Heriot-Watt University

Robotics Project

Deep Reinforcement Learning Agent for Enercities Game

Students:
Al Arafat
Flávia Dias Casagrande
Marcel Sheeny de Moraes

Subject Professor:
Yvan Petillot
Project supervisors:
Helen Hastie
Srini Janarthanam

A project submitted in fulfilment of the requirements for the subject of Robotics Project

in the

VIBOT - Computer Vision and Robotics Master Programme
Heriot-Watt University

December 17, 2015

ABSTRACT

The purpose of this project is to develop an intelligent robotic tutor to play one out of the three of the roles of a collaborative game called Enercities, which aims the construction of a sustainable city. The robot's agent will be implemented by using Deep Reinforcement Learning (DRL) techniques. Basically, DRL is a set of Reinforcement Learning methods that learns an optimal policy for a learning agent based on high-dimensional sensory data. In this project, the learning agent will be provided with raw data such as the state of the game and conversation. Based on these inputs, the agent should learn to choose optimal actions (both game moves and communicative actions). The agent will interact with random agents (representing each of the other two players), which were also implemented as part of the project. The DRL agent was implemented and several networks were designed (with different number and types of layers, execution time and setting parameters), although none of the results proved the agent learned something.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Deep learning is a branch of Machine learning which evoked increasing amount of interest in research community during recent years. It uses Neural Network with many deep layers and a large amount of dataset to solve computational and perceptual problems and contributes to make better representation and create models to learn features from a large-level unscaled data. Deep learning offers various learning structures and has been applied to different fields. Use of deep learning in Intelligent tutoring systems is one of the possibilities where the use of deep learning can be proved effective.

Intelligent tutoring system has received research attention in recent times given the technological society we live in. As such, systems able to facilitate learning are becoming frequently explored and applied within educational settings.

Reinforcement learning allows an intelligent system (e.g. robotic tutor) to learn from trial-and error interaction with its environment. It helps the system to determine the optimal sequence of actions to take in order to reach a goal by observing the results of its actions (i.e. awarded rewards). Thus, a combination of deep learning and reinforcement learning allows a learning agent to learn an optimal policy to control a system by using a deep neural network which extracts relevant features from the high-dimensional dataset.

Inspired by the possibilities of robotic tutors, we worked with the researchers working on the EMOTE project (an FP7 European-funded project) to develop an intelligent robotic tutor that will deliver geography education to late primary school children.

In this project, we create a Game Player module using Deep Reinforce-

ment Learning techniques for a robot who plays Enercities game with two other random agents or human players. Enercities is a three player game where players collaborate to build a sustainable city. We model a grid map for the locations of the game and a database with all the available structures, upgrades and policies according to the specified role of the agent for different levels of the game. Our learning agent takes raw data such as the current states of the game as input. The states contain the currently available location to build structure, the types of structures, upgrades to the built structures, and policies to implement. The network to be designed takes the current states as input and chooses an optimal action. The action builds a chosen structure at a selected location. Based on the goal, we use Q-learning to reward the selected actions.

We begin our exposition in Section 1, with some of the Related Works. In Section 3, we briefly introduce the theoretical background. Section 4 contains information about the Enercities game. In Section 5, we elaborately discuss about our programming implementation for this project. To this end, in Section 6, we describe the experimental results. In later Section 7, the time schedule toward finishing this project is described. Finally, in Section 8, we conclude the work and illustrate the scope of future work for this project.

2. RELATED WORKS

There are previous successful attempts in using reinforcement learning to derive optimal actions from high-dimensional sensory inputs. In such cases, they faced complexity regarding the representations of the environment from high-dimensional sensory input, and using these to generalize past experience to new situations. They used deep neural networks for training to develop deep Q-network, that learned successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning. They implemented this agent on the challenging domain of classic Atari 2600 games12 [8].

There is almost no studies in applying deep reinforcement learning to Intelligent Tutoring System (ITS). Introducing robots as tutor in ITS is even less common in this research field. Recent researches on socially intelligent robots show robots as partners that collaborate with people [9] and has made the use of robotic platforms in experimental learning more approachable [10]. Also, social supportive behavior in robotic tutors has been shown to have a positive impact on students learning performance [11]. On the other hand, educational games that provide learning content to players in addition to entertainment - have been used successfully for learning purposes [12]. Such games have shown to be very effective in helping students to learn new concepts [13]. Likewise, recent interaction technologies (e.g., multi-touch tables) have also proved to be important pedagogical tools, enhancing collaboration and interest in the learning process.

One previous implementation presents developing dialogue dimensions for a robotic tutor in a collaborative learning scenario grounded in human

studies. Their implementation identified seven dialogue dimensions between the teacher and students interaction from data collected over 10 sessions of a collaborative serious game [14].

3. THEORETICAL BACKGROUND

3.1 Reinforcement Learning

Reinforcement Learning is a way of programming agents by reward and punishment without needing to specify how the task is to be achieved [1]. The agent that learns then must do it through trial-and-error interactions within a dynamic environment.

Figure 3.1 presents the standard Reinforcement Learning model, in which an agent is connected to its environment T via perception and action. For each interaction the agent receives from the environment an input i, which is its current state s. Then, the agent generates an action, sends it to the environment and it changes its state. The state transition has a value which constitutes a scalar reinforcement signal r. The behaviour B of the agent chooses actions in order to maximize the sum of values of the reinforcement signals acquired during the time, and this generates the so-called policy π.

The reinforcement learning may be formalized in terms of Markov Decision Processes (MDP), which consists of:

- Discrete set of environment states, S.
- Discrete set of agent actions, A.
- The probability of moving to s' given that the agent is in state s and executes action a, $P(s' \mid s,a)$.
- The expected immediate reward from executing action a and moving to state s' from state s, $R(s,a,s')$.

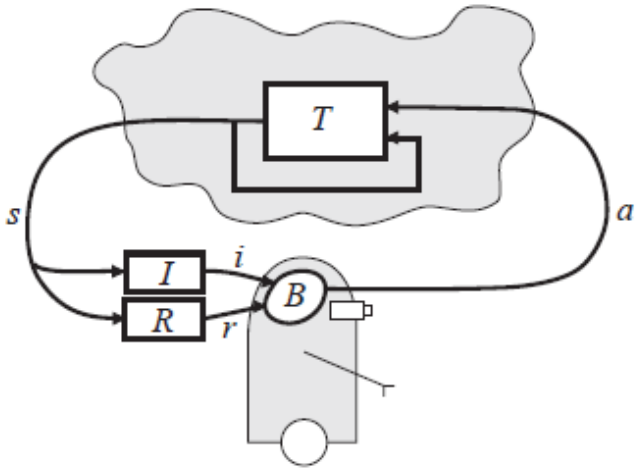

Fig. 3.1: Standard Reinforcement Learning model [1].

- The discount factor γ which presents the difference between future and current rewards.

Figure 3.2 shows a MDP example.

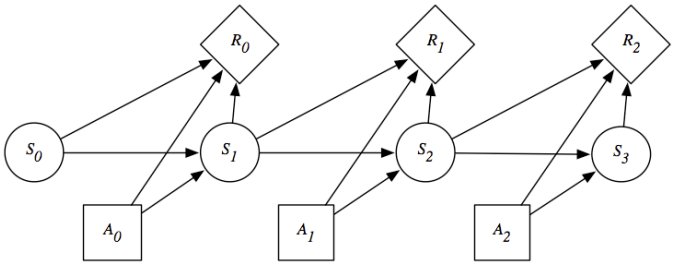

Fig. 3.2: Decision network representing a finite part of an MDP [2].

Reinforcement Learning is not a MDP in all senses though, since the agent, initially, only knows the set of possible states and the set of possible actions. Therefore, the *P(s' | a,s)* and *R(s,a,s')* are initially unknown. The agent acts to achieve the optimal discounted reward, with discount factor γ.

Thus Reinforcement Learning is concerned with how to obtain the optimal policy when such model is not known *a priori*. The agent must interact

with its environment directly to obtain information which can be processed to produce an optimal policy. For this, there are two models to consider:

- Model-free: it learns a controller without the knowledge of a model.
- Model-based: it learns a model and from it designs a controller.

The one used in this project is the model-free Q-Learning, which will be here described.

3.1.1 Q-Learning

In Q-learning and related algorithms, the agent tries to learn the optimal policy from its history of interaction with the environment [2]. A history of an agent is a sequence of state-action-rewards: $\langle s_0, a_0, r_1, s_1, a_1, ... \rangle$. It means the agent was in state s_0, executed action a_0, and received reward r_1, and so on.

We call then an experience the tuple $\langle s, a, r, s' \rangle$.

Let $Q^*(s, a)$, where s is a state and a is an action, be the expected value of doing a in state s and then following the optimal policy. Q-learning uses temporal differences to estimate the value of $Q^*(s, a)$ (Equation 3.1. There is a table of $Q[S, A]$, where S is the set of states and A is the set of actions. $Q[s, a]$ represents its current estimate of $Q^*(s, a)$.

$$Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma max'_a Q[s', a'] - Q[s, a]) \tag{3.1}$$

The Q values will converge to the optimal values, independent of how the agent behaves while the data is being collected (as long as all state-action pairs are tried often enough). Although it may converge quite slowly to a good policy.

Q-learning is the most popular algorithm and seems to be the most effective model-free algorithm for learning from delayed reinforcement [1].

3.2 Neural Networks

Artificial Neural Networks (ANN) are an attempt at modelling the information processing capabilities of nervous systems [4].

Natural neurons (Figure 3.3) receive signals through synapses, which are located on the dendrites. When the signals received are strong enough, the neuron is activated and emits a signal through the axon.

The idea of ANN is taken from the natural neurons. The artificial neurons (Figure 3.3) is basically to receive inputs, which are multiplied by weights (strength of the respective signals), and then compute mathematically the activation of the neuron. Another function calculates the output that follows to a next neuron of the end of the chain. ANNs combine artificial neurons in order to process information.

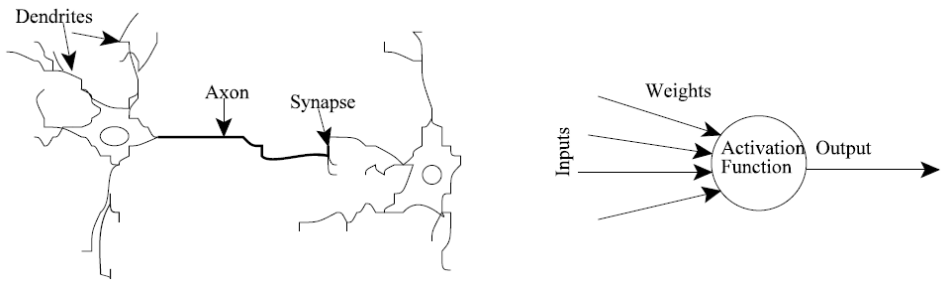

Fig. 3.3: Natural neuron (left) and artificial neuron (right) [3].

Figure 3.4 presents a typical artificial neural network structure. It represents a function Φ evaluated at the point (x, y, z). The nodes are the functions $f1, f2, f3, f4$ and the different weights $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are produced, in order to generate different network functions.

Three elements define a model of artificial neural network:

- The structure of the nodes;
- The topology of the network;
- The learning algorithm used to compute the weights.

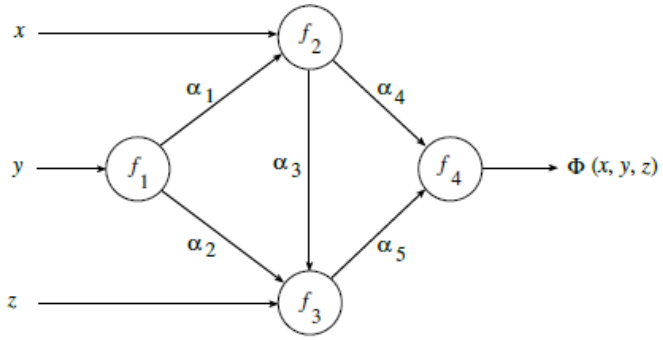

Fig. 3.4: Functional model of an artificial neural network [4].

Neural networks are normally organized in layers (Figure 3.5). The layers consist in interconnected nodes which contain the activation function. There is an input layer, which communicates to one or more hidden layers where the actual processing is done, computing the weights. The hidden layers then connect to the output layer.

The higher a weight of an artificial neuron is, the stronger the input which is multiplied by it will be. By adjusting the weights of an artificial neuron we can obtain the output we want for specific inputs. They can be adjusted by learning or training algorithms, in order to obtain the desired output from the network.

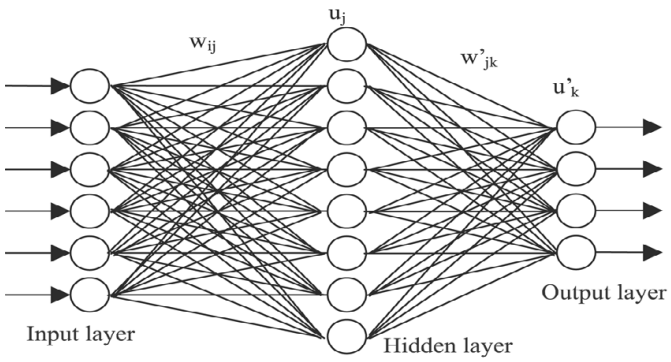

Fig. 3.5: Layered artificial neural network [5].

The layered neural network presented in Figure 3.5 is called feed-forward.

Signals only go one way, from input to output layers. This is the one used in the project. Another architecture could be the feedback networks, that by introducing loops can have signals in both directions. They are dynamic which means their state changes continuously until it reaches an equilibrium.

3.3 Deep learning

Deep-learning networks (DL) are distinguished from single-hidden-layer neural networks by their multi-step process. More than three layers (including input and output) is defined as "deep" learning.

In deep-learning networks, each layer of nodes is trained on a distinct set of features based on the previous layers output. The more layers, the more complex the nodes' features can recognize, since they aggregate and recombine features from the previous layer.

Depending on how the architectures and techniques are intended for use, DRL can be categorized into three major classes [15].

- Deep networks for unsupervised or generative learning: it tries to capture high-order correlation of the observed or visible data for pattern analysis or synthesis purposes when no information about target class labels is available. This is the one used in the project.

- Deep networks for supervised learning: it provides discriminative power for pattern classification purposes, often by characterizing the posterior distributions of classes conditioned on the visible data.

- Hybrid deep networks: the goal is discrimination which is assisted with the outcomes of generative or unsupervised deep networks.

Deep learning discovers intricate structure in large data sets by using the Backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer [8].

Backpropagation is a method of training artificial neural networks used with an optimization method such as gradient descent. The method calculates the gradient of a loss function with respect to all the weights in the network. The gradient is then fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function.

3.3.1 Deep Reinforcement Learning

With all the deep learning techniques that exist, it seems natural to ask whether they could be beneficial when adding to them reinforcement learning [8].

It is very challenging to use both together, since they have opposite characteristics such as:

- Normally, DL applications succeed when there is a large amount of handlabelled data, while RL must learn from a scalar reward;
- Most of DL algorithms assume data samples to be independent, while RL uses sequences that are highly correlated in time;
- In RL the data changes as the algorithm learns, while in DL the structure is fixed.

Although, as it was mentioned in the Related Works section it is possible to make both techniques work together and generate very good results.

In this project, the Deep Reinforcement Learning was the technique used.

4. PROJECT EMOTE

EMOTE, which stands for EMbOdied-perceptive Tutors for Empathy-based learning, is an European Union project focused on empathy-based robotic tutors [16]. More information can be found in <http://gaips.inesc-id.pt/emote/>.

The overall goal of the project is to develop an empathic robot tutor for 11-13 year old school students in an educational setting. The robots will be able to understand learners affective and motivational states, adapt to the learners' needs and display appropriate responses in order to motivate children when learning.

There are two learning scenarios:

- Scenario 1: virtual learning environment focusing on map exploration;
- Scenario 2: based on Enercities (EU-funcded existing game), which deals with sustainable energy awareness.

This project is implemented for Scenario 2.

4.1 Enercities Game

Enercities is a collaborative game played by three players (a mayor, an environmentalist and an economist) and the objective is to build a sustainable city over time (Figure 4.1) [6]. The players can build and improve structures such as housing, businesses, public amenities, etc. in order to grow the population of the city.

The players' performance is measured in terms of points scored for economy, environment and well-being.

Fig. 4.1: Screenshot of Enercities game [6].

The three roles share resources such as oil, power and money. Each role can build their specific structures and some structures can be built by any role.

There are four levels in the game. The players can advance to the next level when they achieve the goal population set for each level.

The game is over when the team completes the final level of the game. The team loses the game when they run out of non-renewable resources. The challenge is to manage the balance between several factors: growing the population, keeping them happy, keeping the environment green, generating enough power, building businesses to make money, etc.

During each turn, a player can perform one of four possible actions:

1. Build a structure: the player selects one of the structures available for his role and choose one of the places available in the city.

2. Upgrade existing structure: the player can perform up to three upgrades at turn to make them more efficient (i.e., produce less CO2 emissions, consume less power, etc.).

3. Implement a policy: there are five available policies, which improve several aspects and structures of the city.

4. Skip the turn.

The game allows the players to develop two aspects: they can collaborate and work as a team to build a healthy and sustainable city and also compete in order to maximize their own individual scores.

The EMOTE project for this scenario aims to build a system where the game will be played by two students and a robotic tutor (Figure 4.2). The objectives of the robotic tutor will be to play the role of a team member, play the game, contribute positively to the discussions, and tutor the students on various underlying educational concepts as the game progresses.

Fig. 4.2: Robotic tutor playing Enercities with students [6].

4.2 EMOTE Project Architecture

Figure 4.3 shows the architecture for the EMOTE project under development.

When an event happens in the map application, a message is sent to the learner model. The learner gathers this information with information received from the Perception and Affect Perception modules, to estimate the current state of the interaction.

The Affect Perception module uses these data to update the affective states and sends a message to the learner model.

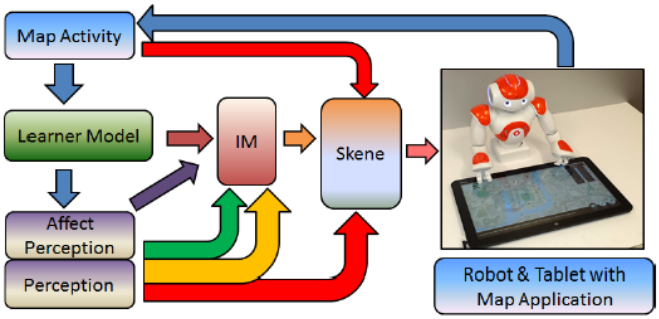

Fig. 4.3: Structure of the EMOTE project [7].

The Interaction Manager (IM) uses the received data to select an appropriate next high-level system action.

The Skene module transforms the high-level action specification into a concrete set of words and behaviours for the Robot to perform. It also uses low level information from other modules to gaze at the learner and map locations on the touch-screen device.

The Map Application module and the Map Application are the Enercities game module and interface, respectively.

The project set-up consists of a Nao robot and a large (55") touch-screen table.

5. PROJECT DEVELOPMENT

5.1 Methodology

The project consists in creating a Deep Reinforcement Learning agent to play one of the roles in the collaborative game Enercities. It was defined by the team that the two other players would be random agents. As a random agent, the player will do any action in the game, on condition that is an allowed move.

It is clear though that many other small tasks should be executed before the DRL is implemented. Therefore, the methodology for the development of the project consisted in several steps.

Firstly, before even starting any real implementation, it was needed to read several bibliographies about the project itself and about the theoretical background that would be useful.

Also, the code of the Interaction Manager (under development in Heriot-Watt) was provided for us, in which we would include our DRL agent and the random agents as well. If the DRL agent works properly, it will replace the current AI planning used for identifying the next best game move.

The code provided, besides the fact we would put our code on it, would also be useful to learn how to receive and send data from/to the game.

After understanding the code, a dummy AI module was implemented. It would perform as a random agent, for two of the players (environmentalist and economist It was also useful to construct it to understand the game rules and some of the functions would be used in the DRL agent to be developed.

Then the network implementation should be done. For that we needed

the communication between Java and Javascript to work properly. This was also implemented.

The Interaction Manager (IM) was written in Java. It was decided that it would be beneficial if the Game Player was implemented in the same language. We used an off the shelf toolkit called ConvNetJS to build the learning agent using DRL <http://cs.stanford.edu/people/karpathy/convnetjs/>.

After that, the experiments would be performed.

Figure 5.1 shows the configuration of our project.

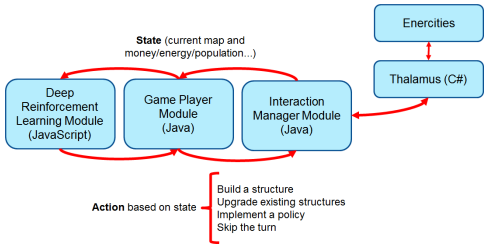

Fig. 5.1: Configuration of the project.

The DRL agent to be implemented and the Game Player GP implementated communicate with the IM, which sends the state of the game. The GP treats the state data to send it to the DRL agent (the scores and actions that were executed) and the DRL replies with an action to be executed. The IM interacts with the Thalamus, which is a programme developed to manage all the modules (from all the project's partners) and that communicates with the Enercities game.

5.2 ConvNetJS

ConvNetJS is a Javascript library for training Deep Learning models (mainly Neural Networks). All the information of the library, including demos and the documentation can be found in http://cs.stanford.edu/people/karpathy/convnetjs/index.html.

The library is available on Github under MIT license and also on npm for use in Nodejs. It was originally written by Andrej Karpathy, a PhD student at Stanford, and has since been extended by contributions from the community

There are two ways of using the library: inside the browser or on a server using node.js. This second way is the one we used for our project.

Many reasons led us to choose this library in our project:

- It possesses a Deep Q-Learning module already implemented, different from other libraries that have it in separate modules;
- Other colleagues from Heriot-Watt University have already used it and it proved to be easy to use and successful;
- There is no need of specific software requirements, compilers, installations or GPUs, avoiding other minor problems before implementing the project.

With this library it is possible for example to design a framework for creating and training a network. It can be built by specifying layers of various types. For instance, some of the layers used in our implementation:

- Convolutional: it computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and the region they are connected to in the input volume.
- Pool: it performs a downsampling operation along the spatial dimensions (width, height).
- Fully-connected (FC): it computes the class scores. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

Once the network is defined, it can be trained either by using backpropagation or to minimize a sum of squared errors to learn arbitrary data in regression applications.

In the case of this project, the Deep Q-reinforcement learning class was used to learn to play games given only the game state.

5.3 Dummy AI agent implementation

The dummy AI is a random agent to play the game as the environmentalist and economist.

The first step to develop it was to create a dynamic database. It was provided some XML files in which we could get some rules of the game:

- structures.xml: who can build each structure, the cost and the game level they can be built.
- structureupgrades.xml: which structures can receive each upgrade.
- upgrades.xml: the cost of each upgrade.
- surfaces.xml: which structures can each kind of field accept.
- grid.xml: which field is contained in each cell.
- policies.xml: the cost of each policy.

Figure 5.2 presents the class diagram for the whole implementation.

The XML files were read in the Database class and classes were made of them in order to have our own database (Structure, Cell, Policy and Upgrade). With them we could check allowed actions selected by the agent randomly and have the current map of the game (what each cell has as structure if any, which upgrades it has, which policies were already implemented).

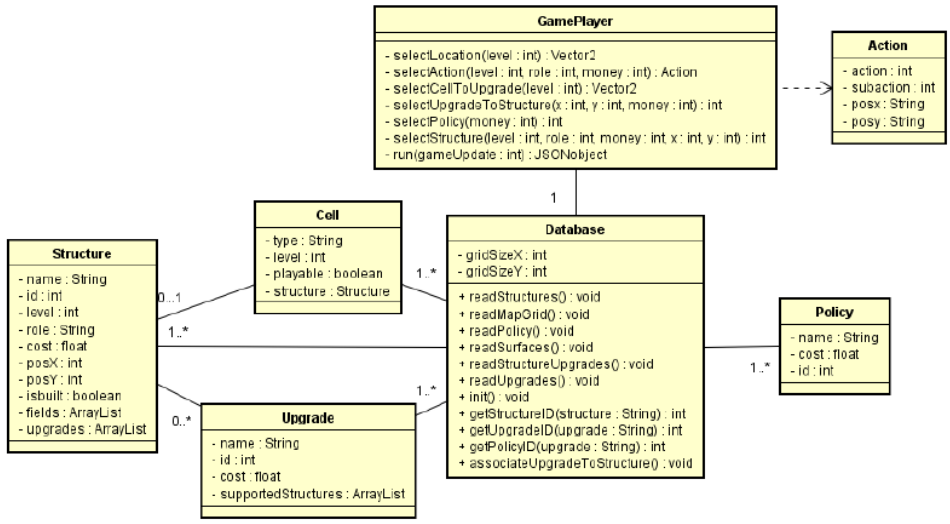

Fig. 5.2: Class diagram of the implemented dummy AI.

The action class contains the identification of the action (skip, build, upgrade or implement policy), the subaction (what to build, which upgrade or policy) and the position (in the case to build and upgrade).

The GamePlayer class is the one that connects to the IM. It receives the state of the game, and executes all the functions to select actions, subactions and cells. Finally, it sends a JSON to the IM, that will execute the action in the game.

5.4 Connection Java/Javascript

In our project, the Javascript, which is the client, is connected to the server via a WebSocket connection and the server periodically publishes messages. The server side application runs on Jetty, which is a servlet container.

Web applications are mostly built around the request/response pattern of HTTP protocol. In this sense, the web browser always initiates the protocol by requesting a resource from the server. When this happens, the server application handles the request and returns a response to the browser. The

problem is that the server cannot send any information to the client unless the client explicitly requested it.

WebSockets can solve this by allowing a server to communicate with a browser without needing a request from it.

A WebSocket is a communication channel over a single TCP connection between two parties (e.g. browser and server). This channel is always on unless either party finishes the communication. In HTTP, on the other hand, each request-response interaction occurs over a distinct connection. Moreover, WebSockets are full-duplex. This means simultaneous bidirectional communication is allowed that is not the case for HTTP which is half-duplex.

One example of communication is shown in Figure 5.3.

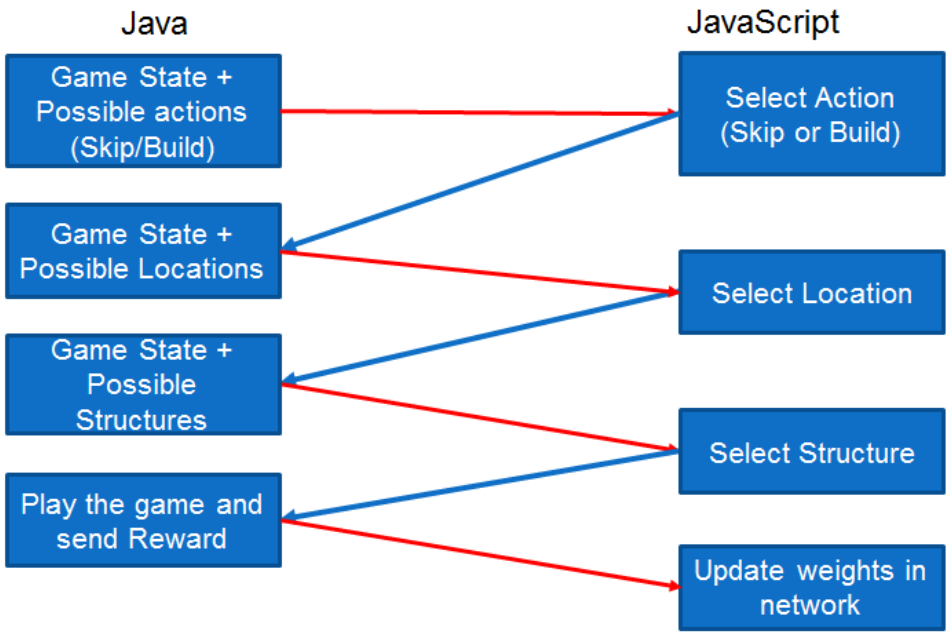

Fig. 5.3: Example of communication of Java/Javascript for the building a structure action.

5.5 *Network Design*

Figure 5.4 shows the designed model for the networks to be created. After sending a state (which contains current map of the game, the policies and game scores), there is a first network that will select an action out of the four possible ones.

For each of the actions, a network should be created, except in the case of "skip" action. For "build" and "upgrade", the following network are made for selecting the location within a range of possible locations and next a network for selecting the structure and upgrade, respectively.

For the policy, only one network is created, for selecting which one would be implemented.

In order to have a faster period of training, the group has decided for designing only the networks marked in the red box of Figure 5.4.

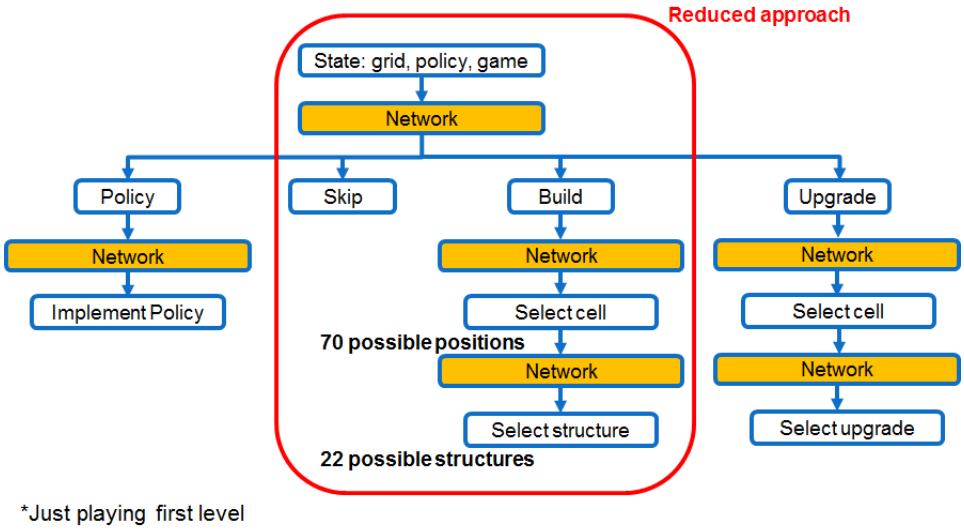

Fig. 5.4: Network model designed.

The input of the first network is shown in Figure 5.5. It receives in total 504 features. From the total, 400 features are from the map of the game, composed by cells. Each cell is represented in seven features: the first one

gives the structure ID as seen in the game and the next six are the upgrades the structure has implemented (maximum is six per structure), represented as "0" if done and "1" otherwise.

After the map, there are six features reserved for policies. Again, "0" if implemented and "1" otherwise.

Last, there are the game scores: economy, environment, well-being, energy, money, oil and population.

There are two of the features that assumes value "0" so that the convolutional network can be designed as size 7.

Finally, the network is designed. Figure 5.6 presents one of the models tested. The first layer is the input with all features, which are connected to a convolutional layer of size 7. A subsampling is then performed with the pool layer. One more convolutional and pool layers are next and the reduced set of values goes to two hidden layers of size 20. Last, the result of 20 features is obtained from them.

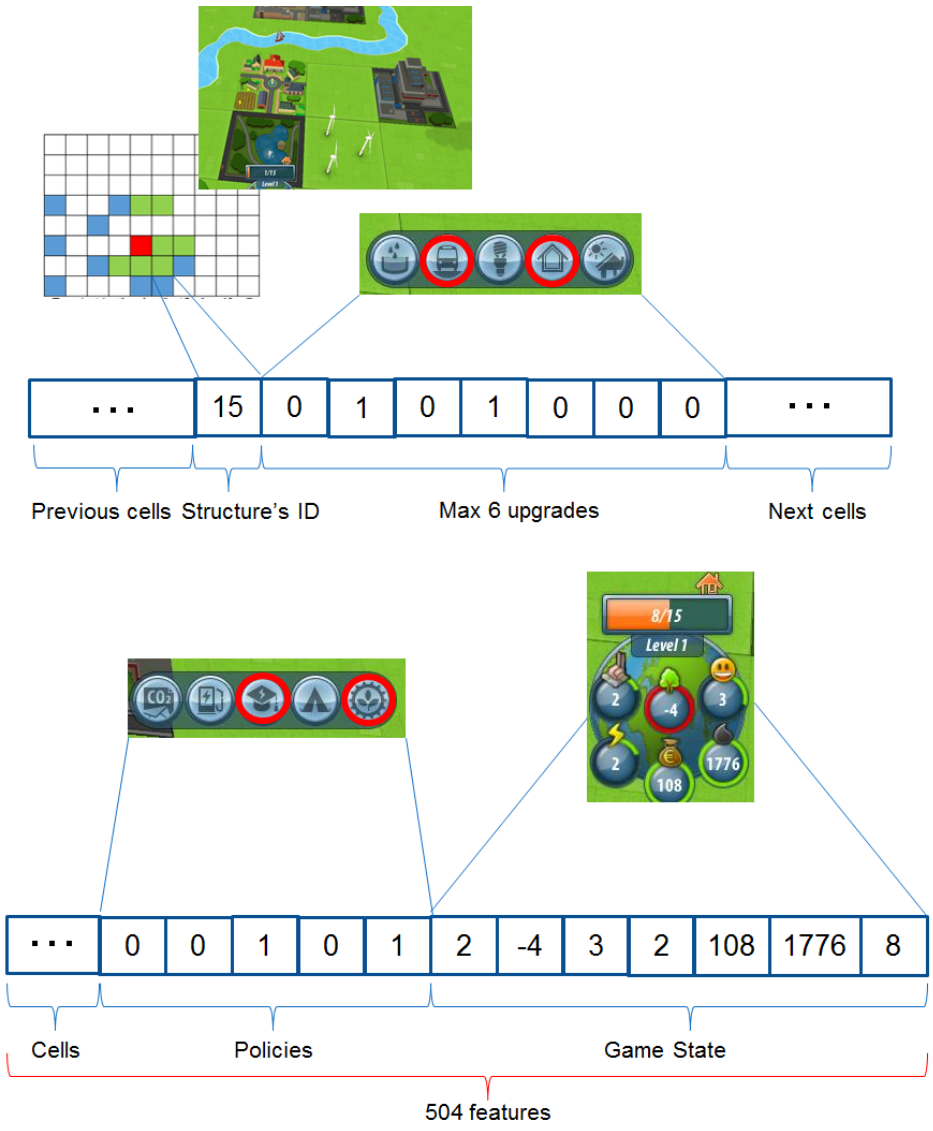

Fig. 5.5: Input string in the first network.

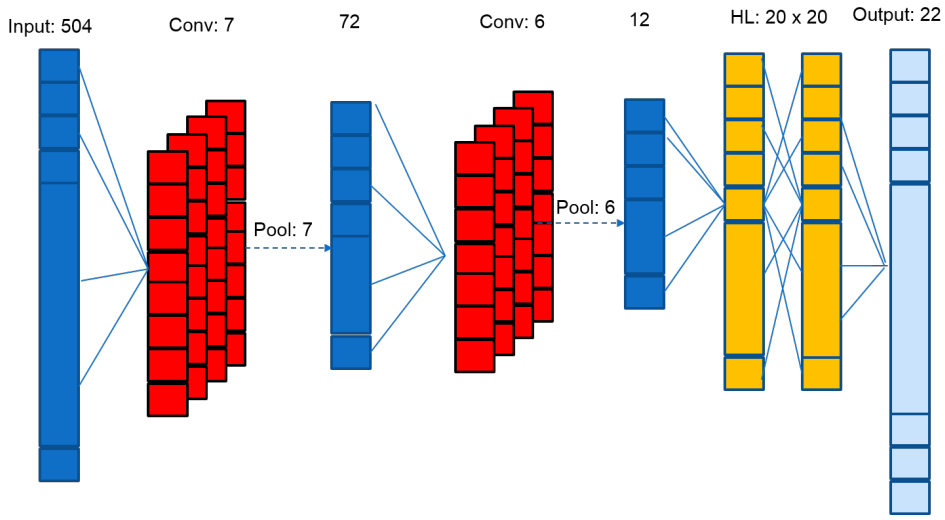

Fig. 5.6: Example of designed network for test.

6. RESULTS

Enercities is a complex game: there are many possible actions and our input state vector is very long (504 features). As mentioned, the mayor would be our DRL agent, and the environmentalist and economist players would use the Dummy AI. This approach was decided to make the mayor learn without being influenced so much by other players, since they would play randomly.

While generating the first results, it was realized that every time the agent reaches the next level, we needed to press a button to confirm. This was a big problem of our development, since we would need someone always pressing a button to go to next level. Therefore, it was decided that we just want to see if our agent could learn passing the first level and reaches level two.

It was made a simplification in the game in which the agent and the other players could just build a structure or skip. However our input vector contains all the state of the map with policy and upgrade information (actions that can be executed by other players).

To generate the results, many types of networks were designed. Our first approach was just to use a normal neural network. In the website of ConvNetJS there is an example for Deep Q-Learning for an agent that has the goal of getting red balls (https://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html).

Even being a really different game, our first neural network was built based on this one.

6.1 Test 1

Networks' Configuration

The specification for our network was 2 hidden layers of 40 neurons fully connected. -1 was given as a reward every mayor's turn, 1000 was given as a reward if the agent reaches level 2, -1000 otherwise. By episode, 18 turns maximum were played.

This simple reward function was designed to make the agent learn the moves that can help it finish the level successfully. 1000 was given as a reward to make it win a lot of points and show the actions that he did were really good. In addition, -1000 was given to really punish him, and do not repeat those actions. We give -1 to every step to punish it, to make our agent try to play less steps to reach next level.

Other parameters: $\epsilon = 5\%$, $\gamma = 0.7$ were given to all networks.

Training results

Following, all the results are shown and discussed. The plots have is x-axis the number of episodes and in y-axis the average reward of each 10 episodes.

Figure 6.1 presents the results for this network, trained for 90 minutes.

As we can observe, the result was random. This first test did not give us better results.

6.2 Test 2

Networks' Configuration

We decided to create more hidden layers to check if our agent could learn just by using neural networks. For this network it was created 4 hidden layers of 40 neurons and 11 turns per episode. This network was trained for 45 minutes. We planned to train it for more time, however the game crashed during the execution.

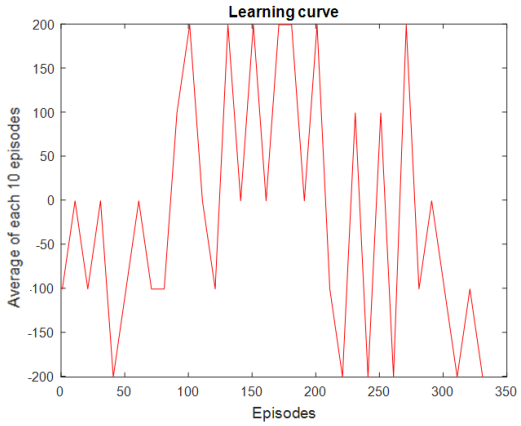

Fig. 6.1: Test 1. Rewards: (-1,1000 ,-1000). Execution time: 90min (18 turns). Two neural networks layers of size 40. .

Training results

Results are shown in Figure 6.2.

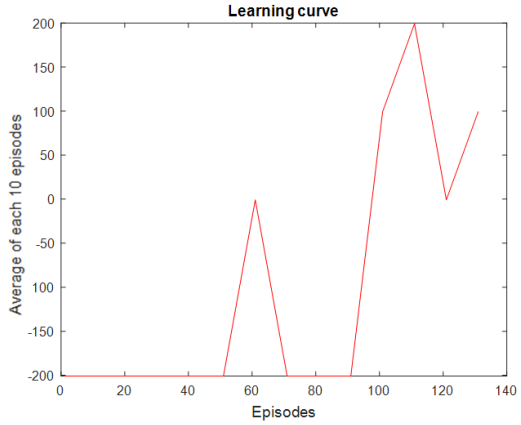

Fig. 6.2: Test 2. Rewards: (-1,1000,-1000). Execution time: 45min (11 turns). Four neural networks layers of size 40.

As we can see, this network seems to be learning, it increased our average reward results, but was not conclusive, since it did not run for a long time.

6.3 Test 3

Networks' Configuration

It was decided to create a similar network and run it for longer time. It was used 6 hidden layers of 100 neurons and 11 turns each episode, giving the same reward as the previous test. The network started to become more complex and expected to get better results.

Training results

In the Figure 6.3 we can visualize the result for this network.

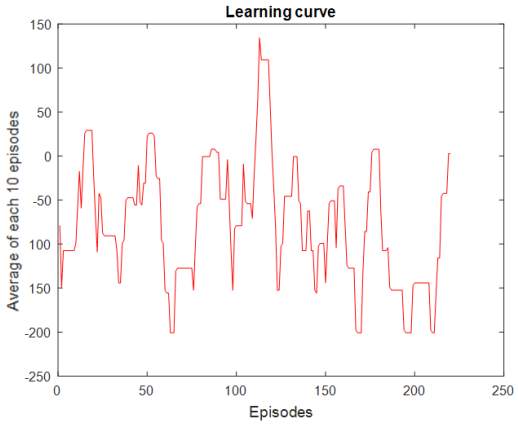

Fig. 6.3: Test 3. Rewards: (-1,1000,-1000). Execution time: 280min (11 turns). Six neural networks layers of size 100.

The curve shows the agents was well exploring, receiving good rewards until episode 120, however it started to decrease.

The network is still getting random results, so it was decided a new approach.

6.4 Test 4

Networks' Configuration

As in the Atari paper, Convolutional Neural networks were used to design the next networks. Since we are using a 1D vector for our input, it was created many 1D convolutional masks to be trained.

This new network was a fully convolutional neural network. It starts with 16 convolutional masks with size 7, then a pooling layer of size 7 was created, followed by 20 convolutional masks with size 5, then a pooling layer of size 6, another 20 convolutional mask with size 5, and finally a pooling layer with size 3. The reward function is still the same as the previous tests and 11 turns per episode.

Those pooling layers values were decided according to the size of our input, since the convolutional layer gives an output for each seven positions. Therefore, it would retrieve the most significant part from those 7 values. The values of 6 as pooling layer in the next steps were created to reduce the size of our input image until we each only 22 values to represent our input data. This is the main concept of a Convolutional Neural Network, it starts with a long input data, and the network will be able to get the most important information from the image and represent it in much lower data.

Training results

The Figure 6.4 shows the result for this network. It was executed for 90 minutes.

As we can see, the agent receives good reward, then bad ones, and it remains in a cycle. It showed that even trying to use a better approach, our agent still makes random actions.

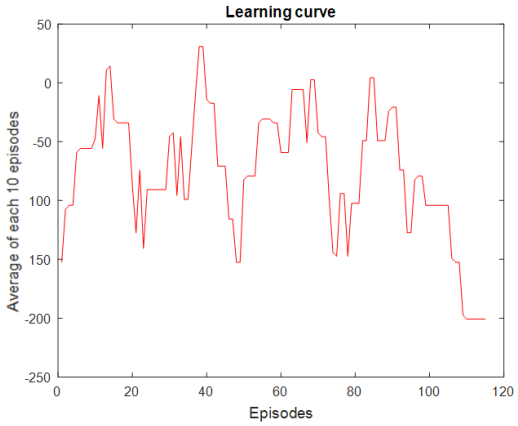

Fig. 6.4: Test 4. (-1,1000, -1000). Execution time: 90min (11 turns). Convolutional neural network and pooling of size 7×7, followed by a convolutional neural network size 5×5 and pooling of size 6×6, and convolutional neural network size 5×5 and pooling of size 3×3.

6.5 Test 5

Networks' Configuration

Since creating a simple neural network and a pure convolutional neural network was not giving us good results, it was decided to create a hybrid approach. This hybrid approach starts with a convolutional neural network in the begging to create our data in a lower dimensional, and then create a simple neural network that will decide which action our agent should do.

For this network, 16 convolutional masks with size 7 was created, then a pooling layer of size 7, followed by a 16 convolutional mask of size 5, then a pooling layer of size 6. After it was created a neural network with 2 hidden layers of 20 neuron fully connected. The same reward function from previous tests were used.

For this test, we started with a 504 features, transformed into 14 features to be the input of our neural network.

Training Results

Figure 6.5 presents the result for this network. It was executed for 180 minutes.

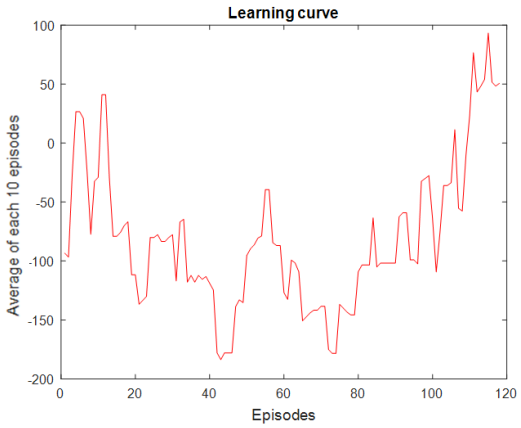

Fig. 6.5: Test 5. (-1,1000, -1000). Execution time: 180min (11 turns). Convolutional neural network and pooling of size 7×7, followed by a convolutional neural network size 5×5 and pooling of size 6×6, and two neural networks of size 20.

In the end of the episodes executed it seemed to be learning, however the game crashed during the learning process. It was being executed for 180 minutes, so it was decided to run it again for 280 minutes.

In the Figure 6.6 we can see the result for this new training.

As we can see, it was receiving good rewards at some episodes, but started in the end only bad rewards, showing that our agent is still playing randomly.

6.6 Test 6

Networks' Configuration

Our last trial was to change the reward function in order to make it learn how to reach level two. In the game, it is needed to build two suburban structures to reach the new level. So it was created a new reward function

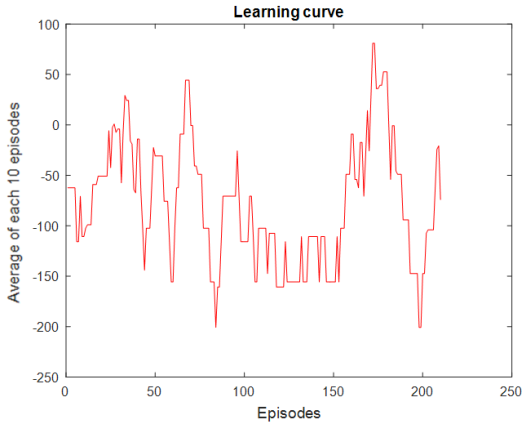

Fig. 6.6: Test 5. (-1,1000, -1000). Execution time: 280min (11 turns). Convolutional neural network and pooling of size 7×7, followed by a convolutional neural network size 5×5 and pooling of size 6×6, and two neural networks of size 20.

that gives 1000 times the number of suburbans built. The configuration of the network remains the same.

Training Results

In the Figure 6.7 we can see the results for this test. It was executed for 60 minutes.

As we can see, in the first episodes, it seemed to be learning, however it became really bad, and it was giving always really low results.

6.7 Discussions

It was tried different settings for the our network to make it learn, but we did not succeed in any case.

From what we think, it happened for two main reasons. First, the game is pretty complex comparing with other successful AIs using deep q learning. Most of those games were much simpler, with a smaller set of actions.

Another reason was that we did not have a simulator to make it train

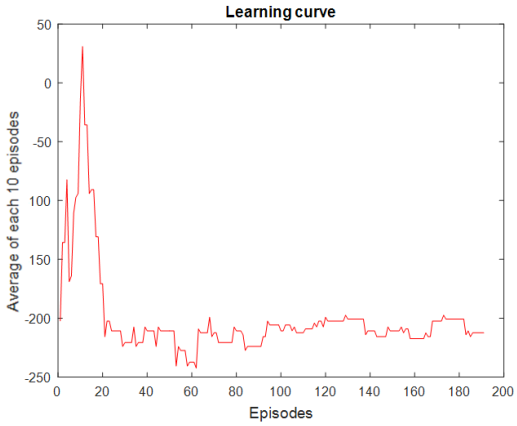

Fig. 6.7: Test 7. Reward: (-1,n*1000, -1000). Execution time: 120min (11 turns). Convolutional neural network and pooling of size 7×7, followed by a convolutional neural network size 5×5 and pooling of size 6×6, and two neural networks of size 20.

faster. Our game was taking one minute per episode more or less. If we had a simulator it could run faster and after some hours we could have more conclusive results.

Besides that, for the training other options should be tried such as more turns per episode and more episodes to see if the agent can learn.

Above all, it is important to say that the Emote researchers will take the framework we developed forward and run further experiments and tests, so it has been a very worthwhile endeavour.

7. TIME SCHEDULE

The group was initially given a total of 12 weeks to accomplish the project. During the execution, the deadline was extended to 13 weeks.

The proposed time schedule with its respective tasks are shown in Figure 7.1.

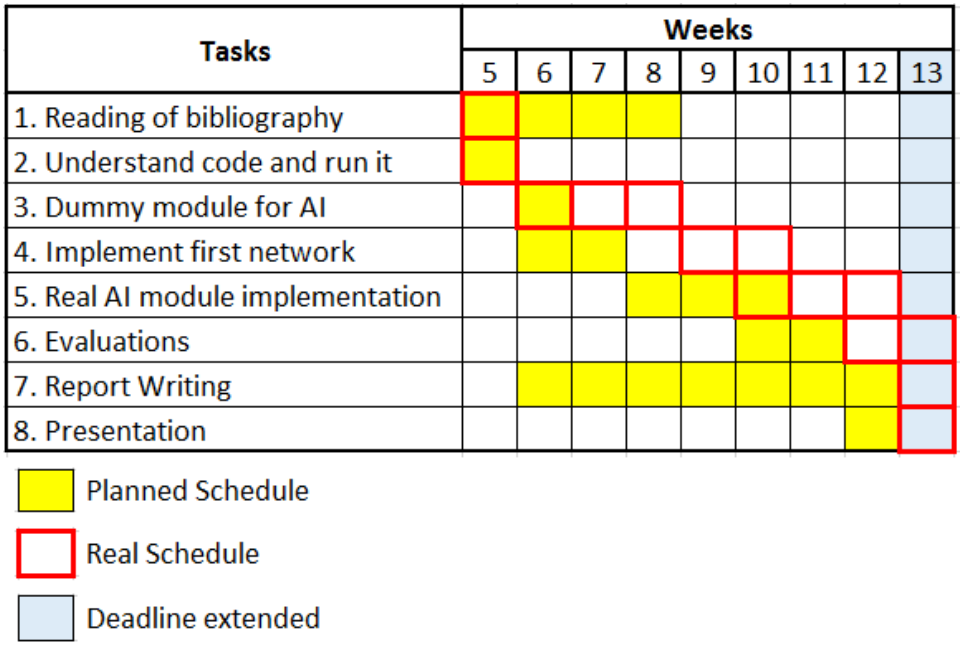

Fig. 7.1: Project Time Schedule

Tasks number 1 and 2 were executed on time, but since task number 3 the project was delayed. It happened because it was thought that the dummy AI would be simpler.

The reason for taking so long is that the game possesses many restrictions

that should be taken in account. Most of the restrictions are contained in XML files provided, but some are not. Besides, the dummy should be a very consistent player (even though it has random moves), since it would be used for two of the three players in the cooperative game and interfere in the Deep Learning agent.

After that, it can be seen the other tasks took approximately the time it was firstly predicted.

8. CONCLUSION AND FUTURE WORKS

Conclusion

The main goal of the project was to develop an intelligent agent to play the collaborative game Enercities. The agent implemented has a DRL behaviour and play the mayor, one of the three roles of the game. Other tasks executed in order to accomplish the main goal was implementing a random agent to play as the other two roles (economist and environmentalist) and providing the connection between Java and Javascript modules.

In the network designed, the DRL agent has three levels of neural networks to select the optimal actions. Our implementation is limited for level 1 of the game and also to two actions out of four: building structures and skipping.

Unfortunately the DRL agent did not prove to be successful in learning how to play the game. Many reasons can be listed, such as the game complexity and the way the networks are designed, for instance. In next section some future works are listed in order to determine what could have been done wrong.

Implementing a DRL agent is very challenging, but it is worth the try since it would be a better alternative for the AI planning, which has to be modelled. In other words, for the AI planning all the rules of the game and possible actions should be coded, while the DRL plays by itself after the training period.

The main outcome of this project is that we could learn different topics such as Deep Reinforcement Learning, Convolutional Neural Networks and

JavaScript (which can be used for other purposes rather than Web Development).

Future Works

Several tasks are considered for the future regarding this project:

1. To tune the network for better results. It seems the learner agent will take more time than we were expecting. Since the game takes so long to play and therefore to have a result, the next experiments will be performed for a much longer time. Also many other parameters have to be tested, it is too many to change and with a huge variation each.

2. Create other approaches for the Input. We think that might be possible that the input vector is too big for the agent to learn with a decent accuracy. Other inputs are though to be tested (i.e. more than one input, or several types besides string such as JSON).

3. To implement a different way of network (more humanized one). Instead of selection an action, then a subaction and the position sequentially, the implementation could be made regarding the scores of the game. For instance, if the energy score is low, which actions and subactions must be made to increase it.

4. To develop the DRL agent to play Enercities up to level 4. As before mentioned, due to technical reasons we could only train the agent up to level 1. After the problem is solved, the other levels will also be in the training (which will increase even more the training time required).

5. To add the other actions to be selected by the DRL agent (implementation of policies and upgrading of structures). In order to have a faster training, we only implemented the network for the skip and building structure actions. This should be also changed for next tasks.

6. To test the implementation of DRL agents for environmentalist and economist. Currently the game is using random agents for the environmentalist and economist. It was suggested to put them two also as DRL agents. Testing this idea might be worth it.

BIBLIOGRAPHY

[1] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. URL http://people.csail.mit.edu/lpk/papers/rl-survey.ps.

[2] Poole David L. and Mackworth Alan K. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, New York, NY, USA, 2010. ISBN 0521519004, 9780521519007.

[3] Carlos Gershenson. Artificial neural networks for beginners. *CoRR*, cs.NE/0308031, 2003. URL http://arxiv.org/abs/cs.NE/0308031.

[4] Raúl Rojas. *Neural Networks: A Systematic Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1996. ISBN 3-540-60505-3.

[5] Graham Templeton. Artificial Neural Networks are changing the world. What are they?, 2015. URL http://www.extremetech.com/extreme/215170-artificial-neural-networks-are-changing-the-world-what-are-they.

[6] Patrcia Alves-Oliveira, Srinivasan Janartham, Ana Candeias, Amol Deshmukh, Tiago Ribeiro, Helen Hastie, Ana Paiva, and Ruth Aylett. Towards dialogue dimensions for a robotic tutor in collaborative learning scenarios. In *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, pages 862–867, Edinburgh, Scotland, August 2014. IEEE, IEEE.

[7] Amol Deshmukh, Aidan Jones, Srinivasan Janarthanam, Helen Hastie, Tiago Ribeiro, Ruth Aylett, Ana Paiva, Ginevra Castellano, Mary Ellen Foster, Lee J. Corrigan, Fotios Papadopoulos, Eugenio Di Tullio, and Pedro Sequeira. An empathic robotic tutor in a map application. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 1923–1924, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-3413-6. URL http://dl.acm.org/citation.cfm?id=2772879.2773506.

[8] Yoshua Bengio Yann LeCun and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015. doi: 10.1038/nature14539. URL http://www.nature.com/nature/journal/v521/n7553/full/nature14539.html.

[9] C. Braezeal. Role of expressive behaviour for robots that learn from people. *Philosophical Transactions of the Royal Society B*, 364:35273538, 2009.

[10] Iolanda Leite, Ginevra Castellano, André Pereira, Carlos Martinho, and Ana Paiva. Modelling empathic behaviour in a robotic game companion for children: An ethnographic study in real-world settings. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '12, pages 367–374, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1063-5. doi: 10.1145/2157689.2157811. URL http://doi.acm.org/10.1145/2157689.2157811.

[11] Martin Saerbeck, Tom Schut, Christoph Bartneck, and Maddy Janse. Expressive robots in education - varying the degree of social supportive behavior of a robotic tutor. In *28th ACM Conference on Human Factors in Computing Systems (CHI2010)*, pages 1613–1622, Atlanta, 2010. ACM. doi: 10.1145/1753326.1753567.

[12] Video games in education: Why they should be used and how they

are being used. *Theory Into Practice,*, 47:229–239, author = Annetta, Leonard A., 2008.

[13] Antonio Brisson, G. Pereira, Rui Prada, Ana Paiva, Sandy Louchart, Neil Suttie, Theo Lim, Ricardo Lopes, Rafael Bidarra, Francesco Bellotti, Milos Kravcik, and Manuel Oliveira. Artificial intelligence and personalization opportunities for serious games. In *Proceedings of AAAI Workshop on Human Computation in Digital Entertainment and Artificial Intelligence for Serious Games, co-located with AIIDE 2012 - 8th Conference on Artificial Intelligence and Interactive Digital Entertainment*, Stanford, CA, oct 2012. URL http://graphics.tudelft.nl/Publications-new/2012/BPPPLSLLBBKO12a.

[14] Patrcia Alves-Oliveira, Srinivasan Janartham, Ana Candeias, Amol Deshmukh, Tiago Ribeiro, Helen Hastie, Ana Paiva, and Ruth Aylett. Towards dialogue dimensions for a robotic tutor in collaborative learning scenarios. In *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, pages 862–867, Edinburgh, Scotland, August 2014. IEEE, IEEE.

[15] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(34):197–387, 2013. ISSN 1932-8346. doi: 10.1561/2000000039. URL http://dx.doi.org/10.1561/2000000039.

[16] Amol Deshmukh, Ginevra Castellano, Arvid Kappas, Wolmet Barendregt, Fernando Nabais, Ana Paiva, Tiago Ribeiro, Iolanda Leite, and Ruth Aylett. Towards empathic artificial tutors. In *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, HRI '13, pages 113–114. ACM/IEEE Press, March 2013.