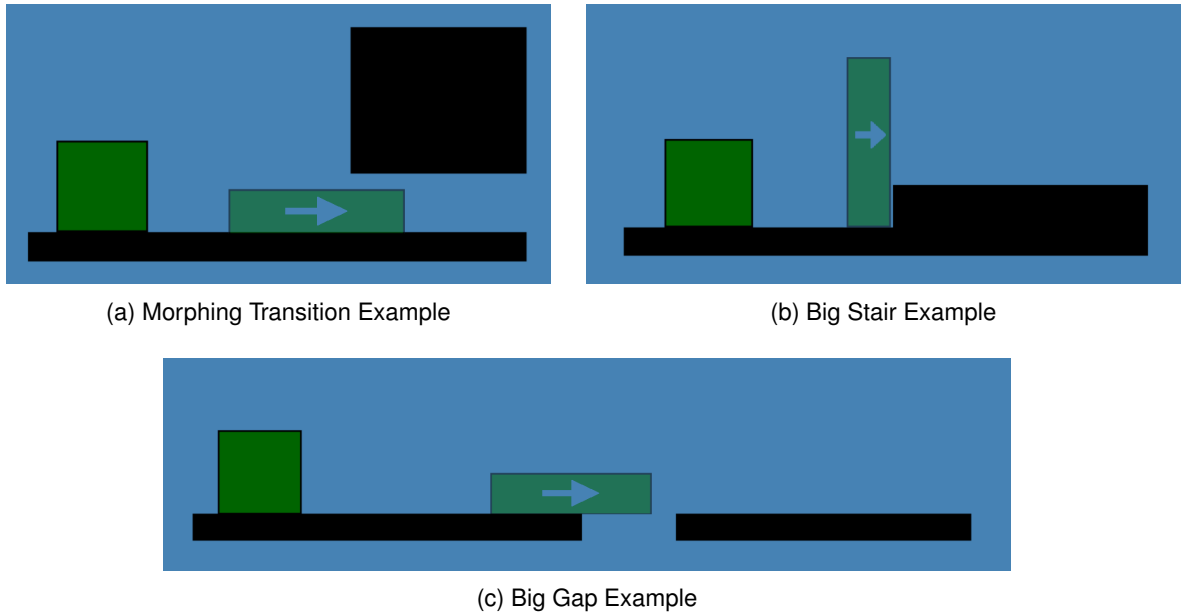


Figure 5.6: Rectangle TRANSITION Moves

5.2.3 FALL Moves

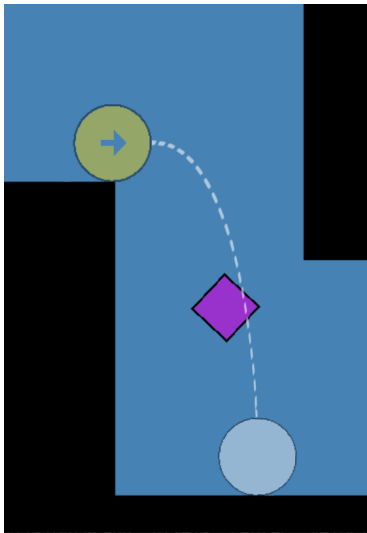
The FALL moves are studied using the motion equations 3.1 and 3.2) (such as agent KIT - Section 3.1.4). Using the platform corners as starting positions simulates the trajectories with different initial horizontal velocities. These values are discretized (in intervals of 20) and limited to the acceleration that the platform length allows (maximum value of 200). This means that in the worst case scenario, ten initial horizontal velocity values are tested for each direction (left and right).

Diamonds that are intercepted by the trajectory are recorded in Move structure. In case of obstacles, this trajectory is considered invalid. However, there is a special sensitivity to two types of collision:

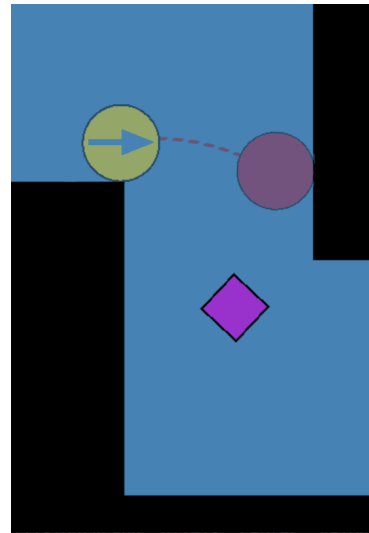
- **Ceilling Collision:** if the top of the character is the first to come into contact with an obstacle, it is considered ceiling collision (it has its own boolean flag in Move). The trajectory is not invalidated but the horizontal and vertical velocities are reduced to one third of the pre-collision values.
- **Platform Landing:** if the bottom of the character is the first to come into contact with an obstacle that is a platform, then the target platform and landing position of the Move have been found.

This trajectory simulation method works very well for the circle, with only a higher margin of error after collisions. This problem gets worse in the rectangle because the rotation of the character during the fall influences the study of collisions (something that does not happen with a circular shape). Moreover, currently, the game does not give access to the inclination angle value of the rectangle. The solution found for the rectangle (and experimentally tested with good results) was to simulate its trajectories as if it were a circle (of diameter equal to its smallest edge).

One feature that does not exist in the circle is the possibility of falling without initial velocity through GAP platforms. The reason for the existence of those platforms is precisely to facilitate the identification of this type of move.

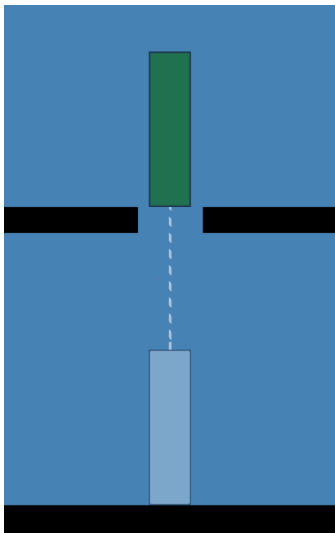


(a) Successful fall with minimum velocity

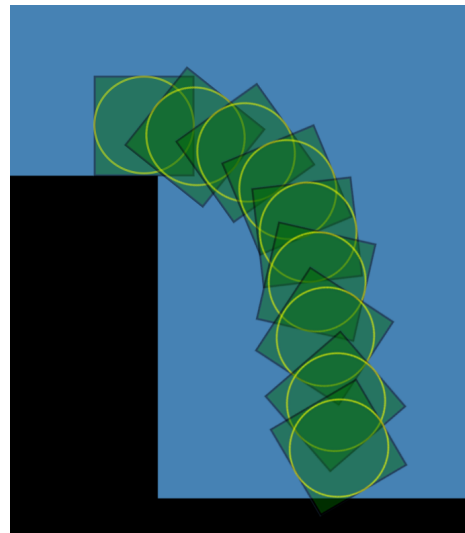


(b) Unsuccessful fall with maximum velocity

Figure 5.7: Circle FALL Moves



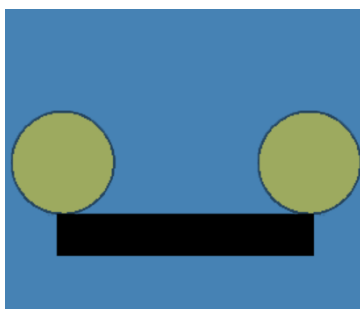
(a) Fall from GAP Platform



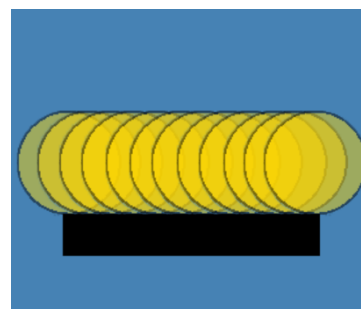
(b) Simulation of a rectangle fall as circle fall

Figure 5.8: Rectangle FALL Moves

5.2.4 JUMP Moves



(a) FALL Move



(b) JUMP Move

Figure 5.9: Possible Trajectory Initial Positions

The simulation of jumping movements is also done using the motion equations, in a very similar way to the falling ones. However, instead of considering only trajectories with initial position at the edges, other positions are tested along the platform. Not all positions are tested since they are discretized (by intervals of 16 units) as shown in Figure 5.9. The main difference is obviously the initial vertical velocity value associated with the jumping capacity that only the circle has.

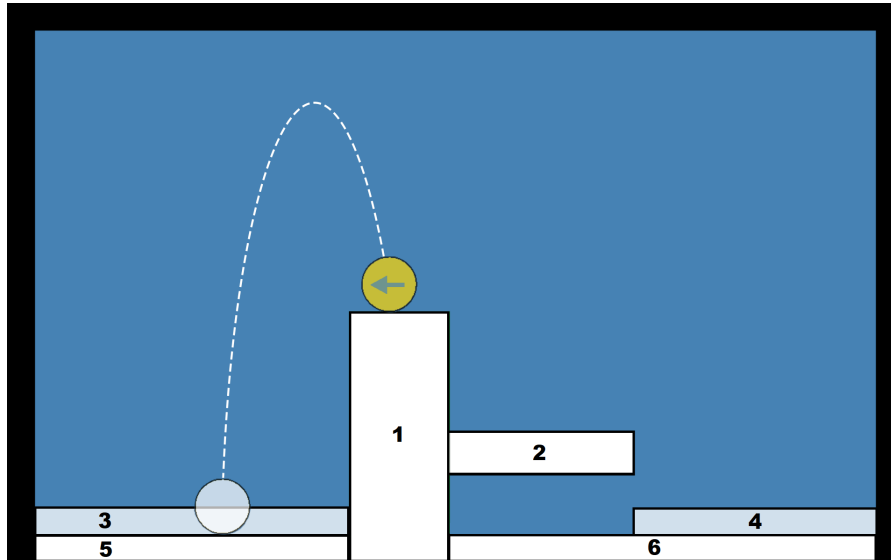


Figure 5.10: Example of a JUMP Move (level from Figure ??)

5.2.5 COOPERATIVE Moves

Given our approach to the cooperation problem in which the circle takes the lead, the COOPERATIVE moves initially exist only in this character graph. They do not require new trajectory simulations by the Moves Manager. In fact, they are identified simultaneously with FALL and JUMP moves. This is possible due to the existence of the COOPERATIVE platforms, described in Section 5.1.3, which represent the possible positions of the rectangle. Given this and the fact that the Ride Position is the basis of cooperative movements in our system, there are two ways for a FALL or JUMP move to be classified as COOPERATIVE:

- Starting COOPERATIVE Platform: if the circle is falling/jumping from a COOPERATIVE platform.
- Landing COOPERATIVE Platform: if the circle lands on a COOPERATIVE platform.

If either condition is met, the position of the rectangle is registered in the Move structure itself, as `partner_state`. For example, in the jump simulation of Figure 5.10, the circle landed on a BASIC platform. However, prior to that, Moves Manager checked a possible landing on platform 3 of type COOPERATIVE. Before it proceeded to identify the JUMP move, the COOPERATIVE move was registered with its respective rectangle position as `partner_state` (see Figure 5.11).

Knowing that the rectangle can be in a position does not mean that it is there in that moment, so for this type of movements it is necessary that the agents are synchronized. This is a special feature of

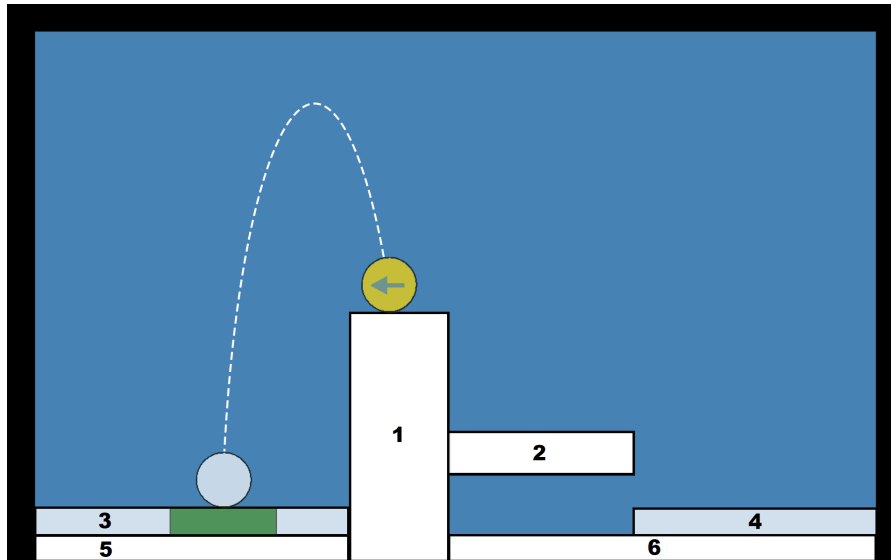


Figure 5.11: Example of a COOPERATIVE Move (level from Figure ??)

COOPERATIVE moves that sets them apart, from the circle perspective, from a FALL or a JUMP. For synchronization to exist, the rectangle must receive the COOPERATIVE Move that the circle wants to perform (through Cooperation Manager, as explained in Section 6.2).

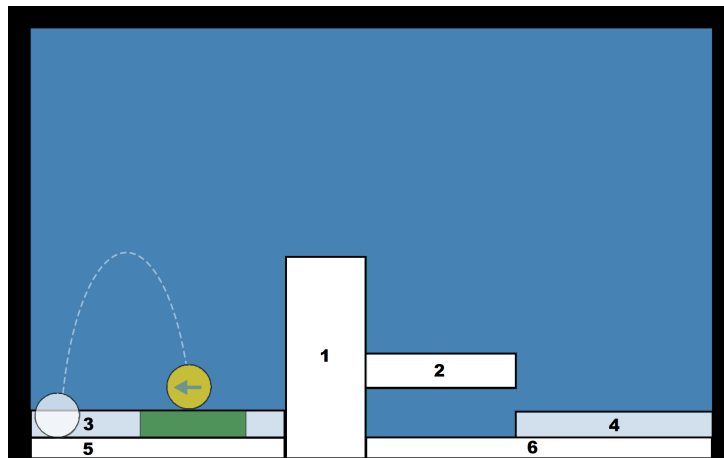
When received, this move is incorporated into the rectangle graph with just one adaptation: the state property is replaced with that of `partner_state` and vice versa to match the correct player. From a rectangle perspective, performing this move is very similar to a COLLECT one, giving that, not considering the required synchronization, it only has to assume a certain position.

Dynamic Height

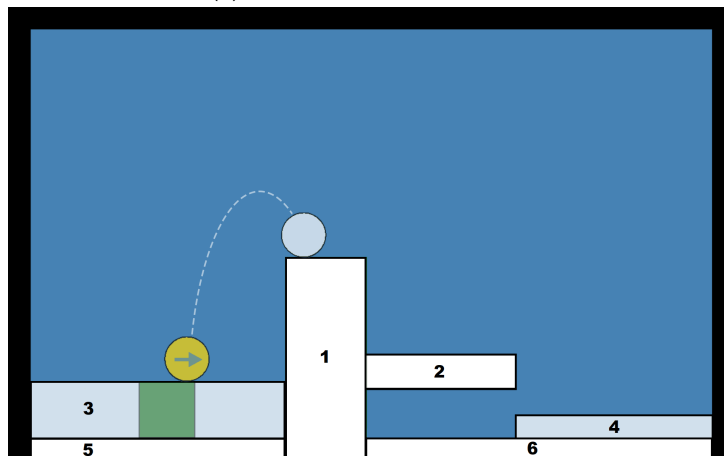
The previous Figure 5.11 is an example of a cooperative move where the landing platform is of type COOPERATIVE. However, as we mentioned, these movements also exist for COOPERATIVE starting platforms.

To identify such movements, these platforms have a special property: Dynamic Height. The reason for this is the fact that the rectangle can assume not only the horizontal form but also the square and vertical form (as we introduced in the Figure 4.2). Basically, a COOPERATIVE platform can be seen as three different BASIC platforms for each rectangle height and for which the Moves Manager simulates every possible movement.

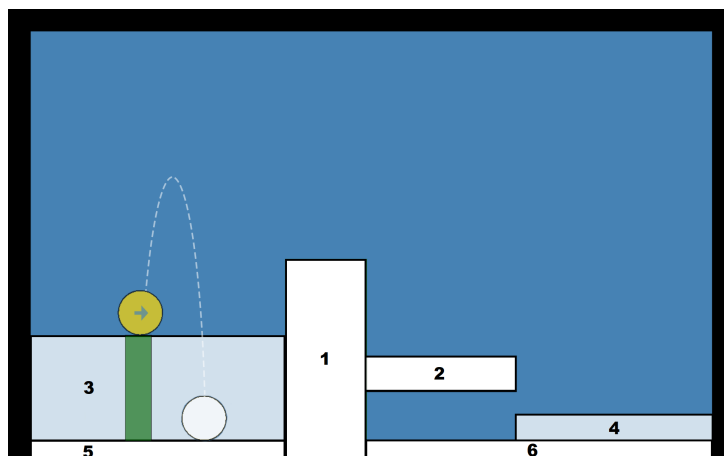
This process is shown in Figure 5.12, following the previous example. Again, for each cooperative move, the rectangle `partner_state` is recorded, in this case, with the associated platform height.



(a) Low COOPERATIVE Move



(b) Medium COOPERATIVE Move



(c) High COOPERATIVE Move

Figure 5.12: Example of COOPERATIVE Moves from COOPERATIVE platforms (Dynamic Height)

Chapter 6

Control Module

The Control Module is responsible for the agents' behaviour from the moment they start playing the level. In this chapter, we explain how the graph and training records of the agent, produced by the Mapping Module and Learning Module, respectively, lead to the level solution. In fact, these two structures are used in two distinct stages of this system component, as shown in Table 6.1.

Control Stage	Input	Strategy	Output
Planning	Sensors Information Graph	Subgoal A* Search	NEXT MOVE
Action	Sensors Information Next Move	Training Records (<i>Q</i> -Tables)	NEXT ACTION

Table 6.1: Control Module Stages

Table 6.1 is a simplification of agent operations flow because the possibility of cooperative movements brings more complexity to planning and action stages. Specifically, each agent has a Cooperation Status that indicates which phase of synchronization with the partner they are in, or whether they are playing individually. This is also a tool for Cooperation Manager to verify the need for messaging between players.

Before describing the sequence of operations for each agent, we show how to represent and obtain the necessary synchronization for cooperative movements.

6.1 Cooperation Status

Agents in our system can play both individually and as a team, depending on the type of move they select to perform. These distinct gameplay modes are represented by the Cooperation Status. In fact, in order to achieve cooperation, the agent goes through more than one state due to the synchronization process. In Table 6.2, we list the possible values of CooperationStatus.

Cooperation Status	Type of NEXT MOVE	Agent State
SINGLE	COLLECT FALL JUMP TRANSITION	Agent playing individually.
UNSYNCHRONIZED	COOPERATIVE	The agent has not reached the cooperative move state (position and horizontal velocity).
SYNCHRONIZED	COOPERATIVE	The agent has reached the cooperative move state (position and horizontal velocity).
RIDING	COOPERATIVE	Agent and its partner are moving while assuming the Ride Position.

Table 6.2: Cooperation Status Values

As we mentioned in the introduction of our cooperation approach (Section 4.4), our main focus on this aspect is the Ride Position. This is why, in addition to the CooperationStatus needed for synchronization, we have a value specifically for the riding movement. Knowing its meaning, we now explain its dynamics and the consequences of its values on agent operations.

6.2 Cooperation Manager

The CooperationStatus value of one agent often depends on the partner CooperationStatus 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 (if they are not playing individually).

In fact, the CooperationStatus of an agent can go through four types of transition, as modeled in Figure 6.1.

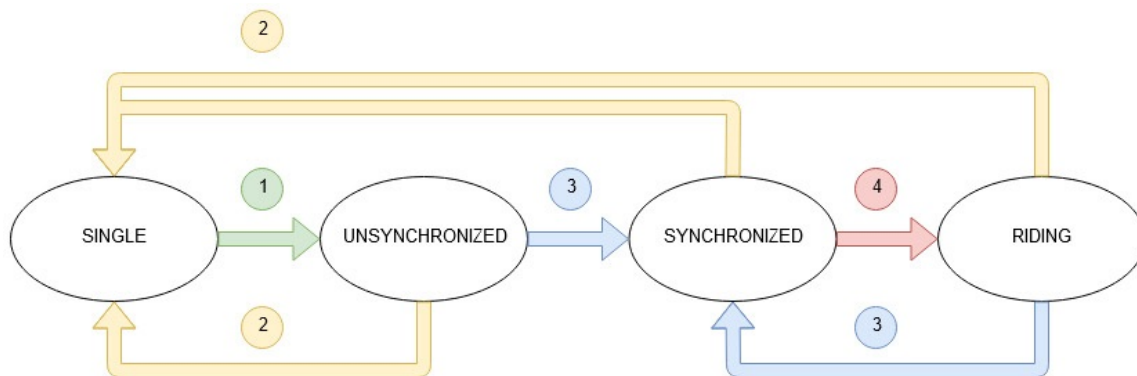


Figure 6.1: Cooperation Status Diagram

Note that, of all the transitions present in the model of Figure 6.1, only the type 3 transition does not involve communication between agents. Let us describe in more detail each type of Cooperation Status transition:

- In transition type 1, the circle, previously playing individually, selects a new NEXT MOVE of type COOPERATIVE. A message with NEXT MOVE attached, is sent to the rectangle. Unlike the circle that becomes promptly UNSYNCHRONIZED, the rectangle does not make this transition immediately (see Section 6.4).
- In transition type 2, a cooperative move finishes successfully as SYNCHRONIZED/RIDING or unsuccessfully as UNSYNCHRONIZED, 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, which means they are SYNCHRONIZED (while assuming the Ride Position). They both receive that information from their own sensors so no communication is needed.
- In transition type 4, similar to transition type 1, the new NEXT MOVE selected by the circle is of type COOPERATIVE. Likewise, a message with NEXT MOVE attached, is sent to the rectangle. The difference between these transitions is that both agents start SYNCHRONIZED and the Ride Position must be maintained until they reach the states required by the move.

It is in these transitions' descriptions that the leader-follower strategy is most noticeable. This approach led to the implementation of unidirectional communication: sending messages only occurs from the circle to the rectangle. No message goes the other way. The reason for this is the fact that it is the circle that makes the decisions on the cooperative side and takes responsibility for sharing them. On the other hand, the rectangle only commits to receiving the indications and following them.

In the next two sections, we'll explore how all of this is used in the Control Module by looking at the circle and rectangle flow of operations while they playing.

6.3 Circle Control Flow

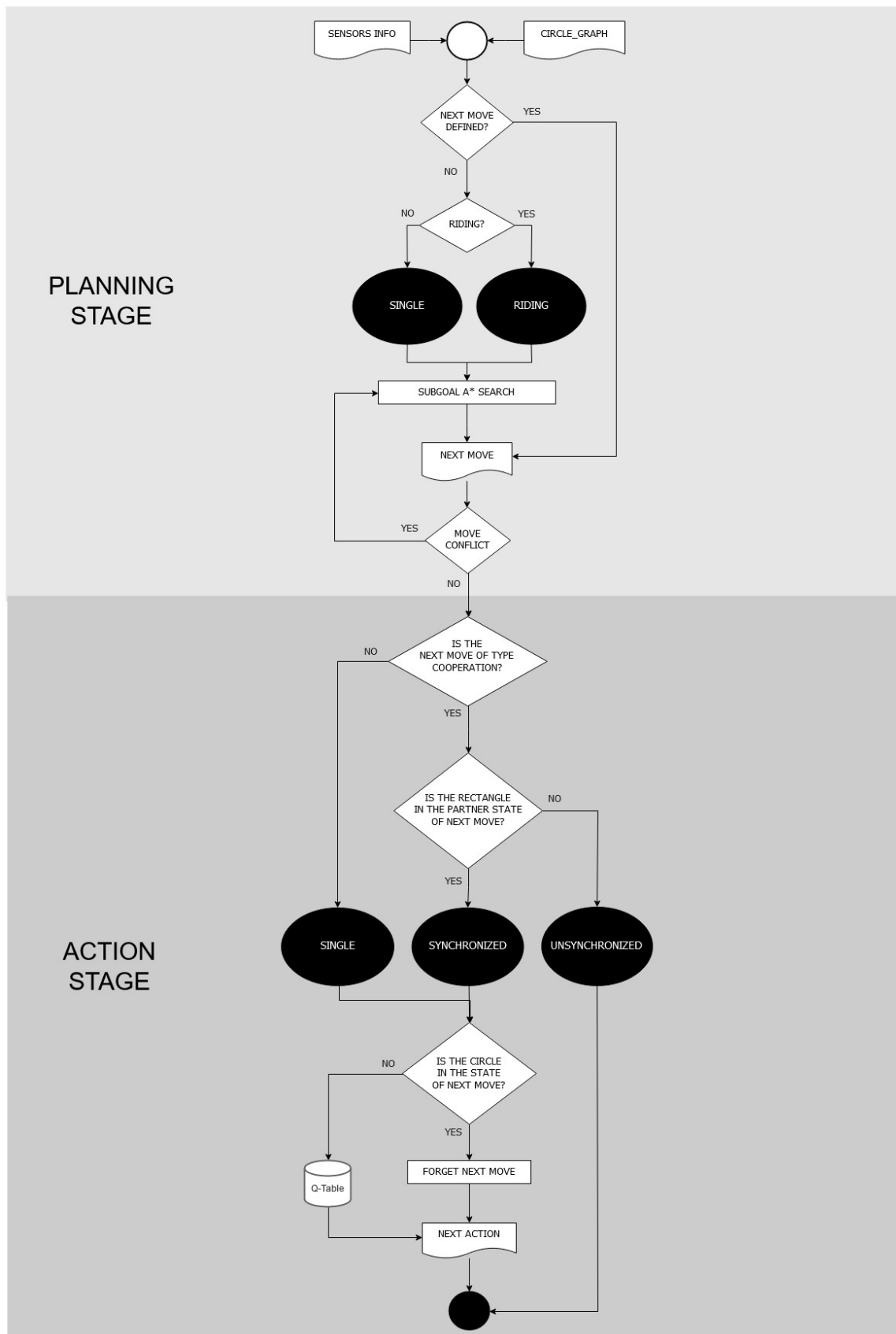


Figure 6.2: One Iteration of the Circle Control Flow

6.3.1 Circle Planning Stage

As we mentioned at the beginning of this chapter, an iteration of the control flow can be separated into a Planning Stage and Action Stage. The first does not occur every time because the NEXT MOVE may have been previously selected. In fact, this is the most common case since a move is rarely chosen and executed entirely in a single iteration. In short, if NEXT MOVE already exists, the Control Module can proceed to Action Phase. Otherwise, it is obtained through Subgoal A*.

However, there is still one more step before running the search algorithm, relative to the CooperationStatus value. The reason for this is that if the circle is in the Planning Stage to select a NEXT MOVE, it is probably because it has just performed one. It's not just about updating Cooperation Status but also the transition and communication that can result from that (from the ones modeled in Figure 6.1). Note that in this flowchart the values of Cooperation Status do not necessarily mean a transition (the previous status may be the same).

In this case, the condition to check through the agent sensor data is if the characters are in the Ride Position. The CooperationStatus is then SINGLE or RIDING according to the veracity of this condition. What can happen here, for example, is that the circle comes from a cooperative movement and is no longer in ride position. This is a type 2 transition between RIDING and SINGLE (as shown in Figure 6.1), requiring a message that releases the rectangle to also play individually.

Between Subgoal A* Search on the circle graph and the Action Phase, there is one more operation: move conflict identification. This mechanism is explained in more detail in Section 6.3.3 but basically, if there is conflict, the search is repeated (temporarily deleting the old NEXT MOVE from the graph). Otherwise, the agent can finally proceed to the Action Stage.

6.3.2 Circle Action Stage

This stage begins with another CooperationStatus check: if the type of NEXT MOVE is COOPERATIVE. If not, it means that the circle is playing individually (whether or not previously). Otherwise, the circle as system leader, must confirm 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 not, the agents remain UNSYNCHRONIZED and the circle control flow iteration ends there (with NEXT ACTION output being no action at all).

Given this, the last step is to use the training results (the output of the Learning Module). Specifically, the agent queries the Circle Q -Table if NEXT MOVE is individual or the Team Q -Table if it is a "riding" movement as the ones in Figure 4.2.

This step is skipped only if the circle is already in the state required by NEXT MOVE. If this is the case, the NEXT ACTION corresponds to the move type: JUMP if JUMP type, ROLL (in the move direction) if TRANSITION type, or no action if none of these. Additionally, NEXT MOVE is forgotten as it has already been executed. If this is not the case, NEXT ACTION is the best one learned for that state during agent training in order to reach NEXT MOVE required position and horizontal velocity.

6.3.3 Move Conflict Verification

One of the biggest difficulties in multi-agent environments is the existence of conflicts in their moves. This does not depend on whether or not they are competitive with each other. In our system, despite cooperative, conflicts can be very common and varied. Therefore, it was necessary to implement a mechanism for its identification and solution.

Due to time constraints and a large scope of possibilities, our mechanism does not cover all conflicts. In fact, it is actually very basic, far from what would be a final product. It works more as a corrector of some flaws in the Mapping Module. Specifically, there is a more complex type of cooperative move in which the circle jumps vertically (without horizontal velocity) while the rectangle positions itself in the jump spot, allowing for the circle to land and, consequently, assuming both the Ride Position. These movements are not identified by the Mapping Module because the position of the circle and rectangle required to perform them coincides (a kind of delay would have to be implemented).

One consequence is that the circle and rectangle are not able to change their relative positions. This means that if the circle is to the left of the rectangle on the same platform, this condition will not change. More importantly, move conflicts arise if the circle selects a movement in which it requires the rectangle to be positioned in the opposite direction. This situation is exemplified in Figure 6.3a. Our move conflict verification mechanism is able to identify these occurrences and solve them. It achieves this by removing these moves from the graph and only accepting similar moves that lead to the same platform and do not cause the same problem. An example of non-conflicting alternative is shown in Figure 6.3b.

In Chapter 8, we address the potential and aspects for improvement of this mechanism.

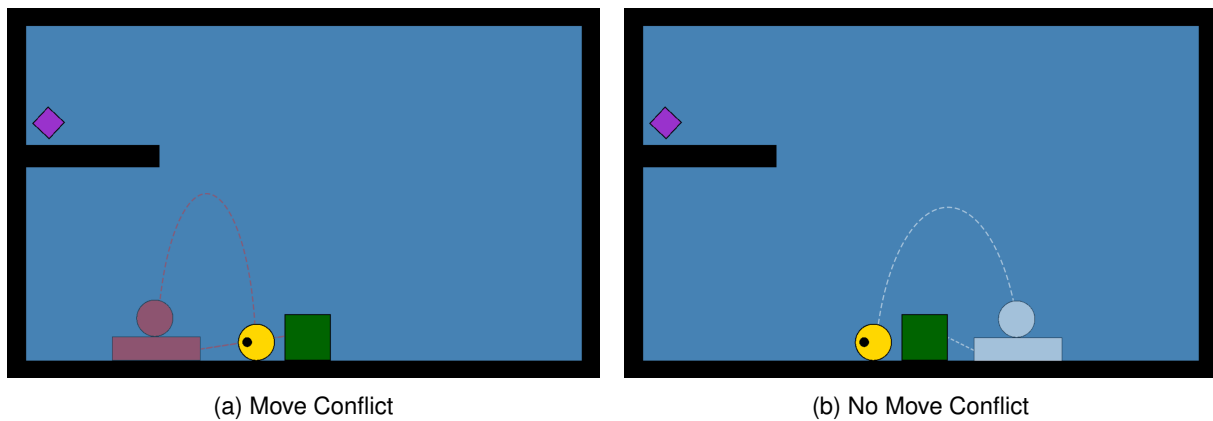


Figure 6.3: Move Conflict Verification

6.4 Rectangle Control Flow

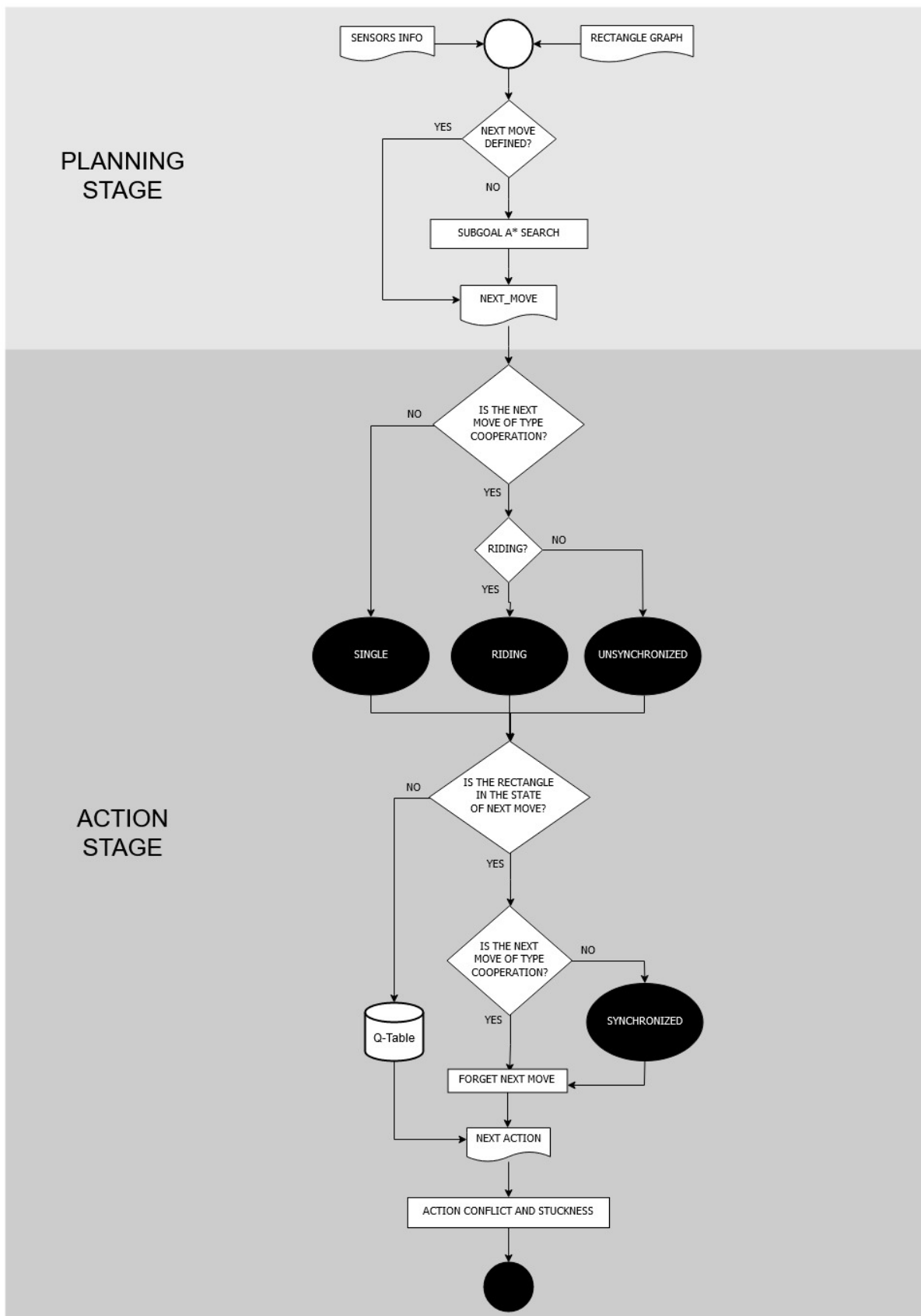


Figure 6.4: One Iteration of the Rectangle Control Flow

6.4.1 Rectangle Planning Stage

The rectangle Planning Stage is simpler than the circle one. The reason for this is the fact that the rectangle is the follower in this system leader-follower strategy. This way, the agent does not need to perform initial checks on its Cooperation Status. Nevertheless, similar to the circle, its Planning Stage includes a Subgoal A* Search on its graph if its NEXT MOVE was not previously obtained.

6.4.2 Rectangle Action Stage

Knowing the NEXT MOVE, the rectangle starts the Action Stage by updating its Cooperation Status. Having verified the move chosen as cooperative, this value is RIDING if the agents are in Ride Position and UNSYNCHRONIZED otherwise. Being an individual move, Cooperation Status becomes SINGLE.

The rest of the algorithm shares some similarities with the circle one. For example, if the rectangle is not yet in the necessary state to execute NEXT MOVE, it uses the rectangle or team Q -Table depending on the move type. If, on the other hand, it has already reached it, it updates itself as SYNCHRONIZED (only in case of cooperative NEXT MOVE). Not only that but it also erases the performed NEXT MOVE (whether cooperative or not).

The logic behind NEXT ACTION is the same as that of the circle, whether it comes from the consulted Q -Table or from the final execution of NEXT MOVE.

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.

6.4.3 Action Conflict Verification

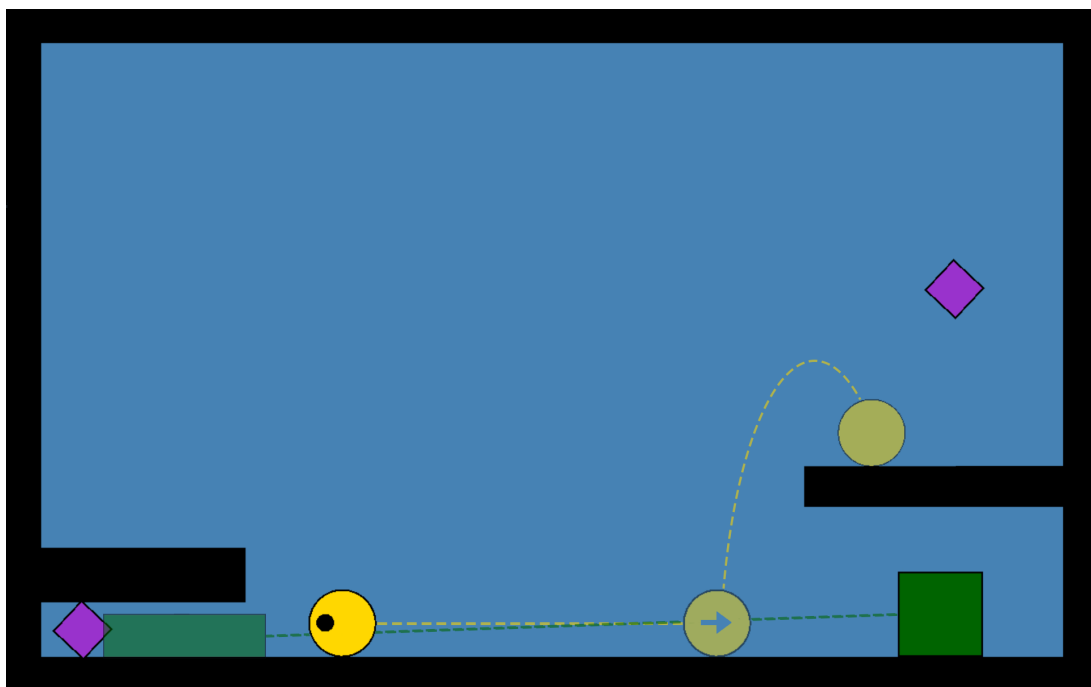


Figure 6.5: Example of a solvable Action Conflict

It's not just the circle that identifies potential move conflicts. In fact, the rectangle does as well but with a different approach due to the leader-follower strategy. The circle resolves conflicts still in Planning Phase, leveraging its leader status to change the NEXT MOVE.

The rectangle, as a follower, does not have this privilege but can make a correction on the Action Stage to prevent conflict. Specifically, it may not act as long as there is one. This means that the rectangle's NEXT ACTION is equal to NO_ACTION while it waits for the conflict to end. The situation is very similar to that of Section 6.3.3 and also for the same reasons, this mechanism is far from covering all action conflicts.

In Figure 6.5 we show an example of a solvable action conflict. If both agents execute the movements they want from their current positions, a conflict will occur. However, by identifying it, the rectangle will wait. During this interval, the circle performs its NEXT MOVE, freeing the platform for the rectangle's NEXT MOVE (shown in Figure 6.6).

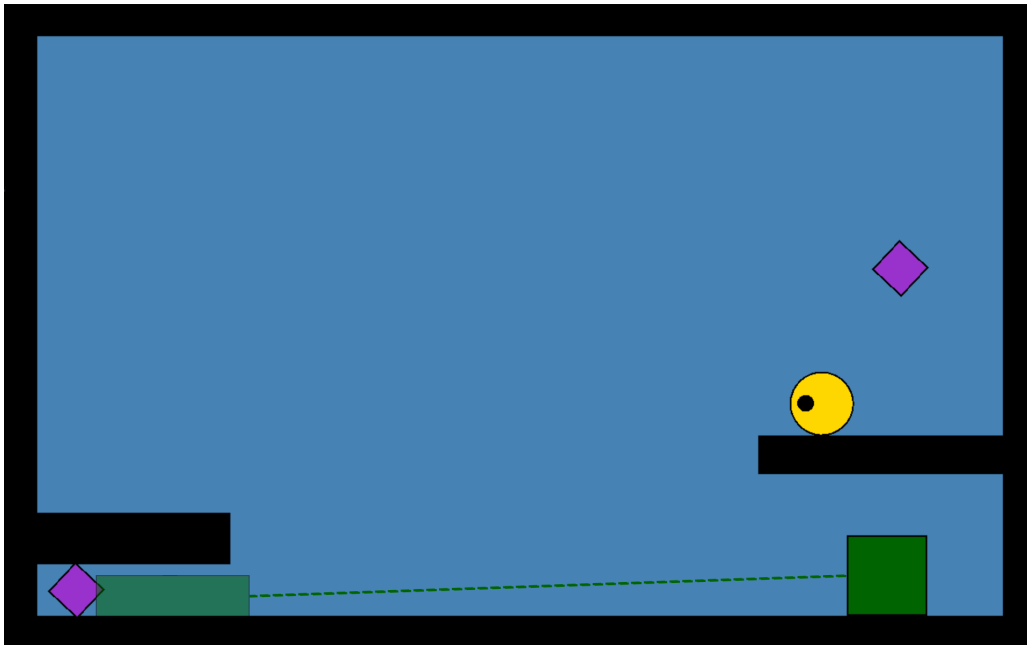


Figure 6.6: Solved Action Conflict

6.4.4 Stuckness Verification

The rectangle's morphology makes it especially prone to situations where it gets stuck. This happens when the NEXT ACTION selected by the Control Module does not change the rectangle state, leading to an infinite loop as NEXT ACTION will always be the same and have no state effect. In Figure 6.7, we have an example of this kind of situations where the rectangle moves left unsuccessfully because it is stuck in a gap between platforms.

Identifying these types of situations involves studying the character's recent past. The number of previous actions and states stored for this purpose is given by *AGENT_MEMORY_SIZE*. The general idea is to check if all these states and actions have remained the same. If so, the rectangle is considered stuck.

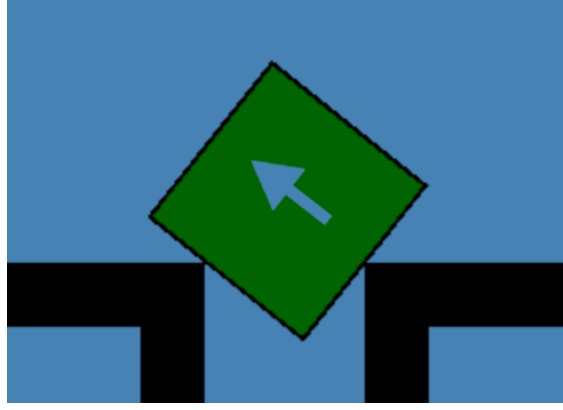


Figure 6.7: Stucked Rectangle

Our solution to stuckness is to choose random actions. Only one can have no effect on the agent's state while too much can sabotage its strategy. Therefore, once an action is drawn, it is executed a specific number of times: *STUCKNESS_FACTOR*. Both this value and *AGENT_MEMORY_SIZE* were obtained experimentally and are used in stuckness verification as shown in Algorithm 3.

Algorithm 3 Stuckness Verification

```

1: STUCKNESS_FACTOR  $\leftarrow$  10
2: AGENT_MEMORY_SIZE  $\leftarrow$  5
3: function STUCKNESS(stuckness, s, a, past_states, past_actions)
4:   if stuckness > 0 then
5:     stuckness  $\leftarrow$  stuckness - 1
6:   else
7:     stuck  $\leftarrow$  true
8:     if past_states.SIZE = AGENT_MEMORY_SIZE then
9:       for i from 1 to AGENT_MEMORY_SIZE do
10:        if stuck = false then
11:          break
12:        if a = NO_ACTION or
13:           s = NEXT_MOVE.STATE or
14:           past_states(i)  $\neq$  s or
15:           past_actions(i)  $\neq$  a then
16:             stuck  $\leftarrow$  false
17:       if stuck = true then
18:         stuckness  $\leftarrow$  STUCKNESS_FACTOR
19:         a  $\leftarrow$  RandomAction()
20:       enqueue s into past_states
21:       enqueue a into past_actions
22:     return a

```

Chapter 7

Learning Module

The Learning Module is responsible for training the agents and storing the results, making them available later to the Control Module. There are three training sessions in our system: circle training, rectangle training and Ride Position training. Although different, they all use the same technique: Q -Learning. In fact, we had to use the Team Q -Learning variant for Ride Position training, only differing in the joint action aspect.

As we introduced in Section 2.3.1, the basis of Q -Learning is the Q -table that measures the quality of each action for all possible states. These action state pairs are initialized to zero and then updated during the training, which is a sequence of episodes. In each episode, the agent, in a state s , selects an action a and updates the table entry $Q(s, a)$ with Equation 7.1.

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a)) \quad (7.1)$$

About Equation 7.1:

- α is the learning rate e γ is the discount rate, assuming respectively the values of 0.3 and 0.999;
- r is the action reward, worth +200 if it reaches the target and -1 otherwise;
- s' and a' correspond respectively to the next state (resulting state) and the action with the best Q value in that state.

The agent chooses an action in each episode following the ε -greedy approach (see Section 2.3) with $\varepsilon = 0.6$. This means that a random action is selected with 60% probability, otherwise it selects the greedy action (the one with best Q -value in the current state) with 40% probability. The value of ε as well as the learning rate, discount factor and rewards are those successfully used by the KIT agent.

Thus, the differences in the agents' individual and team learning lies in the definition of states, actions and target. We use the next sections for that.

7.1 Circle Learning Problem

The circle problem has already been defined by agent KIT. Not only that, but it has already been trained and therefore solved. Nevertheless, in order to introduce and compare the remaining problems, we show its definition in Figure 7.1.

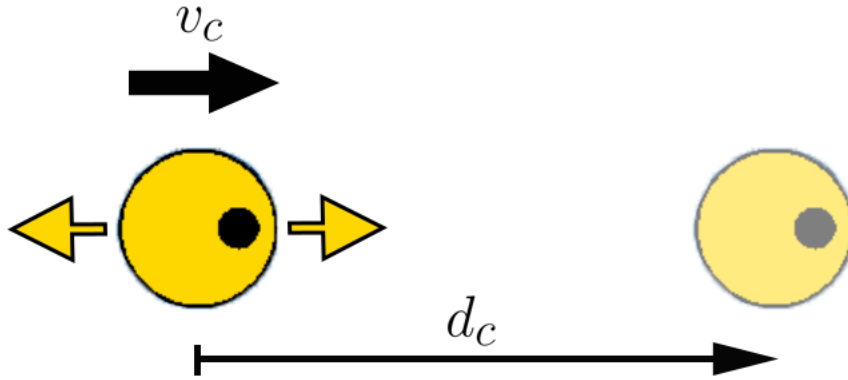


Figure 7.1: Circle Learning Problem

The target consists of one position and one horizontal velocity value. Knowing this, we only considered ROLL_LEFT and ROLL_RIGHT as circle possible actions. As for the state, it is represented by two values: the agent horizontal velocity v_c and the relative distance to the target d_c .

7.2 Rectangle Learning Problem

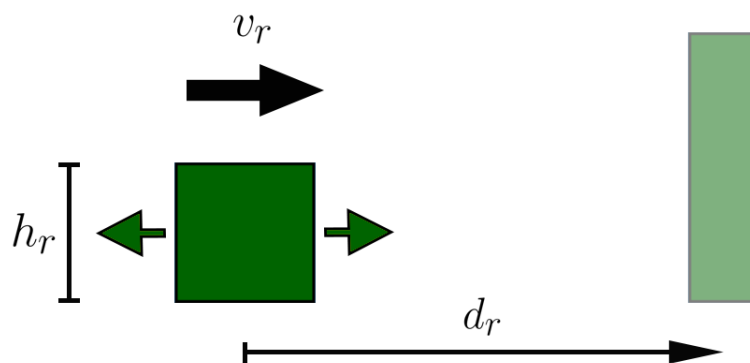


Figure 7.2: Rectangle Learning

The rectangle case does not differ much from the circle one. In fact, it only has the particularity of considering the height h_r in the agent state definition. For the rest, instead of rolling sideways, the possible actions include MOVE_LEFT and MOVE_RIGHT. The state horizontal velocity and relative distance to target are represented by v_r and d_r , respectively (see Figure 7.2).

7.3 Ride Position Learning Problem

The Ride Position case is more complex but even so, it shares some similarities with the previous individual problems.

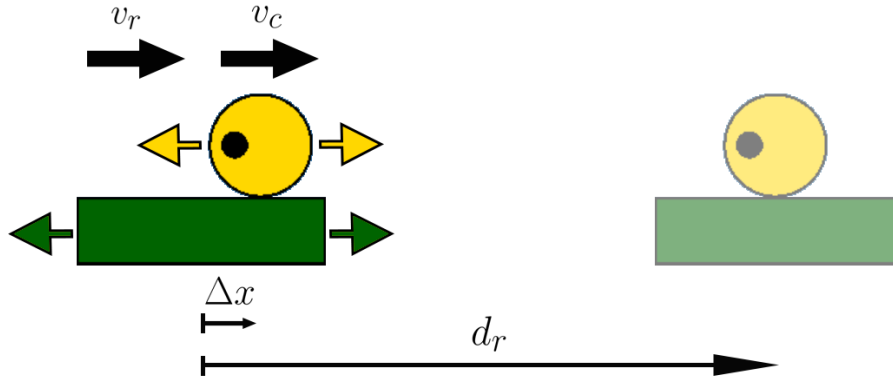


Figure 7.3: Ride Position Learning

Both agents keep their possible actions: the circle can ROLL_LEFT or ROLL_RIGHT while the rectangle can MOVE_LEFT or MOVE_RIGHT. As a system, the result is all possible combinations of actions, called 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 Δx . 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.

7.4 Values Limitation and Discretization

Considering all possible values for the elements enumerated in the previous sections, training would be nearly impossible. A strategy is then needed to reduce the number of training for each agent and system as well as the size of the resulting Q -Tables. The solution is to discretize and limit these values, cutting the state and target space covered by the Learning Module.

In fact, we started doing this right from the definition of rectangle actions. When we approached the rectangle learning problem in Section 7.2, we refer to how the state includes its height. This idea is contradicted by the fact that MORPH_DOWN and MORPH_UP are not possible actions. What happens is that the rectangle performs these two actions, as needed, before the movement begins. Nevertheless, as we will see later, the agent is trained to the three heights but the morph actions do not count to the problem. Although a simple measure, being only two actions and not four, it means a Q -Table four times smaller.

All information about values limitation and discretization for each learning problem is summarized in Table 7.1 followed by our comments.




	Action	State		Target
		Limits	Discretization	
	ROLL_LEFT ROLL_RIGHT	$ d_c \leq 200$ $ v_c \leq 200$	4 10	$d_c = 0$ $v_c = \{0, 20, 40, \dots, 160, 180, 200\}$
	MOVE_LEFT MOVE_RIGHT	$ d_r \leq 300$ $ v_r \leq 200$	4 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$ $ v_c \leq 90$ $ v_r \leq 90$ $ \Delta x \leq 60$	4 10 10 4	$d_r = 0$ $v_c = 0$ $v_r = 0$ $\Delta x = 0$

Table 7.1: Learning Problems Overview

- The circle, as a successful implementation, is very useful when comparing values between the three learning problems. 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 were discretized in intervals of 4 while the velocity ones were 10. The same approach was used for the other cases.
- For the rectangle, we only changed some value limitations. The distance to the target d_r had to be increased to 300 because the character has less acceleration. In the circle, the 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. This velocity constraint is a measure that affects early on the identification of rectangle moves in the Mapping Module. Finally, the considered heights correspond with the rectangle approach explained in Section 4.3.
- 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 Section 4.4 (Cooperation Approach), we only considered the movements in which both agents end with zero speed and still in Ride Position.

7.4.1 Learning Symmetry

There is a reason why target speeds are only positive values: learning symmetry. The idea is the same as that implemented in the KIT agent. Specifically, the positive direction is defined as the direction of the target velocity. This means that the signal of the agent's horizontal velocity depends on whether it shares the same direction as the target velocity.

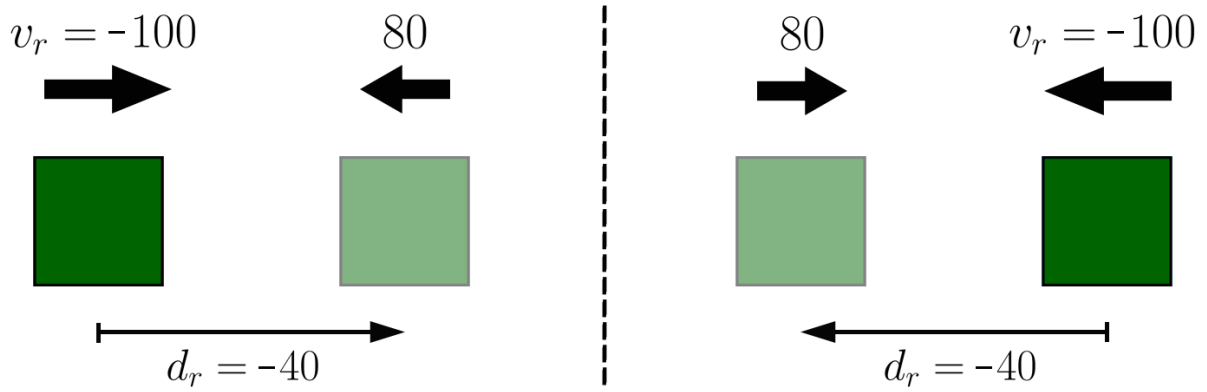


Figure 7.4: Learning Symmetry: example of two equal rectangle learning states

For example, in both states from Figure 7.4, the velocity v_r and relative distance d_r are negative values because both have opposite direction relative to the target velocity. In fact, although they are physically different positions, these rectangle learning states are exactly the same since they have equal values. What we just exemplified for the rectangle also happens for the circle and Ride Position.

This symmetry property is another factor that reduces the state space of our problems and, consequently, the necessary training for our agents/system.

7.5 Training

After specifying learning problems, the choice of training level was quite straightforward: an obstacle-free level (as used by the KIT agent). These levels are shown in Figure 7.5.

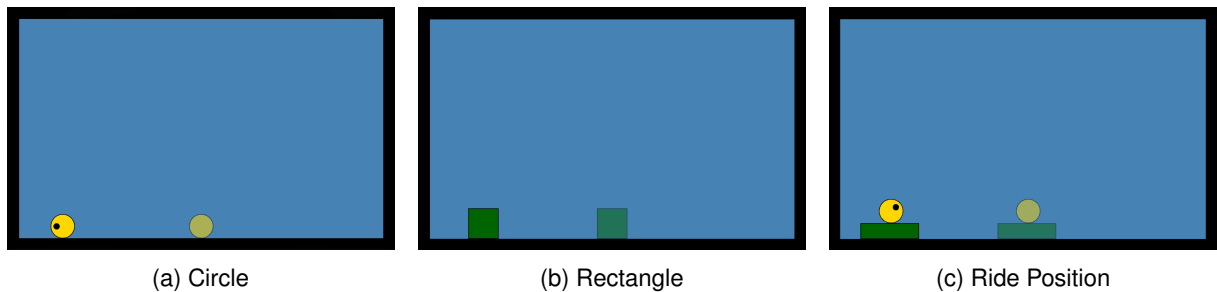


Figure 7.5: Training Levels

Figure 7.5 also shows an example of agent target. It is always positioned at the center of the level but varies in the horizontal velocity (for the circle and rectangle) and/or height (for the rectangle only).

We still need to address the result of the Learning Module: the size of Q-Tables produced as well as the number of training runs. In Table 7.2 we present the final numbers to show and compare the consequences of the decisions made when defining each learning problem.




	Problem				Training Runs		
	Number of Actions	Number of States	Number of Targets	Total (Q-Table Entries)	For Target	Total	Ratio
	2	4.000	20	160.000	5.000	100.000	1
	2	6.000	15	180.000	7.500	112.500	1,125
	4	486.000	1	1.944.000	1.215.000	1.215.000	12,150

Table 7.2: Q-Table Size and Training Runs

The first number to stand out in the case of the rectangle is the number of states (6000) which represents a 50% increase over the circle. Our solution to minimize Q -Table training impact and number of entries was to consider fewer targets (we only considered 5 speeds and 3 heights for the rectangle). Thus, inspired by the number of training runs required for the circle, we obtained only a 12.5% increase in the need for training.

The case of Ride Position is much more complicated. Just increasing the number of actions (technically joint actions) to four means double everything else. However, it is in the number of states that is the most difficult. Despite our efforts to limit these values, the number of states is more than 120 times larger than the circle. All these factors led us to decide, as we mentioned, to consider only one target: the motionless Ride Position. This is what makes us "only" need a number of training runs approximately 12 times the circle.

7.6 Implementation

During the development of this multi-agent system, we focused on achieving a solid and organized mapping, planning and control phase. After that, the idea was to expand the agents' training and the use of Reinforcement Learning progressively. However, we ended up sacrificing this last step as we explain in this section.

One of the first things we did was to build the rectangle agent such as the circle KIT, ignoring for example its ability to morph. Initially, we used the Q -Table obtained by training the circle to get the best actions also for the rectangle (just translating the actions of ROLL LEFT and ROLL RIGHT respectively to MOVE LEFT and MOVE RIGHT). However, the results proved so viable that our priority became all the remaining agent phases other than agent training.

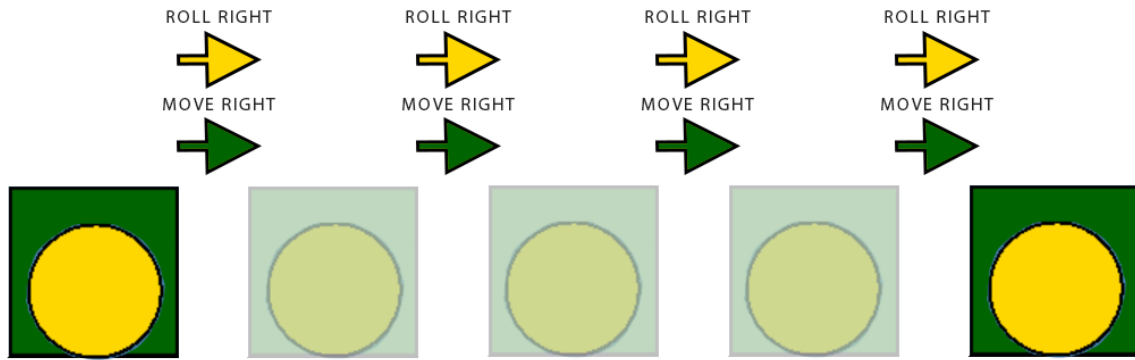


Figure 7.6: Rectangle Agent using the Circle Q-Table

In the meantime we started implementing the system by adapting each agent's Mapping Module and Control Module to consider cooperative moves and to communicate between them in order to achieve synchronization. Once again, we realized that we could focus on this task without any training. The alternative was for the agents to ensure individually the Ride Position with each one's Q -Table (in this case, only the one from circle) while maintaining a deviation between their positions below a certain value (which we used 70 as defined in the Ride Position training in Section 7.4).

In the final stretch of our project, given the time constraints and the size of the learning problems discussed in Section 7.5, we were unable to train agents enough and obtain Q -Tables for the rectangle and Ride Position with acceptable performance (at least in relation to the alternatives mentioned above). Nevertheless, this allowed us to implement a solid system based on all other phases, in which we were successful enough to the point of considering agent learning as an optimization for our system rather than its primary tool.

Chapter 8

Results And Discussion

In this chapter, we show and analyze the results of the 2019 Geometry Friends competition as well as the evolution of our post-competition agents. In the end, we identify the limitations of the final version of our system.

8.1 2019 Competition Analysis

By mid-August 2019, the deadline for agent submissions in the Geometry Friends competition was over and the results were announced. Our solution won the Rectangle Track and Cooperation Track, beating all the other agents. Among these was the RTT proposal, which was the best in both so far (baseline). In this section we show how our rectangle and multi-agent system outperformed competitors.

8.1.1 2019 Rectangle Track

The Rectangle Track results obtained by RTT (Table 8.1) and by us (Table 8.2) were influenced by a game bug in which the rectangle was blocked if it fell on one of its faces. This mainly affects the levels that require a high and partially unimpeded fall (few obstacles around). In this case, levels 2, 3, 6, 7 and 8 (marked in Table 8.1 and Table 8.2). The score is affected if the fall performed by an agent includes landing on that problematic rectangle face.

For example, our agent is severely affected at levels 2 and 8, and slightly at 7. Levels that our agent successfully completed in a bug-free version of the game. So, considering that the other competitors may also have been affected, we ignore these five levels for comparison purposes.

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).

At bug levels 3 and 6 where our agent was not affected, there was also an excellent performance, completing those levels in a short time.

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 8.1: RTT Results in 2019 Rectangle Track (Runner-Up / Baseline)

Level	Collectibles	Time(s)	Score
1	2.0 (2)	12.685	1494.29
2	0.9 (3)	90.349	517.09
3	3.0 (3)	22.195	1715.04
4	2.8 (3)	31.513	1577.39
5	2.0 (2)	12.572	1495.23
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	1779.89
10	2.0 (2)	9.423	1521.47
bug level	Total (bug-free)		13945.72(7868.27)

Table 8.2: Our Results in 2019 Rectangle Track (Winner)

Note that there was no stuckness verification in this agent version. That is why level 4 does not record all diamonds collected. These results confirm that it is possible to apply to the rectangle the same approach followed by the KIT agent for the circle. Not only that but even with an incomplete implementation, it is the best recorded rectangle agent.

8.1.2 2019 Cooperation Track

In the Cooperative Track, once again, we do not compare the two systems at levels where the bug is possible. They are also marked as gray in Table 8.3 and Table 8.4 for RTT and our proposal, respectively.

Like the Rectangle Track, our proposal is superior to RTT in bug-free levels (except level 10 where both only get one collectible). This time, much more clearly, getting a total score with an increase of approximately 470% compared to the baseline.

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 8.3: RTT Results in 2019 Cooperation Track (Runner-Up / Baseline)

Level	Collectibles	Time(s)	Score
1	2.8 (3)	32.55	1568.72
2	1 (1)	15.70	1169.19
3	2.5 (3)	65.77	1201.89
4	2.5 (3)	58.96	1258.68
5	2.8 (3)	32.38	1570.14
6	3 (3)	21.24	1723.02
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 (7221.5)

Table 8.4: Our Results in 2019 Cooperation Track (Winner)

Regarding the bug levels, our agent was not affected in any of them. That's why at levels 5, 8 and 9 we see the same excellent performance. The source of the level 7 problem is a minor flaw in the rectangle platform identification process, corrected post-competition.

While solving level 10, another flaw was identified, this time in identifying cooperative moves. Since we were still halfway through the implementation of this system component, it was naturally rectified. Another thing to note in the table is that the collection rate is rarely 100% (only at levels 2 and 6). This phenomenon occurs because cooperative movements, such as circle jumping and landing on top of the rectangle, are more complex to execute. What happens is that there is often a different bounce that leads to trajectory failures. The main problem is later in NEXT MOVE replanning, which in this version of the agent did not have a move conflict verification (later implemented).

The conclusion is that despite some flaws in an incomplete system, our solution has brought a significant improvement in the performance of cooperative agents in competition.

8.2 Post-Competition Analysis

After the competition, we implemented a number of minor enhancements and mechanisms mostly in the Control Module. Within these, we highlight:

- Stuckness Verification (Rectangle Agent)
- Action Conflict Veriification (Rectangle Agent)
- Move Conflict Verification (Circle Agent)

As we have been referring to in the analysis of previous results, these processes have corrected all problems associated with competition levels. For the most part, the goal was to allow the correct re-planning of moves and to prevent unnecessary collisions between characters. However, the truth is that these mechanisms, explained earlier in their sections, are rudimentary algorithms and temporary solutions to a larger problem than those present at competition levels. Before we address these limitations of our system, we show what was achieved after the competition.

In Table 8.5 and Table 8.6, we show post-competition results already in a bug-free version of Geometry Friends. The score was obtained through the game's native batch simulator, based on Equation 2.1 in which $V_{Completed} = 1000$ and $V_{Collect} = 300$ (as used in the competition).

Having a more complete implementation, conflict prevention mechanisms and bug fixes led to a significant improvement in post-competition performances. In fact, all levels have been completed with a similar or higher score. In the Rectangle Track, the final score was 16221 against the previous 13945.72 (16% improvement). In the Cooperation Track, the final score was 15799 against the previous 11886.56 (32% improvement).

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

Table 8.5: Post-Competition Results for 2019 Rectangle Track (Bug-Free Game Version)

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

Table 8.6: Post-Competition Results for 2019 Cooperation Track (Bug-Free Game Version)

8.3 Limitations

8.3.1 Rectangle Agent

The fact that the rectangle has not undergone training adapted to its characteristics leads to a lack of fluidity in its movements. However, this is not a limitation as it does not prevent any kind of movement (on the other hand, it is not the optimal solution).

Note also that the problematic simulation of ceiling collisions associated with circle agent KIT, does not occur in our agent. This is because it only affects jump trajectories (ability that the rectangle does not possess).

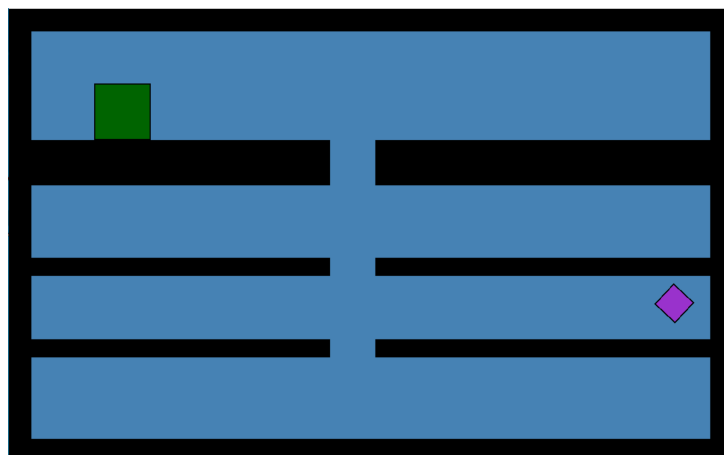


Figure 8.1: Unsolved Rectangle Level: Morphing While Falling

A real limitation of our rectangle agent is a particular type of falls. Specifically, when morphing while falling allows the character to reach certain platforms that, otherwise, it could not (exemplified in Figure 8.1). The reason for this is that it requires that each trajectory point for each rectangle height be checked for the possibility of changing platforms. Not only that, Moves Manager has to calculate at what point

during the fall, the rectangle can and should perform the necessary morphing, taking into account the surrounding obstacles. These operations were not implemented in our rectangle agent, although there are structures like the abstract type of GAP platforms that make it easy to identify such movements.

8.3.2 Multi-Agent System

Like the rectangle, the Ride Position also shows little fluidity in its movements due to lack of proper training. Similarly, that is not a limitation of our system, unlike those that we will describe. Interestingly, it is very much related to the existence of conflicts in the cooperative movements for which we have implemented the mechanisms mentioned in the post-competition section. Nevertheless, solving conflicts in Control phase is difficult and inefficient because there are many possibilities to be covered. The strategies described were a good solution for the type of conflicts present in the competition levels but they are far from eliminating all types of them.

One thing we realized too late was that the source of the conflicts was in the Mapping Module. More precisely, it failed to identify some key cooperative movements to avoid collisions between agents.

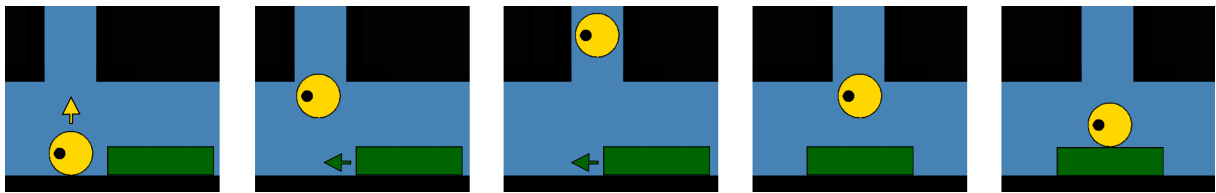


Figure 8.2: Uncovered Cooperative Move: Breaking Circle Fall

The first example, shown in Figure 8.2, is breaking circle fall. We saw in Section 5.2.5 (COOPERATIVE Moves) how we considered all the moves in which the circle jumps and lands on top of the rectangle. However, there is a situation/trajectory that is not covered: when the circle jumps with zero horizontal velocity while the rectangle moves to the circle's jump position so that it falls on top. This is a more complex cooperative movement but is essential for spaces where agents have less freedom.

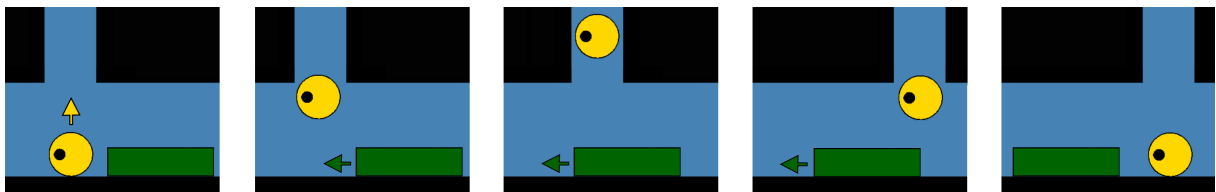


Figure 8.3: Uncovered Cooperative Move: Changing Positions

The second example, shown in Figure 8.3, although it does not seem so, is very particular. It is a cooperative movement because it requires synchronization between agents, but it does not involve any COOPERATIVE platform. This means that the circle does not start or land on top of the rectangle (the Ride Position is not assumed by the system anywhere in this move). In fact, this goes against the very definition we have made of a Move because it does not involve platform change or a diamond collected for the circle character. Nevertheless, this type of movement should exist in our system because the intersection of agents' trajectories as in Figure 8.3 is extremely common at multiplayer levels.

The two types of movement mentioned and exemplified would allow our system to solve more cooperative levels (as the ones in Figure 8.4 and Figure 8.5). The optimal solution would be to prevent conflicts from appearing during level resolution. This would imply that the Mapping Module only identified non-conflicting moves and that the entire planning phase would also be pre-level. The truth is that, as we have seen in previous proposals, when it comes to Geometry Friends, it is better not to underestimate the problem and to effectively divide it into sub-problems.

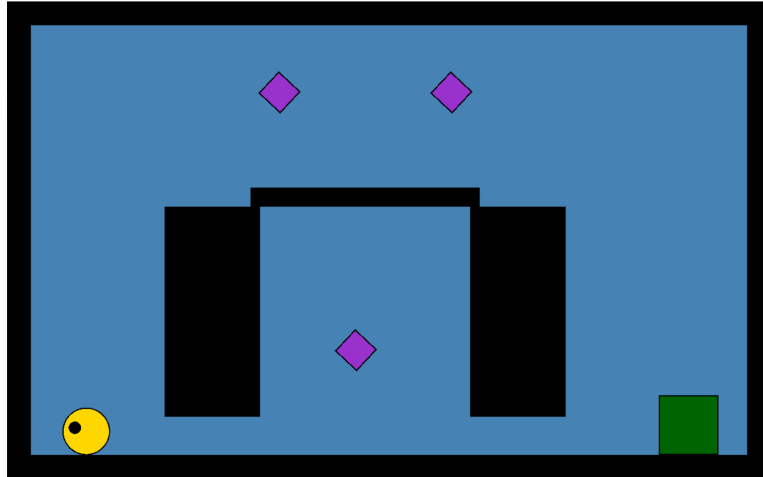


Figure 8.4: Unsolved Cooperation Level Example 1

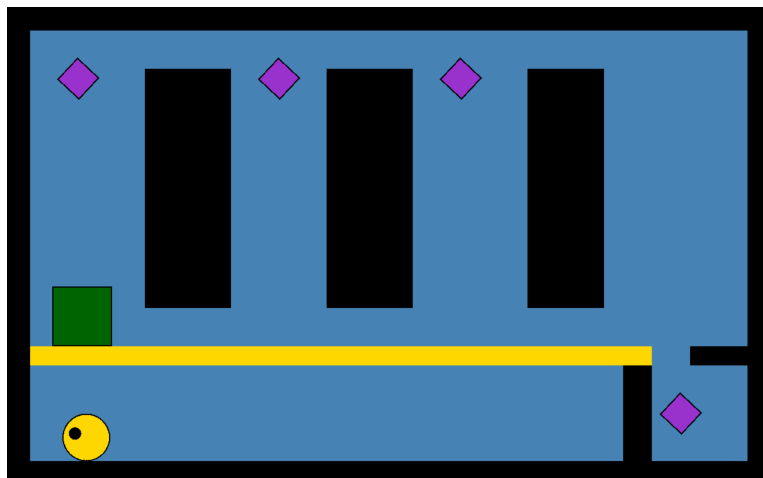


Figure 8.5: Unsolved Cooperation Level Example 2

Chapter 9

Conclusions

We are now in a position to verify if the objectives we proposed in Chapter 1 have been met. As we saw in the previous chapter, our rectangle agent and multi-agent system won the Rectangle Track and Cooperation Track of the Geometry Friends competition respectively. More important than this recognition was the performance achieved both during and after competition. These were the first two goals.

The last objective was a modular system construction, structured to support cooperation between agents and future explorations of Machine Learning techniques. Our solution is divided into components dedicated to learning, mapping and controlling characters, each with its own sub-problems and strategies for solving them. They all function independently and consistently, allowing for possible change in their mechanisms and/or extending their influence.

In order to evaluate our contribution and legacy, the new Geometry Friends state-of-the-art after our work was done is presented in Table 9.1.

Subproblem	State-of-the-Art (before)	State-of-the-Art (after)
Pathfinding	Subgoal A*	Subgoal A* (used by our system)
Circle Agent	KIT Agent (using Subgoal A*)	KIT Agent (used by our system)
Rectangle Agent	RTT Rectangle Agent	Our Rectangle Agent
Cooperative Agents	RTT Circle Agent & RTT Rectangle Agent	Our Multi-Agent System

Table 9.1: Geometry Friends State-of-the-Art (before and after September 2019)

First, regarding the pathfinding and circle agent problems, Subgoal A* and the KIT agent remain state-of-the-art respectively. As we mentioned at the beginning of this dissertation, these works inspired us and were even incorporated into our system. After studying them in detail, we designed a system architecture that allowed us to further develop and expand those strategies to the rectangle and cooperation problem.

For the rest, our proposal outperformed RTT agents and won the competitions. With this, and looking

at the Geometry Friends state-of-the-art as of September 2019, all solutions are developed or incorporated by us. This means that our legacy is a multi-agent system that solves all problems.

It is true that we have not been able to complete the implementation we propose in this dissertation but it just proves that the potential of using Reinforcement Learning in this game does not stop here. Our proposal is the first to use it successfully in the cooperative problem, obtaining the best performance ever recorded in competition.

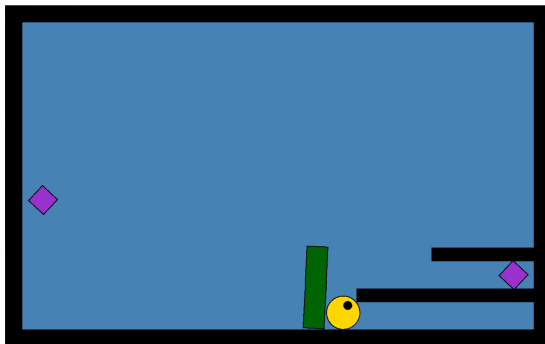
9.1 Future Work

The first step after this paper would be to correct the implementation of the Learning Module by completing what was proposed in Chapter 7 or by simplifying the definition of the problem. This way the movement of the rectangular agent and the Ride Position would be optimized respectively.

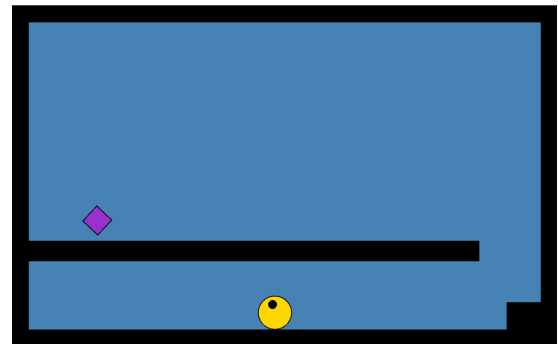
Nevertheless, significant improvements would come if movement conflicts were solved. Covering the cases, depicted in Section 8.3, in the mechanisms we have already implemented would be an important update. The alternative is to change the cooperation approach entirely because, as we mentioned, also in Section 8.3, the root of the problem of conflict lies in the Mapping Module, more precisely in identifying COOPERATIVE moves.

With the previous measures correctly implemented, we could explore new situations of cooperation such as the circle serving as the ramp to the rectangle (such as the level of Figure 9.1a). We could even go back and look to agent-specific situations, such as rectangle morphing during the fall (previously shown in Figure 8.1) or circle bouncing off wall for levels like the one in Figure 9.1b.

One of the most interesting cases regarding future work would be to achieve agent cooperation without communication between them. This would allow us to make a full assessment of our agents in collaboration with humans. The truth is that our circle is very close to accomplishing this since it does not require reception of communication (in our system it is unidirectional circle to rectangle).



(a) Circle serving as Ramp for Rectangle



(b) Circle Bouncing Off Wall

Figure 9.1: Future Work Examples

Bibliography

- [1] S. Rabin. *AI Game Programming Wisdom*. Charles River Media, Inc., Rockland, MA, USA, 2002. ISBN 1584500778.
- [2] I. Millington and J. Funge. *Artificial Intelligence for Games, Second Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2009. ISBN 0123747317, 9780123747310.
- [3] G. Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, Mar. 1995. ISSN 0001-0782. doi: 10.1145/203330.203343. URL <http://doi.acm.org/10.1145/203330.203343>.
- [4] M. Campbell, A. Hoane, and F. hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57 – 83, 2002. ISSN 0004-3702. doi: [https://doi.org/10.1016/S0004-3702\(01\)00129-1](https://doi.org/10.1016/S0004-3702(01)00129-1). URL <http://www.sciencedirect.com/science/article/pii/S0004370201001291>.
- [5] S. Sodhi. Ai for classic video games using reinforcement learning. Master's thesis, 2017.
- [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] 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. URL <http://arxiv.org/abs/1702.06230>.
- [8] 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.
- [9] R. Prada, P. Lopes, J. Catarino, J. Quiterio, and F. Melo. The geometry friends game ai competition. pages 431–438, 08 2015. doi: 10.1109/CIG.2015.7317949.
- [10] D. Fischer. Workflow of subgoal a * agent solving the rectangle track of geometry friends, 2015.
- [11] R. S. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 1998.
- [12] R. Soares, F. Leal, R. Prada, and F. S. Melo. Rapidly-exploring random tree approach for geometry friends. In *DiGRA/FDG*, 2016.

- [13] 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.
- [14] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, UK, 1989.
- [15] C. J. C. H. Watkins and P. Dayan. Q-learning. In *Machine Learning*, volume 8, chapter 3, page 279–292. Springer US, 1992.
- [16] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968. doi: 10.1109/TSSC.1968.300136.
- [17] R. Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957. ISSN 0022-2518.
- [18] L. Buşoniu, R. Babuška, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, 2008.
- [19] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, Nov. 2005. ISSN 1387-2532. doi: 10.1007/s10458-005-2631-2. URL <http://dx.doi.org/10.1007/s10458-005-2631-2>.
- [20] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 746–752, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. ISBN 0-262-51098-7. URL <http://dl.acm.org/citation.cfm?id=295240.295800>.
- [21] S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. 01 2002.
- [22] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542. Morgan Kaufmann, 2000.
- [23] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *In Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann, 1993.
- [24] M. Ghavamzadeh and S. Mahadevan. Learning to communicate and act in cooperative multiagent systems using hierarchical reinforcement learning. 2004.
- [25] M. L. Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55 – 66, 2001. ISSN 1389-0417. doi: [https://doi.org/10.1016/S1389-0417\(01\)00015-8](https://doi.org/10.1016/S1389-0417(01)00015-8). URL <http://www.sciencedirect.com/science/article/pii/S1389041701000158>.

- [26] K.-J. K. D.-M. Yoon. Cibot ball agent description. *Geometry Friends Game AI Competition*, 2014.
- [27] K.-J. K. H.-T. Kim. Cibot square agent description. *Geometry Friends Game AI Competition*, 2014.
- [28] D.-M. Y. K.-J. K. Hyun-Tae Kim. Cibot cooperation agent description. *Geometry Friends Game AI Competition*, 2014.
- [29] A. D. B. Vallade and T. Nakashima. Geometry friends competition (cig 2014) opu-scom. *Geometry Friends Game AI Competition*, 2014.
- [30] A. D. B. Vallade and T. Nakashima. Geometry friends competition (cig 2015) opu-scom. *Geometry Friends Game AI Competition*, 2015.
- [31] L.-j. W. Y.-w. Lin and T. h. Chang. Using a * and q-learning algorithms to implement the task of geometry friends single ai track. *Geometry Friends Game AI Competition*, 2014.
- [32] J. Quitério, R. Prada, and F. S. Melo. A reinforcement learning approach for the circle agent of geometry friends. Master's thesis, Instituto Superior Técnico, Lisboa, Portugal, 2015.
- [33] D. Brandão, R. Prada, and F. Melo. A deep learning approach to the geometry friends game. Master's thesis, Instituto Superior Técnico, Lisboa, Portugal, 2017.
- [34] H. Oonishi and H. lima. Improving generalization ability in a puzzle game using reinforcement learning. pages 232–239, 08 2017. doi: 10.1109/CIG.2017.8080441.

