Open University of Cyprus

Faculty: School of Pure and Applied Sciences

Postgraduate (Master's) Program of Study *Cognitive Systems*

Postgraduate (Master's) Dissertation

Opponent Selection in a Multi-Agent Gaming Environment under Resource Constraints

Sotirios Zikas

Supervisor

Dimitris Kalles

November 2021

Open University of Cyprus

Faculty: School of Pure and Applied Sciences

Postgraduate (Master's) Program of Study *Cognitive Systems*

Postgraduate (Master's) Dissertation

Opponent Selection in a Multi-Agent Gaming Environment under Resource Constraints

Sotirios Zikas

Supervisor
Dimitris Kalles

The present Postgraduate (Master's) Dissertation was submitted in partial fulfilment of the requirements for the postgraduate degree in Cognitive Systems Faculty of Pure and Applied Sciences of the Open University of Cyprus.

November 2021

Summary

The RLGame is a strategy game of two players that is continuously developed since 2001. It was created in order to conduct research at the field of Reinforcement Learning. In this dissertation we suggest a number of scenarios, where the agents (avatars) that are competing in the RLGame could be studied under the presence of resource constraints.

The goal of these scenarios is to investigate whether there are indications of behavioral adaptation when the rules of the game essentially change and therefore could lead to explicit or implicit opponent selection. A number of scenarios are theoretically described and some of them are implemented using Eclipse, a Java integrated development environment.

These implementations provided some indications that the existence of some resource constraints might affect the behavior of the synthetic agents and at the same time set the base for future research on the RLGame in order for a more dynamic approach to be implemented and the behavioral adaptation of the agents to be investigated in depth.

Keywords

Multi-agent Systems, Reinforcement Learning, Synthetic Agents, Resource Constraints.

Περίληψη

Το RLGame είναι ένα παιχνίδι στρατηγικής μεταξύ δύο παιχτώ, όπου αναπτύσσεται συνεχόμενα από το 2001. Σκοπός της δημιουργίας του ήταν η έρευνα στον τομέα της Ενισχυτικής Μάθησης. Στην παρούσα μεταπτυχιαή διατριβή, προτείνουμε έναν αριθμό σεναρίων, μέσα από τα οποία οι πράκτορες (avatars) όπου διαγωνίζονται στο RLGame μπορούν να μελετηθούν υπό την παρουσία περιορισμών πόρων.

Στόχος αυτών των σεναρίων είναι να ερευνηθεί το αν υπάρχει ένδειξη συμπεριφορικής προσαρμογής, όταν η κανόνες του παιχνιδιού ουσιαστικά αλλάζουν και κατά συνέπεια αν μπορεί να οδηγήσει σε άμεση ή έμμεση επιλογή αντιπάλου. Ένας αριθμός σεναρίων περιγράφονται θεωρητικά και μερικά από αυτά υλοποιούνται στο με χρήση του Eclipse, ολοκληρωμένου περιβάλλοντος ανάπτυξης για τη γλώσσα Java.

Αυτές οι υλοποιήσεις ανέδειξαν κάποια αποτελέσματα που δείχνουν ότι ίσως η συμπεριφορα των πρακτόρων επηρεάζεται από την ύπαρξη κάποιων περιορισμών πόρων, ενώ παράλληλα έθεσαν τη βάση για μια πιο δυναμική υλοποίηση πάνω στο RLGame όπου θα επιτρέπει μια πιο εις βάθος έρευνα πάνω στην προσαρμοστική συμπεριφορά των πρακτόρων.

Λέξεις Κλειδιά

Πολύπρακτορικά Συστήματα, Ενισχυτική Μάθηση, Συνθετικοί Πράκτορες, Περιορισμός Πόρων.

Acknowledgements

I would like to thank my supervisor, professor Dimitris Kalles for his valuable guidance and understanding.

Table of Contents

Chapter 1

Introduction

The field of Multi-Agent Systems (MAS) is considered a subfield of Distributed Artificial Intelligence (DAI), which aims to develop distributed solutions for problems. The MAS applications have experienced a rapid growth during the last decades. The main reason is their promise of making decisions in real-world (complex) problems. The general idea behind the MAS applications is that a number of intelligent autonomous agents are deciding for their actions within a complex environment having as goal to maximize the individual or the team score depending on the environment (competitive or cooperative), while at the same time the complex problem is being divided in smaller tasks which are assigned to the agents. Obviously, the target can be different depending on the approach. Even though there are many approaches and techniques to complex problem solving [Chen et al., 2008: 1-2] we are particularly interested in experimenting within a multi-agent environment, because we believe that first, such environments are the closest simulation to the real-world in terms of complexity and manipulation of a dynamic environment from the agent [Helleboogh et al., 2007: 29] and second, that they bring us closer to the cognitive systems paradigm [Langley, 2012: 4], which briefly describes the ultimate target of AI projects, to achieve cognition that is close, if not similar to the human one, therefore closer to the human behavior that we want these complex systems to reproduce.

For example, during a complex task such as driving a car, a human, who can be also an unexperienced driver, must receive information from the environment (car, road, traffic-lights etc.) and at the same time from the other drivers, while the driver is adapting her

behavior to the continuously changing environment. Therefore, in a MAS simulation, where each car is an agent and we cannot foresee all the possible scenarios that could take place, an intelligent agent must learn on its own and adapt to every change, thus the autonomy (ability to choose the actions without an intervention outside the environment) that is given to it. Knowledge could be acquired from its actions, the environment and from the agent's interactions with the other agents.

The definition of what an agent or what intelligence is, has troubled the researchers since a long time ago [Franklin & Graesser, 1996: 7] and during all these years we have seen many definitions that slightly differ from each other, making their appearance in the literature. For example, in [Dori et al., 2018: 2] the authors hinted at this problem of the literature and tried to provide a general and simpler definition. In order to avoid all the confusion regarding the definition, during this thesis we have decided to use the definition of an *agent* as it is introduced in [Russel & Norvig, 2010: 34], which in our opinion is very accurate and describes an agent as "…anything that can be viewed as perceiving its environment through sensors and acting upon that environment…" and for a *synthetic agent*, the definition that is used in [Kiourt, 2017: 2], which targets agents that act in a gaming environment, and describes a synthetic agent as "… artificial with different characteristics (different playing profiles), which interact with other agents…".

Further we are going to explain our main scenarios in which we suggest the conduction of some of the suggested experiments within a multi-agent system and more specially in a gaming environment, where synthetic agents will interact under resource constraints and how these constraints might affect or not the implicit or explicit opponent selection. Throughout this thesis the word *avatar* has been used to describe these *synthetic agents* that interact within the RLGame platform, and their definition is given in detail later in Chapter 2.

1.1 MAS environment – Characteristic Features

Above, we have briefly described the general idea behind the function of a multi-agent system. A number of agents, interacting with each other and with the environment (in terms of basing their decision on the information that is provided from each agent and the environment that the agents are located), having as goal to solve a complex task. We are going to provide a slightly more detailed description of a MAS and its agents and also briefly present the applications and the challenges that we could meet in a MAS and more specifically during a high-level implementation of our suggested scenarios.

We should mention that the environment and the agents that are acting in it, are different things, meaning that the agents are deciding according to their environment and not vice versa. That also depend on the problem's nature, that is strongly connected to the environment's dimensions. For example, the environment could be a network and its agents different software programs or in our case the environment is a gaming board, and its agents are avatars that play the game and gain experience after every game. The environment is an especially important factor in a MAS, for the reason that it can affect the decisions of the agents from the information that it provides. As we have mentioned earlier the dimensions of the environment determine to a large extent how the agent is designed, and which techniques is it going to apply when acting within the environment. As it is described in [Russel & Norvig, 2010: 42] the environment can be *fully observable* or *partially observable.* When the agent obtains constantly through its sensors, information about the environment's state, then we are talking about full observability of the environment. The partial observability could be due to noise in the agent's sensors or missing data. It is obvious that in general we would like a fully observable environment because this means that the agent does not have to keep track of the world. Another dimension is *single-agent* or *multi-agent* environment. For instance, an agent that solves mathematical equations is acting in single-agent environment, whereas a game against two or more agents is obviously a multi-agent environment. At the beginning, it was mentioned that a multi-agent problem is closer to a real-world problem, therefore it is more complex. An environment can be also characterized *deterministic* or *stochastic.* Almost everyone is familiar with these two philosophical terms,

but when it comes to computer science, an environment can be deterministic if the next state of the environment depends on the current state and of course the agent's action. Any other description is connected to a stochastic environment. Another dimension is when we have to determine whether the agent needs to think ahead or not, meaning that if the decision taken at a state *s*, affects the state *s'.* If the short-term actions have long-term consequences then we are talking about a *sequential* environment, otherwise we have to do with an *episodic* environment. *Static* or *dynamic*, *discrete,* or *continuous* and *know* or *unknown* are also some dimensions that can describe an environment and determine the agents design and actions. In Table 1 we can see some examples of task environments and their characteristics.

But it is not only the environment that can differ but also the agents that act within it. The features of an agent can affect the decision-making process and in order to generate complex solutions as we see in [Garcia et al. 2010 as cited in Dori et al., 2018: 3] and in [Rabuzin et al., 2006: 158] these specific features are that the agent is capable of work without human intervention and make decision on its own, i.e., *autonomy*, *proactivity*, meaning that the agent explores new possibilities and is able to take the initiative and *adaptability*, so that the agent is able to adapt to changes that take place in the environment.

Regardless of all the aforementioned characteristics that could define an agent, some key components remain the same. An agent that is operating within an environment is affected from it (perception) and the agent's actions affect the environment too (action). As we can see in Figure 1, the main components of an agent that acts autonomously in a single agent or multi-agent environment are perception, action, and reasoning. Through interaction with the environment, the agent receives information, selects actions and observes the effects that these actions have on the environment. Over time, after many repeated interactions and observations from the side of the agent the learning process improves reasoning.

This brief description of MASs and agents will help us in the next sections to determine and describe the reinforcement learning technique in the gaming environment, the agents that are taking action in it and what does it mean to act under resource constraints in a more detailed way.

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Table 1:Task Environments and their characteristics [Russel & Norvig, 2010: 45]

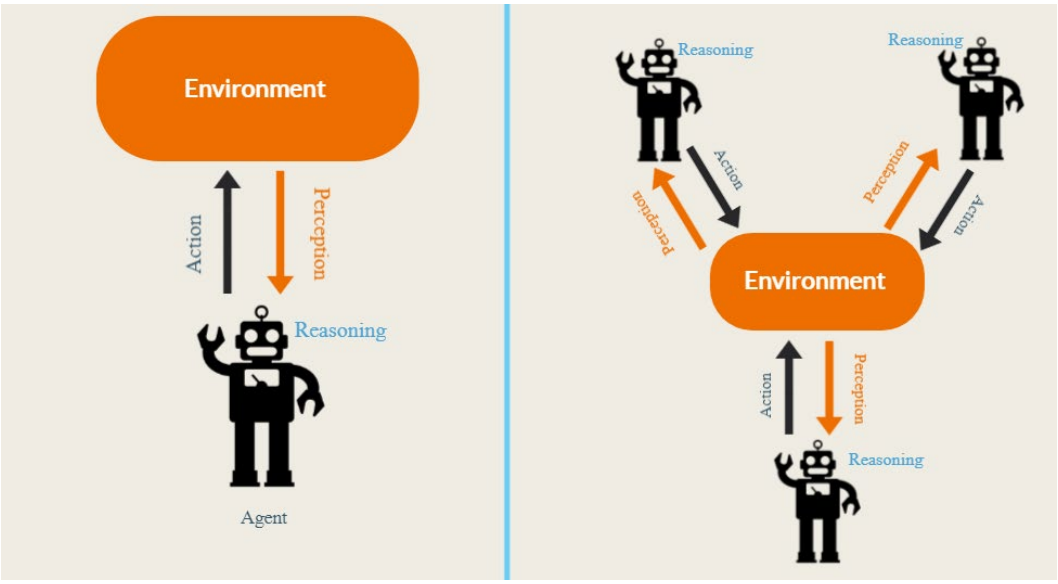

Figure 1: Framework of an autonomous agent

1.2 Reinforcement Learning in Multi-Agent Systems

Reinforcement Learning is an approach of machine learning inspired by behavioral psychology and the general idea behind it is that it is mimicking the way animals and humans learn to take decisions, therefore the agents that act within an environment use the notion of *reward* (positive or negative feedback received from the environment) in order to learn and maximize their success. This means that the agents observe the effects of theirs actions and learn from their observations, as a result in the future, if they have to make a decision on a specific state, they know whether a move is in their favor (maximizes the reward) or not. Many multi-agent systems use *reinforcement learning* techniques in order to train their agents. As it has been mentioned above, most of the times a multi-agent system is considered to be effective, because it can divide its main task into smaller subtasks and share it across multiple agents. As we can imagine, such systems need particularly good collaboration (assuming that the action takes place in a cooperative environment) and coordination between the agents. In order to reach a satisfying level, meaning that the agents have reached a level where their next moves are rational, the system needs lots of hours of training and learning. For instance, everyone is nowadays familiar with the story of DeepMind's AlphaGo and its famous game against the human world champion Lee Sedol. AlphaGo's system uses reinforcement learning approaches in order to train. It started from day zero and a random behavior and reached a satisfying level more or less after 40 days of training and 29 million games of self-play (no human intervention). [Silver et al., 2017]

Most of the times, in order to model the interactions in reinforcement learning between the agents and the environment, especially in the stochastic ones, we use the Markov Decision Process (MDP) technique. Essentially MDPs are probabilistic models of sequential decision problem. Reinforcement learning was originally developed for MDPs. A brief description of a state at a particular point in time t would be that the agent finds itself in a state s and at this state has to make a decision that leads to an action a. Since, we have reinforcement learning we expect from the environment a reward r. Learning is achieved by the continuous agent-

environment interaction and by the agent's observation of the effect of its actions. The trial-and-error characteristic is one of the most important features of reinforcement learning. At the end, what we are expecting from the agent is to find out a policy π that contains actions that maximize the summary of rewards. The best actions that an agent can make form the optimal policy π^*. As *policy* is defined the probability that an agent has at state *s* to choose an action *a*. Generally, in reinforcement learning there are two basic function that are used in order to identify how 'valuable' is a state and the value of an action *a* at a specific state *s*. These functions are the *value function* and the *action-value function* respectively.
Now, let us say that we have the following variables for a single agent acting within an environment:

- *S* is the set of states. $S = \{s_1, s_2, \ldots, s_n\}$.
- *A* is the set of actions. $A = \{a_1, a_2, \ldots, a_n\}$
- *P* is a transition function. In simple words, it describes the to which state *s'* the action *a* takes us if we are in state *s*, i.e., *P (s,a,s').* It essentially describes the probability of transitioning to another state from a current state by doing an action.
- R is the reward function. *R (s, a, s')* is the rewarded that an agent is expected to get when it does the actions *a* in order to transition from state *s* to state *s'*.

All the above are represented in a tuple *(S, A, P, R)*. The alternation between action and perceptions helps the agent to interact with the MDP.

Now another possible representation of a multi-agent environment, as it is presented in [Guo & Buerger, 2019] is like the following. Consider that we have a team of *N* agents, we would use an excessive form of MDP as a tuple to describe the current condition of each agent *n*, where *S* is again the set of states in the environment, *A* is again the set of actions that each agent can execute, *C* is the set of pairs of the states and the actions of the other agents, *R* the reward function and *T* the transition function.

$$M_n = (S_n, A_n, C_{-n}, R_n, T_n)$$

As the system is learning and evolving it ends up to the ultimate 'path', that is the best reward, and it would be in the form of a sequence of action-state pairs. For example, an agent at the state S_4 can choose the a_1 action and end has 0.6 probability to end up at the state S_5 with reward zero and 0.4 probability to end up at the state S_6 with reward zero as well. (Fig.2).
Of course, depending on the features and the complexity of the environment the MDP tuple changes, e.g., in a single-agent environment the variable *C* would not exist, since it describes other agents and not the one that we are observing.

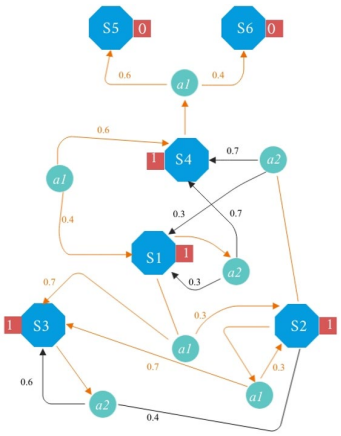

Figure 2: An MDP example of six states.

There are several reasons why the problem of learning MDPs is coped with reinforcement learning. In many cases, learning MPDs can be practically difficult, because the MPD should be known from the beginning, in large state-spaces the computational time exceeds the desired amount and in continuous state-actions spaces direct use cannot be applied. Fortunately, reinforcement learning can deal with all the aforementioned problems, with the only thing that is actually required is the agent-environment interaction. The input of the current state *s*, which usually can be seen as a vector that contains all the information that is available from the environment at a particular state and the reward *r* for the particular state *s* represent the interaction. Essentially the reward *r* is the input that is coming from the environment and is directly associated with the training of the agent. The agent is consisted of three important parameters. The *model* where the states and the actions are mapped, the *learning update*, where based on the rewards the model is being updated and the *policy*, where action selection takes place.

In general, the goal of reinforcement learning is for the agent to learn the optimal policy π, that maximizes the reward received with a particular transaction. As we can see in the following formula, the optimal policy is a policy that provides the best value function for every state.

$$\pi(s) = \arg\max_{\alpha} \sum P(s, a, s')(R(s,a,s') + \gamma V(s'))$$

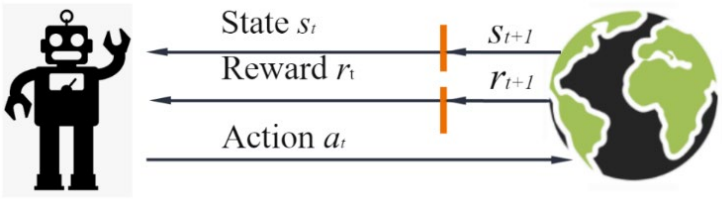

Figure 3: A simple depiction of reinforcement learning architecture

1.3 Exploitation/Exploration trade-off

During the introduction section, we have mentioned that we are mostly interested in conducting our experiments within a gaming environment of synthetic agents, defining them as agents with different characteristics. In reinforcement learning, there is always a discussion on the right balance between the knowledge that the agent has already acquired and the use of it (exploitation) versus the unexplored territories of the search space (exploration) and how much the agent should depend on it. There are several parameters that can affect the exploitation/exploration trade-off. The first is $\lambda \in [0, 1]$, which determines the *degree* that the updates are going to be influenced from events that occur later in time. A good value for λ depends on other parameters, but usually a value between 0.6 and 0.9 is considered an acceptable one. Another parameter is the *discount rate* $\gamma \in [0, 1]$. This factor determines the current value of future rewards. For instance, a reward that is received after *k* steps worth γ^{k-1} times what it would worth if it would have been received now. Within the aforementioned domain, the agent's behavior is determined between a *risky* or a *conservative*

frame. For $\gamma = 0$ the agent considers only present rewards and does not care about the future when on the other hand for $\gamma = 1$ the agent is acting under a more farsighted strategy. Certainly, we do not want our agents to be neither short- nor farsighted. A rather famous approach to balance the problem, is the ε – Greedy, approach, which is a mechanism that specializes in action selection. The factor $\varepsilon \in [0, 1]$ denotes the *probability* for the best move to be selected when the agent is based only on the knowledge that it currently possesses. The probability $1 - \varepsilon$ denotes a random move. As we are going to see in a later section, these three parameters (ε-γ-λ), so called *characteristic values* are the ones that determine the agent's playing profile.

In the last couple of years, the significant progress has been achieved in the field of artificial intelligence regarding the balance between exploration and exploitation. For instance, we have seen in [Badia et al. 2020 as cited in Leonardos et al. 2021: 1] policies that are parameterized from trained neural networks in order to select a between an exploratory and exploitative policy. Other examples of progress regarding the aforementioned dilemma, is the ranking of agents that participate in tournaments, according to their performance, introducing a principled evolutionary dynamics methodology, [Omidshafiei et al. 2019: 22] that even can involve noisy outcomes [Rowland et al. 2019: 9].

Despite the progress that has been mentioned it is worthy to clear that still there is not a reliable method in order to confirm that the agent has fully understood the environment that it is acting within. In simpler words, we cannot fully confirm that an agent has visited all the possible states and has performed all the available actions, therefore it should terminate its exploration. There is always a chance that there is a specific action, at a specific state, that will provide the agent with a much more beneficial reward. Usually, for the exploitation-exploration trade off to be determined it is used the ε-greedy policy, a policy that we also used in our experiments. The probability 1-ε that we define earlier, in this policy describes the probability to act in order to get the highest reward, while the probability ε on its own denotes the random selection of an available action at the current state. For $\varepsilon = 0$ the agent acts only accordingly to the optimized policy, without exploring the environment. On the other hand, for $\varepsilon = 1$ the agent acts under a fully explorational approach.

All the previous information was mentioned, in order for the reader to get an idea what is an agent, an environment and how they can interact and affect each other. Also, the aforementioned characteristics of the agent and their parameters, could be used, as we will see further to create avatars in the RLGame platform that differ from each other in terms of these characteristics and therefore investigate potential behavioral characteristics when these types of avatars are competing in resource constraints scenarios.

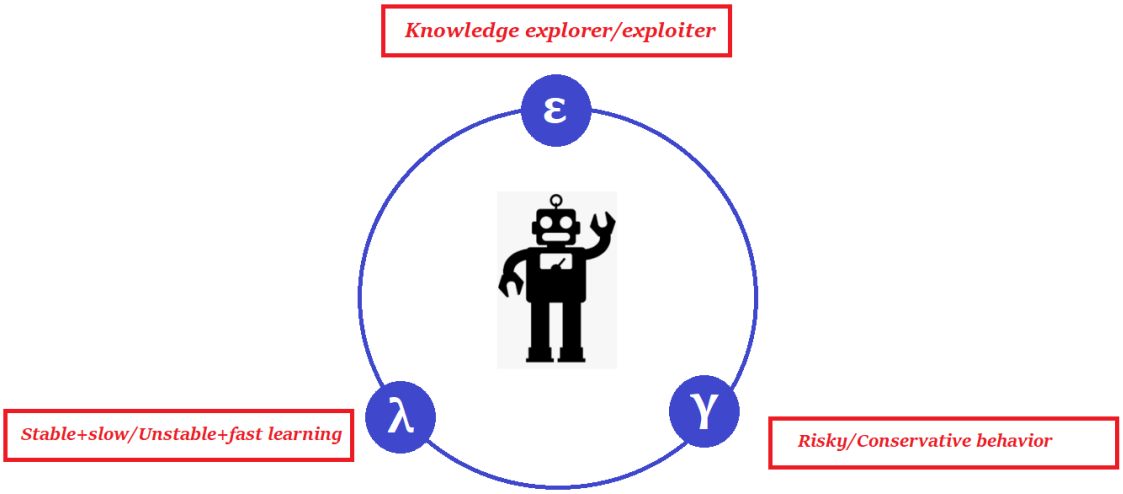

Figure 4: Characteristic values (blue) and possible playing profiles (red)

Chapter 2

RL Game

In this chapter we are explaining the concept and the rules of the RL Game, its development through time from the previous research on it and briefly explain what the term *resource constraint* could mean in the RLGame and in multi-agent systems in general.

2.1 Environment and Rules

The RL Game as we have mentioned in the previous section is basically a combination of reinforcement learning and neural networks and it was first introduced in [Kalles & Kanellopoulos, 2001] as an application of reinforcement learning in order to deal with playability and learning issues. Since then, the RL Game has been frequently used and further developed during many theses. Our inspiration had been the direction for future investigation that was suggested in [Kiourt, 2017: 145], regarding the opponent selection in the RL Game under resource constraints. Further we are going to describe the environment of the game and analyze what these constrains could be, but first, we will briefly introduce the game and its rules.

The RL Game rules are simple. Two players compete against each other, in a board game of *N x N* squares. Each player has a "base" where her pieces (if it is agreed between the players can be more than one for every player) start (Fig.2). The piece at its first step, can be moved only at the squares just outside the base (Fig.2 red squares) and not inside the base. Outside the base each piece can move one square around its current position (Fig.2 green squares

assuming that a piece is located at the blue square). In order to win the game, a player must move to the opponent's base or eliminate all the opponent pieces.

Earlier we have given the definition of a synthetic agent. In RLGame, every human player that is created, can own a number of *avatars*. Avatars are agents that belong to a human player, but they play and learn the game on their own. All the scenarios that we describe in further sections are based on competitions between avatars. In Chapter 3 we can see in detail how an avatar is connected to a human player and the interaction between two avatars in the RLGame platform.

As we have mentioned earlier, the RL Game has been used frequently as a base for research in several academic theses. For the purposes of these theses a platform has been developed, in JAVA and JAVA script, in which players can play the specific game and at the same time investigate the agent who plays the game. Our experiments and the scenarios' implementations were conducted on this platform.

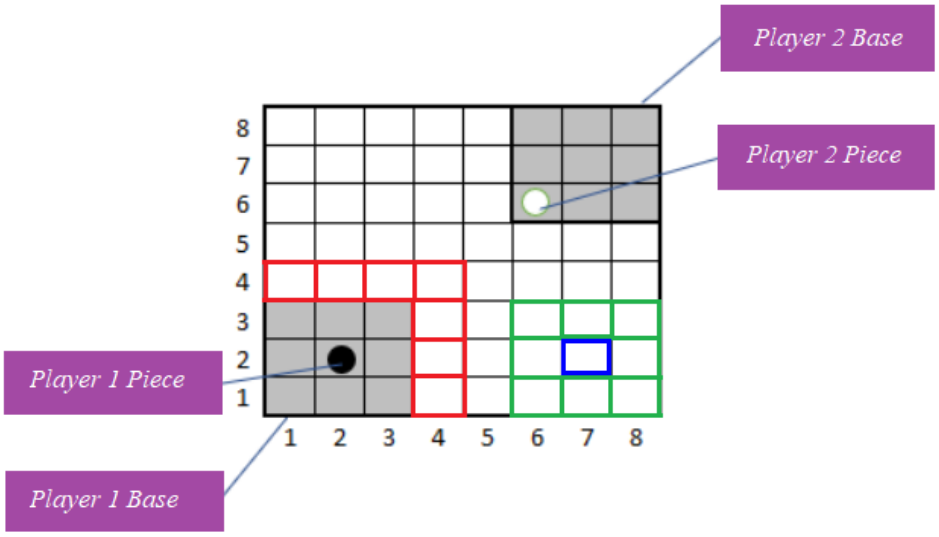

Figure 5: RL Game - Initial state and possible moves. Hellenic Open University (site)

Now, the RL Game environment's description, can be based on the features that we have described in the previous section. We can say that it is an *accessible* environment, since the agents, that act within the environment, can get complete information about the environment's state, meaning that the position of every piece inside the gaming board is

known to the agents. Also, the environment is *discrete* and *static*, since there is a set of moves that could be done since the final result of the game and the environment itself is only modified by the actions of the agents. And of course, is not *deterministic*, because the decisions of the agent on their next moves do not have a guaranteed, although there is a rule that says, when a piece reaches a position from which it cannot move, all the possible moves are blocked from opponent or allied pieces as a result the blocked piece is out of the game. This particular phenomenon causes a form of *deterministic discontinuity* in terms of how the game continues over time.

2.2 Resource Constraints in MAS and RLGame

In a MAS environment, information needs to be exchanged between the agents in order for the maximal gain to be achieved. For instance, consider a team of robot agents in the real world, that are deployed to complete a task of transporting a big number of heavy boxes from point A to point B. The agents need to constantly communicate with each other, sharing information regarding the amount of boxes left, the amount of robots at the pickup station and since our example takes place in the real world, we can imagine that they share information on their energy left e.g., battery level. The latter is an indication that the agents are acting in a complex environment under resource constraints. Their energy level indicates that they need constantly to update each other about their efficiency at the current time, always taking the best decisions in order to maximize the target, which at this specific example is to transfer if not all the largest amount of the boxes, that their energy level allows. We can imagine the same idea in an example of software agents. Many problems that take place in a MAS are well-known to be NP hard, due to their exponential increase of the state-action space [Cappart et al. 2020: 2]. There are several solutions that have been developed in order to tackle such problems. Optimization tools and deep reinforcement learning has shown great potential in handling this type of complexity, especially the latter, even though it is still hard to apply deep reinforcement learning techniques in many problems [Foerster et al., 2017: 6].

Another example problem that can describe the complexity of decisions in a resource-constrained MAS, is the bandwidth consumption. It is important that the agents communicate and share information with each other, but when this happens at an indiscriminate level it can be problematic because important amount of bandwidth is consumed in information exchange that it is not critical to reach the maximal reward [Dutta et al. 2007: 3].

In RL Game we do not have the problem of the energy limits like in our first example, but we can assume that if the game design reaches overly complex levels the system will fail due to the computational power needed. As we understand the "resource constraints" take another meaning in the platform. For instance, resource constraint from the scope of the agent could be considered the number of games that an agent plays against another. If we would try to create performance rankings of all the agents with only a few games played and allow to all of them to access this ranking information, it would be interesting to watch if the agents decide to compete against a specific opponent in order to maximize their rewards or maybe skip another one to avoid a big loss. Another resource constraint of course could be computational power that our personal computers can provide (measured in FLOPS or benchmarks), even though the first constraint probably sheds more light on the agent's behavior since it is a closer example of the human behavior in cases that we have to create a strategy and choose opponents. There is always though the possibility to investigate the behavior of the agent under computational constraints and watch how they would share their power in order to reach the maximum reward at a specific position.

Our main idea based on the aforementioned constraints, is to first execute a different number (which would be a limited one, hence the constraint) of games between the avatars in order first to identify a value for the number of moves that can be executed in one game (number of moves constraint) and observe at which point (number of games) this starts to affect the decisions of the agents knowing of course the performance ranking of all the agents in the environment, when this is combined with a constraint on the total number of moves. We would like to see if the avatars will reach more or less the same performance ranking and if this is the case what will drive them to select a specific opponent under these circumstances. At the next sections we are going to describe the previous research that has been conducted

on the RL Game platform and also our approach on the game, which briefly is that we would like to try several numbers of games between the agents in order to develop avatars with different experience (score) and at the same time try to explain our observations and conclusions regarding the different ranking of the agents and sharing our ideas on the possibility of a more dynamic expansion.

2.3 Previous Development and Research

As we have earlier mentioned, the RL Game had been used many times for research purposes. Its simplicity makes it an exceptional game to try out different algorithms and approaches on reinforcement learning and not only. But in order for us to be able to try our approach on the RL Game platform, which we will explain in more detail in the next section, lots of researchers, during several experimentations and approaches have brought the platform to its current form. Since the development in [Papageorgiou, 2008] the researcher is able to conduct experiment, while modifying the reinforcement learning and neural network parameters. The usage of the platform through a simple internet web browser and the access to the agent's information was first presented in [Vlasi, 2008]. Two remarkably interesting and helpful developments were the results in [Georgas, 2012] and in [Nikolaidis, 2014] where the researchers examined the development of the neural networks regarding the different gameplay and also achieved human vs human, human vs avatar and avatar vs avatars games, respectively. Now it is also possible to train an agent exclusively with human intervention, spectate games [Sarantinos, 2015] and choose the board size of the games and the number of pawns as well [Nikolakakis, 2016]. The latest developments on the platform were made in [Giantzoglou, 2017] and [Vasilopoulos, 2019], where the researcher is able to compare possible moves with or without reinforcement learning in the two teams and set a new parameter in the new game settings, which is the parameter of repeated games between two avatars. The latter is something that we are going to use extensively during our experiments, due to the different levels of agents that we would like to create in order to study their behavior against agents with significantly different scores.

Chapter 3

User-System Interaction

This chapter provided to the reader an idea on how to interact with the system of the RLGame, through several platforms and environments, such as *Eclipse, PhPmyAdmin*, and *Windows Power Shell* in order to set up the parameters of the avatars and the game and start playing.

3.1 Platform Set Up and Agent Training

As discussed in previous sections, we would like to first to create several avatars that have different levels of experience (score) and then try to create several gaming scenarios and observe their behavior when it comes to opponent selection. The level of experience would depend on the games that they have played during the training phase. For instance, if an avatar has been trained in 20 games and another in 2000, certainly the level of experience highly differs. The number of the agents and their levels is something that we have to investigate as well in the first phase of our experiments, which means that we need to conduct experiments under different score levels and also within the tournament the number of the agents must differ.

Within the tournament, the agents as we have mentioned, will operate under resource constraints. This means that an agent does not have the luxury to play a relatively big number

of games in order to create the policy that maximizes the reward. For instance, in a tournament of twenty agents, that each of them has a unique score level, an agent knows the score of the rest and also know the number of the available games, that means that an agent that is in a low level needs to create a strategy that maximizes the reward within the specified from us constraints range. We are running the RL Game server, using the *XAMPP* application (Fig 7), which offers the necessary installations of *Apache* and *MySQL* in its packages. These are used, so we can have a clear image of what is going on in our database, what is the name of the avatars, what's their score (experience) etc. In the next section we present the whole set up of the repeated games function using *Eclipse*.

ID	Name	Username	Password	Is_Human	Owner_Player_ID	Score
1	Player_1	player01	pw01	1	*NULL*	0
2	Player_2	player02	pw02	1	*NULL*	1
3	P1A1(RL)	avatar0101	12345	0	1	8449
4	P2A2(RL)	avatar0201	12345	0	2	7205
5	P4A1(MM)	avatar0401	12345	0	8	433
6	P3A1(MARL)	avatar0301	12345	0	7	1369
7	Player_3	player03	pw03	1	*NULL*	0
8	Player_4	player04	pw04	1	*NULL*	0
9	Player_5	player05	pw05	1	*NULL*	0
10	P5A1(RL2)	avatar0501	12345	0	9	147

Figure 6: The RLGame server in phpMyAdmin showing the list of players and their avatars.

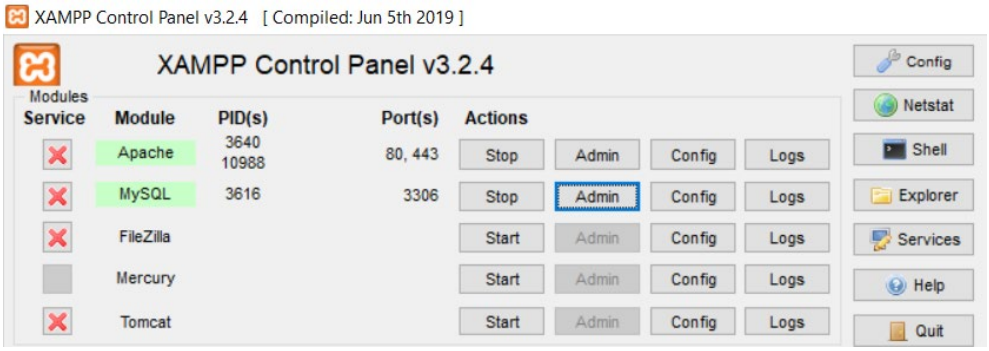

Figure 7: The XAMPP Control Panel operating Apache and MySQL.

In Fig.6 we can also see a table of players with their avatars, that use different algorithms (parentheses at the 'Name' column). The table shows the avatar owners (human users) and the avatars' scores (wins). We can see that the avatar0101 (avatar 1 of the player 1 trained with RL algorithm) has won 8449 games and the avatar0301 (avatar 1 of the player 3 trained

with MinMax algorithm) has won 433 games. As we have mentioned earlier, the previous developments on the RL Game platform offer to the user plenty of possibilities. During our experiments all the games between the avatars were set in a 6x2x5 game design (board size *x* base size *x* number of pawns) but the user can set different parameters (Fig.8). We choose always "create game as avatar", because as we have mentioned, we want to train the avatars and observe their behavior, without human intervention. Then we are just spectating the game (Fig. 9) as the avatars play against each other. Our target during the first phase is to run a significant number of games in order to create avatars with different levels of experience (more games played or trained with different algorithms) and observe the possible behavioral patterns in the scenarios that we will describe in the next sections. To achieve that we will use the 'repeated games' function of an updated version or RLGame which allows us to set a number of repeating games and allow the avatars to compete against each other until the number of selected games is completed.

Figure 8: Setting one game parameters in RL Game online platform.

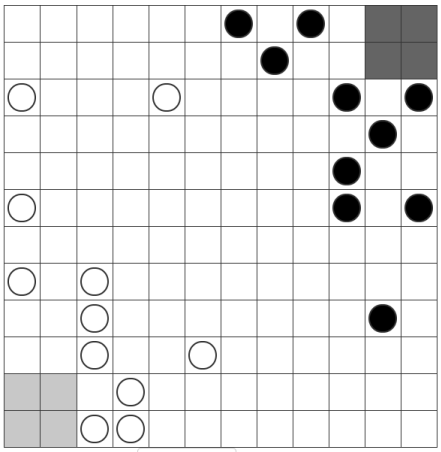

Figure 9: Spectatorship of an RL Game between two avatars

3.1.2 Repeated Games in RLGame

The setup of the repeating games and the scenarios is taking place in the Eclipse integrated development environment. The reason is that eclipse is offering a great flexibility in terms of changing the code of the program and testing it. As we can see in Fig.10 by setting up the project workspace in the eclipse platform it very easy for the user to instantly see all the files that complete the java project folders and make instant changes. In our example, we see how we can easily select the avatar and its algorithm (player type option). Eclipse offers an easy way as well to deeply search the code and make the changes regarding the player's profile. For example, in Fig.11 we see the different parts of code, for the players' training algorithms. This allows us to fast locate the place where the changes are needed to be implemented in order to change some parameters.

Now, in Eclipse in order to start our server, again the XAMPP application needs to run at the same time in administrator mode. Again, we run the Apache and MySQL services from there, and then in Eclipse we select the RLGame Server file and run it. Then we select the Client file and run it as well (Fig.12). The Client file allows the player with the corresponding avatar to be connected to our server. From the *Settings* file (Fig.10) we select the avatar that we want to start with (the player that starts has the white pawns). We run the second avatar outside eclipse, simply by using the *Windows Power Shell* and run the java command to star the Client file (Fig.13). Then our players are ready to begin a process of repeated games where their avatars are competing against each other. We login with the two players and automatically, since the client files are running, their avatars are connected too. Then we select the game parameters (board size, base size and number of pawns), the opponent and the number of repeated games and the training of the avatars starts.

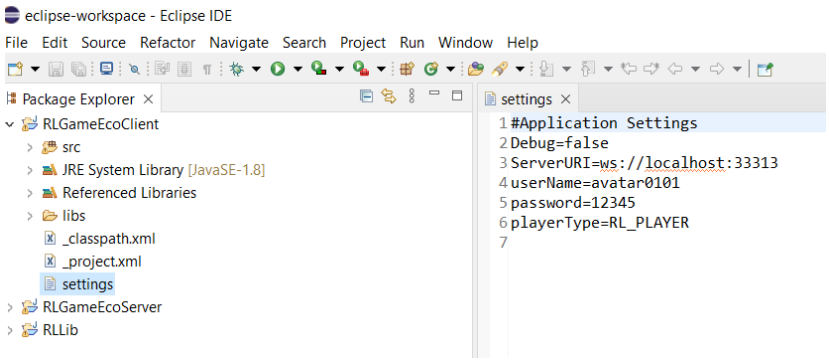

Figure 10: Selecting Avatar in Eclipse IDE

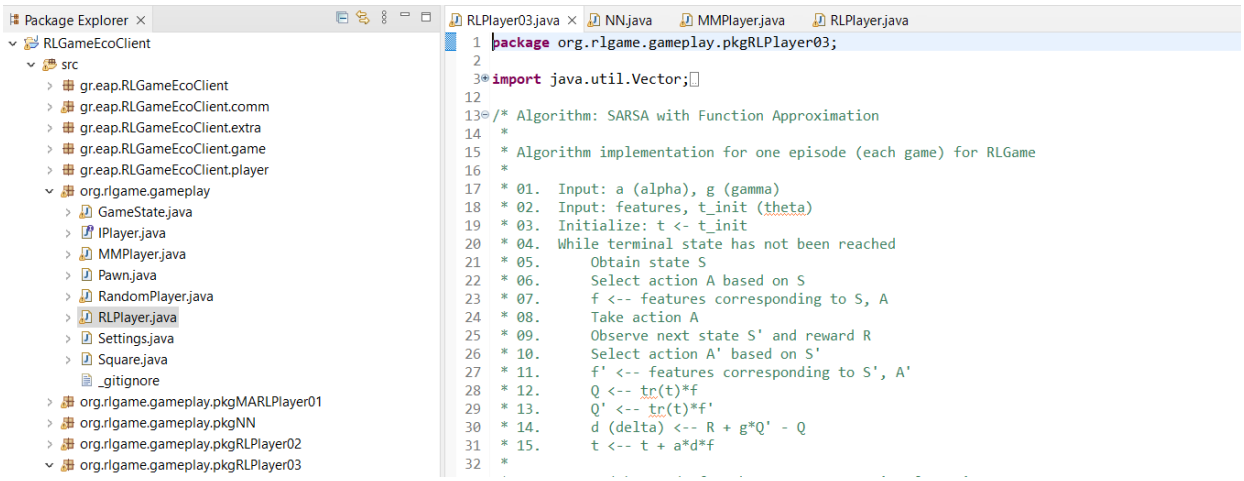

Figure 11: Easy access to the Java code using Eclipse IDE

Figure 12: Server and Client (Avatar) running in Eclipse IDE.

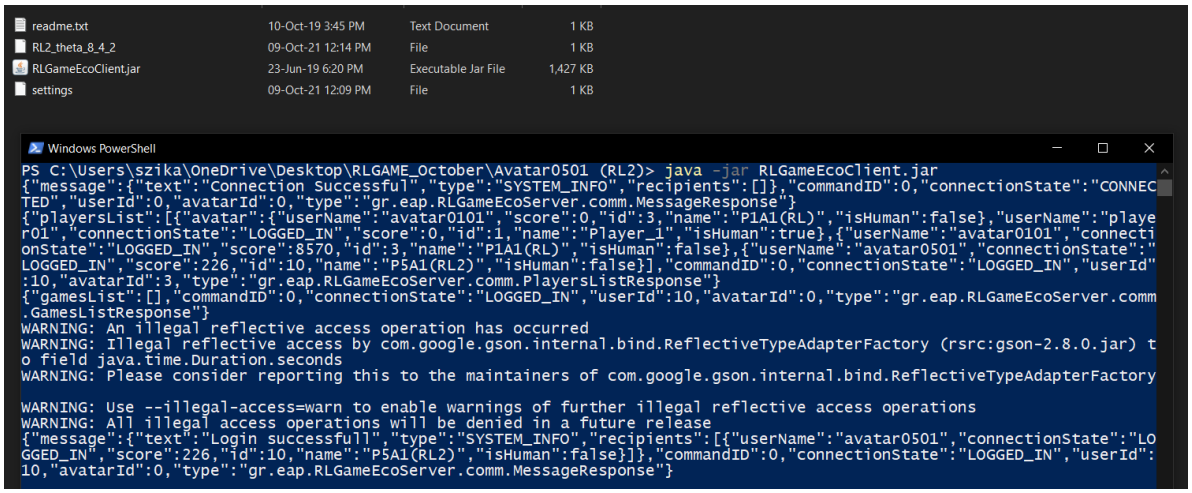

Figure 13: Running the second avatar with Windows Power Shell

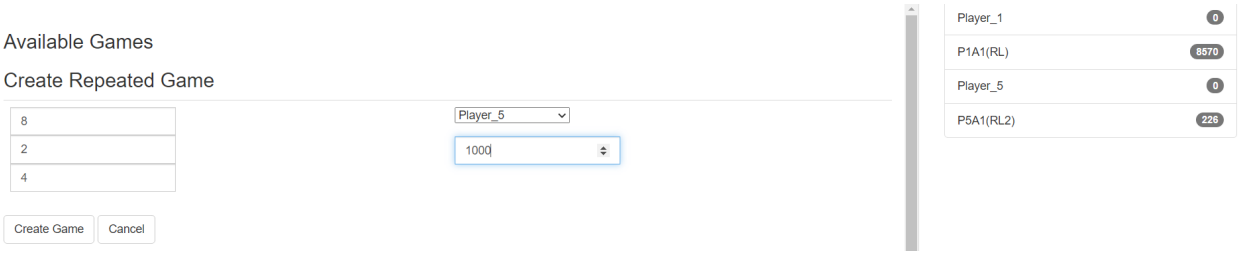

Figure 14: Setting up a process of 1000 repeating games between two avatars

Chapter 4

Introducing Resource Constraint Scenarios in RL Game

In this chapter, we propose different scenarios, in which the behavior of an agent could be investigating when it is acting under resource constraints. In each scenario, it is suggested a different type of constraint within a multi-agent environment. The scenarios were developed having as an example the RLGame platform, but they can be implemented on every similar multi-agent gaming platform.

Before we start describing the different suggested scenarios and the underlying behavioral patterns that we might discover, we need to clarify that the scenarios, are not explicitly connected to the spectrum of '*opponent selection*' but are closer to the '*behavioral adaptation*' of an agent while the circumstances of the game are changing, leading to an explicit or implicit opponent selection depending on the scenario that is being implemented. For instance, if an agent due to the applied resource constraint, chooses to stop a game in order to save some resources, because the opponent is stronger and does not want to waste its resources, it is an implicit opponent selection, although the agent is adapting its strategy mostly because of its resources.

The following scenarios could be characterized as scenarios of increasingly difficult implementation complexity, since the already existing code of the game, needs to be modified with the aim of creating different circumstances for the agents in order for them to adapt

their behavior correspondingly. We have previously described, that the RLGame is a complex system, that requires the usage of many tools at the same time in order to function properly. Therefore, from a minor change can occur many problems to the system since most of its parts are connected. This did not allow us to be as flexible as we wanted with the changes in the complex code and the implementation of different ideas and scenarios Nevertheless, as we will see in the next chapter, we have succeeded to set the fundamentals for a deeper analysis and implementation of our ideas.

4. 1 Scenario 1 (number of moves constraint)

During scenario 1 the agents can be studied under the *number of moves (n)* resource constraint. Each agent can execute only n moves in each game. If no agent wins until the n_{th} move then the game is considered a 'drop' from both sides (both players are participating in a 'drop' even though one decides it, therefore the result is a loss for both). The idea behind this scenario, is to investigate how many actual resources (moves) do we need in order to achieve a result (win) and which number of moves (n) can be considered a constraint. Obviously, setting n=1 or n=100 would be meaningless; therefore, we have to investigate the range of the number of games that we could design our experiment. Despite naming this process a 'scenario' it is not one per se but it can lead to the design of more dynamic scenarios as we will see later.

Now, a first simple approach in order to identify the number of moves is to run a significant number of games (1000), between agents that have different characteristics, therefore different game strategy and record the number of moves that lead to a win.
For instance, if the average number would be 17, the number of moves would be a number smaller than 17. Then again we would gather a sample of 1000 *winning* games which would be held under the 'at most 17 moves' constraint and see if it affects the behavior the results, therefore the agents' behavior. When the agents are competing under a constraint, obviously there are fewer options for interaction between the agents. Since,they would be under a

constraint and the option to drop the game, this means that we would have to run more than 1000 games to get a sample of 1000 winning ones.

Now, a more sophisticated approach in order to investigate the number of moves and get the whole picture of the agent's behavior, would be again to run the 1000 games and record the number of moves that lead to a win. The number of moves would be the data points that would help us design a distribution curve and examine what we can learn from its behavior. If we would see that the distribution curve has a sharp form we would expect that a change in the number of moves would have a significant impact on the agents' behavior at a certain point.
On the other hand, if the distribution would end up being in a flat form, a change in the number of moves, probably won't have the impact that we expect. Essentially, the distribution would show us if the probability to drop a game is increasing when we are setting *the number of moves* constraint.
The real question here is if we would end up having significantly different distributions when the players are of different characteristics and when the players are of similar ones. If we notice a noteworthy difference, this will mean that the *number of moves* constraint affects differently the agents that have different characteristics, therefore it might lead to an approach that ends up using different *number of moves* constraint for different kinds of agents.

However, if we will see that the distributions overlap, is considered an important result as well, because it is basically showing to us that the agents with different characteristics could be constrained under the same number of moves.

Between the two approaches, the latter is considered statistically better because it gives a better picture of the winning patterns. The 'average of winning moves' approach, could work but the only problem that someone should consider is that the decision regarding the *n* is slightly random, since there is no rule regarding where the constraint should be placed. For instance, if the average of winning moves is 18, we are not sure, which number <18 would be effective in order to affect the behavior of the agents, but the second options gives us a range.

4.2 Scenario 2 (budget constraint)

This scenario is based on the *budget* constraint, of every agent. The *budget* is the number of total moves that an agent can perform on all the games together, within a tournament. For instance, in a round robin tournament of 30 agents, each agent starts with a budget of 1000 moves. This means that there correspond about 30 moves per agent for each game. In this scenario, the number of average moves per game can be exceeded since there is no limitation for maximum number of moves. It is important to mention that every agent has the choice to *drop* the game at a certain point if it would think that it's not beneficial for its learning process. During scenario 1 explanation, it has been mentioned that if a player decides to drop the game, both of the players are participating into that decision. This is mentioned only to clarify that the result of a draw could not occur.
Let's imagine the following situation: Agent A plays against agent B. It is the 27th game of the tournament, therefore both of them have spent some moves on their games. Agent A has a remaining budget of 180 moves and the agent B has only 30 moves left, meaning that on average, up to now, agent A was concluding its games faster than agent B, not necessarily by winning them, of course. Obviously circumstances look better for the first agent rather than the second. The interesting fact here is that the agents are not aware of the opponent's situation but only of theirs. Agent B does not know that the opponent has a budget of 180 moves. The decision (play or drop) is based only on the budget and on the number of games left.

This scenario aims to investigate how agents exploit their number of total moves and trying to provide an answer to the question of whether an agent is capable of maintaining its budget until the last game but at the same time spend enough in order to compete and gain experience?

4.2.1 Scenario 2.1

In this more complicated version of scenario 2, independently of a constraint's presence, at the end of each tournament a table would be shared, showing the results of every agent. For instance, agent 10 has 20 wins and 80 loses and agent 24 has 50 wins and 50 loses. When the new tournament starts, the agents that are about to compete against each other (in our example agent 10 and agent 24) would first take a look at the table and then decide how many moves they might attempt to commit to that game. Essentially this is a more complex version of the second scenario, with the difference an agent is able to examine the results and make a decision on how many moves they would spend, regardless the constraints. Here it is needed to be clear that the two agents that are participating in a game of a round robin tournament, are affected only from their remaining budget, the remaining budget of their current opponent and from the number of remaining games. The number of remaining games is always known from the agents. For instance, if two agents currently play against an opponent for the 26^{th} game of the tournament, it is clear that the two agents are aware that for both is the 26^{th} game. Clearly, the complication of this scenario is mostly connected to its programming nature since the agents would have to examine an external table in order to reach a decision.

4.3 Scenario 3 (Board/Base size constraint)

In this scenario we are focusing on the *board size* constraint. Imagine, a simple game between two players at an RL Game platform, it is used to be held in an 8x8 board and a 2x2 base for each player. Let's assume again that we are conducting a round robin tournament experiment. Several agents of different characteristics are about to compete against each other. Obviously the agents, are trained in a certain pattern of board size (most probably at 8x8). As it is shown in Fig.1, the number of possible moving area is notably decreased when we increase the base size of each player from 2x2 to 3x3 within an 8x8 board size. Now, as we can see in Fig.2, if the base stays as it is, and we decide to decrease the board size to 7x7 the area of possible moves is decreased even more (from 56 to 41 free squares).

If now, instead of letting them compete within an environment that they already are familiar with, we would either increase the base size or decrease the board size. Both approaches would lead to an environment that would allow a significant number of less moves for each player. It is important to clarify that the information that each of the players receives before every game, is the board size number, the base size number, and the experience of the opponent.

It is probable that an underlying behavior would be revealed during this scenario, that is, the experienced player would adapt to the changing environment knowing that the game is against a less experienced opponent, and the less experienced might decide to drop the game knowing that the opponent is more experienced, and the environment is unknown.

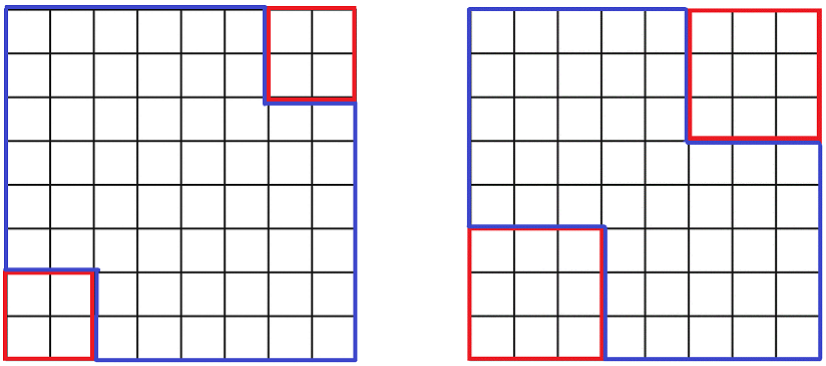

Figure 15: Increasing the base size, decreases the area in which the pawns can move (blue)

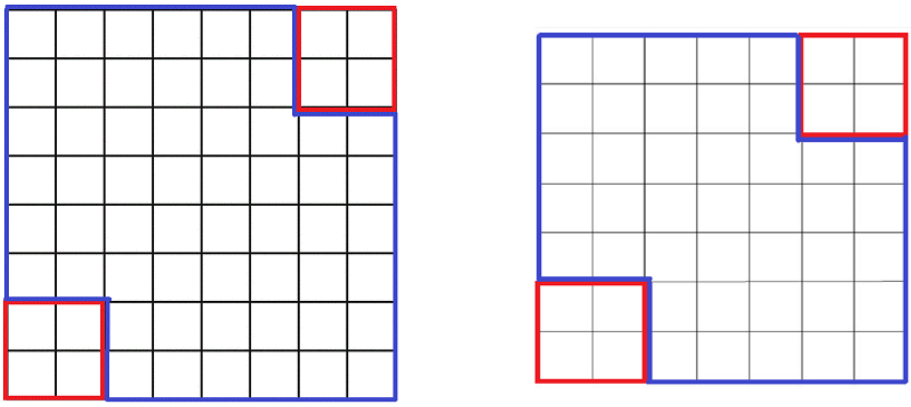

Figure 16: 8x8 and 7x7 board size respectively.

4.4 Scenario 4 (Block move constraint)

In this scenario the players are facing the *block move* constraint. For example, in a match between two agents, before they begin to play, they are informed that some of their moves are going to be blocked, which means that the opponent has an advantage of playing two times in a row, which obviously offers a great strategy advantage to the agent. In case that the two agents are of similar experience, they are going to be blocked the same number of times.

On the other hand, when the players are of different experience, the strongest player is going to be blocked more times than the less experienced agent. For instance, in a game between a strong and a weak agent, the strong agent would be blocked two times and the weak agent none.

During this scenario we can investigate several things. The first is the investigation of the experienced agents when they are facing the block constraint. Will they continue playing, because they are sure that the opponent is going to lose anyway, or the constraint of the blocked moves is going to make them drop the game.

The second question is whether between two agents, that are of similar experience but with different strategy characteristics (ε-γ-λ) we would find different behavioral patterns.

The third question, simpler than the aforementioned ones, is whether eventually the less experienced agent will continue to play, while knowing that its opponent has a number of blocked moves. Basically, we investigate the first question but t from the side of the weaker agent.

Chapter 5

Implementing the Scenarios

In this chapter we present in detail all the steps of the analysis that was conducted in order to identify the number of moves that has been described in Chapter 4 and the set up of the budget constraint additionally with its implementation in different situations, such as a small round robin tournament or many round of repeated games between two avatars, focusing on different directions each time.

5.1 Number of Moves: Analysis and Identification

As we've described in Scenario 1, our goal was to run multiple times a significant number of games in order to identify the *number of moves* constraint. This process was executed multiple times, each time creating new avatars and starting the round of 1200 games again. We chose the number 1200 for the number of games, because the code in eclipse, was returning sometimes '0' for the number of total moves in a game. This is due to a small bug in the code which still needs to be identified and fixed. Nevertheless, after some trials we realized that the number of zeros in our results was never exceeding the number of 100, hence the 1200 games between the avatars. This would give us more than 1000 clean data points in order to plot and observe the behavior of the data. We've decided to create a code that exports the zeros in our data though, so a future researcher can confirm our numbers.

The games took place on the eclipse IDE, where in the next section we will present the changes that we had to implement at the java scripts. The data analysis was conducted using different software programs. The cleaning of the data was done with Microsoft excel, where the exported data were analyzed and visualized in RStudio using scripts of R language.

5.1.1 Changes in the Java code

We've set a simple counter inside the *public class MoveCommand()* that keeps track of every move (from both avatars) that's being executed in each game. The user can keep track of the process of the game on the Server's screen where it shows the number of the current move. Then, at the end of each game the counter is simply nullified in the *class ConfirmGameStart()* when the next game is about to start. At the end of every round of 1200 games, a txt file is being exported from the *WriteResults() class* indicating firstly the description of the round setup (board size, number of pawns, date, time etc.) and then on every row the number of game, the winner and the total number of moves (summary of the moves of both players in the game).

```
public class MoveCommand extends Command {

    private int pawnId;
    private int toXCoord;
    private int toYCoord;
    private UUID gameUid;

    public static int x=0; //Counter for number of moves


    public MoveCommand(){
        this.setType("gr.eap.RLGameEcoServer.comm.MoveCommand");
        x++; //Increases with every move
        System.out.println("Move:" + x); //Prints on the Server screen the number of the current move
    }
```

Figure 17:MoveCommand() counter

```
{"pawnId":0,"toXCoord":3,"toYCoord":4,
Move:26
{"pawnId":1,"toXCoord":3,"toYCoord":0,
Move:27
```

Figure 18: Server number of moves indication

```java
@Override
public void execute() {

    Player player = PlayersRegister.getInstance().getPlayerById(getUserId());
    Game game = GamesRegister.getInstance().getGameByUid(getGameUid());
    if (player.equals(game.getWhitePlayer().getTeamLeader())){
        game.setWhitePlayerReady(true);
        MoveCommand.x=0; //Nullify Counter of moves when a new game starts.
    } else if (player.equals(game.getBlackPlayer().getTeamLeader())){
        game.setBlackPlayerReady(true);
        MoveCommand.x=0; //Nullify Counter of moves when a new game starts.
    }
```

Figure 19: New game comfirmation sets the counter to zero

```
Date: 2021/10/17 14:17:53
WHITE PLAYER Name: P3A1(RL_Moves)
BLACK PLAYER Name: P4A1(RL_Moves)
Game Configuration: 6x2x5
Number of Games: 1200

0: white 0
1: black 28
2: white 23
3: black 21
4: white 25
5: white 0
6: black 0
7: black 29
8: black 0
9: black 17
10: black 49
11: white 45
12: white 24
13: white 129
14: white 22
15: white 30
16: white 23
17: white 20
18: white 22
19: white 20
20: white 50
```

Figure 20: Sample of the exported txt file at the end of every round, where we can also observe the 'zeros' that were mentioned at the beginning of the current chapter.

5.1.2 Cleaning the data in Excel

From the exported txt file, we could easily copy paste the data into an excel file and from there by the use of a division and a round up function we were ending up on the number of moves that the winner made on each game. For instance, in Fig.21 we can see that for a number of 63 total moves in Game 2, this number is divided by 2 (*Winner Moves* column) and then it is rounded up in the *Moves* column. At the end of every process, we save the data into a CSV file and continue are analysis in RStudio.

Game	Moves	Winner Moves	Total Moves
1	0	0	0
2	32	31.5	63
3	16	16	32
4	14	14	28
5	27	27	54
6	29	28.5	57
7	29	28.5	57
8	28	28	56
9	14	14	28
10	23	22.5	45

Figure 21: Excel Dataset sample

5.1.3 Visualization in RStudio

In RStudio, we've created a simple script that reads the csv file from the directory where it is placed, then cleans the data, imports the necessary libraries, and plots the data in a histogram. In Fig.22 we can see the script for Round 1 of the 1200 games and the plot of the data. As we can see at the visualization of the 10 rounds of 1200 games, every time, we ended up having an almost normal distribution, with the indication being that each time most of the games were won within the range of [15,25] moves. This is an indication that a constraint could be set for *n>15* in order to set a constraint and probably expect a difference at the agent's behavior.

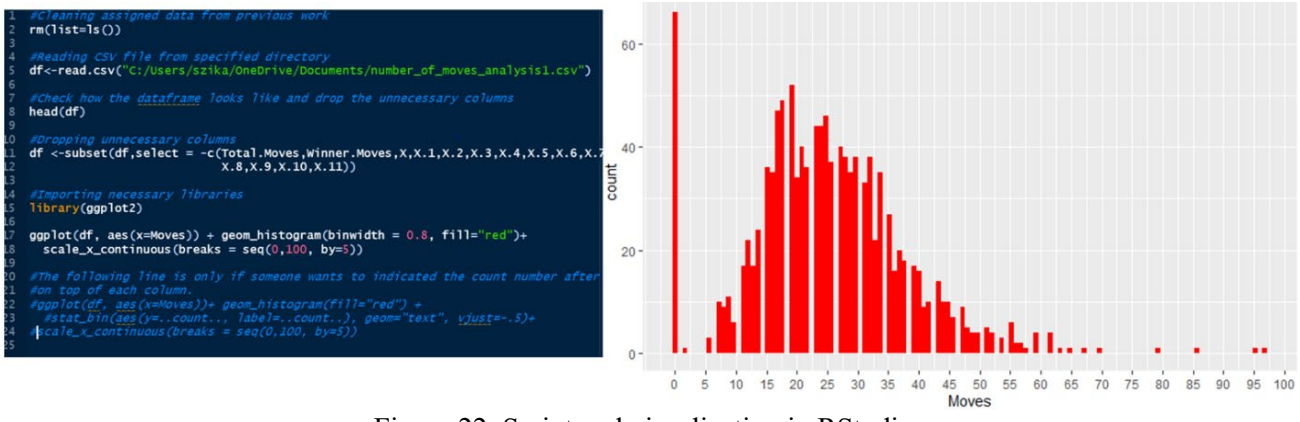

Figure 22: Script and visualization in RStudio

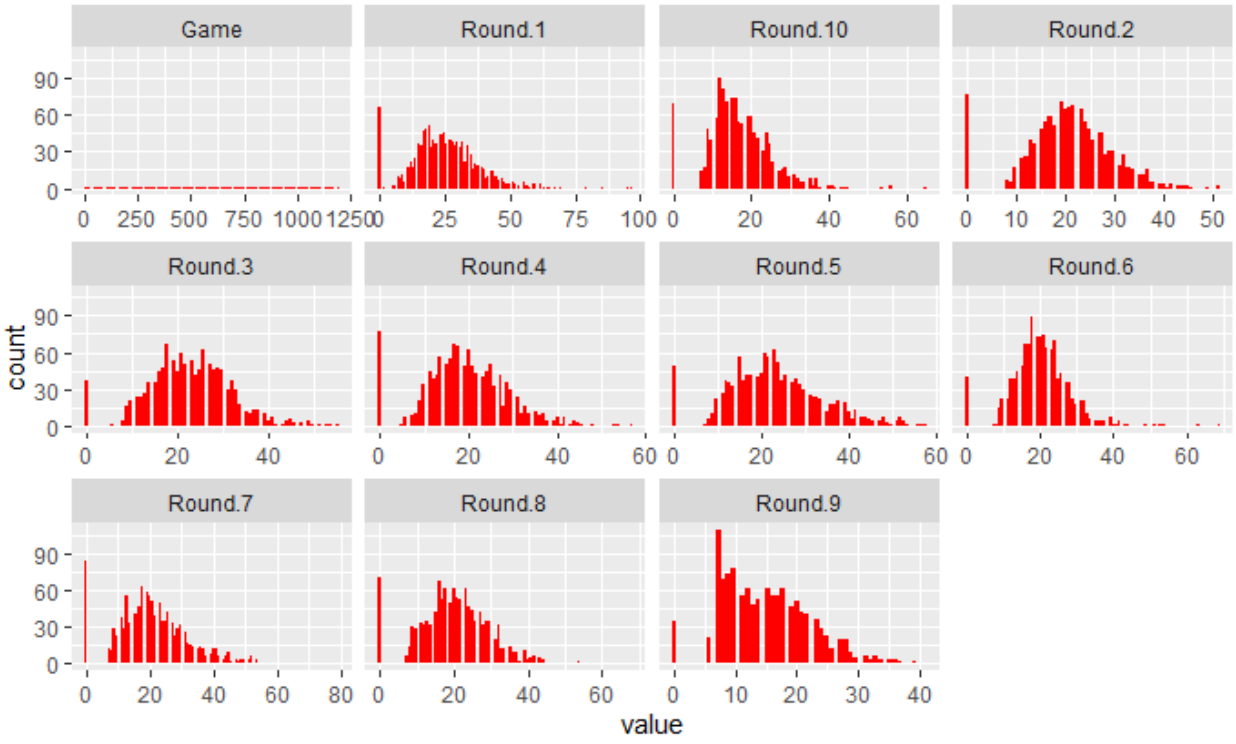

Figure 23: Histograms of all the rounds

5.1.4 Usefulness of the *n* identification

Now that we have a better idea on the range of the number of moves *n* constraint, it is important to clarify its usefulness. The main questions here are the following; first, how the number *n* can be used in one game and what could we actually reap from it, and second how the number *n* can be used in a tournament between avatars and what again are we expecting to obtain from its use.

Regarding the first question, the number *n* when set as a constant constraint within a game, could possibly lead the avatar to adopt another behavior when it comes to its playing style. For instance, when a game between two avatars starts with a maximum number of moves set to 15 for each player and after following the same approach that we followed in order to identify a range for *n* we could see in the plots a totally different behavior in comparison to the rounds that no constraint was present. As we have mention in Chapter 3, if no avatar has won until the n^{th} move the game would be considered a drop and a loss would be charged to

both avatars. In case that the number of total wins is a significantly low one, we can reach the conclusion that the avatars have a really hard time to identify a strategy in order to win when the constraint is set, and maybe we might reach the conclusion that setting a constraint for the number of moves is crucially affecting the ability of the avatars to adapt and win the game. The aforementioned question could be also approached statistically, with the same way that we've identified the number of moves *n*. The idea behind this approach is to run again a large number of games and plot the results. The difference now is that the number of moves constraint would be present and the avatars would be charged both with a loss in case no win has been achieved until the n^{th} move. When we have a large number of wins achieved, again we could create the histogram and examine the behavior of the moves this time. It would be interesting to investigate, whether the avatars are adapting on this constraint and teach themselves to win with less moves a or their behavior does not change significantly.

The answer to the second question is a little bit more complicated. If we could apply the *n* constraint within a tournament between avatars, the conclusion that we might reach would probably be the same as in the previous paragraph. However, when the number *n* constraint is combined with the *budget* of total moves constraint that we've mentioned in Chapter 4, we might observe a different behavior in comparison to the first question, when it comes to the management of the moves. For example, in a tournament between a number of avatars, let's assume that the total budget is *b* moves for each avatar and the maximum number of moves within each game is the number of moves constraint *n.* During each game an avatar would have the option to drop the game when is in the game or even when before the game starts. In these cases, we could observe whether the behavior is changing for avatars of different experience (score) in the RLGame. Of course, there is the possibility that we could see no difference at the behavior of different avatars, which would actually reveal that the degree of randomness in RLGame is bigger than we thought.

5.2 Implementing the Budget Constraint

During the explanation of the 2nd scenario in Chapter 4 we've briefly described the concept behind the *budget constraint*. The two avatars, have at their disposal a certain number of total moves that they can execute in a series of repeated games, therefore a tournament. As we will see in the next section, this number is expressed as a total number of moves that are allowed from both of the avatars. For instance, a budget constraint = 1000 means that both of them together can execute 1000 moves. This means that when the first game starts, the white player makes the first move and then the black player makes another move too, the budget would be 998. Further we are going to explain the changes in the code in order to achieve the implementation of this constraint, and later we are going to present the comparison between two tournaments of four avatars, where the one was constraint free and the other one was conducted under the presence of the budget constraint.

5.2.1 Changes in Java Code

Firstly, in order to implement the budget constraint, we had to interact with the existing system and make a few changes in the code. Again, working in Eclipse IDE, and in MoveCommand() class, we have simply created a variable with a specific value (the number of the budget, described as *b* in Fig.24), which reduces with step -1 when an avatar makes a move. For now, the user can only set the value of the *b* variable in the class, save the changes, and then execute the code as we've described earlier. Another change that we've added in the code, is that now after the execution of every move, the user can see on the Server's screen, the remaining total budget. In Fig.24 we see an example of game between two avatars, that started the series of repeated games with a budget of 500 total moves together.

After the end of every series of repeated games, our results are exported in a .txt file as we've seen previously, during the number of moves constraint analysis. Now, the .txt file contains, not only the moves that needed to be executed in order to reach a win but also the remaining budget after every game (Fig.25).

To ensure the continuation of the repeated games process, we have set the budget variable to be positive as a prerequisite for a pawn to move. When the budget reaches zero, the server terminates, and the process of the repeated games stops. These changes took place in the GameProcess.java file. Something that is worth to mention for a future researcher, is that the zero number of moves problem that we've described in the previous section, can be studied now in comparison to the remaining budget number at the end of every game. In Fig.25 we can see that the remaining budget continues to reduce at the end of every game, while sometimes the total number of moves number is zero.

Figure 24: The code for the budget constraint implementation

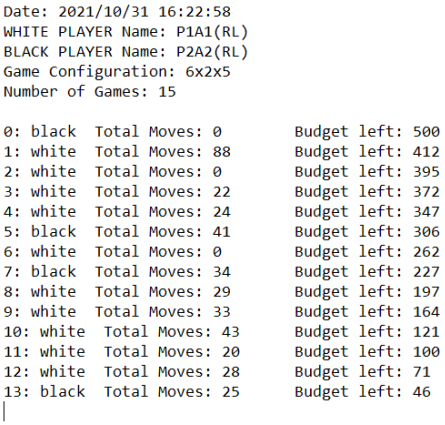

Figure 25: Exported .txt file for the budget constraint

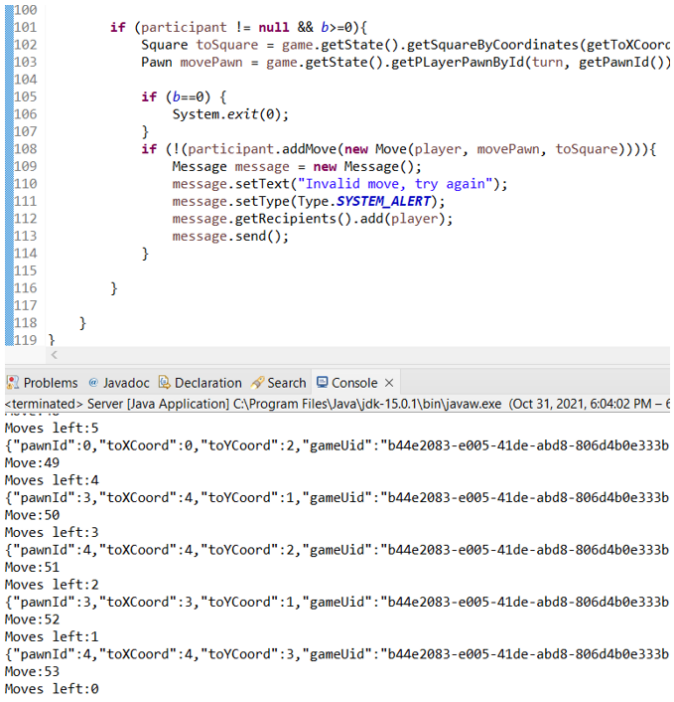

Figure 26: The series of games is terminated when there are no moves left on the budget.

5.2.2 Tournament under the budget constraint

After setting up the code in order to implement games under the *budget* constraint, we first decided to create two small round robin tournaments between four similar avatars of zero experience, one in normal mode (no constraints) and one under the existence of the *budget* constraint. All the avatars that participated in the tournaments were using the RL algorithm, that is available from the settings of the RLGame in the Client set up. Each avatar had to compete in 10 games against the opponent with a total budget on every round at 300 moves. This number was based on our previous analysis that was made to identify the number of moves per game. We've showed that a number of 15 moves per avatar would be considered a value that could probably constrain the avatars and their game, therefore 15(moves) x 10(games) x 2(avatars) provides us with the value of 300 total moves for the *budget* variable. In the following tables we can see the schedule of the games and their average moves per game for the tournament without and with the *budget* constraint respectively.

Round	Opponents	Average moves/game in 10 games	Average moves/game until n^{th} game
1	A1 vs A2	45.5	53.5
2	A1 vs A3	43.25	45.8
3	A1 vs A4	48.2	49.6
4	A2 vs A3	40.5	39.5
5	A2 vs A4	49.7	53.6
6	A3 vs A4	51.9	50

Table 2: Data from the tournament without any constraints

Round	Opponents	Games completed	Average moves/game until budget runs out in *n* games
1	A1 vs A2	5	43.75
2	A1 vs A3	6	39.4
3	A1 vs A4	6	50
4	A2 vs A3	6	59.75
5	A2 vs A4	5	43.6
6	A3 vs A4	4	48.25

Table 3: Data from the tournament under the presence of the budget constraint

The number of average moves per game was selected as a comparison point in order to investigate whether there is an indication of a more careful approach when the avatars are playing under the budget constraint. In Table 2 we can the column *average moves/ game until n^{th} game*. This category was inserted so the avatars that competed under normal circumstances can equally compared to the avatars that competed under the budget constraint, because statistically is wrong to compare a round of 10 completed games with a round of <10 completed games. Now, if we compare the last columns of the two tables, we can see that indeed the number of the average moves is most of the times higher when the avatars are competing in normal mode. Now, even though we see this small indication, we cannot conclude that indeed the avatars are approaching the game more carefully, since our data sample is small and for someone to conclude such thing would require a very large

number of games, which would have been extremely time consuming with our manually repeated approach. Despite our last results, we've decided to investigate the behavior of the avatars under the budget constraint with another approach.

5.2.3 Rounds of Repeated Games with Budget Constraint Between Two Avatars

First we created two new avatars, similar to the other ones, meaning that they have zero experience and are using the RL algorithm, and then we let them compete in 60 rounds of 10 repeated games under the budget constraint. The difference now is that the two avatars play significantly more games, therefore they are carrying more experience (despite the result). After each round of 10 games, we were saving the number of completed games. We thought that if the number of the completed games is slowly increasing, then this might be an indication that the avatars are adapting their strategy, so they complete more and more games with a given budget. As we can see at the next diagram (Diag.1), as the rounds are increasing, indeed the number of completed games seems to increase. Notice that we observe more 7s at the beginning of the diagram and then the 7s become less and the 8s and 9s more.

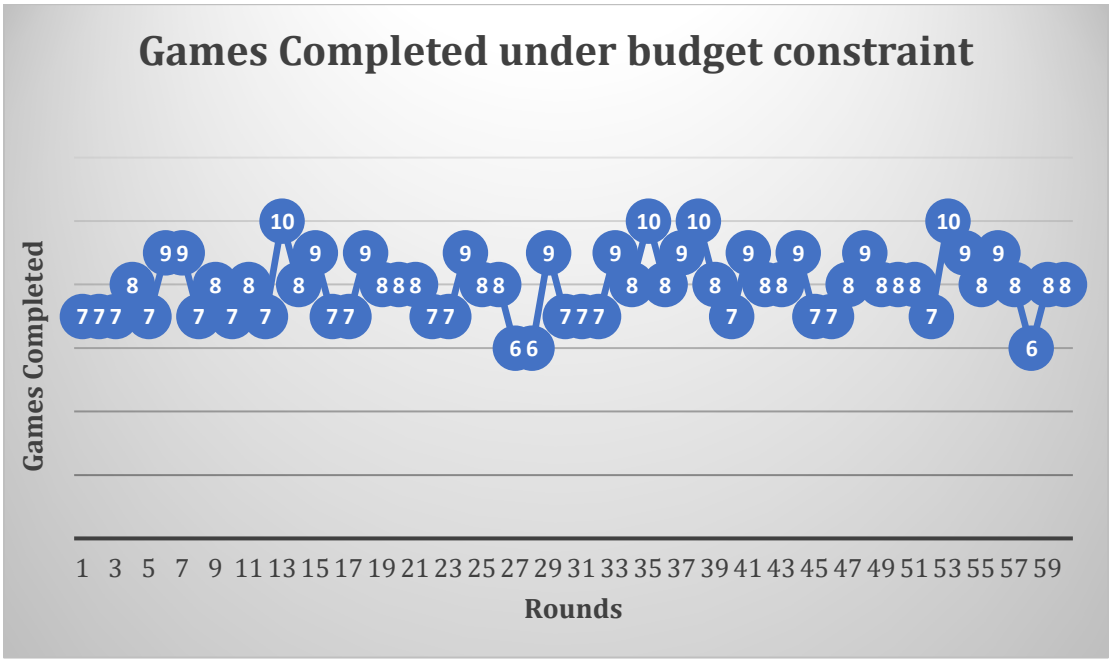

Diagram 1: Games completed under the budget constraint in 60 Rounds of 10 games

As we have mentioned earlier, the repeating process of the rounds was done manually, meaning that every time that a round of 10 games was fully completed or stopped because the avatars run out of budget we had to start again the server of the game in eclipse, activate the avatars, login and start again the process. It is indeed a very time-consuming process, that in future research on RLGame could be automated an allow the user to run more rounds, therefore collect more data for the analysis. Even though our data collection is relatively small, we can observe a minor indication of improvement in terms of spending the budget and completing more games.

As we have seen earlier, we started first by focusing on four avatars competing against each other in a small tournament, we continued by focusing only on two avatars competing only against each other and at the end we have decided to look a little bit closer to the avatars' behavior and observe only one avatar and more specifically its winning percentage. We run again 60 rounds of 10 repeated games between two avatars, but this time in normal mode. The data we have collected were compared with the data of the previous process of the 60 rounds under the budget constraint, focusing on the winning percentage of the *white* avatar. The presentation of the results is divided into two parts. One part is where the results of the winning percentage of the white avatar under the budget constraint is compared with the winning percentage in a completed normal mode (60 rounds, with 10 completed games each). The second part is where we compare again the results of the winning percentage of the white avatar under the budget constraint but this time we look its winning percentage in the normal mode until the n^{th} game. For example, as we can see in Diagram 1, the first round under the budget constraint had 7 completed games. According to our data the white avatar won 5/7 games. This means 71.5% (rounded up) winning percentage. In the first round in normal mode, obviously since there is no constraint, 10 games were completed. The white player won 3/10 games, which means 30% winning percentage. Now we wanted to see what the white player's percentage of wins until the n^{th} game was, which is the game 7 of the corresponding round under the budget constraint. In the next section we present the results of both of the comparisons.

5.2.4 Budget VS Completed Normal Mode

In Diagram 2 we can see that the blue bar charts, are representing the winning percentage of the white avatar at the completed games. From the example that was given earlier, we can see that the winning % of the white avatar for the first round under budget reached the 71.5% and in normal mode the 30%, which is represented by the orange line on the bar chart. We can easily observe that the winning rate of the white avatar is evidently better when the budget constraint is present. Theoretically this could be an indication of a behavioral adaptation skill since the avatar seems to adapt to the constraint and spend its moves more strategically. But as we have mentioned earlier, the data are not enough in order to support efficiently this indication, therefore this is also something that could be studied in the future. In Diagram 3, we can more clearly observe that the winning rate of the white avatar was 53% better than in normal mode, 42% better in normal mode and 5% of the time the corresponding rounds ended up with the same winning rate.

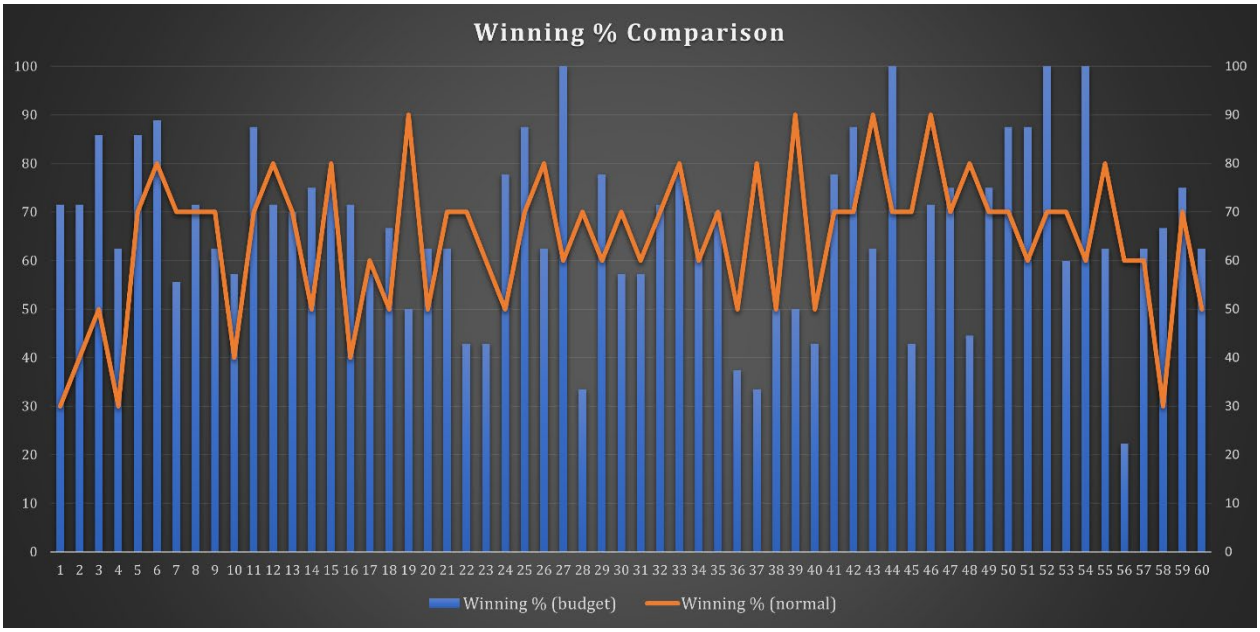

Diagram 2: Winning rate comparison. Budget mode VS completed normal mode

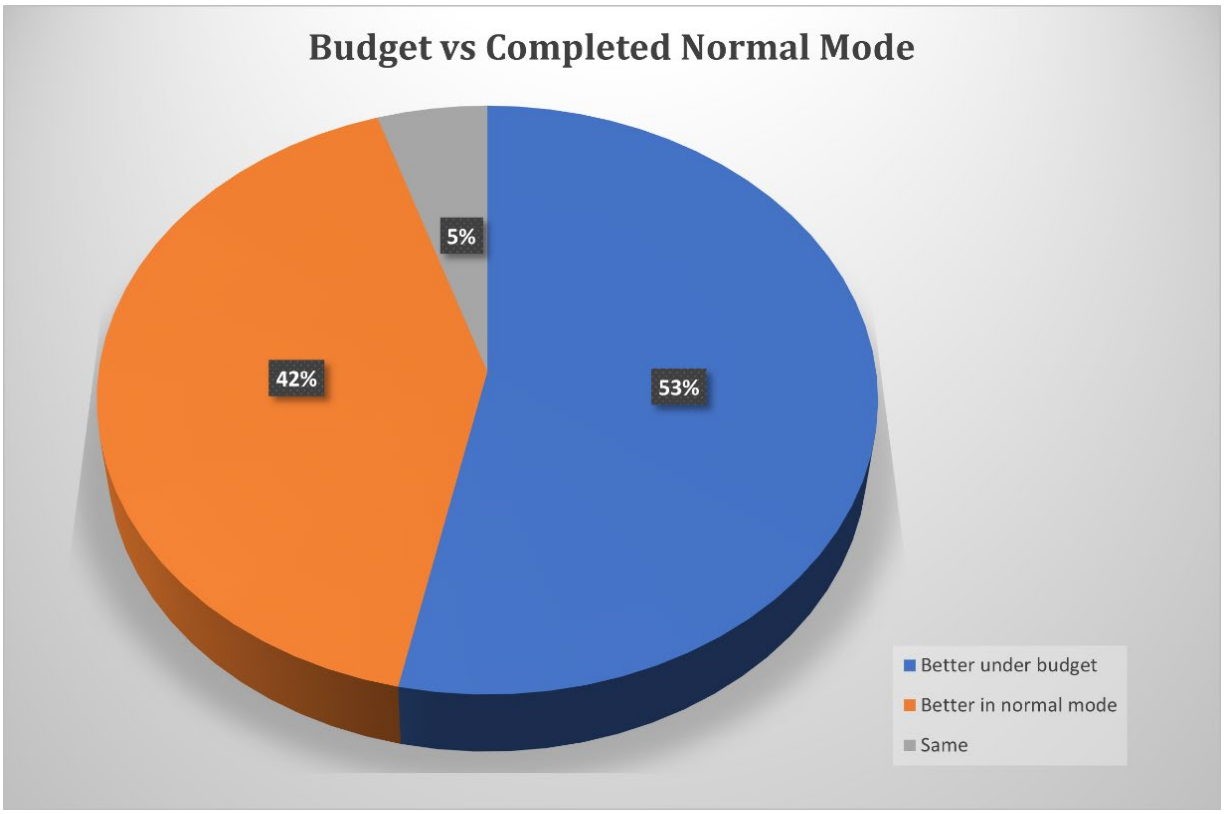

Diagram 3: Budget vs Completed Normal Mode Pie Chart

5.2.5 Budget VS Normal Mode until n^{th} Game

As we have earlier explained now these results are a comparison of the white avatar's winning rate under the existence of the budget constraint and the winning rate in normal mode until the n^{th} game, with n being the number of completed games at the corresponding round under the budget constraint. We observe that the winning rate in normal mode is until the n^{th} game is now lower, and the '*same*' category that represents equal winning rates at the corresponding rounds, is higher. Also, the difference of rates seems to have an equal displacement from '*better under budget*' to '*same*' and '*better in normal mode*'/ '*better in normal mode until the* n^{th} *game*, a fact that will be explained at the conclusion section. An interesting fact when comparing the two bar charts, is that in several rounds, e.g., round 35 and round 38, the white avatar's under budget constraint winning rate under budget constraint seems to be significantly better (Diag.4) but when the normal mode is fully completed they ended up having the same winning rate (Diag.2).

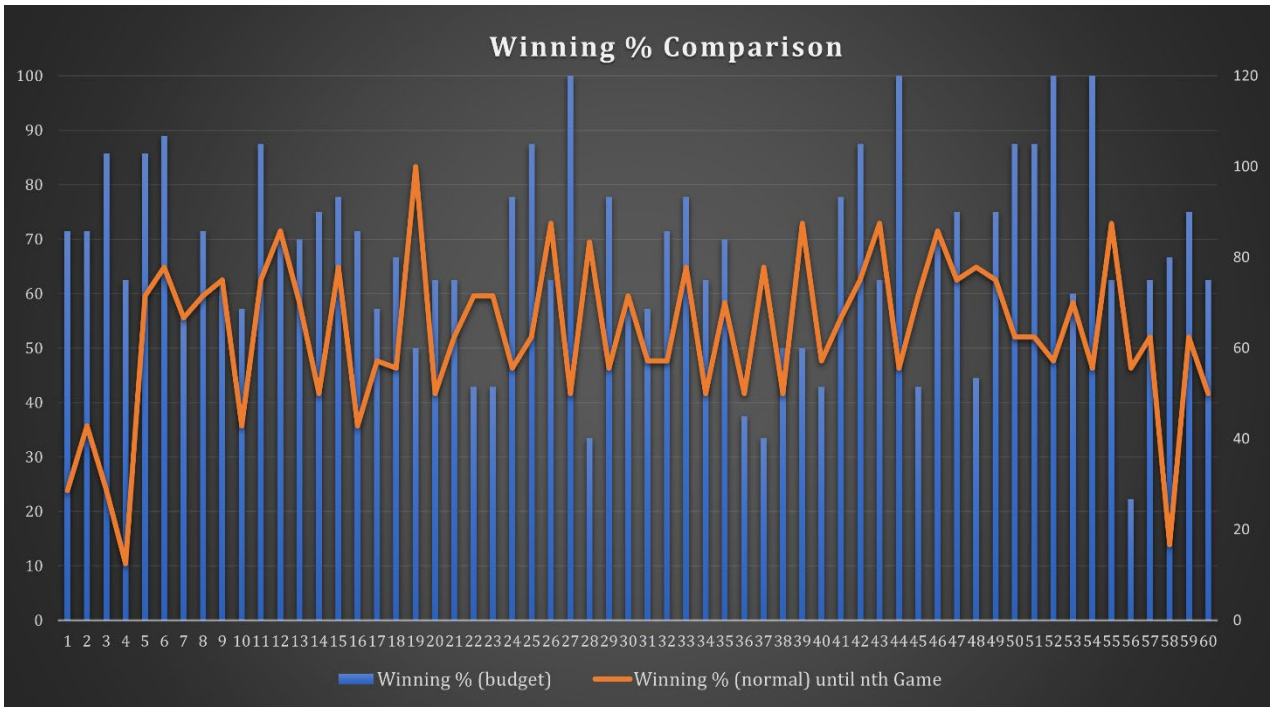

Diagram 4:White avatar's winning rate comparison. Budget mode VS Normal Mode until the n^{th} Game

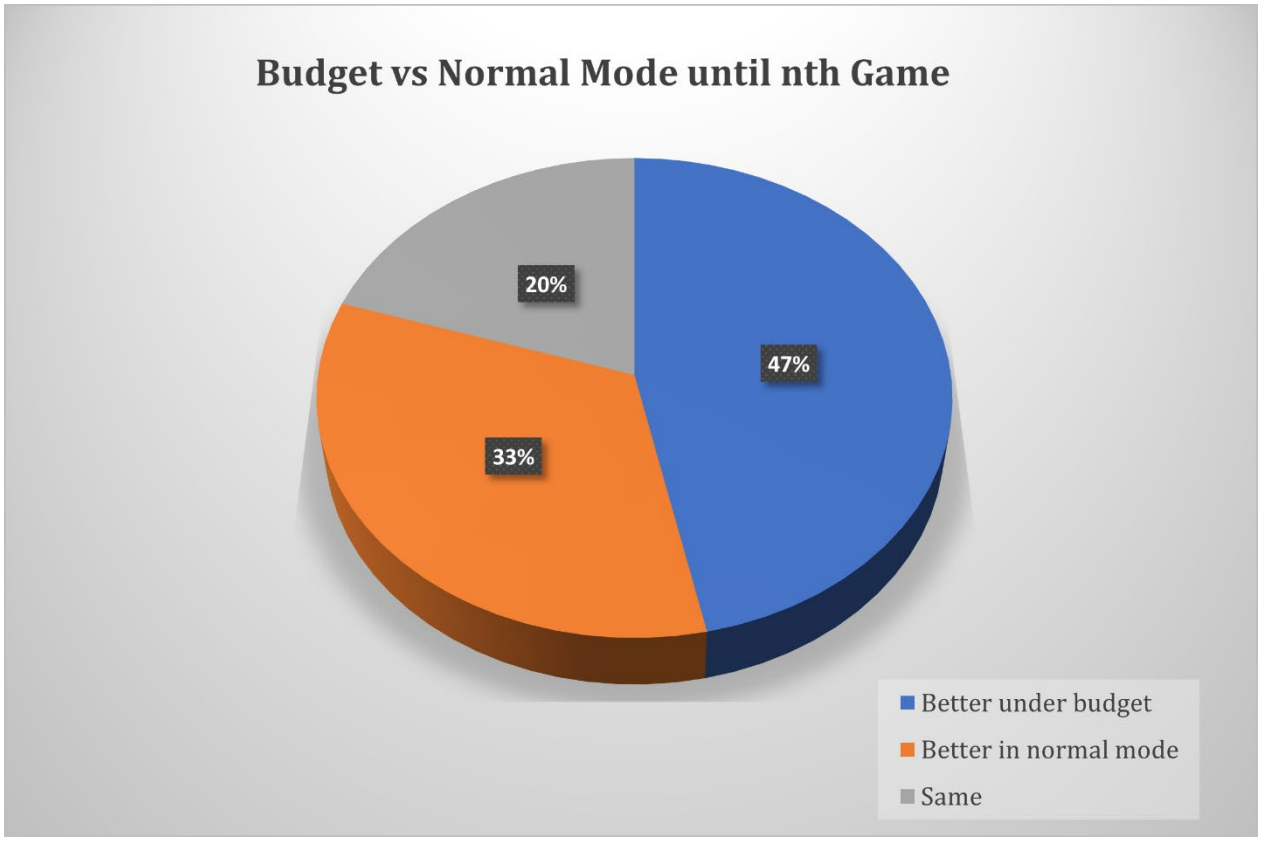

Diagram 5:Budget vs Normal Mode until the n^{th} Game Pie Chart

Chapter 6

Conclusions

The description of a number of scenarios in the RLGame and the implementation of a part of them, gave us a different scope under which the game and the synthetic agents that are participating in it (avatars) could be examined. We have seen that the implementation of resource constraint scenarios, provide a good framework to the researcher in order to investigate whether there are behavioral adaptation signs from the side of the avatars. First with the number of moves constraint we have found a range where a future researcher could try and experiment with several values. Our choice to select the number 15 for our number of moves constraint, provided as with some interesting indication when it was combined with the value of the total budget in a round of games. We saw, that in our experiments, there are small indications of behavioral adaptation, when it comes to spending the budget of total moves. Also, we have seen that as the rounds are increasing and two avatars are playing together, the number of completed moves is slightly increasing when playing under the budget constraint. A future approach could be the application of different values for the number of moves, therefore for the total budget constraint, and examine whether the distributions that we have seen in chapter 5 differ when the characteristics of the players are different. In order to do that, as we have said earlier, the researcher should repeat a very large number of repeated rounds and extract a significant number of data points that could support the confirmation or the rejection of our indications. Also, two different variables could be use in the code in order to represent the white avatar's budget and the black avatar's budget separately. This might lead to even more detailed results that will allow to a more thorough analysis on the behavior of the avatars.

There are a couple of things that we would like to point out when it comes to our experiments with the budget constraint. First, as we have said in the last chapter, that there is an equal

displacement of rates between the three categories, noticing that there is a 53-47 percent displacement between the categories. This can be characterized as normal since the parameters of the game have not change much when the budget constraint is implemented. The avatars are competing in the same environment like in normal mode with the only difference that the now the server stops when the budget runs out. In the future the choice of '*drop*' that has been described in chapter 4, could implemented, meaning that the avatars could choose whether or not they will continue when they take under consideration their budget and the strength of the opponent. At the end of the repeated games, the system could export information, about who chose to drop, against whom, what was the remaining budget and how many games left until the end. These data could allow the researcher to go through a deeper and more thorough analysis about the avatar's behavior. We would like also to point out, that this displacement of the rates, is also a result of the randomness that exists within the RLGame, and it was something that we were already expecting. In the future it could be investigated whether the changes that we have described, provide a more unbalanced displacement.

In the beginning of this dissertation, we have talked about avatars of different experience and characteristics. The ε-γ-λ parameters that are responsible for an avatar's playing style (Fig.4) where not changed in our experiments, keeping a balanced share between these three parameters. These parameters can be easily found in the RLGame code and be changed from the user. Even though we have explained the different possible gaming profiles of the players, the goal of this dissertation was not to implement these profiles, since an investigation of all the different styles, can be a dissertation on its own, but to suggest ideas and future areas of interest.

At last, another suggestion for future investigation is the combination of our suggested board size scenario and the factor of deterministic discontinuity that has been mentioned in section 2.1 and in [Vasilopoulos, 2019: 38].

References

[Badia et al. 2020] Adria Puigdomenech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vtvitskyi, Daniel Guo and Charles Blundel (2020). *Agent 57: Outperforming the Atari Human Benchmark.* arXiv:2003.13350

[Cappart et al. 2020] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Premont-Schwarz and Andre Cire. *Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization.*

[Chen et al., 2008] Serena H. Chen, Anthony J. Jakeman & John P. Norton, (2008). *Artificial Intelligence techniques: An introduction to their use for modelling environmental systems.* Mathematics and Computers in Simulation (MATCOM), Elsevier, vol. 78(2), pages 379-400.

[Dori et al., 2018] Ali Dorri, Salil S. Kanhere & Raja Jurdak, (2018). *Multi-Agent Systems: A survey.*

[Dutta et al. 2007] Partha S. Dutta, Claudia V.Goldman & Nicholas R. Jennings, (2007). *Communicating Effectively in Resource-Constrained Multi-Agent Systems*. IJCAI

[Foerster et al., 2017] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli & Shimon Whiteson, (2017). *Counterfactual Multi-Agent Policy Gradients.* 32nd AAAI Conference on Artificial Intelligence (AAAI-18).

[Franklin & Graesser, 1996] Stan Franklin & Art Gaesser, (1996). *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents.* Proceedings of the 3rd International Workshop on Agent Theories, Architectures and Languages. Springer-Verlag.

[Garcia et al. 2010] Alejandra P. Garcia, Juan Oliver & David Gosch, (2010). *An intelligent agent-based distributed architecture for smart-grid integrated network management.* Local Computer Networks (LCN), 2010 IEEE 35th Conference, pp 1013-1018.

[Georgas, 2012] Apostolos Georgas, (2012). *Συστήματα για παιχνίδια στρατηγικής: Δικτυακές εφαρμογές στο υπολογιστικό πλέγμα.* [Systems for strategy games: Network applications in the computational grid]

[Giantzoglou, 2017] Kyriakos Giantzoglou, (2017). *Εφαρμογή για παιχνίδι στρατηγικής: Ανάπτυξη Οικοσυστημάτων.* [Application for strategy game: Ecosystem development]. Master dissertation, Hellenic Open University, Patra, Greece.

[Guo & Buerger, 2019] Meng Guo & Mathias Buerger, (2019). *Predictive Safety Network for Resource-constrained Multi-agent Systems.* Bosch Center for Artificial Intelligence (BCAI), Renningen, Germany.

[Helleboogh et al., 2007] Alexander Helleboogh, Giuseppe Vizzari, Adelinde Uhrmacher & Fabien Michel, (2007) *Modeling dynamic environments in multi-agent simulation*. Auto Agent Multi-Agent Syst. 14:87-116

[Kalles & Kanellopoulos, 2001] Dimitris Kalles & Panagiotis Kanellopoulos, (2001). *On Verifying Game Designs and Playing Strategies using Reinforcement Learning*. ACM Symposium on Applied Computing, Las Vegas, 2001, pp. 6-11

[Kiourt, 2017] Chairi Kiourt, (2017). *Collective Artificial Intelligence and Personalized Educational Interactions in Learning new Skills.* PhD Dissertation, Hellenic Open University.

[Langley, 2012] Pat Langley (2012). *The Cognitive Systems Paradigm.* Advances in Cognitive Systems 1 3-13.

[Leonardos et al., 2021] Stefanos Leonardos and Georgios Piliouras. *Exploration – Exploitation in Multi-Agent Learning: Catastrophe Theory Meets Game Theory.* (2021). 35th AAAI Conference on Artificial Intelligence. 2-9 February 2021. arXiv:2012.03083

[Nikolaidis, 2014] Spyros Nikolaidis, (2014). *Τεχνητή νοημοσύνη και ενισχυτική μάθηση: Ανάπτυξη παιχνιδιού στρατηγικής ως εφαρμογής πολλαπλών συσκευών.* [A.I. and R.L.: Strategy game development as a multi-device application]. Master dissertation, Hellenic Open University, Patra, Greece.

[Omidshafiei et al. 2019] Shayegan Omidsfhafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M. Czarnecki, Marc Lanctot, Julien Perolat and

Remi Munos. *α-Rank: Multi-Agent Evaluation by Evolution.* (2019). https://www.nature.com/articles/s41598-019-45619-9

[Papageorgiou, 2008] Christina Papageorgiou, (2008). *Συστήματα για παιχνίδια στρατηγικής:Ανάπτυξη εκπαιδευτικού υλικού.*[Systems for strategy games: Training Material Development] Master dissertation, Hellenic Open University, Patra, Greece.

[Rabuzin et al., 2006] Kornelije Rabuzin, Mirko Malekovic & Miroslav Baca, (2006). *A survey of the properties of agents.* University of Zagreb. Journal of information and organizational sciences, Volume 30, Number 1.

[Rowland et al. 2019] Mark Rowloand, Shayegan Omidshafiei, Karl Tuyls, Julien Perolat, Michal Valko, Georgios Piliouras and Remi Munos. *Multiagent Evaluation under Incomplete Information*. (2019) arXiv:1909.09849

[Russel & Norvig, 2010] Stuart Russell & Peter Norvig, (2010). *Artificial intelligence: a modern approach.* 3rd ed. Upper Saddle River, NJ: Prentice Hall.

[Sarantinos, 2015] Panagiotis Sarantinos, (2015). *Ενισχυτική μάθηση σε παιχνίδια στρατηγικής σε γράφους χαρτών.* [Reinforcement learning in strategy games on map graphs]. Master dissertation, Hellenic Open University, Patra, Greece.

[Silver et al., 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Boltov, Yutian Chen, Timothy Lillicrap, Fran Hui, Laurent Sifre, George van den Driessche, Thore Graepel & Demis Hassabis. *Mastering the game of Go without human knowledge.* Article, Nature, 19 October 2017. doi:10.1038/nature24270

[Vasilopoulos, 2019] Georgios Vasilopoulos (2019). *Συνεργαστικοί πράκτορες σε συστήματα για παιχνίδια στρατηγικής.* [Co-operative Multi-agent systems for strategy games]. Master dissertation, Hellenic Open University, Patra, Greece.

[Vlasi, 2008] Athanasia Vlasi, (2008). *Συστήματα για παιχνίγια στρατηγικής*. [Systems for strategy games]. Master dissertation, Hellenic Open University, Patra, Greece.