

UNIVERSITAT DE BARCELONA

FUNDAMENTALS OF DATA SCIENCE MASTER'S THESIS

Model-free video game personalization for educational serious games

Author:

Sergi CEBRIÁN GRES

Tutor:

Dr. Eloi PUERTAS

Supervisors:

Dr. J.A. RODRÍGUEZ-AGUILAR
Dr. Josep Lluis ARCOS

*A thesis submitted in partial fulfillment of the requirements
for the degree of MSc in Fundamentals of Data Science*

in the

Facultat de Matemàtiques i Informàtica

July 3, 2018

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc in Fundamentals of Data Science

Model-free video game personalization for educational serious games

by Sergi CEBRIÁN GRES

This MSc final project focuses on the problem of reconciling the fun aspect of an educational serious video game with its learning objective. For this, we propose bringing video game personalization to educational serious games in order to help players achieve their learning objectives by modifying the game environment dynamically in response to the player's behavior. Most applications of video game personalization, however, require detailed models of the player using a lot of information that is often not practical or even possible to get from them.

Despite the fact that reinforcement learning in the field of video games has mainly been used for playing, we believe that it is a valuable tool which we will use to achieve an adaptive environment without needing a model of the player. Specifically, we will use the Q-Learning technique to train an AI to help simulated players with simple behaviors fulfill learning objectives in a simplified version of the game.

With a virtual reality archaeological educational serious game as our case study, we collected a lot of data and obtained insight into this problem. First, this project studies this data to then try to clearly formalize the problem and decide an appropriate approach. We will also use this data to cluster our players into four types to use for the simulations of different players.

Lastly, we turn our attention in a way to evaluate the policies obtained from training and visualize them to better understand what was learned. We will also propose the best course of action when we do not yet know what type of player we are dealing with in a particular playthrough.

Acknowledgements

I would like to thank both of my supervisors, Juan Antonio Rodriguez-Aguilar and Josep Lluís Arcos, for their great help, their continuous support, and for teaching me so much.

I would like to thanks my tutor, Eloi Puertas, for his endless encouragement and sharing his knowledge of how this MSc values its theses.

I would also like to thank the people at the Archaeological department at the UAB for all their tough work in our serious game and for being overall amazing and kind.

Finally, I would like to thank my friends and family for putting up with me over the last few months, which I know was not always easy.

Thank you, sincerely.

Contents

Abstract	iii
Acknowledgements	v
1 Introduction	1
1.1 Motivation	1
1.2 Case study: Archaeological serious game	2
1.2.1 La Draga	3
1.2.2 La Draga VR: A serious game	3
1.2.3 Collected gameplay data analysis	4
1.3 Problem definition	5
1.4 Approach	6
1.5 Contributions	7
1.6 Guide to the document	8
2 Background	9
2.1 Reinforcement learning	9
2.1.1 Model: Markov Decision Process and rewards	9
2.1.2 Q-Learning	9
2.1.3 Q-Learning example	10
2.2 Automatic algorithm configuration	13
2.2.1 Itrace	13
3 Model free reinforcement learning for AESG	15
3.1 Simplification and discretization	15
3.1.1 Game world simplification	15
3.1.2 Game entities	16
3.1.3 Time discretization	16
3.2 AESG problem as an MDP	16
3.2.1 AI guide's state and action spaces	16
3.2.2 Rewards	17
3.2.3 Markov property	17
4 Empirical evaluation	19
4.1 Experimental setup	19
4.1.1 Learning scenario generation	19
4.1.2 Player behaviors	19
4.1.3 Hyperparameters	20
4.2 Training results and evaluation	21
4.2.1 Examples and visualization of policies	21
4.2.2 Evaluation criteria	23
4.2.3 Evaluation	24
4.2.4 Relative evaluations	25

5 Conclusions and future work	27
5.1 Summary	27
5.1.1 Problem and approach	27
5.1.2 Assumptions and simplifications	27
5.1.3 Results	28
5.2 Future work	28
5.2.1 On player behaviors and guide actions	28
5.2.2 On reinforcement learning techniques	29
5.2.3 On human player testing	29
A Links of interest	31
A.1 Github repository	31
A.2 Videos of la Draga	31
A.3 Downloads	31
Bibliography	33

Chapter 1

Introduction

Serious video games are an incredible tool for teaching, since they make the learning process fun. To define them and other educational media, the term *Edutainment* was coined, neologism blending *education* and *entertainment*, since they aim to educate through entertaining. Artificial Intelligence has made great strides in video game studies, and we believe it can be used to great effect in making these serious games better and adaptable to the user's needs, which is the main goal of this project.

In this chapter, we will first explain our motivation for this project by explaining the state of AI in video games and serious games, and pose our main research question. We will then present the case study for this thesis, an archaeological serious game set in La Draga, Banyoles (Spain). Next, we will define and name our problem, and then, describe our approach to solve it. Finally, we will discuss our contributions made in this project.

1.1 Motivation

Video games have always been an interesting field for Artificial Intelligence research [1]. The fact that they provide a virtual environment with concrete and controlled rules offers a lot of practical advantages to design and test machine learning algorithms. Since the first appearance of video games, AI research in this field has been focused on two main areas:

- Game Playing AIs: There has been a lot of progress in making AIs capable of learning to play games at human level or above. From the world-renowned, yet game specific, Deepmind and AlphaGo [2] to AIs tasked to be able to play any given video game, known as General Game Playing (GGP) [3]. These AIs can be used to provide an adversary for the player, like in chess or Real Time Strategy games, or simply to test the potency of a Machine Learning algorithm.
- Procedural Content Generation: The ability to automatically and procedurally generate content (e.g. levels, environments, and even enemies or weapons) [4] is incredibly valuable for video games. They can cut costs and speed up development time while also providing replay value since every playthrough is in some way unique.

Another less known field that has been taking off recently is the research in video game personalization. In what follows, we will give a general understanding of this field of research and it will serve to better understand our project's main goal.

Video game personalization has as its goal providing a tailor-made experience depending on the player's taste, skill or behavior [5]. This, however, can only be

based on a prediction of what the player actually wants unless explicitly told. The problem of generating such predictions is called *Player Modeling*, the study of computational models of players in games [6]. There are a lot of different ways to model video game users (e.g. personality, preferences, experience, performance, in-game behavior) and which way to choose depends on the available information and intended goal. The applications of video game personalization mostly depend on what player information is used to model them. As an example, modeling by performance is usually chosen to personalize difficulty, number of enemies, frequency of jumps, etc. However, as seen in [6], player modeling needs either explicit information either by filling out forms, or by long enough gameplay sessions to observe the players in, sometimes even with sensors attached to gather information such as heart rate or perspiration. As we will explain in our case study (section 1.2), we worked on a serious game with short gameplay sessions and in an environment that did not favor filling out forms. This contexts made it so that we would like to avoid these problems with player modeling completely and thus propose a model-free version.

A serious video game is a video game designed for a main purpose other than entertainment. There are serious games with many different objectives, such as teaching and professional training tools, city planning, engineering, etc. [7]. However, serious games are still games and, thus, must be *fun*. When designing serious games, then, one has to tread a fine line between focusing on this main educational purpose and ending up with a boring experience for the player, and focusing on fun and letting the actual serious goal fade to the background. This makes it clear that, in an educational serious game, which is the main interest in this MSc, players and designers have different objectives: the player's objective is basically to have *fun* and the designer's objective for the player is for him or her to learn. From now on, we will call these the *player's objective* and the *learning objective* respectively, even though it would still hold for non-educational serious games.

Given how the player's objectives and learning objectives might not always be aligned, this project's main research challenge is how to reconcile player and learning objectives in educational serious games using video game personalization foregoing player modeling.

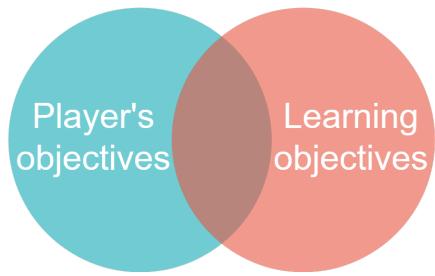


FIGURE 1.1: Player's goals and learning objectives might not always be aligned.

1.2 Case study: Archaeological serious game

Our motivation for this project came in great part from our previous experience designing and testing a serious game. This was a virtual reality serious game developed in the context of the Recercaixa project "*The Digital Reconstruction of the Prehistoric Past: Virtual Reality and Artificial Intelligence for understanding social life in the Neolithic*" from February 2016 to February 2018 and involved a multidisciplinary team of archaeologists, designers, and programmers. It was meant to be part of a museum exhibition and was therefore designed to be suitable for all audiences. This game became our main case study.

This section will first present the neolithic settlement in which the game takes place and explain its significance which is directly tied to the learning objectives. Afterwards, we will present the VR serious game with its objectives and mechanics. We will end the section with an analysis of the gameplay data obtained during showings of our game at different events and the player behaviors observed to show how different types of players tackle the game.

1.2.1 La Draga

La Draga is a neolithic deposit dated from about 7000 years ago [8]. It was first uncovered in April 1990 and has been worked on ever since. A big part of the deposit is actually found buried underwater in the lake near the Spanish municipality of Banyoles. The settlement is very significant, as it is one of the oldest stemming from an agricultural and cattle raising people in the Iberian peninsula. One of the advantages of it being underwater, is that most of the wooden components of tools and architecture, together with other organic materials, have been almost completely preserved giving amazing insight as to their functionality.

1.2.2 La Draga VR: A serious game

The serious game designed for la Draga had the learning objectives of classifying different objects between *neolithic* and *non-neolithic* and visiting some of the most archaeologically significant landmarks reconstructed in a 3D virtual reality environment. To this end, the story of the game was the following:

The player is a time traveler that has stumbled in la Draga but the portal she came through malfunctioned and spewed out some objects belonging to other time periods. Now she has to find all the objects that do not belong to the neolithic and throw them back into the portal before it closes.

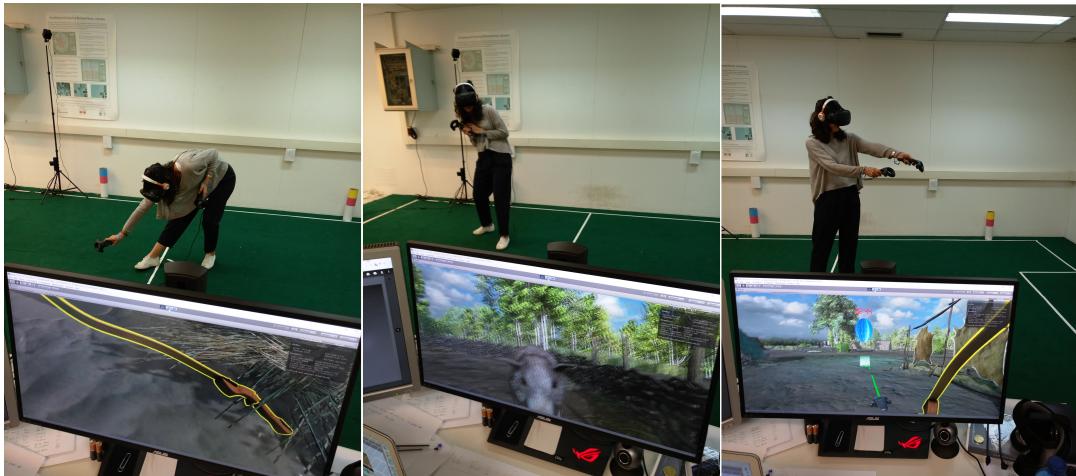


FIGURE 1.2: Pictures of gameplay. The user picks up a non-neolithic bow (left), faces a boar (middle), and decides to bring the bow to the time portal (right)

This story defines an explicit objective that matches the learning objective, and doesn't focus on the exploration of points of interest (implicit in the search). To complete her objective the player has to:

- Teleport to move around the space (standard practice for VR movement).
- Find interactable objects (highlighted in yellow) scattered across the level.
- Grab interactable objects and inspect them.
- Decide whether or not they belong to the neolithic.
- If she decides they do not, throw them in the portal.

When throwing an object in, the player gets feedback from the portal indicating whether or not it was correctly classified. She also has knowledge of how much time she has left to explore.

1.2.3 Collected gameplay data analysis



FIGURE 1.3: Heat map of all players' collected gameplay. Each red dot represents a teleport destination.

The game has since been shown to the public at conventions such as El Saló de l'Ensenyament, the opening of La Draga exhibition at the Archaeological Museum of Catalonia, or el Festival Ciència de Barcelona. These settings explain the reason why a time limit of 3 minutes and 30 seconds was imposed: to avoid bottlenecking and allow every visitor the same playtime. At every showing, gameplay data was collected to analyze later. This data includes number and position of every teleport, objects grabbed and number of mistakes, time spent actively looking around. Our analysis will be looking into general trends and correlations in the data as well as study the spatial distribution of teleports which will give an understanding of which points of interest are popular and which are mostly ignored (figure 1.3). In this analysis we noticed a few interesting things.

First, there exists a huge spectrum of players in both skill and adaptability to the VR environment. Since most people still haven't tried out virtual reality, the period of adaptation to be comfortable with the technology may, for some players, be longer than the in-game timer, even though we provided a short tutorial for the controls and objectives. Figure 1.4 shows the huge differences that exist between players when plotting their final score and teleport count. When applying some simple clustering techniques, we can first observe two clusters of well performing players (blue and

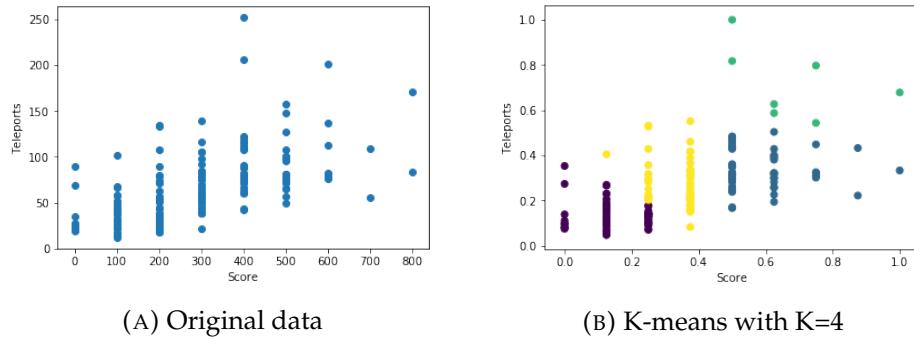


FIGURE 1.4: Plot of all finished games played by number of teleports, which shows activity, and score, which shows proficiency.

turquoise in figure 1.4), and two clusters of badly performing players (purple and yellow). Turquoise players are more active and teleport around much more than the rest of players but are, as a group, less efficient (have to teleport more to find the same amount of objects) than blue players, who get very high scores without teleporting too much. Yellow and specially purple players are much more inactive and almost do not interact with the environment and therefore get a low score and explore very little of the environment. Also, we noticed that even the most skillful players usually didn't visit the entirety of the game level and missed out on some things that archaeologists wanted them to see and focus on, such as some cabins and other buildings.

In conclusion, we can see that the player's objective of finding objects and having fun and the designed learning objective of visiting the points of interest in the settlement are not completely aligned. How to get the users to visit them, in turn, is what motivated this project.

1.3 Problem definition

We have seen that the player's objective and our learning objective are not completely aligned. Let us focus on the learning objective consisting in visiting the points of interest in the level. In a game, the player's main goals are her explicit objectives (in our case study, to classify objects between neolithic and non-neolithic), which is the only way to score points, and to have fun. There's no real incentive to achieve the implicit learning objective (to visit points of interest) but we want to encourage exploration in the player. We want to find a way to align these objectives. However, the player's preferences and skill level, which are unknown to us, also affect their in-game behavior and difficult doing this manually. Having it be automated also has the advantage of allowing for a different experience every game.

With this in mind, we want an algorithm that can reconcile these two objectives, getting the player to visit the most points of interest before the timer runs out. We are working under the assumptions:

- We have the ability to modify the environment during gameplay.
- This problem is constrained in time by the in-game timer.
- We do not have a model of any player. Our only knowledge of each player, thus, comes from observations of her actions and their effects.

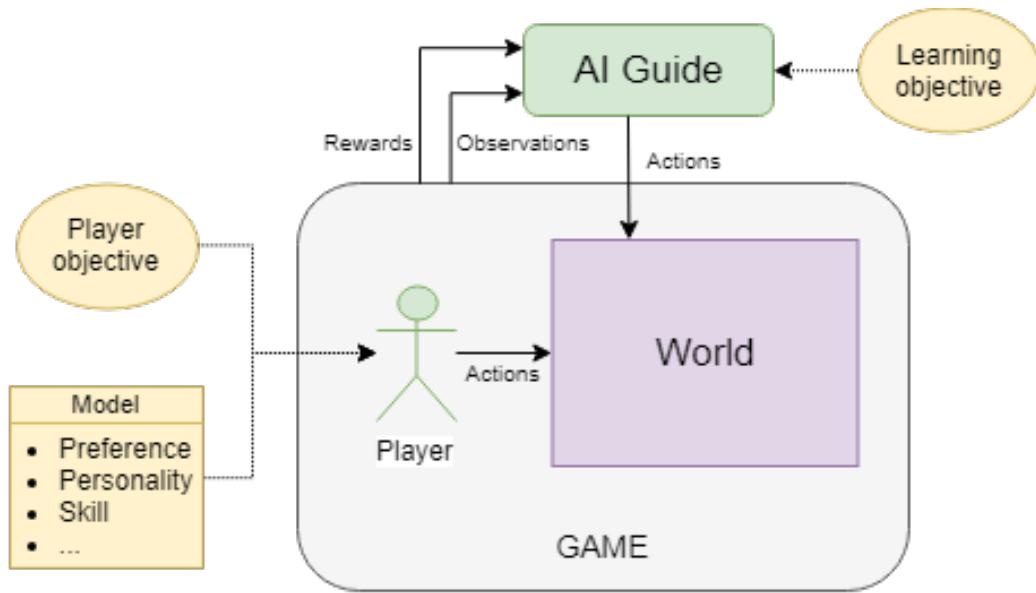


FIGURE 1.5: Diagram of our approach for our problem and its components. The AI guide observes the player, the world, and how they interact and then decides the best course of action.

This problem can thus be seen as an optimization problem consisting in finding the best sequence of actions that modify the environment to maximize the objective function, i.e. number of visited points of interest. We will call this problem *Adaptive Environment for Serious Games*, or *AESG*. Since the algorithm chooses a way of modifying the environment depending on the player’s actions and her general in-game behavior, the AESG problem is a type of *video game personalization* classifying solely based on *in-game behavior* [5].

1.4 Approach

In this section we are going to present this project’s approach to solve the AESG problem. The assumptions we have made in the problem definition allow us to describe our problem with the following components:

- **The game world:** This is the playing environment. It is populated with static objects, some moving objects, and the player. The game world also has a few points of interest we’d like our players to visit that represent the learning objective.
- **The player:** She interacts with it freely within the limitations of her available actions (e.g. teleporting). The player also has as an objective finding and classifying objects, differing from the learning objective of visiting all points of interest. We do not have a model for this player; things like player preferences, personality, skill level, etc. are not known and we also do not have sensors for more explicit information (e.g. heart rate, perspiration). For our purposes, the only information we can use comes from the player’s actions and their results in the world.

To solve the AESG problem, above this structure we will add **The AI guide:** This is an AI that observes both the player and game world and modifies the game world following our desired algorithm. This guide has as its goal that the player reaches

her learning objective (See figure 1.5). The actions that our AI guide takes should happen in a different time frame than the actions taken by the player. This is due to the fact that we shouldn't be modifying the game world every game frame but rather in response to some player event or every t seconds. Defining specific action times for our AI guide allows us to discretize our algorithm in time.

The main problem of not having a model for the player is not knowing how she is going to react to these AI guide's actions modifying the game world. However, we can observe those reactions, see if they get the player closer to the implicit learning objective (visit points of interest), and learn from these observations and subsequently adapt. In this sense, we can try to use some model-free reinforcement learning algorithm on the AI guide to teach it to solve our AESG problem. This algorithm training the AI guide will depend on its knowledge of the state of the game, mainly on the number of the points of interest left for the player to visit and their position relative to the player's. The training goal for the AI will be deciding which is the best action to take at any given state, known as a *policy*.

Our proposed reinforcement learning technique to learn this policy will be *Q-learning*. We have chosen *Q-learning*, explained in detail in section 2.1.2, for its simplicity of implementation and because it is guaranteed to converge on an optimal policy for any given finite Markov Decision Process, a standard model for reinforcement learning problems which we will define and explain in detail in the next chapter.

Our goals, then, are to find a way to model the AESG problem as a Markov Decision Process and train this AI in a controlled environment with simulated players. Our approach is to simulate different player behaviors, with increasingly erratic and unpredictable movement, based on the ones we observed in our real players. To simplify this work and help with discretization of the problem, we opted to train in a 2D grid-like representation of the game (see section 3.1.1).

1.5 Contributions

With this MSc project we made four main contributions:

- **On the analysis of behaviors of players in a VR serious game:** We gathered information from over 180 playthroughs across a few public showings of our game. We used this information to visualize the problem of some points of interest being generally ignored. We also managed to cluster users based on their proficiency and efficiency. We used this results in order to model simulated players to generate artificial playthroughs.
- **On the definition and formalization of the AESG problem:** We coined the term Adaptive Environment for Serious Games, or AESG, to define our goal of having an adapting game scenario that conducts the player towards the learning objective. We also formalized this problem with assumptions and its components.
- **On the modeling of the AESG problem as a reinforcement learning problem:** We showed how our AESG problem can be modeled as a Markov Decision Process stochastic over player behavior. A few optimal policies were then derived from repeated games with different initial conditions using the Q-learning algorithm.

- **On an empirical evaluation method for the results obtained from the Q-learning algorithm:** We proposed an empiric way to evaluate how well our policies work. We also compared them against each other and drew conclusions from these results.

1.6 Guide to the document

The remainder of this MSc thesis is organized as follows:

- **Chapter 2:** We provide theoretical background for reinforcement learning and, specifically, Q-Learning. We also present and explain the *irace* package.
- **Chapter 3:** We describe the steps to model the AESG problem as a reinforcement learning problem. We define the state and action spaces to be used in our experiment.
- **Chapter 4:** We apply those steps and run our experiment. We visualize the learned policies and evaluate their performance.
- **Chapter 5:** We conclude the thesis by providing a summary of our contributions and outlining possible future lines of research.

Chapter 2

Background

In this chapter we will provide the necessary information to understand this project. We will give a short technical overview of Reinforcement Learning and how it models machine learning problems. Afterwards, we will examine the main RL algorithm we used Q-learning, which is one of the most prominent RL techniques for Finite Markov Decision Processes, which we will also define. Finally, we will explain the hyperparameter selection algorithm *irace* we used to fine-tune the learning parameters for Q-Learning.

2.1 Reinforcement learning

Reinforcement Learning, or RL, is an area in the field of machine learning with the goal to find the actions that agents should take in a given environment to maximize some reward. We will first define how reinforcement learning models the decision-making problems and then define and exemplify the RL technique we will be using in this MSc thesis, Q-Learning.

2.1.1 Model: Markov Decision Process and rewards

Reinforcement learning models the agent's environment as a Markov decision process (MDP). MDPs are defined with the following elements:

- S : A set of environment and agent states.
- A : A set of actions of the agent.
- $P_a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$: The probability of getting to state s' from state s while performing action a . This distribution follows the Markov property, which means that this probability distribution does not depend on previous states or on the current time-step t .

To this model, RL adds a function $R_a(s, s')$ that returns the immediate reward after getting to the state s' from s , performing action a . There has to be an *interpreter* which translates the results of an action by looking at the environment and feeds the agent with both the reward earned and its new state. As such, an RL MDP can be defined as the 4-tuple (S, A, P, R) .

2.1.2 Q-Learning

For most RL decision-making problems, the objective is to decide which action $a \in A$ is the best to take given a state $s \in S$. Functions that map state space to action space are commonly known as policies, defined as functions $\pi: S \rightarrow A, s \mapsto \pi(s)$.

Q-learning is a reinforcement learning technique that converges on the optimal policy for any finite Markov Decision Process, or FMDP, without needing its transition model. The optimal policy is the one with maximum expected reward at the end of a game. Q-learning was invented by Christopher J. C. H. Watkins in his thesis [9] who later also proved its convergence to an optimal solution in [10]. The technique gets its name from the function $Q: S \times A \rightarrow \mathbb{R}$, $(s, a) \mapsto Q(s, a) = r_{s,a}$ that maps every state-action pair to its expected reward value, which it learns through repeated playthroughs. This function Q is sometimes referred to as a *Q-matrix* with rows being states and columns being actions, storing the Q-values. The algorithm follows these steps:

Algorithm 1 Q-Learning

```

1: Initialize  $Q, s$ 
2: loop
3:   Choose action  $a$  based on  $s$  and  $Q$  and an exploration strategy (e.g.  $\epsilon$ -greedy).
4:   Use action  $a$  and observe  $r, s'$ .
5:    $a^* \leftarrow \arg \max_a Q(s', a)$ 
6:    $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$ 
7:    $Q(s, a) \leftarrow Q(s, a) + \alpha \delta$ 
8:    $s \leftarrow s'$ 
```

This basic Q-learning algorithm is based on a simple value iteration update and depends on two hyperparameters:

- **Learning rate** $\alpha \in (0, 1)$: The rate at which Q-values are updated. The lower this value is, the more resilient to change the values are.
- **Discount factor** $\gamma \in [0, 1]$: The factor that defines how far into the future we look for rewards. A $\gamma = 0$ implies a myopic agent that only cares about immediate rewards and a higher γ means that the agent also takes into account future rewards.

As for exploration strategies, a lot of work has been put in finding the best ones since it directly affects the convergence time and can affect finding optimal solutions [11]. To these hyperparameters, if we use the exploration strategy ϵ -greedy, we have to add the **Exploration rate** $\epsilon \in [0, 1]$. This is the probability of choosing a random action other than the one with maximum expected reward, which is picked with probability $1 - \epsilon$. Without this exploration we might be foregoing a very high reward action with a low probability associated.

2.1.3 Q-Learning example

This will be a theoretical example of Q-Learning with no actual computation behind. Let's say we want to teach a robot to navigate a set 2D maze (figure 2.1). The robot has only knowledge of its position and, therefore, the set of all possible positions for the robot represents its state space. The robot has 4 available actions: Moving up, left, right and down. However, the robot does not always walk in the desired direction; 80% of the time he goes in that direction, 10% of the time he walks in the direction immediately left, and 10% of the time he walks in the direction immediately right. This introduces some stochasticity in the process (figure 2.2).

To begin formulating the problem, we start by giving every state a name, from 1 to 8 and we call the bad end-state X and the goal G. We keep in mind that

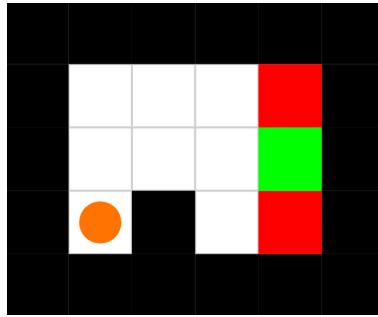


FIGURE 2.1: This is a 3x4 maze. The agent starts in the orange circle. Black tiles are impassable, red tiles are lava pits, and the green tile is the goal

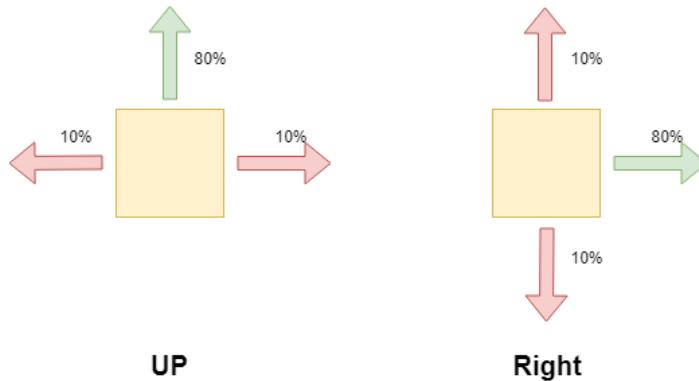


FIGURE 2.2: Examples of the expected behavior for the robot's actions "moving UP" and "moving RIGHT".

the robot starts at state 1. We have: $S = \{1, 2, 3, 4, 5, 6, 7, 8, X, G\}$ and $A = \{UP, RIGHT, LEFT, DOWN\}$. The probability distribution $P_a(s, s')$ marking transitions between states given an action a is given by this walking behavior just described and the map itself, but it is not used (nor is it learned) by the robot.

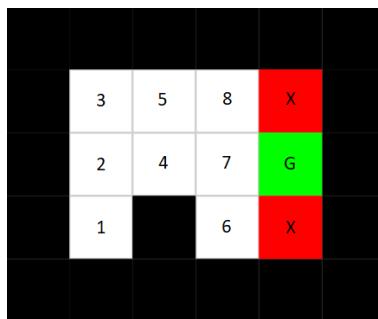


FIGURE 2.3: Names of states.

We do need, however, to define rewards for each transition. The transitions $(8 \rightarrow X)$ and $(6 \rightarrow X)$ both will have a negative reward $r = -1$, while the transition $(7 \rightarrow G)$ will have a reward of $r = 1$. We then define the Q-matrix (8×4) of eight possible states by four possible actions and initialize it with zeros and start the iterated learning process:

The robot will start at state 1 with no idea of what that means, so it will simply pick random actions until, eventually, it gets to either state G or state X . Then it will

be either positively or negatively rewarded. Let's say we are using the hyperparameters $\alpha = 0.5$; and $\gamma = 0.5$ and the robot just got to state G from state 7 with the action *RIGHT*; The value of $Q(7, \text{RIGHT})$ will be updated as follows:

$$Q(7, \text{RIGHT}) \leftarrow Q(7, \text{RIGHT}) + \alpha(1 + \gamma Q(G, a^*) - Q(7, \text{RIGHT}))$$

$$Q(7, \text{RIGHT}) \leftarrow 0 + 0.5 \times (1 + 0.5 \times 0 - 0) = 0.5$$

Since $Q(G, a^*) = 0$ because there are no actions to be taken from end-state G .

After this initial game, the robot already knows *something*, and this knowledge will spread. Since state-action pairs that bring the robot to state 7 will now also get some of its Q-value, discounted by a factor γ . After many games, the robot will have tried all possible state-action pair multiple times and keep good track of their expected rewards. Figure 2.4 shows a graphic visualization of a possible resulting Q-matrix:

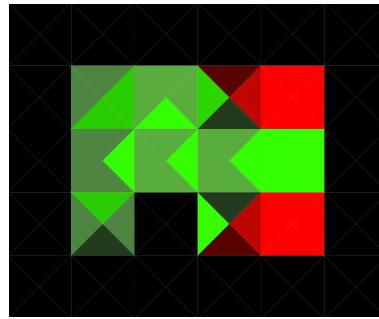


FIGURE 2.4: Each square (state) has the four possible actions drawn inside. Green/red represent positive/negative Q-matrix values. The more intense the color, the higher the magnitude of the value.

After hundreds of games, the Q-matrix can be used to get a policy π by simply picking for every state s , $\pi(s) = \arg \max_a Q(s, a)$, i.e. the action with the highest value in that particular state. The optimal policy given these rewards would be:

s	$\pi(s)$
1	UP
2	RIGHT
3	DOWN
4	RIGHT
5	DOWN
6	LEFT
7	RIGHT
8	LEFT

At the beginning of this simple example, the robot did not know its goal, what the level layout was, or even the results of its actions. With only the rewards and its observations of the world in the form of states, he is able to successfully navigate the maze. If one looks at states 6 and 8, it might seem strange that the robot has learned to value higher the action *LEFT* than the action *UP* directly towards state 7. Remember, however, that the robot sometimes misses his moving action, so trying to get to state 7 would occasionally kill the robot. Therefore, walking left is actually the safest action to *sometimes* get to state 7. However, different initial rewards, like giving all non-ending transitions a reward of $r = -0.2$, would make the robot take

more risks, since by staying alive it racks up negative rewards. This type of reward schema are used to ensure that the robot finds a fast solution.

2.2 Automatic algorithm configuration

Most modern optimization algorithms require setting up a large number of hyperparameters to optimize their performance. The goal of the field of automatic algorithm configuration is to find, automatically, the best distribution of parameters for an optimizer. The *irace* package is a software package that tackles this problem. [12]

2.2.1 Irace

The main goal of the *irace* package is to find the best configuration of setup parameters for a given optimization algorithm and does so by racing different candidate configurations and after some races, eliminating the worst performers and picking new contestants from an actualized distribution. When done, *irace* returns the top 3 performing configurations of parameters and their respective importance. To work, the package requires the following elements:

- **Parameter space:** We define the space of parameters that we want to optimize. *Irace* allows for four types of parameters: Categorical (c), ordinal (o), integer (i) and real (r). These parameters must also have a specifically defined domain.
- **Instances:** The racing scenarios. This may include subsets of data, level layouts, or parameters for automatic instance creation. These must be runnable in a significantly shorter time than the main problem to solve.
- A **target runner:** A software program that, given an instance and a set of parameters, returns an evaluation in the form of *cost*. The evaluation criteria can be any function and can depend on the time spent, on accuracy, or on any other measures of interest for the given algorithm.

Irace is an R package for R, available for download at [CRAN](#).

Chapter 3

Model free reinforcement learning for AESG

In this chapter we will explain the steps to take and simplifications to apply in order to model the AESG problem as a reinforcement learning problem. In particular, we will exemplify it with our case study of la Draga (section 1.2).

In the end, we will have a Markov decision process model with rewards for the problem, following the definition given in the background chapter in section 2.1.1.

3.1 Simplification and discretization

3.1.1 Game world simplification

A 3D game environment, such as the one found in our VR serious game is both harder to work in and slower to render and train in than a 2D environment. To reduce complexity, we have simplified our game level to a 2D grid-like representation of the level. This allows us to discretize both the positions and the actions for modifying the environment. The idea behind this 2D representation of the environment was trying to paint a top-down view of the game level with sprite versions of all points of interest, as seen in figure 3.1.



FIGURE 3.1: La Draga settlement recreated in 2D in the style of old-school role playing games.

As we will see during in the next chapter (section 4.1.1), however, we ended up using other randomly generated learning scenarios which allowed us to both generalize the problem and make resulting policies robust to changes. This game world will be populated by game entities which interact between them and the world itself.

3.1.2 Game entities

Our simplified game world (or level) is an empty 2D grid. A learning scenario, however, is the level and the initial distribution of entities populating it.

The first entity we will talk about is the *player entity*. This is a simulation of a user playing our game, from now on simply called *player*. It interacts with the world and behaves according to some internal logic and its knowledge of the world (we describe this further in section 4.1.2). Its standard available actions are to rotate to look around, walk in 8 directions and picking up *object entities*, from now on *objects*, removing them from the game.

Additionally, we also have to define *goal entities*, or *goals*, which represent the points of interest we would like the player to visit to achieve its learning objective. They are distributed initially around the level and disappear once visited.

Finally, the goal of this experiment is to create the *AI guide* entity. This entity does not occupy a physical space in the level but interacts with it to try to get the *player* to visit the *goals*. This *guide*, then, is the learning agent for our experiment. Which information of the game world it encodes and how is described in the next subsection 3.2.1.

3.1.3 Time discretization

We have also opted to discretize time into small units: *ticks*. Every action taken by the player entity takes exactly one tick to be completed and, at every tick, the player takes an action. This allows us to talk henceforth about durations and times in relation to how many actions has the player done.

3.2 AESG problem as an MDP

Here, we will look at the 4-tuple (S, A, P, R) that is an MDP (section 2.1.1) and see how our AESG problem can be formulated as such. Notice that, as explained in section 2.1.2, Q-learning is a model-free RL technique and, as such, the agent does not need the transition probability distribution $P_a(s, s')$.

3.2.1 AI guide's state and action spaces

In reinforcement learning, the desired output is a policy mapping *states* (or observations) to *actions*. This policy is learned through playing by filling a matrix defining an expected value reward for each State-Action pair. Since, in this experiment, the whole game state depends on the position of the remaining goals, of already placed objects, and of the player, we thought it best to simplify states and actions and frame them both around the player's current position.

We will only consider as an observation the nearest goal to the player, and therefore define a state $s \in S$ as $s := (i, j)$, the 2D vector from the player to this goal.

The action space can be any finite set of actions by which the world is modified. In our experiment, each action represents a position in which to spawn an object. To further simplify the experiment, we only allow some specific positions around the player's position. We have 8 possible directions, relative to the player, to spawn the next object in. They can appear at a distance of 1, 2 or 3 (See figure 3.2). Thus, the action $A = (r, d)$ where $r \in \{0, \dots, 7\}$ (clockwise from east to north-east) and $d \in \{1, 2, 3\}$ represents spawning an object in the direction r at distance d . This limit in distance has been chosen both to limit the action space size and to match the vision range of the player for this experiment (explained later in section 4.1.2).

3.2.2 Rewards

The AI guide only executes an action every 12 ticks. We selected this value because it represents the worst case scenario for the player to find and pick up the spawned object (8 ticks to turn around, plus 3 to move, plus 1 to pick it up). The reward for the last action is determined right before taking the next one. If during the 12 ticks after the guide takes an action, the player goes through a goal, the reward for that state-action transition is of +1; otherwise, the reward is of -0.01. This negative reward guarantees that the guide will prioritize the shortest path to the goal to avoid these punishments from stacking up.

3.2.3 Markov property

Even though we do not need to define probability distribution of transitions P , we need to verify that it follows the Markov property. For this, if we remember that $s_t, s_{t+1} \in S$ are the 2D vectors pointing from the player to the nearest goal at times t and $t+1$ and that $a \in A$ denotes where a new object has been placed, we can clearly see that the distribution $P_a(s_t, s_{t+1})$ does indeed not depend on the previous state history or on the time t .

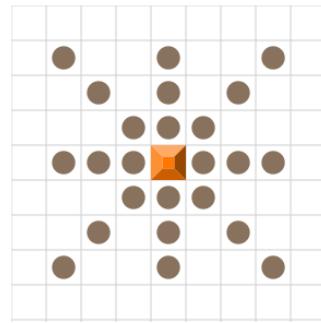


FIGURE 3.2: Action space used in experiment: The brown dots mark all possible object spawning positions relative to the player (orange square).

Chapter 4

Empirical evaluation

In our experiment, our main goal is to figure out a way to modify the playing environment during the game by spawning objects that attract the player, ensuring so that she visits every point of interest in the shortest amount of time. With this objective in mind, we will train an *AI guide* via the Q-Learning algorithm in a simplified 2D grid-like environment.

We will train a different policy for each combination of learning scenario and player personality (see sections 4.1.1 and 4.1.2) for a total of 15 policies. Afterwards, we will show our results, define our evaluation method, and analyze how the policies perform relative to each other.

4.1 Experimental setup

We followed all steps defined in chapter 3 to simplify and discretize our problem. Here, we more closely look at the specific learning scenarios and at the different behaviors for the *player entity* we have trained our guide with. We will also explain our use of irace to select the hyperparameters we used.

4.1.1 Learning scenario generation

As mentioned in 3.1.1 we will experiment in simplified learning scenarios in the form of simple 2D colored grids. For this experiment, we created three different learning scenarios of increasing size and number of points of interest, thus making the state space bigger in each one. The layout of the points of interest and the player's beginning position are randomly chosen to create scenarios following these blueprints (see figure 4.1):

- *Small* layout: A 9 by 9 grid with 4 points of interest
- *Medium* layout: A 20 by 25 grid with 10 points of interest
- *Big* layout: A 40 by 60 grid with 25 points of interest

After being created, they are saved and not changed during the learning phase. We evaluate later with different layouts to see if the learned policies can be exported to different levels or games.

4.1.2 Player behaviors

Here we will discuss the behavior logic of our simulated players. Since the explicit goal of our game's player is to find and classify objects, which also gives them points,

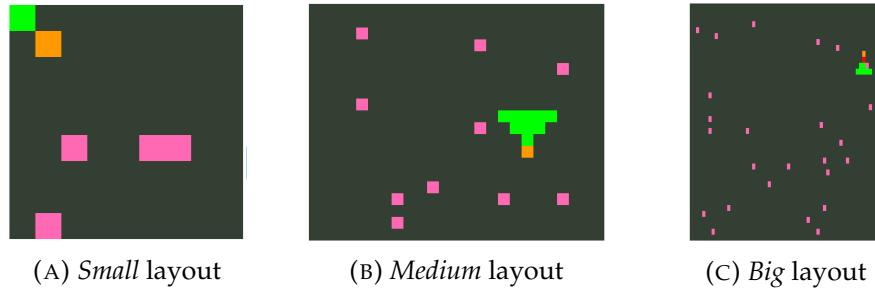


FIGURE 4.1: Examples of randomly generated level layouts.

our player will be attracted to any objects in its field of view (FOV). This FOV is a cone of 90 degrees and has limited vision range of 3 squares (see figure 4.2). This simulates the vision range of a human player in VR looking around for objects.

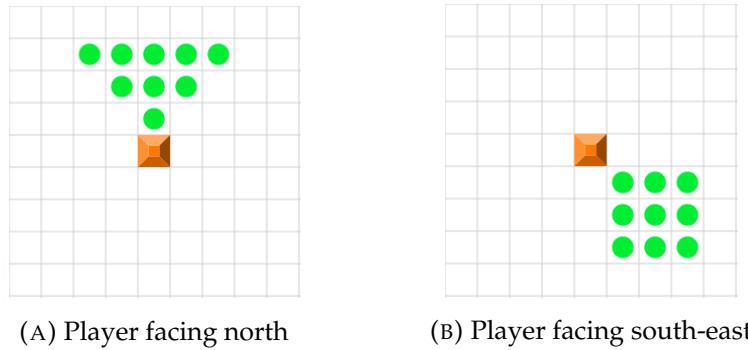


FIGURE 4.2: Field of view examples. Green dots mark cells currently seen by the player (orange square)

Once the player arrives at an object, she picks it up, removing it from the game, and moves on. When not moving towards nor seeing any objects, players engage in an *idle* behavior. Idle behaviors try to model the strategy for different players to find new objects. However, in the context of this experiment we will limit it to a random walk with probabilities of turning around or walking forward.

For this experiment, the simulated players will have one of the following idle behaviors associated $\{(10;0), (9;1), (8;2), (7;3)\}$. At every *tick*, a player with behavior $(X; Y)$ will look around with a probability of $\frac{X}{10}$ and walk forward with a probability of $\frac{Y}{10}$. A low Y value, then, means that the player will mostly look around (rotate) and seldom walk forward, while higher Y values are associated with more erratic behavior. We have limited this list to four behaviors to take into account the different users we have observed at showings of the game (section 1.2.3).

4.1.3 Hyperparameters

As seen in the background chapter (section 2.1.2), there are two basic hyperparameters in Q-Learning: the learning rate (α) and the discount factor (γ). On top of these parameters, one must also choose the exploration strategy. We have picked a static ϵ -greedy strategy. This adds the hyperparameter ϵ to the other two.

Since hyperparameter selection can have a huge impact on convergence time and performance, we have used the irace package for automatic algorithm configuration

(section 2.2.1) to determine how to better initialize our problem. In our case, we used a small executable target runner for the candidates to race in, where an AI trained from scratch during 300 games for the different personalities. After training, it was tested in 10 games for each personality. The loss function is the sum of the means of the number of objects placed before ending these testing games (the shorter the game, the better) for each personality.

	.ID.	exploration	gamma	learn_rate	.PARENT.	.ALIVE.	.RANK.	.WEIGHT.
41	41	0.66	0.52	0.15	6	TRUE	23.0	0.5000000
34	34	0.95	0.46	0.13	29	TRUE	23.0	0.3333333
36	36	0.66	0.55	0.15	6	TRUE	30.5	0.1666667

FIGURE 4.3: Output of the irace execution with three candidate combinations for our parameters α , γ and ϵ

Figure 4.3 show the top 3 configurations with their values and weights. From this irace results, we extract the following initial parameters:

$$\begin{cases} \alpha = 0.15 \\ \gamma = 0.50 \\ \epsilon = 0.75 \end{cases}$$

A few things of note:

- An exploration rate this high means that most of the time we will be trying out new actions or actions we know to be bad. This is necessary since we have an action space of 24 actions and we really do not want to settle for the first one to yield positive rewards.
- To fine-tune the policies, one can have an exploration rate that lowers over time. This allows for a good exploration early on but converges on a policy sooner and explores mainly around it.
- A lot of research has gone into how the learning rate affects convergence [13]. However, in practice, they only affect convergence time and a fixed value is commonly used.

4.2 Training results and evaluation

We have trained five different policies for each learning scenario, one for each of the four behavior types and one picking a random behavior each playthrough (for a total of 15 policies). Each policy trained during 100 thousand games each using the parameters selected at 4.1.3.

4.2.1 Examples and visualization of policies

Once trained, that is once we are done updating the Q-matrix values, whenever we talk about the learned policy, we are talking about the policy $\pi(s) = \arg \max_a Q(s, a)$. We can simulate the result of such a policy by turning off both the learning rate and the exploration rate and rerunning the game. Figure 4.4 shows the path that a player followed while the AI Guide modified the environment following the learned policy for a (10; 0) player in the *medium* scenario layout.

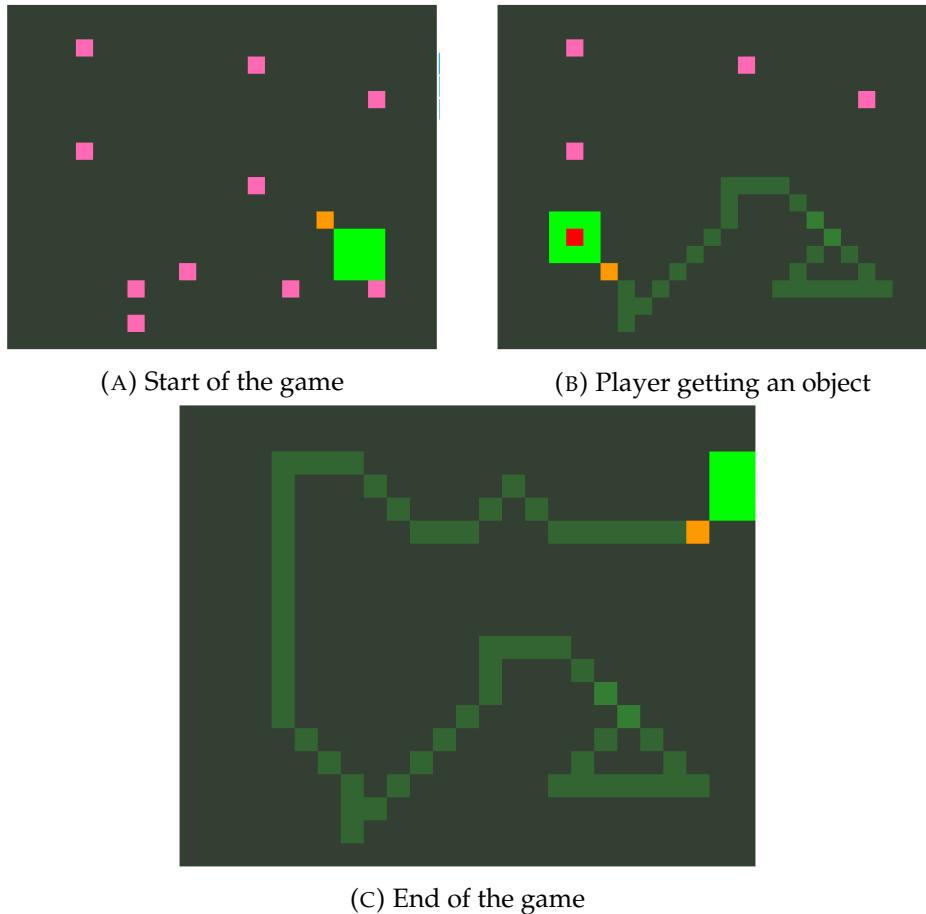


FIGURE 4.4: Gameplay of the policy learned with a well behaved $(10; 0)$ player in a medium sized level.
Color legend: Player, Goal, Player FOV, Object

To help understand the policies learned, figure 4.5 shows 3 different trained policy matrices where the color in each cell (s, a) represents the expected value of action a at state s . To keep the matrix intelligible, we have ordered the state space (the rows) by the angle each state $s = (i, j)$ represents, $\text{atan}2(j, i)$. The 24 actions (the columns) are also ordered first by orientation and then by distance. We do not show any *big* scenario matrices because the state space is too large and, therefore, the image is mostly black.

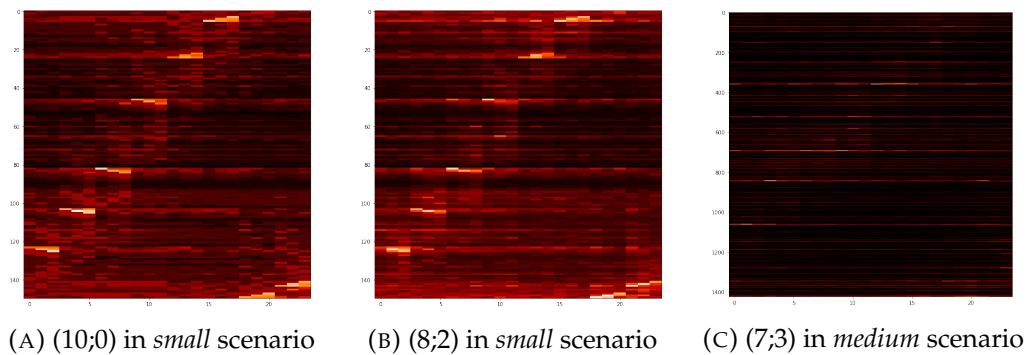


FIGURE 4.5: Q-Matrix visualization for 3 learned policies.

As we can clearly see in the *small* scenarios (also, but less so, in the *medium* one) there is a clear diagonal of very bright squares. These distinctly brighter cells represent the states in which the reward is immediate, that is the states that are 1 object away from getting to a point of interest. The diagonal trend shows that the AI has learned to spawn objects in the general direction of the next goal.

4.2.2 Evaluation criteria

Our AESG problem, in this particular experiment, can also be seen as helping the player to find the best path between all goals. This is a very well known problem: the traveling salesman problem, or TSP [14]. If we define a completely connected graph between all points of interest and weight the edges with the distances between them, the TSP min-distance solution in that graph would equal the least amount of objects we would need to spawn to end the game. These distances, however, have to be interpreted as the quantity of objects that must be placed in order to get from one point of interest to the next, which we will call *object distance*. Knowing the positions relative to the player where objects can be placed (figure 3.2), the object distance is calculated as follows, with $v = p_2 - p_1$ where p_1 and p_2 are the positions of the two points of interest:

$$d_{obj}(p_1, p_2) = \text{ceil}\left(\frac{\min(|v_x|, |v_y|)}{3}\right) + \text{ceil}\left(\frac{|v_x| - |v_y|}{3}\right)$$

where $\text{ceil}\left(\frac{\min(|v_x|, |v_y|)}{3}\right)$ is how many objects should be diagonally placed from p_1 in order to get the player on the same row or column as p_2 , and $\text{ceil}\left(\frac{|v_x| - |v_y|}{3}\right)$ are the objects needed from there to p_2 . Figure 4.6 is an example calculation of object distance between two points of interest p_1 and p_2 :

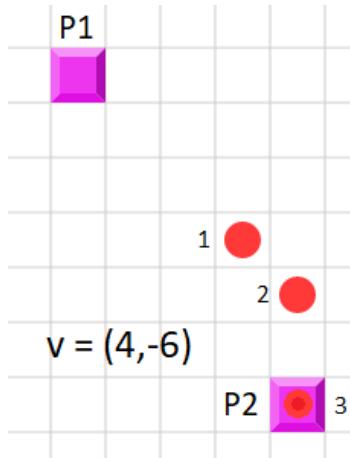


FIGURE 4.6: Example of object distance calculation:
 $d_{obj}(p_1, p_2) = \text{ceil}\left(\frac{\min(4, 6)}{3}\right) + \text{ceil}\left(\frac{|4 - (-6)|}{3}\right) = 2 + 1 = 3$

With this in mind, to evaluate a policy's performance, we will compare its average of objects placed until all points of interest were visited over N games with the result of the TSP algorithm. These comparisons, however, have to be looked at carefully, because TSP has global knowledge of the level layout while our AI only takes into account the closest goal, so it doesn't look for a general path but for a local solution instead. That said, it still serves as a baseline of how good our AI in this

experiment could possibly fare given a more complex state space (see future work section 5.2).

4.2.3 Evaluation

Using the evaluation criteria just presented for our policies we have created the table 4.1 which shows:

- The total distance for the TSP solution in that layout.
- The average over $N = 250$ games of how many objects each policy needed when exploited in their learned scenario with their learned player behavior.
- The increase in percentage from the TSP to the policies' performance. This number will be shown in green in this increase is smaller than 30% and red if it is greater than 100%

In the table, the name for each policy comes from the "*Behavior-Scenario*" they were trained with.

Behavior-Scenario	TSP	Policy	Loss %
10_0-SMALL	6	6	0.000
9_1-SMALL	6	7.352	22.533
8_2-SMALL	6	8.411	40.183
7_3-SMALL	6	10.405	73.417
Random-SMALL	6	7.612	26.867
10_0-MEDIUM	24	22.01	-8.291
9_1-MEDIUM	24	32.224	34.267
8_2-MEDIUM	24	39.573	64.888
7_3-MEDIUM	24	51.647	115.196
Random-MEDIUM	24	36.61	52.541
10_0-BIG	76	83.346	9.666
9_1-BIG	76	123.82	62.921
8_2-BIG	76	178.57	134.961
7_3-BIG	76	206.45	171.645
Random-BIG	76	132.9	74.868

TABLE 4.1: Each row is a policy trained. Columns represent the name of the policy, the TSP minimal distance, the policies performance averaged over 250 games, and the increment in percentage.

In this table, we can appreciate how the more erratic the player's behavior, the worse our policy performs. This is natural since the MDP is stochastic over player's behavior and players more prone to walk aimlessly will often not even see the objects the AI guide spawned. The size of the layout also significantly affects performance, since the state space grows polynomially (order 2) with layout size.

Another interesting thing to point out is how for *10_0-MEDIUM* we get a negative loss, which should not be possible if the TSP solution is correctly calculated. However, watching how the policy behaves we see that it learned to make the player move over some goals without stopping, which still counts as visiting them, in order to cut distance on the following point of interest. This causes the player to end in an advanced position and, from there, one less object is needed to get her to the next goal. TSP simply can not account for that.

4.2.4 Relative evaluations

Every policy was tested playing with the personalities with which they were trained. Here, we are interested in seeing if these guide policies are also useful for players with different personalities. We added a sixth policy for every scenario which comes from averaging all other Q-matrices. We have then evaluated the performance for every combination and saved the results in a table as follow: The value at (row, col) is the percentage of increase in objects placed before ending when testing the policy trained with row played with behavior col , compared to the performance obtained when played with the same behavior. Figure 4.7 shows an example of such a table for the *medium* scenario.

	10_0	9_1	8_2	7_3
10_0	0.000000	4.807001	4.662152	8.534256
9_1	8.834783	0.000000	0.224895	3.169177
8_2	13.147826	2.050320	0.000000	3.026403
7_3	17.621739	1.975309	1.917926	0.000000
Random	13.043478	2.015940	1.379694	7.832452
Average	4.347826	-2.037818	-1.316521	4.474250

FIGURE 4.7: Relative performance of policies tested with every personality for the *medium* scenario.

As was expected, policies work better when exploited with the same behavior they were trained with. That said, we can observe that the policy coming from averaging the Q-matrices learned with every player behavior performs overall very well; so much so that when tested against the behaviors (9;1) and (8;2) it outperforms the policies trained with them. In this sense, if we had a new player and we do not know its behavior, the best policy to follow seems to be the one averaged from all others.

Chapter 5

Conclusions and future work

In this chapter we first summarize the work developed during this MSc final project while drawing the most notable conclusions attained from it. Finally, we outline some lines for future research.

5.1 Summary

5.1.1 Problem and approach

This work revolved around using reinforcement learning techniques to train an AI to solve the problem we named Adaptive Environment for Serious Games, or AESG. The AESG problem consists in helping educational serious video game players achieve their learning objectives by modifying the game environment dynamically. On every showing of the educational serious game we developed over the last 2 years, we collected data which we used in this project in order to define and better understand this problem.

Among all reinforcement learning techniques we opted for Q-Learning for a two main reasons. The first reason was that it is a model-free technique. This means that the AI trained does not need to know how its actions affect the world and the player, but simply observes the results. This was very helpful for us because it allowed us to avoid the problems and caveats that come with player modeling. Secondly, we were attracted by its simplicity of implementation, which allowed us to test the merits of our hypothesis without investing time and effort into more complex RL techniques. Moreover, Q-Learning is guaranteed to converge to an optimal solution for any given finite Markov decision process.

This approach of modeling the problem as an FMDP, stochastic only over the player's behavior, in which we train an AI presents an additional advantage. Once we have a policy that works well with most players, its values can still be updated when facing a new player. Basically, it is based on off-line training to set a baseline and then permits on-line training to better suit any given player.

5.1.2 Assumptions and simplifications

To be able to model the AESG problem as a reinforcement learning problem we had to make a few assumptions. The first assumption may seem evident, but we must be able to change the game's environment. This may not always be true for all educational serious games, in which case our approach could theoretically still be used with other actions with which to *manipulate* the player's behavior. The other important assumption is that we do not have any model for our player; If we had one,

other video game personalization techniques could be used without the need of machine learning. Lastly, we assume that the action and state spaces can be discretized into finite sets. Otherwise, another reinforcement learning technique should be used which could handle non-finite MDPs.

As for simplifications, we first discretized and simplified our continuous 3D game world to three discrete 2D grid-like world of increasing sizes. This permitted both a finite representation of the state and action spaces, and simpler and faster simulations. Another important simplification we applied was in our view of simulated players, where we essentially defined four different types of random walkers with increasingly erratic behavior. They served to have different types of players and to see how this affected performance.

5.1.3 Results

We have seen in our results that indeed it is possible to help the players achieve some learning objective without directly influencing or limiting their actions. We also showed that more erratic player behaviors are more difficult to plan for, but it is still doable, and our AI learned to take fewer risks with them and did not usually spawn objects far from them, for fear they would not be seen.

We have found a way to visualize every policy in the form of heat maps of the learned Q-matrices. These visualizations help understanding exactly what is learned. We can also show an entire simulated playthrough of a game by a simulated player while the AI guides it.

Finally, we proposed a way to evaluate the performance of the learned policies by comparing them to the solution of a Traveling Salesman Problem in the same scenario. We also tested every policy with every possible behavior to know which would be the best to start with if we did not know what kind of player we were facing.

5.2 Future work

5.2.1 On player behaviors and guide actions

In this project, the simulated player behaviors were limited to more or less erratic random walkers attracted to objects. This concept could and should be worked on to allow for more nuanced behaviors more distinct from each other (e.g. explorers who prioritize going to new places). Another option would be, with enough player information and gameplay data, to use Behavioral Cloning or Imitation Learning techniques [15] to used simulated players that behaved in the same way our real users did.

If we added more actions with which to modify the environment and attract players, maybe more intensely or from further away (e.g. a bird flying near the player before resting on top of a place of interest), we could still use the same system but with a larger action space.

5.2.2 On reinforcement learning techniques

Q-Learning has proven to be an efficient and useful technique to achieve our goals. However, since it was first thought of, huge improvements have been done in the field of reinforcement learning since the advent of neural networks. Using for example Deep Reinforcement Learning with double Q-Learning [16], we could avoid the simplification we had to take with our state space (only looking at the closest point of interest), since Deep Q-Networks can look at the whole level and derive a useful state representation. Double Q-learning helps with the intrinsic overestimation issue associated with Q-learning and could be easily implemented [17].

5.2.3 On human player testing

Once shown how our proposed approach is a valid technique for adapting serious game environments in order to help players achieve a given learning objective, we now should test this with actual human players in our 3D game. For this, we would need to take a few steps. First, we would need to discretize the 3D space to maintain a finite Markov Decision Process. Then, we should Train an AI guide off-line with simulated players, as we have done in our experiment. Only then could we begin testing with the learned policy on real players while still training the AI.

Appendix A

Links of interest

In this appendix we post the link to our Github repository page. We also list some links of interest for the reader that wants to learn more about our case study, a serious game based in la Draga, or wants to play it.

A.1 Github repository

Github root repository:

<https://github.com/SergiCebrianGres/Model-free-video-game-personalization-for-educational-serious-games>

A.2 Videos of la Draga

Trailer of la Draga VR video:

<https://youtu.be/vgIadv1620Y>

360 Draga Movie. (Can be controlled with the cursor or can be seen in VR using Google Cardboard):

<https://youtu.be/rLcZOfWcRv8>

Gameplay of the serious game that served as our case study:

<https://youtu.be/p3E15XDIQgE>

A.3 Downloads

Here you can find a two executable downloads for windows. One is our VR serious game and the other one is a desktop version of the same game.

<http://www.iiia.csic.es/draga/downloads.html>

Bibliography

- [1] Yannakakis and Togelius. *Artificial Intelligence and Games*. 2018.
- [2] David Silver. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* (2016).
- [3] Michael Genesereth. "Overview of General Game Playing". In: *Stanford University* (2005).
- [4] Mark Hendrikx. "Procedural Content Generation for Games: A Survey". In: *ACM Transactions on Multimedia Computing Communications and Applications* (2013).
- [5] Stephen Karpinskyj. "Video game personalisation techniques: A comprehensive survey". In: *Entertainment Computing* 5 (2014).
- [6] Georgios Yannakakis. "Player Modeling". In: *ACM Subject Classification* (1998).
- [7] Jean Ecalle. "Serious games as new educational tools: How effective are they? A metaanalysis of recent studies". In: *Journal of Computer Assisted Learning* (2013).
- [8] MAC. *La Revolució Neolítica. La Draga, el poblat dels prodigis*. 2017.
- [9] Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. 1989.
- [10] Christopher J. C. H. Watkins. "Q-Learning". In: *Machine Learning* 8 (3-4 1992).
- [11] Sebastian B. Thrun. *Efficient Exploration In Reinforcement Learning*. Tech. rep. 1992.
- [12] Manuel López-Ibáñez et al. "The irace package: Iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3 (2016), pp. 43–58. ISSN: 2214-7160. DOI: <https://doi.org/10.1016/j.orp.2016.09.002>. URL: <http://www.sciencedirect.com/science/article/pii/S2214716015300270>.
- [13] Eyal Even-Dar and Yishay Mansour. "Learning Rates for Q-learning". In: *J. Mach. Learn. Res.* 5 (Dec. 2004), pp. 1–25. ISSN: 1532-4435. URL: <http://dl.acm.org/citation.cfm?id=1005332.1005333>.
- [14] Gilbert Laporte. "The traveling salesman problem: An overview of exact and approximate algorithms". In: *European Journal of Operational Research* 59.2 (1992), pp. 231 –247. ISSN: 0377-2217. DOI: [https://doi.org/10.1016/0377-2217\(92\)90138-Y](https://doi.org/10.1016/0377-2217(92)90138-Y). URL: <http://www.sciencedirect.com/science/article/pii/037722179290138Y>.
- [15] Alexandre Attia and Sharone Dayan. "Global overview of Imitation Learning". In: (2018).
- [16] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning." In: *AAAI*. Vol. 2. Phoenix, AZ. 2016, p. 5.
- [17] Hado V Hasselt. "Double Q-learning". In: *Advances in Neural Information Processing Systems*. 2010, pp. 2613–2621.