Human sensory process for an Artificial Intelligence

Translating stimuli into numbers

Alex Fuentes Raventós, BCE

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Augmented and Virtual Reality)

Supervisor: Michael Manzke

August 2022

Declaration

Permission to Lend and/or Copy

Abstract

The search for realism and immersion in games is booming with the increasing development of Virtual Reality and the investigation of new Artificial General Intelligence. This paper proposes a new approach to Artificial Intelligence (AI) based on the human senses. Real stimuli curves were analyzed for implementing damage curves that affect an agent controlled by an AI algorithm. The agent can detect its surroundings using three human senses created as realistic as possible: sight, hearing and thermal touch. If capturing the stimuli and their effect on the agent is the main goal, the secondary goal is to teach the agent to make decisions depending on how it is affected by the environment while solving two of the primary human needs: health and hunger. The results of the studies concluded that the creation of a reliable agent capturing its surroundings while being affected by them was successfully created.

Contents

List of Figures

List of Tables

0.1 Introduction

Artificial General Intelligence (AGI) is an Artifcial Intelligence (AI) capable of solving almost all tasks humans can solve. The main difference between AGI and AI is that AI cannot apply the resources outside already defined tasks (Shevlin et al. (2019)). Nowadays, some AI has been created for accomplishing better results than humans in a specifically defined area. However, most of them cannot apply the specific tools to another area. The ones capable of doing that are because of the implementation of Neural Networks (NN) and Transformers (Wolf et al. (2020)), but they also cause impossible errors for the human mind. A good example is the new development of Gato, explained in "A Generalist agent" (Reed et al. (2022)). This agent can work as a multi-modal, multi-task and multi-embodiment generalist policy.

Realism in games and virtual realities has been tried to achieve to create the most comfortable environment for the users. Different algorithms and approaches, (Gilbert (2016)), have been used and combined: Reinforcement Learning, Behavior-based Techniques, Multi-agent systems, NN, and more.

Aim by AGI and the tireless pursuit of realism, the main focus is to create a realistic AI capable of detecting its surroundings through its senses and acting according to needs defined by Maslow (Maslow (1958)). To do this, it has been calculated how the intrinsic variables of the objects are received by the human and how these affect human reactions.

This paper presents a realistic process for an AI agent whose mission is to survive in the world while fulfilling some of the principal human needs. An already defined environment has been created, where objects have been placed, and their intrinsic variables **associated with human senses** have been defined.

The investigation had one main goal, to define a realistic environment where the agent will feel through senses the same way a human would feel. After achieving that, the agent was put to test using AI, creating a secondary goal: to teach the agent how a human would react being in the same situations. To achieve these goals, studies already made with real humans have been used to define how the intrinsic variables directly affect the human needs chosen for the project. These studies have been used, for the first time as it's been found, to find or define curves that are applied to the agent exactly the same way that they would affect a human being. Knowing the effects humans perceive and considering the human responses, can be coded as how this affects the agent controlled by the AI.

Not only the environment and the actions are planned to be realistic. Also, the learning process decided was chosen for applying the most realistic learning process possible. This is why **Reinforcement Learning** is the one used in this project. This type of learning faces a problem by trial and error, learning from its mistakes.

This study will not be only useful for an AGI focuses on emotions and human sensations as present in ”Emotional theory of rationality” (Garcés and Finkel (2019)), or for more AGI theories being created that use sensory modalities in AGI studies like ”levels of organization in general intelligence” (Yudkowsky (2007)). It could also be used in cognitive architecture for Agent-Based Artificial Life Simulation as presented in ”A cognitive architecture for agent-based artificial life simulation” (Vieira et al. (2018)). Building on the knowledge gained using Reinforcement Learning, it can be created a new and more realistic Non-Player-Character (NPC) to react the same way a human would act and use it for gaming, training, and more. In summarizing, this research will be helpful for every type of approach to realism that uses the human senses as a part of it.

0.2 Background

Realism is a key factor for user engagement in virtual worlds. Realism could be increased using different techniques in different elements of the virtual world. The more realistic the objects, the environment, the NPC and the interaction between the user and all of them, the more comfortable the user will be in that virtual world. This realism, influenced not only by the fluency but also by the authenticity of the interaction, helps the user to feel immersed in the experience, (Gilbert (2016)).

Creations and rendering of NPC as humanly as possible is one well-known method to increase the feeling of immersion. New rendering techniques are being created to improve the body, clothes and movement imitation (d'Eon et al. (2007), Haro et al. (2001)). Even game engines are creating entire platforms to create humans with these techniques implemented, allowing easy use of a complete meta-human [1] without much effort (Fang et al. (2021)).

The interaction with this NPC is also important. If its movement is completely fluent, but when it interacts with us, it makes completely non-sense, the realism makes a downgrade suddenly. The interaction with them have to be beliavable, and different approaches could be found: the use of genetic-algorithms (Hussain and Vidaver (2006)), the implementation of dialogue-trees (Collins et al. (2016)), among others.

However, the peak of realism is the AGI. Creating an AGI is focused on giving the ability to an agent to understand or learn any task that a human being is capable of. Instead of focusing on only one task, an AGI should be able to use the knowledge gained in one area to use it in another one. It represents generalized human cognitive abilities:

[1]Meta-human: High-fidelity digital humans

facing an unfamiliar task, the AGI system should be able to find a solution as human can. The AGIs is not only focused on games or the virtual world but also on the resolution of every problem-solving that a human could confront.

The project developed is encouraged by the search for realism. The most important AGI approaches used for the development of this project were:

- ”Levels of organization in general intelligence” (Yudkowsky (2007)), where it is explained the five successive levels of functional organization:
  - Code: Saw by the author roughly equivalent to neurons and neural circuitry.
  - Sensory modalities: In humans are sight, sound, touch, taste and smell.
  - Concepts: Categories or symbols abstracted from a system’s experiences.
  - Thoughts: Built by structures of concepts, the example given by the author is a human sentence.
  - Deliberation: Sequences of thoughts.
- ”Emotional theory of rationality”” (Garcés and Finkel (2019)), where it is explored the importance of emotions on human responses. These responses are taken using the expectation created by the human brain predictive systems as a reference for comparison with the actual information received through the senses.

In addition, the paper that finally was the basis for developing the agent logic was ”A Cognitive Architecture for Agent-Based Artificial Life Simulation” (Vieira et al. (2018)). Its principal purpose include simulating the behaviour of living beings credibly. To do it, the author proposed an architecture which consists of:

- Semantic representation. It defends that the agent needs a semantic-modelled world to be able to reason. Therefore, the world objects and the agents should have attributes representing their meaning.
- The agent. The agent can move and interact with other objects because it has senses which perceive the environment and needs that guide him to achieve its goals.
- Senses and perceptions. The goal of the senses is to obtain information from the surroundings and give the agent enough information to decide. The captured information from the object is available in the form of perceptions to the agent’s mind, like the temperature obtained by touch or the sound intensity captured by the hearing.

- Goals. The goal's objective is to alter the environment. During the whole time, the agent is executing a goal. A *goal chain* is the structure formed by goals. A goal should have: —
    - A set of completion condition, which need to be accomplished in order to considered fully done.
    - Optionally, a previous goal or a prerequisite. The previous goal in the goal chain.
    - Optionally, a next goal. The next goal in the goal chain.
    - Optionally, a target. Depending on the goal, it could require a target. For example, if the agent wants to eat, it will need to eat something.
    - A execution cycle. An algorithm executed to solve the goal.
- Needs. It follows the needs defined by Maslow (Maslow (1958)): breathing, temperature, feeding, safety, health, shelter and being part of a group or community or happiness. These needs define the agent's goal to solve it. The goal would be chosen depending on the priority of the need. Needs should have:
    - Intensity value, that represents how important the need is in that instant of time.
    - Increase rate, which defines how fast the intensity value increases.
    - Bound goal. The goal is executed by the agent when the need is related to is the most important.
    - Relevancy evaluator, that gives the importance of every perception felt by the agent, for every need.
- The mind. It translates perceptions obtained by the senses to goals to satisfy the needs. It consists of the memory and the reasoning cycle. The memory stores the perceptions, and the reasoning cycle is instructed to keep only the most important ones.

With this architecture in mind, research about how to train the agent and about the different human senses was done.

The human senses that have been implemented are hearing, sight and thermal touch. Hearing and sight were implemented because they are the senses that nowadays can influence in a virtual world. Thermal touch was also implemented because further investigations are being done related to haptic gloves (Perret and Vander Poorten (2018)), and it seems to be the sense closest to use in the future virtual worlds.

The technique and algorithm that were decided to use is explained in Section 0.5.

0.2.1 Human senses

Sight

It allows humans to capture and interpret visual information about light, color, shape, distance, position or movement. Its receptor organ is the eyeball (eye). Human have binocular sight, which means that the sense is constituted with two eyes. Each eye alone gives us an approximated 130-degree field of vision. With two eyes, we can see nearly 180 degrees, but only in a portion of those degrees are the images captured clearly. The central field of vision covers an angle of 60°. Within this angle, both eyes observe an object simultaneously.

The environment can be interpret thanks to the rays of light that reach the eye. The visual system detects light stimuli (electromagnetic waves), distinguishing between two characteristics of light, its intensity and wavelength (colours). Luminance is the luminous intensity projected on a given area and direction, and is an objectively measurable attribute. The unit is 'Candela per Square Meter' (cd/m2). Do not confuse with brightness, "the quality or state of giving out or reflecting light.". Luminance is the measurable quality of light that most closely corresponds to brightness, which we cannot objectively measure. It can only be perceived.

It is not risky to say that sight sense is always used in 3D and Virtual Reality Environments, because the user can use its own sight sense to distinguish what is happening.

Hearing

Hearing is the sense of detecting sound. When an object vibrates, it causes movement in surrounding air molecules, and this movement is what is perceived by humans through the ear. However, the vibrations need to be at a certain speed to be heard by humans. The slowest vibration humans can hear is 20 vibrations per second, and the fastest is 20.000 vibrations per second. The loudest we can handle without immediately damaging our hearing carries about a million times more energy than the barely audible. Hearing is fundamentally important for survival, maintenance, and reproduction (Plack (2018)). Sound localization, the ability to estimate just where a sound is coming from, depends on the hearing ability of each of the two ears, and the quality of the sound. Since each ear lies on an opposite side of the head, a sound will reach the closest ear first. Humans can locate sound and estimate the distance that it comes from.

Regarding the intensity, the hearing threshold is around 0 decibels(dB). Sounds above 90dB can lead to chronic hearing damage. Hearing becomes uncomfortable if the sound pressure level is above 110 decibels (threshold of discomfort), and it becomes painful above 130 decibels (threshold of pain) (Yost and Killion (1997)).

3D audio is widely used to create realistic environments and interactions with it. As presented in the video by Playstation, 3D audio is used to notice what's happening in every direction and help you to be more immersive in the game. Also, to have the strongest sense of presence and situational awareness. Sound is one of the keys to realism in games.

Touch

The sense of touch is controlled by four main types of receptors [2]: mechanoreceptors, thermoreceptors, nociceptors and proprioceptors. Proprioceptors give limbs position and give the information to move our body accordingly to the task we want to achieve. Mechanoreceptors transmit texture, pressure or vibrations. It is difficult and almost impossible to do simple things like walking or talking if it cannot be felt the part of the body having contact with another surface, as explained in "The sense of touch" (O'Shaughnessy (1989)).

Nociceptors, also called pain receptors, are distributed in the body and are activated when the pain threshold is exceeded. All the other senses explained above also have pain receptors that can be activated by luminance (sight), sound intensity (hearing) or bitterness (taste). When it is talked pain receptors of the touch sense, it refers about the receptors in the skin. The pain receptors are often subdivided by stimulus type: heat, cold, mechanical force and chemicals.

Thermal touch gives us information on temperature when an object is in contact with the skin. They are part of thermoreceptors, which also give us information about our body temperature and skin temperature. (Sessler (1993)).

In this project, it has been implemented the nociceptors related to heat stimuli. Also, proprioceptors and mechanoreceptors has been implemented in a simple way because it is used the agent position and rotation all the time. Animations from Mixamo web page to make the agent move.

0.3 State of the art

0.3.1 Human senses simulation

Hearing

One of the reasons for the failure of NPCs to behave realistically is that many games prioritize computational efficiency over realism. It is not unreasonable to suggest that an

[2]Receptors: Transducers that convert energy from both external and internal environments into electrical impulses

NPC's most common behaviour is to seek out (and often attack) the player character. This seeking behaviour is usually accomplished primarily through visual perception. The NPC's sense of hearing is often ignored, and it acts like a deaf human.

Regarding sound perception, in "The virtual little albert experiment: Creating conditioned emotion response in virtual agents" (Patrick et al. (2015)), the hearing sense is achieved by placing a trigger into the object, with the radius of how far the sound can be heard. If the actor enters the trigger, the object is saved in a list because it is considered as perceived. Another more realistic approach is presented in "Realistic audio AI: Spatial sound modelling to provide NPCs with sound perception" (Cowan et al. (2020)), where it is applied for NPCs a complex alorigthm named GrAF (Cowan (2020)). This algorithm can estimate occlusion and path length attenuation for a human listener. Although the primary goal of GrAF was to render spatial sound for human players, it can be applied to NPCs to create a more realistic reaction.

It could not be found studies to replicate the human sound discomfort in Artificial Intelligence agents or robots. Only studies with humans have been performed to concrete the human thresholds and preferences.

Sight

Sight sense is well developed in robots and artificial intelligence because without it, it is almost impossible to move around an unknown environment. In games and VR is commonly used a straight ray line that comes from the player and has an angle and a range of view (Hubble et al. (2021), Ripamonti et al. (2017)). This, although useful in some games, is not realistic because with a single ray the player can only perceived the information of one of the elements around him.

Other approaches have been made to have an angle of view (Patrick et al. (2015)), where also brightness was included. Also, one interesting feature of this project is that episodic memory was created to establish relationships (in this case, rat and noise).

One of the most extensive projects regarding the imitiation of the sight sense, is the one developed in "Agent vision in multiagent based simulation systems" (Kuiper and Wenkstern (2015)), where it is created a realistic sight sense while maintaining a low execution time. To achieve this, the author uses the following rules:

- The agent does not have access to global knowledge.
- Vision scope is modelled as a cone to emulate human field of view.
- Uses a dynamic vision cone that changes when the agent moves.
- Eliminates unseen items quickly to increase efficiency.

- Has a vision obstruction algorithm.
- Vision and obstruction is made in 3D.

Regarding the glare discomfort, in " A virtual reality lighting performance simulator for real-time three-dimensional glare simulation and analysis" (May et al. (2020)) it is used the Unified Glare Rating (UGR, (Akashi et al. (1996)) in a VR game to represent the description associated with its results. The subjective evaluation of discomfort glare is performed using deBoer's Scale, a nine-point scale from 1-Unbearable to 9-Unnoticeable. The UGR equation is as follows:

$$UGR = 8log_{10}(\frac{0.25}{L_u}\sum_i \frac{L_i^2 \cdot \omega_i}{P_i^2}) \tag{1}$$

where: L_i – luminance of glare source i in the direction of observer's eye (cd/m2), ω_i - solid angle[3] of the glare source its seen from the observer's eye (sr), L_u – average background luminance (cd/m2), P_i – position index (Guth's index) for the glare source i.

On the other hand Glare Rating (GR) is used in outdoor spaces and is calculated with the following formula (Sawicki and Wolska (2019)):

$$GR = 27 + 24log_{10}(\frac{10 \cdot L_{ai} \cdot \omega_i \cdot cos\theta_i . \theta_i^{-2}}{(0.035 \cdot L_b)^{0.9}}) \tag{2}$$

where: L_{ai} – average luminance (cd/m2), ω_i - solid angle of the glare source , θ_i is the angle between the observer's line of sight and the direction to the individual glare source L_b – average background luminance (cd/m2).

This lasts two investigations were the most realistic ones found and a simplifaction of the Glare Rating simulation was the one that was tried to simulate.

Thermal Touch

Haptic devices have been historically used to give awareness information to the players, based on thermal touch. As can be seen in "Haptic device demo using temperature feedback" (Silva et al. (2013)), haptic devices can be integrated with a PC mouse to give an extra feedback to the player, and different investigation have been made with Game Controllers analysis, as can be seen in "Comparing Thermal and Haptic Feedback Mechanisms for Game Controllers" (Löchtefeld et al. (2017)).

But more complex examples can be found. ThermalBracelet (Peiris et al. (2019)) is a wearable wrist-worn device . Also, thermal feedback was integrated with pneumatics in

[3]The solid angle is a measure of the amount of the field of view from some particular point that a given object covers

a glove-based VR device (Cai et al. (2020)). PneuMod is a modular haptic device that using silicon bubbles can arrange the device for different parts of the body (Zhang and Sra (2021)).

In the investigation "Enhancing Virtual Immersion through Tactile Feedback" (Ziat et al. (2014)), there were used two stainless steel cups to present thermal stimuli to the participants hand. This investigation refers to the human threshold of pain to limit the maximum temperature of the cups. Another research that takes the threshold temperture of pain into account is "The psychophysics of temperature perception and thermal-interface design" (Jones and Berris (2002)). In order to develop a thermal display, it uses the threshold of pain, being this between 22 and 42°C. Because of the nature of haptic devices, which is to improve the perception in realism in games, it could not be found investigations using the haptic thermal device above 42ºC.

0.4 Information Retrieval by senses

0.4.1 Hearing

As explained in Section 0.2.1, humans have the ability to estimate where a sound is coming from. The information retrieved by the hearing sense is the direction from where the sound comes and the sound itself.

The intrinsic variables of the sound are: Period, frequency, wavelength, sound power, sound pressure and duration. This are the ones than are detected in human. The agent will be take into account the sound power and the distance to the points to calculate the intensity, with formula: $I = W/(4*\pi*r^2)$, where I is the sound intensity (W/m2), W is sound power (W), and r is the distance between the source and a measurement point (m). This intensity could be translate to decibels with 10 dB = 0.000000000001, watts per meter square, being the decibel scale logarithmic. The only value that will be used is "how high is the agent hearing the sound", and how the agent reacts to different intensities.

0.4.2 Sight

The information retrieved by the sight sense from the environment and the relation object-human is: distance to the object, angle of view. The intrinsic variables of the object based on the view perception are: color, shape, size, luminance and visual texture.

About the information retrieval, using the Snellen Chart (Messina and Evans (2006)). Being the scale of the object equals to 1 , everything in 6 metres away will be fully detected.

Regarding how the intrinsic variables affect human, color and visual texture does not affect if human receive damage or not, and size and shape won't be taking into account because they can affect the agent only if it interacts with the objects directly.

Luminance, as explained in 0.2.1 is a photometric measure of the luminous intensity per unit area of light travelling in a given direction. The final luminance perceived by the agent is the sum of the natural luminance of the object + the brightness, which is the reflecting light.

0.4.3 Thermal Touch

The most specific sense developed, the thermal touch detects the temperature elements that are in contact with the skin. Two main strand could be differentiated: the object temperature detection and the temperature felt by thermal radiation.

The object temperature is influenced by thermal properties of the material. When the skin is in contact with the object, the thermoreceptors detect changes in the skin temperature. Cold receptors respond to decreases in skin temperature over a range of 5-43 °C, and the warm receptors response at approximately 45 °C . Between 30-36º, there is no awareness of cold or warmth because it is the normal range for skin temperature (Jones (2009)). These receptors work the same way when the contact is with air and not with a solid object. This is what is called thermal radiation.

Above 45º degrees and below 15º, the pain receptors start to respond. At this point, an excessive amount of time exposed to these temperatures can lead to irreversible damage to the skin (Stoll and Greene (1959)).

The temperature stimuli is influenced by the previous environment. Because of this, it is necessary to define an environment with a static temperature. In this research, the thermal touch was used to identify the temperature felt by the agent in contact with the air.

Variables information

To create a realistic information to influence in the agent drives, human real sensations have been used. Regarding the sound intensity, a sound greater than 85dB can result in hearing loss. The permissible noise exposures given by the United States Department of Labor is related with "Ocupational Noise". On the other hand, the permissible noise exposures given in "sociocusis"—hearing loss from non-occupational noise exposure." (Cohen et al. (1971)) is related with "Non-Occupational Noise". As can be seen in Figure 1, the Occupational Noise can be endured longer than Non-Occupational Noise.

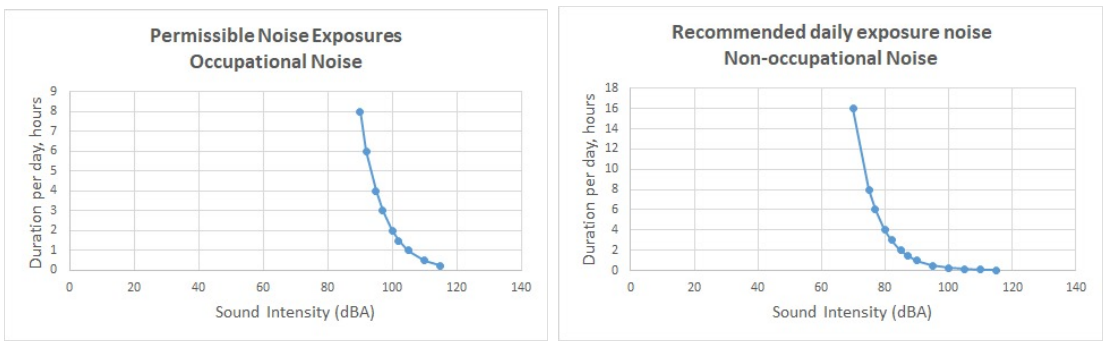


Figure 1: Permissible noise exposures for Ocupational and Non-Occupational noise

In relation with glare, Subjective studies were carried to related GR and de Boer Scale. The results are collected in "The unified semantic glare scale for GR and UGR indexes." (Sawicki and Wolska (2016)) and at Figure 2 is shown the relationship between GR and de Boer Scale.

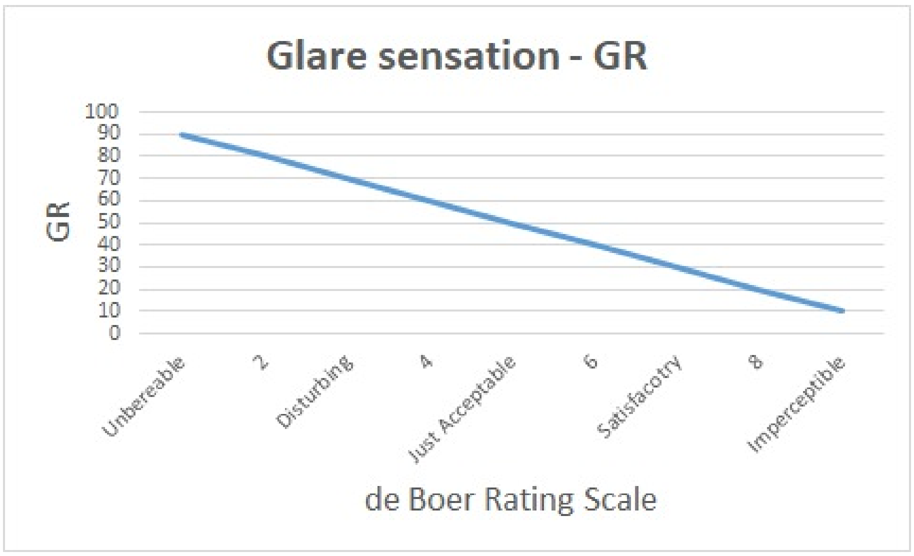


Figure 2: Glare Rating related with deBoer Scale

And the last curve, the temperature, was extracted combined information from " Thermal touch" (Jones (2009)), "Effect of ambient temperature on human painand temperature perception" (Strigo et al. (2000)), while being the basis of the curve from "The nature of pain" (Hardy (1956)). At figure 3 it's shown the relationship between the temperature in °Cand the time exposed to it before irreversible tissue damage.

All the tables with the data that was used to create the curves can be found in the Appendix, in section .1.

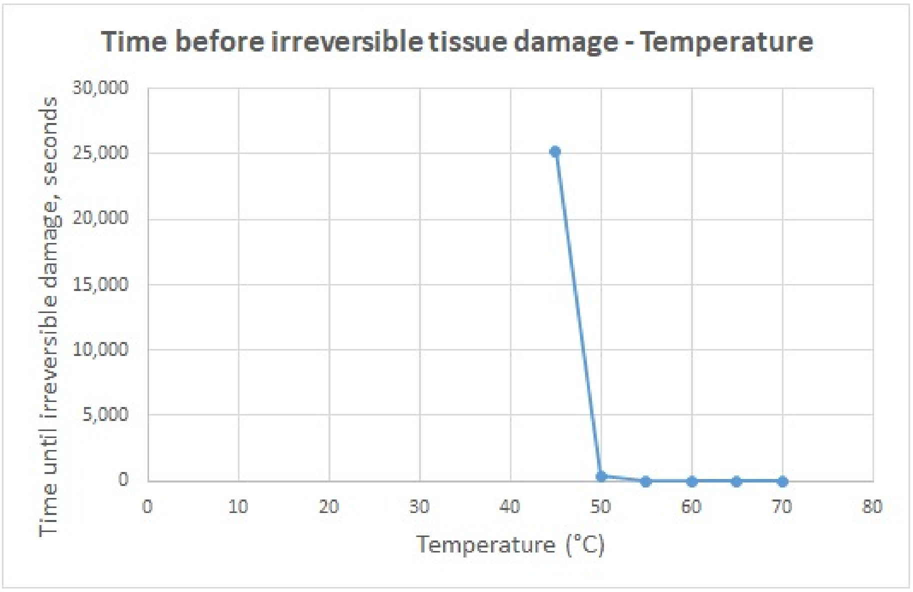


Figure 3: Time exposed to certain temperature before irreversible tissue damage

0.5 Methodology and Experimental design

The first part of the project was to do a research, as extend as possible, to find the information explained above in section 0.4 in order to be able to translate this data into curves of pain. This pain received will be translate into actions, based straightly with the human need "Health".

To be able to traduce real human sensations into an agent, a 3D-environment was created. This 3D world is as much realistic as possible. The first part of the experiment is the testing of the curves created. To do this part, some objects have been placed in the 3D world, and the user will be able to move the agent around the map and in every position will be able to check how the variables are affecting the health of the agent.

In the second part, the agent will have to act depending on the stimuli that is receiving throw it senses while it tries to fulfill his human needs. Same 3D environment and objects are going to be used for this part, but instead of hardcoding [4] all the processes and decisions make by the agent, a Machine Learning technique is used: Reinforcement Learning. This one was choose, not only because it is a well-known technique to train intelligent agents, but also because this type of learning is more like the process of human or animal learning, and it fits exactly what we are looking for: realism.

0.5.1 Reinforcement Learning

The key element of realism in games is the behaviour of computer-controlled characters. Reinforcement Learning is widely used to train the NPC in which an agent acquires experience through continuous interaction with the environment (Wang et al. (2009), Merrick and Maher (2006)) and is based on the Markov decision-making process.

Markov's decision-making process (MDP) is a continuous transition process of the state in which the current state and the next state are not associated. MDP is used to solve the problem of control or decision, and the two key factors are *the agent*, which interacts sequentially over time with the *environment* where it is. At each time step, the agent receives a *state*, the environment representation at that time. After receiving the *state*, the agent decides an *action* to execute. After executing the action and go the next state, the agent receives a *reward*, based on the new *state*, and as a consequence of the performed *action.* In summarizing, the components of an MDP are;

- Agent.
- Environment.

[4]Hardcode: In this context, to define exactly what the agent has to do

- State.
- Action.
- Reward

This process is an essential part of the implementation after the research has been done. In the following sections, MDP is explained in detail. Its understanding is crucial for the development of the investigation.

MDP notation

In an MDP, there is a set of states $\boldsymbol{S}$, a set of actions $\boldsymbol{A}$ and a set of rewards $\boldsymbol{R}$ At every time step t, the agent receives the state $S_t \in \boldsymbol{S}$.

Based on the state received, the agent execute an action $A_t \in \boldsymbol{A}$. With both of them, it is created a state-action pair (S_t,A_t).

Then, time is incremented to t+1, so the environment transits to a new state $S_{t+1} \in \boldsymbol{S}$. This is the moment when the agent receives the reward $R_{t+1} \in \boldsymbol{R}$ for the state-action pair (S_t,A_t). The goal of the agent is to maximize the sum of rewards. Figure 4 illustrates the steps and the process explained above.

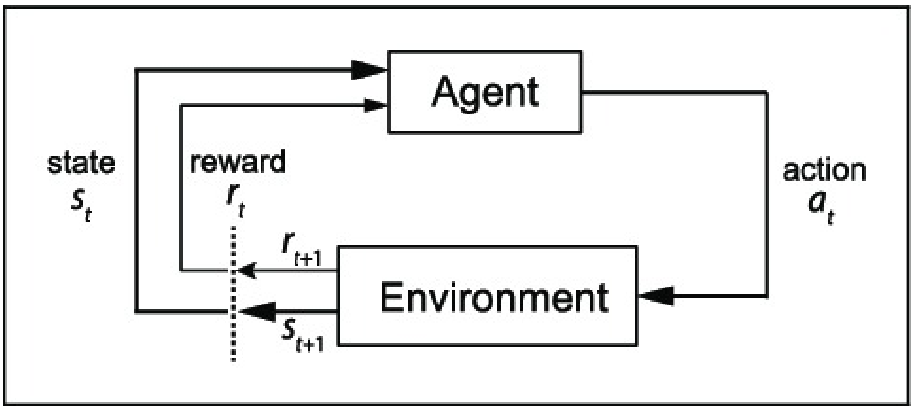


Figure 4: MDP illustration. By Galatzer-Levy et al. (2018)

Transition probabilities

Because sets $\boldsymbol{S}$ and $\boldsymbol{R}$ are finite, all the values assigned to R_t and S_t have some associated probability.

If $s' \in \boldsymbol{S}$ and $r \in \boldsymbol{R}$, there is a probability that $S_t = s'$ and $R_t = r$. The probabilty is determined by the pair of state-action (s, a), where s is the preceeding state and $\in \boldsymbol{S}$ and action $a \in \boldsymbol{A(s)}$. $\boldsymbol{A(s)}$ are the possible actions that the agent can perform at the state s.

For all *s'* $\in$ $\boldsymbol{S}$, $s \in \boldsymbol{S}$, $r \in \boldsymbol{R}$ and $A \in$ ***A(s)***, the **transition probabilities** to state *s'* with reward r taking action a in state s as:

$$p(s', r|s, a) = P_r \{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{3}$$

Discounted return

But not only the immediate reward is important. It is also used the discounted rate γ, a number between 0 and 1, that determines the discount that it is make for future rewards. The bigger the discount rate, less important the future rewards. The *discounted return* is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Applying this, we should keep in mind that now, the goal of Reinforcement Learning is not to find the biggest reward at time *t*. Now, what we are looking for is the biggest *discounted return* of rewards.

Policies Value Functions, and Bellman Optimality equation

Two more components are missing in order to understand Reinforcement Learning: Policies π v_π and Value Functions. Policies address the question *How probable is it for an agent to select any action from a given state?* and Value Functions address the question *How good is any given action or any given state for an agent?*.

The policy π is a function that maps, for each state, the probabilities of selecting each possible action. When talking about policies, it is said that an agent "follows a policy". $\pi(a|s)$ is the probability that $\boldsymbol{A}_t = a$ if $S_t = s$. It describes the decision-making process of the agent and represents a probability distribution for every state over all possible actions.

The Value Functions are functions of states or state-action that estimate how good it is to be in a specific state or to execute an action being in a specific state, depending on the expected return. The rewards the agent can expect to receive in the future depend on what actions it will take. Accordingly, value functions are defined with respect to particular policies and are denoted as v_π.

The state-value function for policy π gives the value of a state under π which is the expected return from starting from state s at time t following policy π. This is defined as:

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$$

The action-value functions is the expected return of starting at state s at certain moment of time, performing action a and following policy π. Is denoted as q_π. and defined as:

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

The action-value function q_π is called Q-function, and the output from the function for any given state-action pair is called a Q-value. The letter "Q" is used to represent the quality of taking a given action in a given state. This is important because the algorithm used during the training and explained in 0.5.2, is straightly correlated with Q-values.

With this in mind, although the final goal of Reinforcement Learning algorithms is to find the biggest discounted reward, another goal has been added. The algorithm has to find a policy that leads him to a lot of rewards. What the algorithm is looking for is the **the optimal policy**

The policy π is better than π' if the expected return is better than the expected return by π'. Following this rule, it can be found the **optimal policy**.

The optimal policy is associated with an optimal state-value function, denoted as v_* and an optimal action-value function, denoted as q_* . The optimal state-value function is the one that gives the largest expected return by any policy π for each state. The optimal action-value function is the one that, for each possible state-action pair, gives the largest expected return achievable by any policy π. q_* has a fundamental characteristic: it must satisfy the **Bellman Optimality Equation** 4

$$q_*(s,a) = E[R_{t+1} + \gamma \max_{a'} q_*(s',a')] \tag{4}$$

The equation states that for a *(s,a)* at time t, the expected return from starting at state s, taking action a, and following the optimal policy, will be equal to the expected reward R_{t+1} in addition with the maximum of expected discounted return that can be achieved from any possible (s',a').

Considering the agent following an optimal policy,the latter state *s'*, will be the state from the agent can take the best possible next action *a'* at time *t+1*.

This equation is a main part of the project, because with it, the optimal q_* can be found to find the optimal policy π, because when q_* is found for any state s, any reinforcement learning algorithmcna find the action a that maximizes q_***(s,a)**.

The Bellman Equation is the one that it will be followed during the all training process, using Q-Learning algorithm.

0.5.2 Q-Learning

The most well-known algorithm related with Reinforcement learning, an the one chosen to train the algorithm. Its objective is to learn the best Q-values for each action-state pair to find the optimal policy.

Q-Learning was the algorithm chosen because it is a direct method that use an straightforward implementation of Bellman-Equation, as explained in "Algorithms for Reinforcement Learning" by Morgan and Claypool Publishers (Szepesvári (2010)). Also is a RL algorithm "typically easier to implement". Just remark that the main goal of the project is to create an agent able to detect its surrounding as a human would do, and detect how its surrounding are affecting them. Because of that, facilitate the implementation of RL with a basic algorithm was a key factor for the well development of the project.

Using Bellman Optimality equation, the algorithm uses the process called *Value Iteration.* The algorithm update iteratively the Q-values for each (s,a) pair, until the Q-function converges to the optimal Q-function q_*.

The Q-values are actualized in every step using the **Q-Table**. In the table are stored all the Q-values for each (s,a) pair. The dimensions of the table are the numbers of actions (horizontal axis) by the number of the states (vertical axis).

Process

With the set of actions and the environment defined, the Q-table has to be initialized with the initial values. These values are pre-defined depending on the project being undertaken, and the strategy that will be folowed, explained in section 0.5.2 After having everything initialized, a number of episodes has to be set. The episodes are the number of times that the agent will play. Every episode has a defined number of steps[5]. Every episode could be finished in two situations: if the agent has arrived to a pair *(s,a)* defined as defeat or victory, or if the agent has execute the maximum steps per episode.

In each step, the agent takes the action with the highest value in the Q-table for the current state. If every Q-value is predefined with the same value, an strategy has to be followed in order to teach the agent how to discover which one is better. To define how the agent choses actions, a balance between *exploration*[6] and *explotation*[7] is done, following the **Epsilon Greedy Stategy**

[5]Every execution of an action is counted as a step
[6]Exploration: Find information of the environement by exloring it
[7]Exploiation: Using the information already known to maximize te return

Epsilon Greedy strategy

To get the balance between eploration and exploitation, an *eploration rate*(ϵ)between 0 and 1 is defined. The exploration rate is the possibility to take a random action instead of taking the action with the best Q-value. For example, if ϵ is set as 1, all the actions will be random. On the other hand, if ϵ is set as 0, all the action will take into account the Q-Table. The idea is to start with an ϵ big enough to explore the enviornment, but if ϵ is too high, the agent will lose time exploring. On the other hand, if ϵ is too low, the agent could not learn all the possible pair *(s,a)* and consequently, miss the best Q-Value possible. After every episode, it is actualized using the *epsilon decay* variable.

Q-Learning equation

Bellman Equation 4 is used to update the Q-value, but one more variable is introduced: the *learning rate*(α). This variable set the importance given to the future rewards. α determines how quickly the algorithm actualize the Q-value in the Q-table. The higher the learning rate, the more quickly the agent will adopt the new Q-value. The final equation to calculate the new value is:

$$q_*(s,a) = (1-\alpha)q(s,a) + \alpha[R_{t+1} + \gamma \underset{a'}{max}\, q_*(s',a')] \tag{5}$$

Being *q(s,a)* the old value and *q(s',a')* the new value.

0.6 Implementation

0.6.1 Environment

The environment is basic in order to prove that the agent could feel throw its senses and learn how to act depending on how it is being affected. It is created using Unity Engine. The agent model and their animations are from Mixamo.

Although the construction in Unity is in 3D, the movement and the positions in the scene are made in 2D. The y-axis will be considered 0 for all the positions and 1 for all the 3D objects. In addition, the movement and the positions are using only natural numbers. This was made to simplify the possible states in the Experiment 2, 3 and 4, Section 0.7.2 using RL. Finally, rotations possible were set only for 0°,90°,180° and 270°

Objects

In the scene for different objects are found: **the Food**, **the Speaker**, **the Bulb** and **the Fire**, and all of them are from Turbo Squid. Some of them were modified using Blender.

Being the objects a Game Object, all of them follow the same parent-child structure:

- The parent, which is an empty Game Object, with an scale of 1.
- The first child, which is the 3D model.
- The second child, which is the radiant temperature collider. Two type of temperatures have been taken into account to create the temperature collider:
  - The first temperature, and the more typical, is the ambient temperature. The ambient temperature is set when the programm starts to run.
  - The second one is the radiation temperature [8].

  Based on thermal radiation and the object temperature [9], an equation has been made in order to make an approximation of the temperature felt by the agent.

  Thermal radiation is calculated by the formula:

  $$T_R = T_o - d \cdot r \tag{6}$$

  Where d is dispersion variable, r is the distance in units, T_O is the object temperature and T_R is the radiation temperature.

  Equation 6 is used until temperature reach ambient temperature.

  If at distance r, thermal radiation temperature does not reach ambient temperature, the final temperature is the maximum temperature detected at that point.

  The radius collider is defined following the equation 6.

- The third child is the light component, strictly related with the light intensity, which is take into account to calculate the glare felt. Luminous intensity is calculated taking into account the Light Component from Unity. Because the Light Component differs between range and intensity, but not in the real-world[10], the decision was to have intensity and range components always equals, being the final luminous intensity as follows as follows:

  $$L_i = ((Intensity + Range)/2) * 1000 \tag{7}$$

[8]Thermal radiation: Electromagnetic radiation generated by the thermal energy. All matter with a temperature greater than absolute zero emits thermal radiation

[9]Object temperature: Temperature that the agent felts if he is exactly in the object position

[10]Luminous range is determined by the luminous intensity of the light, the atmospheric transmission factor and the threshold of iluminance on the eye of the observer (2-1-390) source: the International Dictionary of Marine Aids to Navigation

The logic behind this equation follows the quest of realism. First, the sum and the division by two, so that light in the Unity Scene can be seen intense enough to glare the human sight, based on a subjective view from the author. In addition, the multiplication by 1000 was done to reach luminous intensity enough to affect the human sight, based on the experiments "Model predicting discomfort glare caused by led road lights" (Lin et al. (2014)), and "Predicting discomfort glare from outdoor lighting installations" (Bullough et al. (2008)).

- The fourth child, the sound collider. It's radius is calculated by the inverse square relationship, which is the relationship that follows the sound intensity felt by the agent [11]. The formula to calculate sound attenuation is:

  $$L_p(R2) = Lp(R1) - 20 \cdot Log10(R2/R1) \tag{8}$$

  Where: $L_p(R1)$ = Known sound intensity at the first location (typically measured data or equipment vendor data). $L_p(R2)$ = Sound intensity at the second location. R1 = Distance from the noise source to location of known sound intensity. In this project, 0.01u. R2 = Distance from noise source to the second location.

Finally, to recognize the objects, all of them have a *Tag* on it, related with the type of objects they are: Fire, Food, Bulb or Speaker. This Tags are useful for object recognition and to be able to classify and find the objects using Classes.

All the object are placed above a Board, being finally the world as shown in Figure 5

Environment classes

Two classes were created to control the initialization of the objects variables and components and the organization and the access to the objects variables. All the objects have the class *ObjectClass*. This class is added as a parent component for every object in the scene, and is used to control the variables' intialization values and its access. This class inherit from Mono Behaviour[12] class. When the game starts, *Awake()* method is called. Checking the Tag to detect which type of object has the class as a component, the initialization of the variables are placed there. The variables initialized are:

- Temperature: struct with two variables on it, celsius and dispersion.
- Object type: Equals to the object tag.
- Light Intensity in cd/m^2.

[11]Object sound intensity: Sound felt by the agent if he is at the same position as the object
[12]MonoBehaviour class: The base class from which every Unity script derives.

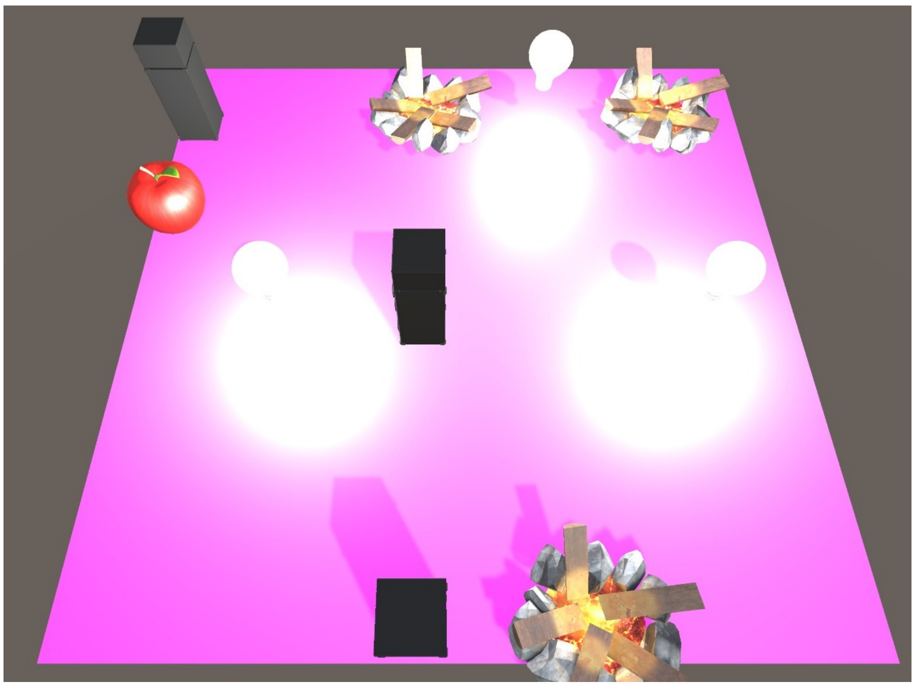


Figure 5: World with objects placed

- Sound Intensity, in dBA.
- Temperature collider: Reference to the child component Temperature Collider.

Also, the sound collider radius is set in the Object Class, but no variable is needed for that. On the other hand, the temperature collider needs a reference because it depends on the ambient temperature, defined in the *World Class*.

To have a control of the access, *WorldClass* has been created, and added as component of the *Board* GameObject. Inside it are the following variables:

- Ambient Temperature, set to 25℃.
- Rows and columns. Initialized depending on how big is the world created.
- A list with the *ObjectClass* component of all Game Objects in the scene.
- A list with the possible positions.

When the game starts, *Start()* method is called, and two functions are used:

- *initPossiblePositions()*: This function was in charge to intialize the possible positions list, without having into account the objects placed above the Board. A double loop, using the number of rows and columns, was made for adding every world position on it.

```
        for (int i = 0; i < rows; i++)
        {
            for (int x = 0; x < cols; x++)
            {
                possiblePositions.Add(new Vector3(x, 0, i));
            }
        }
```

Althought the position were set in 2D, it was decided to save the possible positions as a Vector3. This was to make easier a possible change from 2D to 3D.

- *initObjects()*: This function was in charge of three things: delete the objects positions from the possiblePositions list and set the objects temperature collider. Add The temperature colliders were set here because they depend on the ambient temperature, and the ambient temperature is defined by the World. The objects placed on the scene were found using *FindGameObjectWithTag() method*, which saves in an Array.

```
GameObject[] fireTag = GameObject.FindGameObjectsWithTag("
   Fire");
        GameObject[] bulbTag = GameObject.
           FindGameObjectsWithTag("Bulb");
        GameObject[] foodObjectsTag= GameObject.
           FindGameObjectsWithTag("Food");
        GameObject[] speakerObjectsTag = GameObject.
           FindGameObjectsWithTag("Speaker");

        for (int i = 0; i < fireTag.Length; i++)
        {
            position = new Vector3(fireTag[i].transform.
               position.x, 0, fireTag[i].transform.position.
               z)
            possiblePositions.Remove(position);
            colliderRadius = thermalRadiationEq()
        }

        for (int i = 0; i < foodObjectsTag.Length; i++)
        {
            Vector3 position = new  Vector3(foodObjectsTag[i
               ].transform.position.x, 0, foodObjectsTag[i].
               transform.position.z)
            possiblePositions.Remove(position);
            modifiableObjects.Add(foodObjectsTag[i].
               GetComponent<ObjectClass>());
        }
        for (int i = 0; i < bulbTag.Length; i++)
        {
            Vector3 position = new  Vector3(bulbTag[i].
               transform.position.x, 0, bulbTag[i].transform
               .position.z)
            possiblePositions.Remove(position);
            float colliderRadius = thermalRadiationEq()
        }

        for (int i = 0; i < speakerObjectsTag.Length; i++)
        {
            Vector3 position = new  Vector3(
               speakerObjectsTag[i].transform.position.x, 0,
                speakerObjectsTag[i].transform.position.z)
            possiblePositions.Remove(position);
        }
```

Noted here that *ObjectClass* is initialize using *Awake()* and *WorldClass* is initialized using *Start()*. This is because *Awake()* method is called before *Start()* method, and the variables of the objects must have been initialized to modified them. If the same method is used, it will cause a *Null Reference Exception*, because an attempt would be made to modify the collider, without first initialising it.

During Experiment 1, explained in section 0.7.1, also the name of the object were checked in order to pre-defined the object variables.

0.6.2 Stimulus curves

As it can be seen, each object is directly related with one of the stimulus that are going to be felt by the agent: *Temperature*, *Glare* and *SoundIntensity*.

The data collected presented in 0.4.3, had to be transformed in a curve in order to transform the sensation felt by the agent into a number that represents how the agent is being affected by that attribute.

To do that, the first thing was to set a time for the decision making between actions to answer the following question: how is the agent affected by the variables, during this time in which it did not do nothing? If the time is not set, the information makes no sense. After considering several options, the time "to think" between actions was set to 1 second. This selection of time was because it would be easy to change the time between actions by multiplying the result obtained for the new time set.

The next thing that had to be clear is, the meaning of "being affected". How is the agent "being affected"? It is about pleasure? About discomfort? The choice for this project was to use something tangible. The curves created try to answer the question: how much time does the agent has to be exposed to be harmed irreversibly? Or in another way, how much time can the agent be exposed to a certain temperature/glare/sound intensity to be damaged and not be able to recover naturally?. Noted here, harm is about physical damage, not about pain, although both of them have been used. In the next section, "damage" variable will be used as the one to control how much damage can receive the agent in one second, being one-hundred the maximum damage that the agent can receive before be harmed irreversibly.

Temperature curve

The data collected and shown in 3 is exactly what it is looked for in this project. How much time could the agent be exposed to a temperature, having the same senses that a human being? The information regarding temperatures below 50℃ was discarded to do it, because the difference[13] between the information collected for 45℃and 50℃ was too big. Because the time was set to 1 second, the curve was made between 50℃ and 65℃. Remeber that a human exposed during 1 second to a 65℃ will suffer an irreversible tissue damage. More than 50℃ is not clear how much damage the human can receive, but as explained in "The nature of pain" (Hardy (1956)), the irreversible damage would be produced instantly. The final curve is represented in Figure 6. A potential function has been used to fit as much as possible to the real data. The square error (R^2) and the function equation are shown in the center of the figure

Glare curve

Data about the relation between GR and blindness or eye damage was not found. This is why in that case, an approximation has been done in order to create the curve. The goal of this curve is to not look straight to places where it receives a glare above 90. The damage variable is incremented since receiving a GR of 60, which is between Just

[13]In this context, mathematical substraction. Not about inconsistency

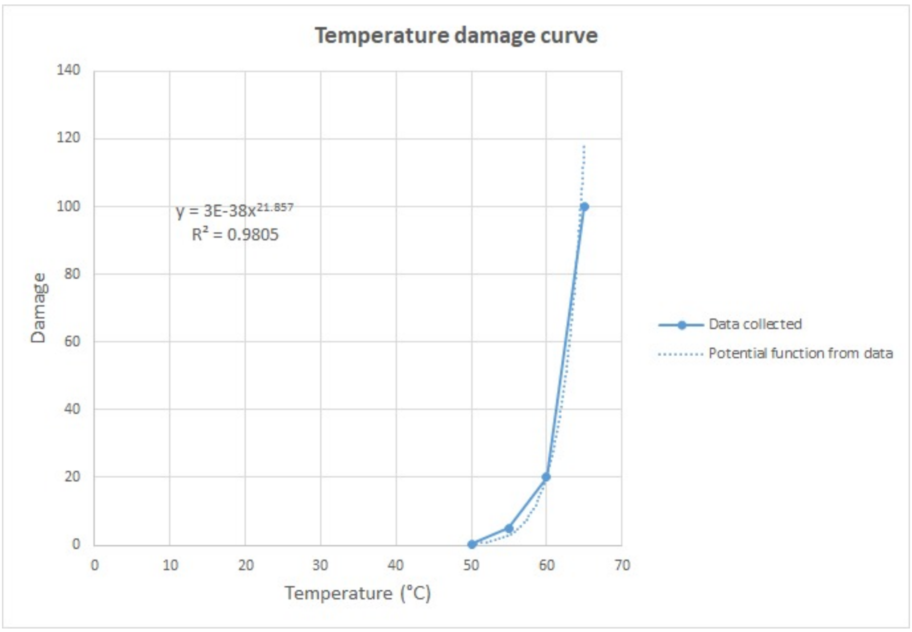


Figure 6: Temperature curve.

Acceptable and Disturbing in the deBoer Scale. Because no relation was found between GR and damage it was decided to follow a lineal equation, the same was done for the damage curve. The final GR curve is shown in Figure 7. A potential function has been used to fit as much as possible to the real data. The square error (R^2) and the function equation are shown in the center of the figure.

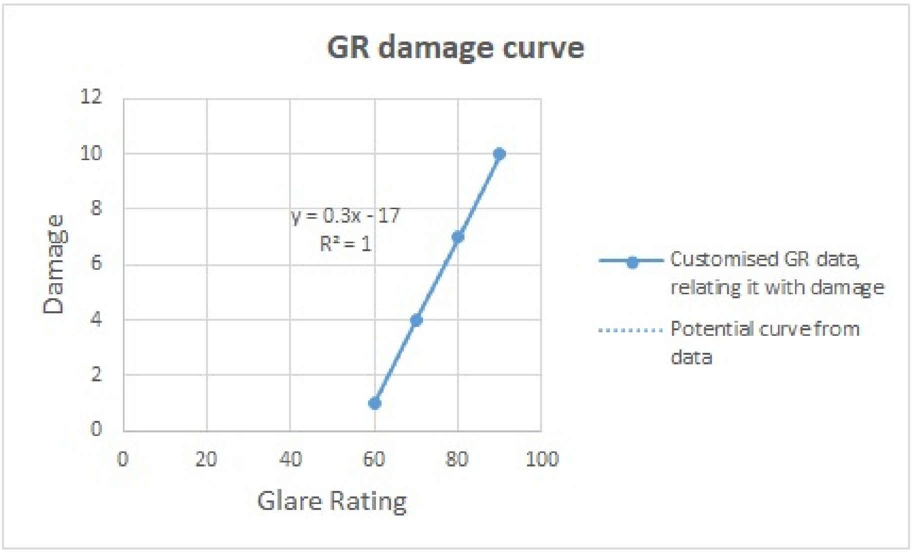


Figure 7: Glare curve.

Sound Intensity curve

Because the project is not related with occupational noise, the data from "“sociocusis”—hearing loss from non-occupational noise exposure.” (Cohen et al. (1971)) was the one used to create the curve. After checking the data, it was seen that it follows a relation that exposure limit is inversely proportional to sound intensity: the daily exposure limit decreases by a factor of 2× for each Sound Intensity = 5 dB-A increase. This relationship was used to increase the data used in the curve. Finally, it was seen that the maximum sound intensity value felt by a human in one second is 154.7 dBA. If 100 of damage is produced after suffering sound intensity of 154.7 dBA, and following the relationship explained above, the result curve can be created Figure 8. A exponential function has been used to fit as much as possible to the real data. The square error (R^2) and the function equation are shown in the center of the figure

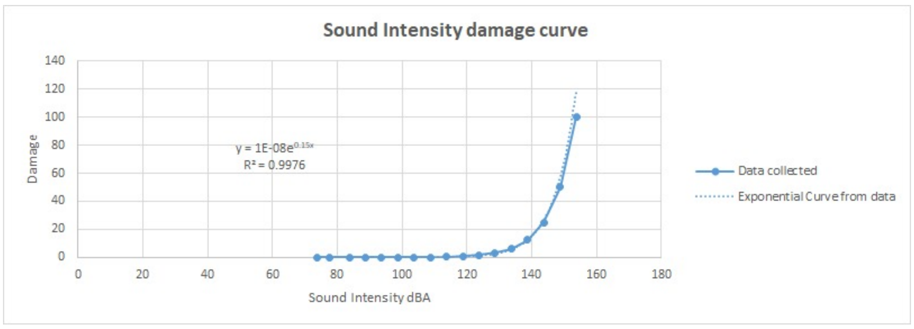


Figure 8: Sound Intensity curve.

0.6.3 Senses implementation

The senses were developed using Unity Engine and programming with C# (Hejlsberg et al. (2008)). Also, a Senses Control class has been developed to manage:

- The sight sense to recognize objects, detect the distance, the angle of view and the glare that is felt.
- The hearing sense, to hear the different sounds and the intensity received.
- A basic touch sense, which is able to feel the temperature received.

The three senses have one list which stores the objects that can feel by that sense.

It should be noted that even though the environment was simplified to be used in 2D, with natural numbers and with limited rotations, the senses developed are prepared to take every real number in consideration.

Sight Sense

The sight sense is developed following "Agent vision in multi-agent based simulation systems" Kuiper and Wenkstern (2015), explained in section 0.3.1. In order to detect the objects, the following steps were followed:

- Use https://docs.unity3d.com/ScriptReference/Physics.OverlapSphere.htmlOverlapSphere. This method detects and stores in a list all the objects that are inside the sphere created around the player. The sphere radius was set to 6u based on the Snellen Chart (Messina and Evans (2006)). The *OverlapSphere()* circle can bee seen produced in Figure 9.
- Check restrictions for every object stored:
  - Is the object inside the angle of view? The angle of view was set to 60º, equals to the central field of view for humans (Strasburger et al. (2011)).
  - Is any other object blocking the view?

. If the object is inside the angle of view and is not being blocked by any other object, it will be saved in the list, which stores all the objects detected by the sight list. In Figure 9, it can be seen how the object blocked is being discarded (red ray) and which objects can be seen by the agent (green ray).

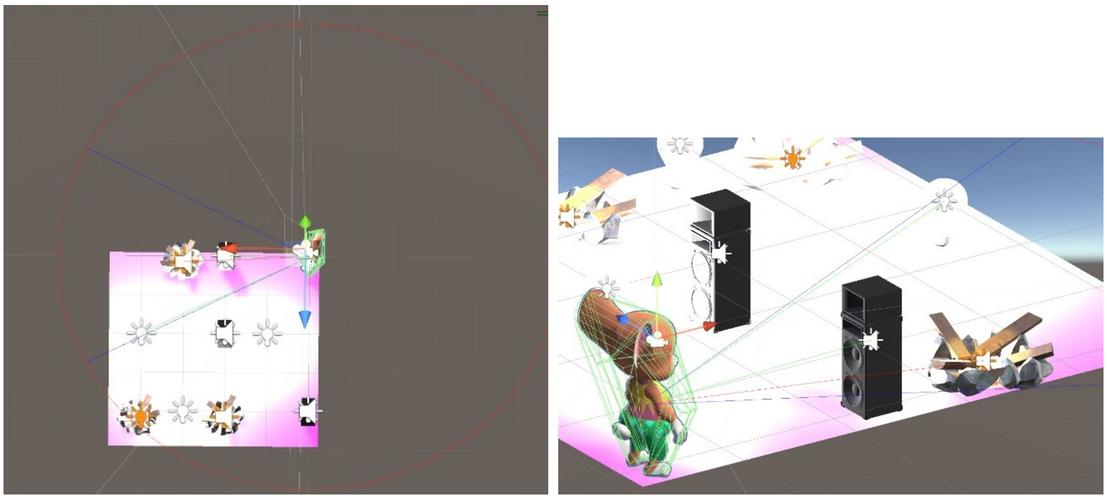


Figure 9: The Sight sense

When all the objects that can be seen have been stored, the GR is calculated for all of them, taking into account the **luminance** variable. To calculate glare, instead of using UGR or GR as explained in 0.3.1, an approximation to take into account luminous intensity and distance has been made. Two experiments, " Model predicting discomfort glare caused by led road lights" (Lin et al. (2014)) and ""Predicting discomfort glare from outdoor lighting installations (Bullough et al. (2008)), were taken as reference to create an equation that finally result in 9:

$$GR = (L_i/(d * 115)) * \cos a \tag{9}$$

Being L_i, the luminous intensity, defined using equation 9 d is the distance between the agent and the object, and a the angle between the agent and the object.

A factor of 115 is multiplied by the distance. This number was chosen after some experiments that have been done before, taking the investigations cited to create equation 7, as a reference. The aim was to reach realistic luminous intensity values taking into account the difference that distance can cause. Moreover, the result is multiplied by the cosine of the angle. This final addition was to de-emphasize the luminous intensity if the angle between the agent and the object increases.

Hearing and Thermal touch senses

Hearing and touch senses are developed using the same logic shown in "The virtual little albert experiment: Creating conditioned emotion response in virtual agents" (Patrick et al. (2015)). Both senses are achieved using their respective colliders as triggers: the radiant temperature collider and the sound collider. When the agent enters the collider area, the object is saved in the sense list, which means that the agent can feel the object's variable (temperature or sound intensity). At the moment that the agent leaves the area, the object is removed from the sense list. This can be made using *OnTriggerEnter()* and *OnTriggerExit()*. Finally, *OnTriggerStay()* was used to update the distance between the object and the agent.

In figure 10, the colliders were set after initialization using equation 8 and Sound Intensity = 115 dbA for the distance and equation 6, temperature = 75 ℃, and dispersion = 15 for temperature variable. The sound collider radius is 5623.4, and the temperature collider radius is 3.33334. Noted how far the agent should go to stop hearing

0.6.4 Agent Learning

The agent architecture is divided into two parts: the agent sensations, already developed and explained in Section 0.6.3 and the agent logic created using Pycharm, that uses Python programming language.

The decision to use Python language was encouraged by the intention to facilitate further research in the future. The standardized environment for Reinforcement learning

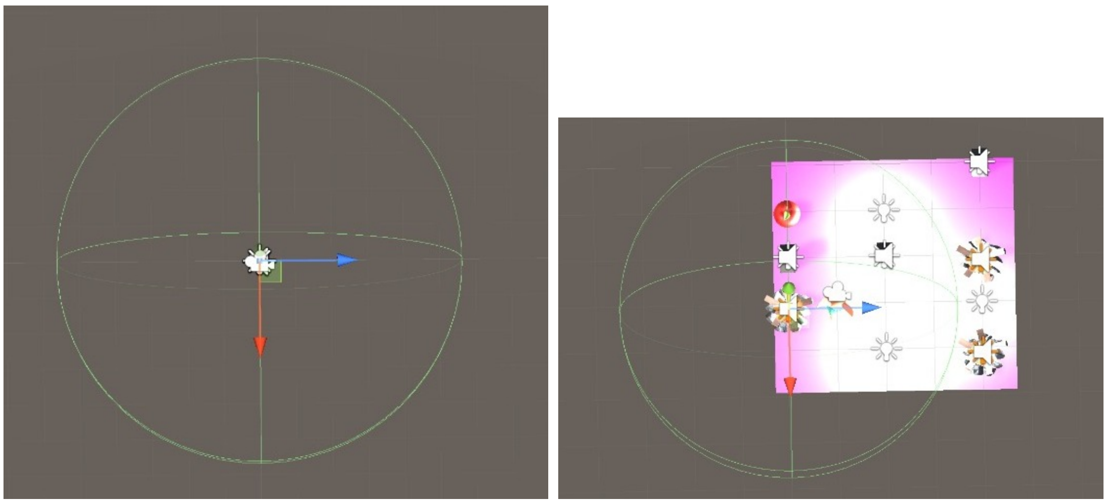


Figure 10: Colliders set after the initializaiton.

is OpenGym AI, a software library that "aims to combine the best elements of the previous benchmark collections, in a software package that is maximally convenient and accessible" (Brockman et al. (2016)). This software had already been the model for developing new projects like "The Optical RL-Gym: An open-source toolkit for applying reinforcement learning in optical networks" (natalino2020optical), "robo-gym – An Open Source Toolkit for Distributed Deep Reinforcement Learning on Real and Simulated Robots" (Lucchi et al. (2020)) and more. The OpenGym AI is based in Python programming language, and if the agent intelligence is developed using Python, it could facilitate future researchers to future implementation of the project presented.

The communication was developed using sockets which provide point-to-point, two-way communication between two processes. This, as explained in "Socket Programming" (Kalita (2014)), is a Client-Server communication [14][15]. In this project, it was decided that Unity would provide the service, and the Client would be created using Python. This decision was made thinking that Unity was the one offering the service (the environment and the agent) and the Artificial Intelligence was the one requesting them. This is not locked and could be changed in future projects. The protocol used for the communication was Transmission Control Protocol (TCP). As explained in "Transmission control protocol" (Postel (1981)), the main purpose of TCP is to be a highly reliable host-to-host protocol, and this is why it was chosen.

In both Unity and PyCharm, a CommunicationClass was created to control the information sent between the agent and its intelligence. The agent sent to the intelligence its surroundings detected, what it was feeling and how its needs were being affected. The

[14]Server: Program that is offering some service
[15]Cient: Program that is requesting some service

intelligence sent to the agent the action that was decided to do, taking into account these stimuli and how the changes in its needs were affecting its goals. Because the communication created is out of the scope of this investigation, more information can be found in the Appendix .3.1.

Following the structure of the paper "A Cognitive Architecture for Agent-Based Artificial Life Simulation" (Vieira et al. (2018)), it had already been defined the semantic representation and the senses and perceptions. Still, it had to be defined:

- Needs: The artificial intelligence action choices are aimed at its needs. Instead of take all Maslow human needs explained in Section 0.1, have only been used: **feeding** , **health**. These needs are the agent **drives**.
- Goals: The agent's goal is to survive as much time as possible.
- Agent interactions: The possibles actions executed by the agent were:
  - Up: Move one positive unit in the Z world axis.
  - Rotate Up: Look to the positive Z world axis
  - Down: Move one negative unit in the Z world axis.
  - Rotate Down: Look to the negative Z world axis.
  - Right: Move one positive unit in the X world axis.
  - Rotate Right: Look to the positive X world axis
  - Left: Move one negative unit in the X world axis.
  - Rotate Left: Look to the negative X world axis.
  - Idle: Do nothing.
  - Eat: Tries to eat

  Every action is instantly executed. The only stimulus that the agent received were the ones during one second between actions, as explained in Section 0.6.2. The actions are restricted by the possibility of doing it or not. The agent could always execute rotate and idle actions. On the other hand, *eat* and movement actions were restricted. Regarding the movements, if the agent had an object in the place where that action led, that action was classified as "impossible to execute". This means that in that position, the agent could not even try to execute that action. For example, if the agent is in the position (0,0), he could not execute the *up* action if there was an object in position (0,1). Finally, eat action was only possible to execute if the agent had an object in front of him. It could try to eat every type of object, not only the food, but it was restricted from executing the *eat* action if in front of him was nothing.

Execution implementation

The mind has two major tools: Memory and the reasoning cycle. In this project, the memory was the Q-table, and the reasoning cycle was the Q-learning algorithm. During this section, PyCharm implementation will be called "the mind", and Unity implementation will be called "the environment".

The first thing to do before initializing the Q-table was to start the connection. When the game is initialized, the connection must be established, and the starting information is shared, using *initCommunication()* method. This method was in charge to established the communication and sharing two attributes and one list:

- Avatar position.
- Avatar rotation.
- Possible positions.

These three variables were sent to the mind. At that moment, the mind could use those three variables. As can be noted, a trick is done that does not follow human reality. It is impossible for a human being to know the position where it is on the world scale and all the positions where he can move. This decision follows the same rule as the choice of Q-Learning algorithm: simplify the learning process.

After the connection had been established, the mind received his drives, being 100 for health and 0 for hunger. It also receives the objects that are being recognized using the sight senses. At this point, the Q-table should be initialized. In this project, following the quest of realism again, the table is being actualized after every action execution instead of creating all the states from the beginning. Although the agent knows where it can move because of the trick explained above, it can not save the state in its memory until it has been on it. After receiving the initial variables, the agent can start to train. The exact process is followed during all the training.

Training process

Developing the training process required the creation of several classes:

- Game. It controls the use of the other classes. It is also in charge of all the training processes. It has an instance of every class except the *Object* class. The *Object* class is used as a parameter when other classes have to interact between them.

- Q-Learning. Where the algorithm takes place, it is in charge of actualizing the q-values. It has stored the Q-table.
- Avatar. This is the agent class. It controls the agent's movement and uses the *QLearning* class to initialize the Q-values.
- Object. Stores the object type and its position.
- Board. It stores the size of the map and the possible positions where the agent could go.

First, it has to be defined the terminate *(s,a)* pair. If the agent is in state s and executes action a, the training episode terminates and another episode starts. Already explained in Section 0.5, the **health** drive is affected by its surroundings, captured by the senses developed. However, how the **hunger** drive is affected?
Food object was used for that. If the food object is eaten by the agent, the **health** drive will be partially solved. If this occurs, another episode starts. On the other hand, if the agent tries to eat the other type of object, **health** drive will be affected, and the episode will continue.

Knowing his state, the agent chooses an action. In order to do that choice, chooseAction() method has been created. This method is responsible for:

1. Save the state where the agent is in the Q-table. The Q-table was created using a Python dictionary. This dictionary has several dictionaries—first, one for every food position. Having food position as a key, there are three string values that represent what it has inside: *notAchieved*, which has the trials that the agent found the food, but it could not eat it, *achieved* which has the trials that the agent ate the food and "epsilon". In the "epsilon" dictionary is saved the ϵ value for every possible food position. Noted here that what was implemented it was not a single Q-Learning process. There were executed multiple Q-learning algorithms, one for every possible food position. Furthermore, it has a value for every state, being every state another dictionary. Inside it, every action is stored as the key, having the **q-value** as a value.

2. Save the actions that can be executed. At this moment is when the algorithm checks the possible positions and detects, for every position, which action is possible.

3. Decide if exploit or explore using the *exploration rate.* Now is when the epsilon greedy strategy is being executed.

4. Check if the avatar has already done more than the maximum steps possible. If this is the case, *"notAchieved"* action will be chosen.

```
if foodPos not in list(qlearning.q_table.keys()):
    qlearning.q_table[foodPos] = {}
    qlearning.q_table[foodPos]["notAchieved"] = 0
    qlearning.q_table[foodPos]["achieved"] = 0
    qlearning.q_table[foodPos]["epsilon"] = qlearning.epsilon
if obs not in list(qlearning.q_table[foodPos].keys()):
    qlearning.q_table[foodPos][obs]={}

    for action in self.totalActions:
        if (self.check_movement(action, board)):
            qlearning.q_table[foodPos][obs][action] = 0

if(self.totalSteps < self.maxSteps):

    self.totalSteps = self.totalSteps+1
    number = random.random()
    if number > qlearning.q_table[foodPos]["epsilon"]:
        action = max(qlearning.q_table[foodPos][obs], key=
            qlearning.q_table[foodPos][obs].get)
        self.executeMovement(action)
    else:
        action = self.random_movement(board, list(qlearning.
            q_table[foodPos][obs].keys()))
        self.executeMovement(action)
else:
    action = "notAchieved"
return action
```

After deciding which action to execute, this one is sent to Unity. If a rotation action is chosen, the *forward* Vector3 is changed to rotate to the direction specified. If a movement action is decided, MoveTowards() method is used. Parameter *Vector3 target* is the place where the agent has to go, and *speed* is a product of a *movementSpeed* variable set to 1000.0f, Time.deltaTime. This was for preparing the agent to execute smooth animations in the future. If the idle action is decided, the agent does nothing and is prepared to

receive another action, and if the *eat* action is taking place, the object list is checked to know if the object being tried to eat is food or not. If it is food, the food object is destroyed using the Destroy() Unity method, and another one is Instantitate() with the same Unity method named as itself, in a random of the possible positions, excluding the actual avatar position. Next, objects list and possible positions are actualizedx.
After that, the object that was tried to eat is saved.
A similar process is made if the action received is *notAchieved*. Because *notAchieved* action is controlled by the Q-Learning algorithm, when this action is decided, the random new food position is decided in Pycharm, and after receiving the action, Unity will receive the new position of the food. Knowing the new position
When an action is finished, *actualizeDrives()* method is called. This update is based on how the intrinsic variables affect the agent in the second that is thinking what to do. ***In actualizeDrives()* method is where the curve equations explained in Section .1 are used to know how every stimulus is affecting the agent**. In addition, if the action executed was *eat*, the saved object is checked, and the drives are actualized.

When the action has been executed and the drives actualized, the new state is sent to the mind, followed by the object that the agent is viewing.

The second step of the Q-learning algorithm takes place: calculating the reward. The reward has been calculated based on how the drives were affected after executing the last action. In the next Section 0.7, this calculations are explained in detail.

Finally, after the reward calculation, it comes the calculation of the new Q-value and the consequently updating of the Q-table. Figure 11 simplifies how the implementation process works.

0.7 Experiments and results

0.7.1 Experiment 1 - Traducing senses to numbers

In the first experiment, the primary purpose was to check if the agent could feel the temperature, sound and glare through its senses. The implementation for this experiment was to create all the senses as realistically as possible, as explained in 0.6.3. Joining the *SensesController*, an *AvatarMovement* class was added to control its movement and rotation throw the keyboard, using Unity *Input* Class, that allows getting the keyboard key that has been pressed. Arrows and 'w', 'a', 's', 'd' keys (up,down,left,right respectively) were used for the movement and 'i','l','k', 'j', (0°,90°, 180°,270° respectively) were used for rotating. The movement was possible only 1 unit by key pressed in the X or Z axis, and the avatar forward arrow was changed every time a rotating key was pressed. These

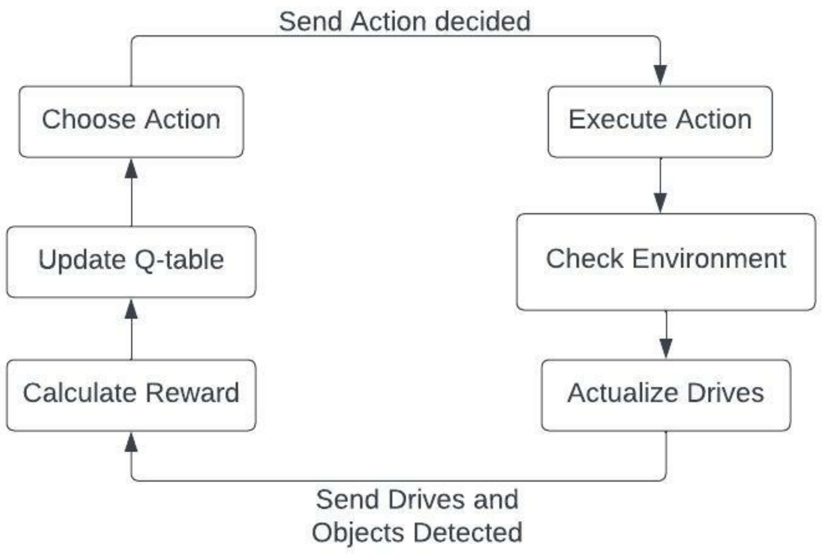


Figure 11: Process execution diagram

limitations were made to define clear positions and rotations where the data would be collected. But the equations and the curves made can calculate how the agent is being affected from every position and rotation possible. Finally, a basic *GameController* was added to control the experiment. If the player wants to check how it is being affected by its surroundings in that specific position and rotation, the "space" key should be pressed. The final class diagram is 12

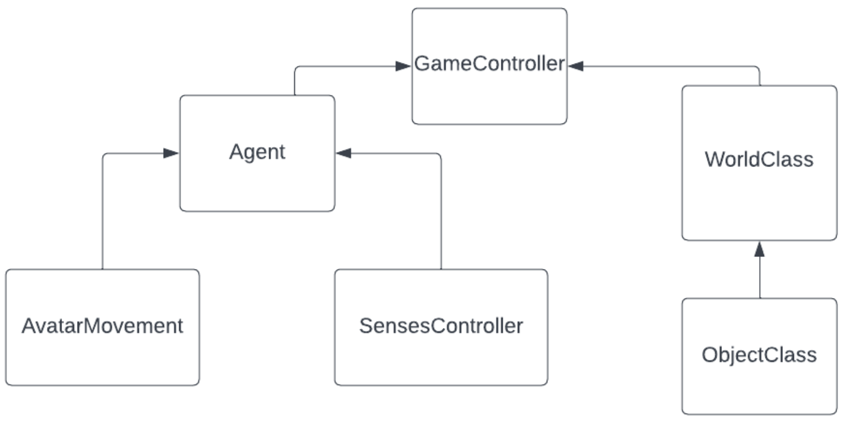


Figure 12: Experiment 1 diagram

For this experiment, the objects and the results after the trial are shown in Figure 29. It can be seen how the avatar is affected for every possible (position, rotation) pair in the map created. Red numbers are the states where the agent received damage above 10. Orange are those between 0 and 10, and green ones are the states where the agent received damage below 1. This map was the one used for in all the following experiments. The variables were set as follows:

- Fire Objects.
  - Fire 1 at position (3,0).
    - Temperature: 60 ℃, dispersion variable = 10.
    - Light Intensity = 10. This was the same for all Fire objects.
    - Sound Intensity = 30. This was the same for all Fire objects.
  - Fire 2 at position (2,4). Temperature: 75 ℃, dispersion variable = 15.
  - Fire 3 at position (4,4). Temperature: 60 ℃, dispersion variable = 10.
- Bulb Objects.
  - Bulb 1 at position (1,2).
    - Temperature: 35℃, dispersion variable = 15. This remains the same for all Bulb objects.
    - Light Intensity = 54.
    - Sound Intensity = 10 dBA. This remains the same for all Bulb objects.
  - Bulb at position (3,4). Light Intensity = 41.25.
  - Bulb at position (4,1). Light Intensity = 54.
- Speaker objects.
  - Speaker at position (1,0).
    - Temperature = ambient Temperature. This remains the same for all Speaker objects.
    - Light Intensity = 0. This remains the same for all Speaker objects.
    - Sound Intensity = 115 dbA.
  - Speaker 2 at position (2,2). Sound Intensity = 100dbA.
  - Speaker 3 at position (0,4). Sound Intensity = 115dbA

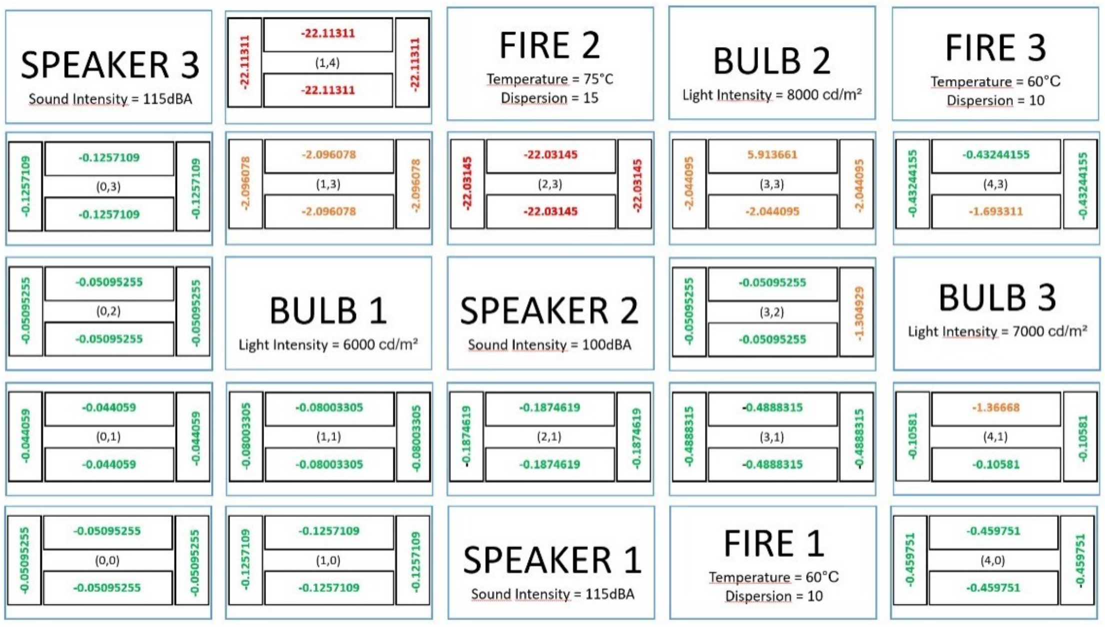


Figure 13: Map used in experiment 1 and 2.

0.7.2 Experiment 2 - Teaching the agent

The Experiment 2 was the first experiment using RL. In order of importance, there were three objectives:

1. Demonstrate that the sight sense was working satisfactorily.
2. The Q-Learning algorithm was well implemented.
3. First check to know if the curves were being learned.

First, it had to be defined how the eat action affected the agent drives. It was set that eating food would decrease the hunger drive by 20 and decrease the health drive by 20.

Second, it had to be implemented reward calculation. The individual reward for the two drive was calculated independently, and the final reward was the sum of both of them. To calculate the health reward it was done the subtraction between the new health and the actual health, and to calculate the hunger reward it was done the subtraction between the actual hunger and the new hunger.

After executing the *eat* or *notAchieved* action, a and placing the food randomly, the agent was not known where the food was placed. That was made to check if the sense sight was well implemented because until detecting the new food position, the agent would walk randomly without considering the rewards. That means the agent would be trapped in an infinity loop if the food could not be detected.

To know if the agent was learning while taking into account the stimulus received, two typical benchmarks were used: rewards achieved and steps done by episode.

For choosing the values of the hyperparameters, an **Ablation study was made**. In the context of machine learning, an "ablation study" describes the procedure to change o remove the algorithm's parameters to detect which combination is better.

The ablation study procedure and its findings can be found in the Appendix, Section .4

The Q-learning variables were set as follows:

- Exploration rate $\epsilon = 0.5$
- Exploration rate decay = 0.01
- Learning rate $\alpha = 0.75$
- Discount Factor $\gamma = 0.99$
- Max steps x episode = 100

It has to keep in mind that in this case, more than one problem was trying to be solved, one for each possible food position. Taking that into account the last variable to set was the number of episodes. Having the Abiation study as a reference .4, it was noted that with one-hundred episodes, the algorithm was already learning.

The result for every possible position can be checked in the Appendix, Section .4.1. In Figure 14 are shown two examples, for position (0,0) and (0,3), showing how the rewards are increasing over time, while steps are decreasing. It can be noted that the stabilisation has barely been established, and more episodes should be run if the main objective was for the agent to learn to go for the food. The procedure has a duration 28minutes and 46 seconds.

0.7.3 Experiment 3 - Stimulus curves and drives

If the Experimet 2 showed how the algorithm implemented and the sight sense were working, now the questions wanted to answer were: are the temperature, sound intensity and glare curves being learned? Are the surroundings affecting the agent? Although the second experiment showed that the agent was being learnt, and it can begin to be seen how the agent is learning the curves, the experiment 2 is not reliable enough because of two key factors:

- when the agent has not yet found the food, its decisions are not affected by its surroundings.

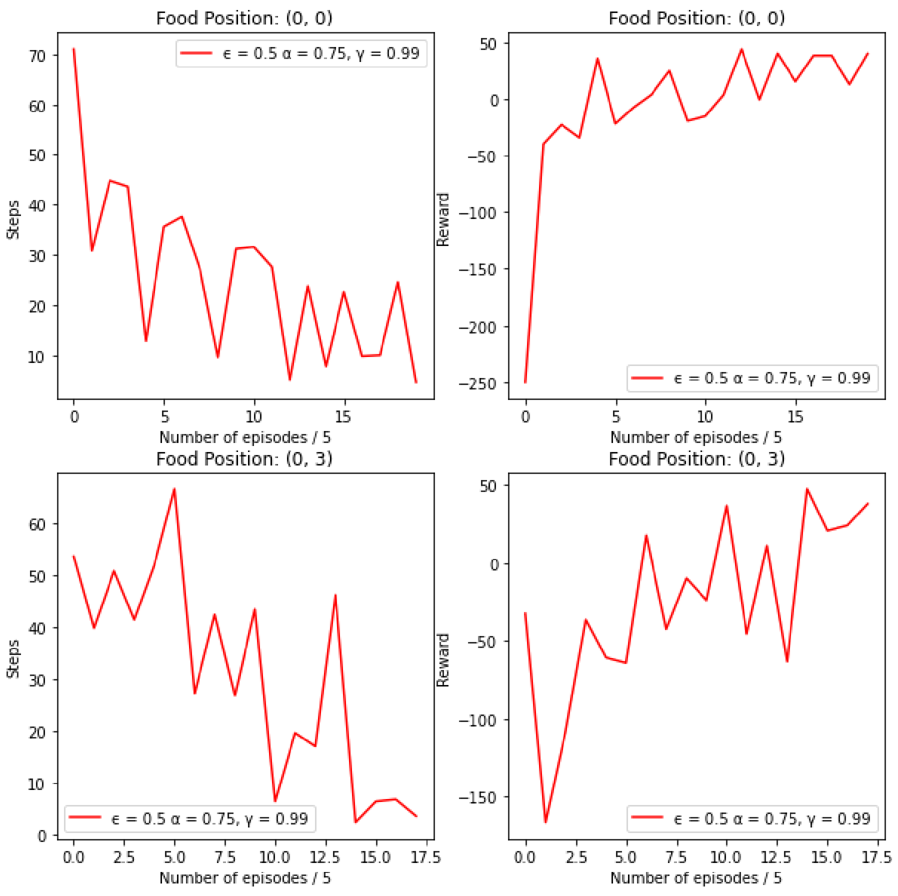


Figure 14: Two examples of reward and steps curves

- The agent is not always in the same place when the episode starts.

In this experiment, this features were changed. When the agent arrives to termination state and he food is randomly placed, the agent already know where the food is placed. Using this new approach, the agent will have always to learn the best way to arrive from every position to every position loosing the minimum health possible. Also, after every termination state, the agent was placed at position (0,0). Because of that, a checking was made to not re-place the food in position (0,0).

In order to be able to learn using its drives, the agent has to have the drives inside the states. At this moment the state was: (position,rotation,health,hunger).

The reward calculation was similar than the one used in Experiment 2, but one features as included:

- After calculating the individual rewards, one variable to control the drive urgency was added called *"nameOfTheDrive"Importance*. This variable was a number between 0 and 1, and muliplied the individual reward. The variable was set following an exponential growth. In the case of the hunger was:

- If the health is between 0 and 50, *hungerImportance* is 0.
- Between 50 and 75, *hungerImportance* is 1/32.
- Between 75 and 87, *hungerImportance* is 2/32.
- Between 87 and 93, *hungerImportance* is 4/32.
- Between 93 and 96, *hungerImportance* is 8/32.
- Between 96 and 98, *hungerImportance* is 16/32.
- Bigger than 98, *hungerImportance* is 1

And in the case of health, it started between 0 and 50 being 0 and 1 for values bigger than 98.

Being consequently with this difference, the health and hunger saved in the states were not the actual avatar drives. They were values related with the *importance* variable:

- 0, 50, 75, 87, 93, 96 and 99 for the hunger drive.
- 100, 50, 25, 13, 7, 4 and 1 for the health drive.

Moreover, the Map used in the experiments done before was changed. This was with the intention of creating "sweet spots", places where the agent does not receive damage. To do that all the decimals numbers were converted to integers. In addition, the temperature of the Fire 1 and Fire 3 was increased to create more places of difficult access. The resulting Map it's shown in Figure 15, with the new values actualized.

At this moment, with all the updates done, everything was prepared to execute the experiment. The hyperparemeters used for Q-Learning were the same as in the previous experiment but changing the ϵ value to 0.8 and the *decay rate* to 0.001. This changed was based on two factors:

- The new map creation. With the new map, there are positions where the agent could be without receiving a reward below 0. This could mean that the agent would explore less possible positions.
- The increase of the states. In the past exercise, only 64 states were possible, 5 for each position in the map. In this exercise, the amount of states was increasing to 4096, incorporating the drives in the states.

The benchmarks used were the same is the previous experiment: rewards and steps per episode. However, in this case, the findings could not be generalized using only the

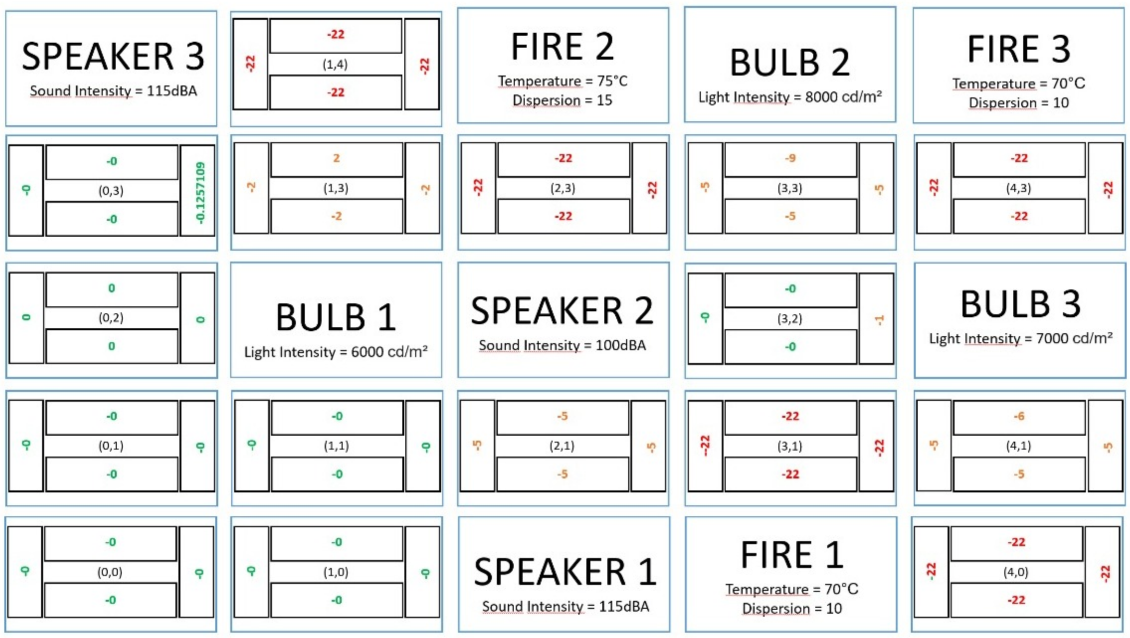


Figure 15: Map used for Experiment 3

food position. After placing the avatar to the position (0,0), the drives used were not re-initialized for trying to know if the agent was able to differentiate its needs. Two cases were studied:

- If at starting state the health drive is 100, but the hunger is 10, it should go straightly to the food. To check this, episodes, when the food was at (4,1), (4,0) and (4,3) were checked.
- On the other hand, if the starting health is 50 or less and the hunger is 0, the avatar should wait or walk around cells that can not affect while it recovers health and, after some steps, go directly to the food. To verify this, episodes, when the food was at (0,3), (0,2), (1,3), were revised.

Finally, the number of episodes run was 2000, with a duration of the experiment of 6h, 37 minutes and 22 seconds. The results can be checked in 16. When it was trying to discern between needs, being hunger more critical, the observations used were the ones with health larger than 50 and hunger below 25. On the other hand, when trying to discern if the agent was capable of deciding to stay in the same place because it had limited life, it was chosen the observations with hunger below 10 and health below 25.

0.8 Conclusions

Related with Experiments 1 and 2, the conclusions extracted were:

- The Map was a confirmation that the agent was not only receiving the stimuli but also being affected by them. The experiment, even though it is the simplest one, was the confirmation that the first objective was achieved: the agent was capturing its surroundings, and the ultra-realistic curves were implemented.
- Glare Variable. Although the simplification of the GR equation 2 makes the calculations simple, and it is being calculated how it should, more variables should be considered. . Checking background luminosity in the investigations taken as reference, angle, and solid angle variables considerably impact the results.
- Temperature variable. The results relating thermal temperature and health extractions are accorded to the equation created and from the model used (Hardy (1956)).
- Sound Intensity variable. The sound intensity in the speaker objects had to be exaggerated to influence the agent. The results are the most realistic ones because the curve is based on proven experiments relating health and sound intensity, and the sound dispersion was made following the real-world relationship.

Also, it was checked in certain positions if the sight sense developed was working correctly. As could be seen, in positions like (3,2) and rotation 270°, the speaker object was blocking the avatar view, which concludes with the avatar not being affected by the glare emitted by Bulb 1. Also, the field of view was working correctly, as can be seen in position (3,3) and rotation 0, when the light source does not influence the agent glare sensation because it is not inside the field of view.

Regarding the Experiment 3, more conclusions can be drawn. In the observations with food positions (4,1), (4,3) (0,3), (1,3), (0,2) it can be observed a rising trend, which would mean that the agent is learning to make decisions based on the two human needs established. Position (4,3) is the most difficult one to reach because it has to get damaged at least three times before eating the food, and its curve is not following an increasing trend. On the other hand, none of the curves was stabilized, and more time would need it for run more experiments to find the perfect combination of the hyperparameters.

With the three experiments done, it was demonstrate that the objectives were accomplished. First, the agent was capable of receiving the environmental stimuli using the senses created. These stimuli were not only captured, but they also affected the agent, using the curves created based on the real world and real sensations. At this point, an the agent was already Working as it should. Finally, Reinforcement Learning was properly used to teach the agent how it should act depending on two basic needs, reaching the second goal established before starting. Even though more research has to be done for mixing senses and needs, it can be concluded that the creation of an ultra-realistic agent was a success.

0.8.1 Summary

This project has been an extensive research and implementation, incorporating different areas into one paper. It also has been the first approach to mixing real-life sensations, basic needs and Artificial Intelligence.
First, an investigation of how human senses work was done in order to replicate them in an AI. Not only were their utilities explored, but also the inputs that can affect a human being in the most basic way: damaging him. Three senses were selected to replicate, sight, hearing and thermal touch, and one stimulus captured by each one was chosen: glare, sound intensity and temperature.
Second, another extended research was done, using several papers to collect data on how humans are affected by this variable, concluding with the information shown in Section 0.4.3.
The next step was to create an environment where recreate the stimulus investigated. Every object was created to be directly related to one of the stimulus. For this process, Unity Engine and C# programming language were used to create the environment, placing the objects on it, and an agent was incorporated. This was the one that was going to feel the stimuli.
Using the stimuli information in Section 0.4.3, a damage curve was created for every possible stimulus. These curves were created taking into account how humans could be damaged irreversibly by the temperature, the sound intensity or the glare.
Having the curve created, it was the moment to integrate them into the agent. To use the curves, the agent had to have sensed for feeling how the objects were affecting him. At this point, the state-of-the-art was extensively revised to replicate the senses in the most realistic way possible.
With an environment affecting the agent, and an agent able to detect it, it was the moment to check if the agent was receiving these stimlus. One experiment was done where the player could move the agent to any place in the world and check if the stimulus were affecting him. But how were they affecting him? To define that, one of the Maslow needs Maslow (1958) was used: the health, giving one step further to replicate the most realistic agent possible. The results of the experiments showed that the stimulus curves

were working, achieving the most crucial objective in this research: the incorporation of real curves in a 3D game.
With the main goal accomplished, one more step was trying to take. Knowing that the agent was feeling the different object, Reinforcement Learning was used to teach the agent how it should act according to its needs, incorporating the hunger need to them. With Reinforcement Learning, another step further was done to realism since its learning is similar to human learning: by trial and error. Using Pycharm IDE and Python programming language, the Q-Learning algorithm, the simplest and the easiest to follow-up Reinforcement Learning algorithm, was implemented.
With both projects created and a communication mounted for sharing information, the last step was to carry out experiments to demonstrate that the agent was able to learn, acting based on how the surroundings were affecting him and his human needs.
The studies clearly demonstrates how a realistic curves can affect the agent which is able to detect them because of the built-in senses. A new path for the pursuit of realism in virtual worlds was open,based on human senses and the capture of their environment. More research has to be done to influence the agent actions based on the stimuli received, but it has been possible to begin to glimpse, based on the results, how the agent began to make decisions taking into account his needs.

0.8.2 Future Work

Although the work done was correctly set, it was only the first step for creating a virtual world where testing agents with human senses. But although it was tried to achieve the most realism possible, improvements could be made.

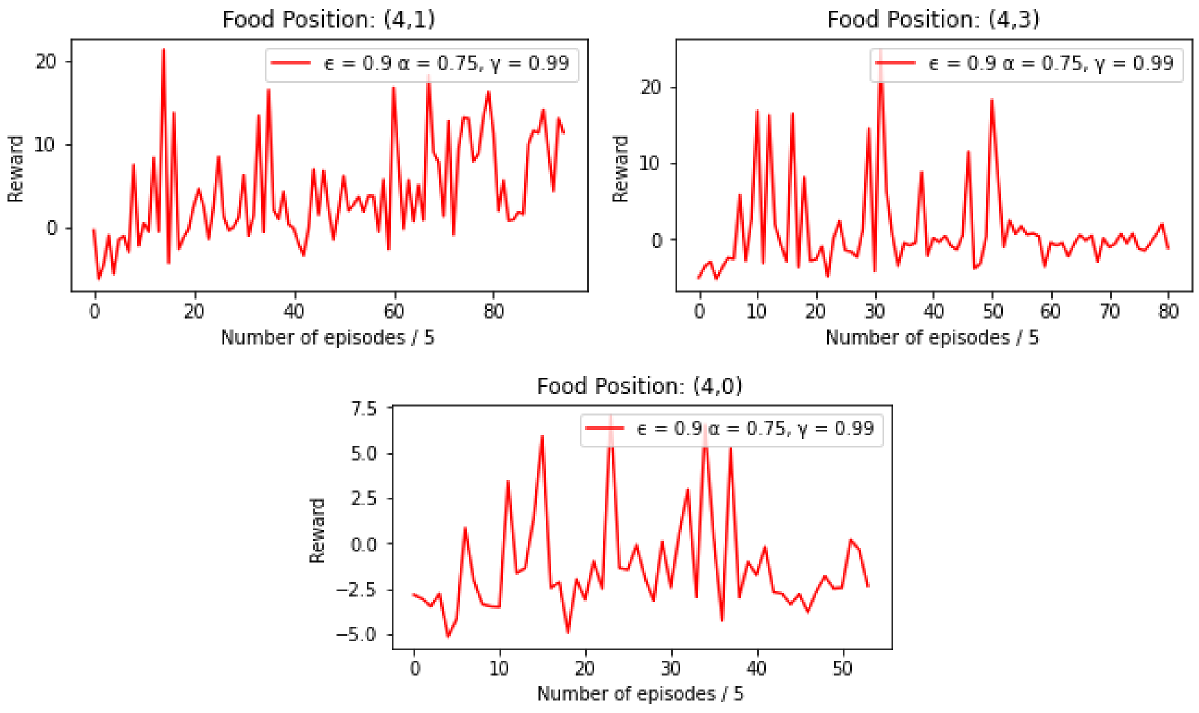
- Combine the objects. At this point, each object is being treated individually, taking the highest value of each variable felt. But how does the sound wave combine with the other sound wave? Light sources create reflections. How do those affect the final glare? And how does the temperature combine?

- Adjust the field of view. Using a general field of view of 60° allows us to test the agent in our environment from a realistic perspective, but the field of view is not that general. Could glare affect the agent if it is outside the central field of vision? Considering the angle of view, how much information can the agent retrieve from an object?

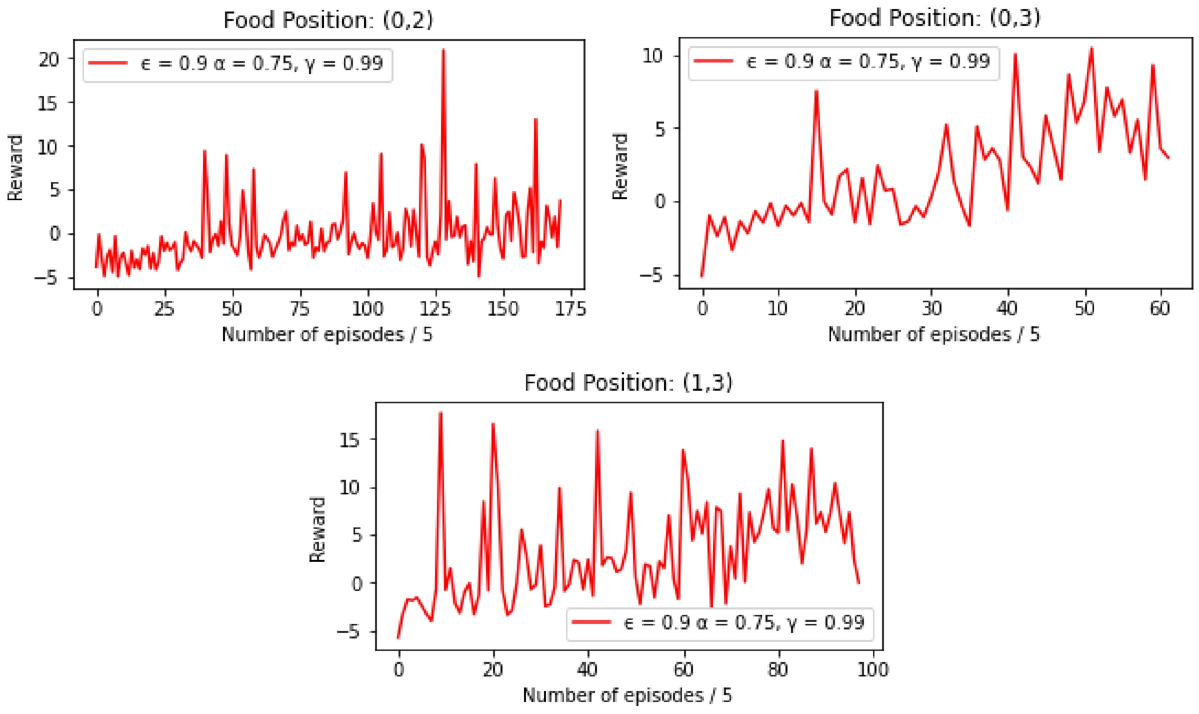
- Implement every other sense. An investigation for developing the taste sense can be found in Appendix .2.

- Concerning the Q-Learning initialization, in this project, the agent already knows its position and rotation in the world and where he can move or not. It would be more realistic if the agent found this information while exploring the map.

Regarding the Reinforcement Learning algorithm choice, Q-learning lets us observe all the changes in the Q-table, ideally for the first step and the first experiments with the new variables curves. But if the difficulty increases, for example, adding more needs to the agent, other algorithms could be used to improve the agent learning process. Deep Q-Learning (Fan et al. (2020)) is interesting to try to go one step further.
A vast field of possibilities is opening up before our eyes, with an immensity of things to explore and test. This project hopes to have laid a foundation for replicating Artificial Intelligence of human senses and the stimulus captured through them.



(a) Discerning between health and hunger, being hunger more important



(b) Discerning between health and hunger, being health more important

Figure 16: Discerning between health and hunger

Bibliography

Akashi, Y., Muramatsu, R., and Kanaya, S. (1996). Unified glare rating (ugr) and subjective appraisal of discomfort glare. *International Journal of Lighting Research and Technology*, 28(4):199–206.

Baharuddin, A. and Sharifudin, M. (2015). The impact of geographical location on taste sensitivity and preference. *International Food Research Journal*, 22(2).

Bartoshuk, L., Rifkin, B., Marks, L., and Bars, P. (1986). Taste and aging. *Journal of Gerontology*, 41(1):51–57.

Beauchamp, G. K. (2009). Sensory and receptor responses to umami: an overview of pioneering work. *The American journal of clinical nutrition*, 90(3):723S–727S.

Beauchamp, G. K., Bertino, M., and Engelman, K. (1983). Modification of salt taste. *Annals of internal medicine*, 98(5_Part_2):763–769.

Bolhuis, D. P., Lakemond, C. M., de Wijk, R. A., Luning, P. A., and de Graaf, C. (2010). Effect of Salt Intensity on Ad Libitum Intake of Tomato Soup Similar in Palatability and on Salt Preference after Consumption. *Chemical Senses*, 35(9):789–799.

Breslin, P. A. (2013). An evolutionary perspective on food and human taste. *Current Biology*, 23(9):R409–R418.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Bullough, J. D., Brons, J., Qi, R., and Rea, M. (2008). Predicting discomfort glare from outdoor lighting installations. *Lighting Research & Technology*, 40(3):225–242.

Cai, S., Ke, P., Narumi, T., and Zhu, K. (2020). Thermairglove: A pneumatic glove for thermal perception and material identification in virtual reality. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 248–257. IEEE.

Cohen, A., Anticaglia, J., and Jones, H. (1971). "sociocusis"—hearing loss from non-occupational noise exposure. *Journal of Occupational and Environmental Medicine*, 13(5):267.

Collins, J., Hisrt, W., Tang, W., Luu, C., Smith, P., Watson, A., and Sahandi, R. (2016). Edtree: Emotional dialogue trees for game based training. In *International Conference on Technologies for E-Learning and Digital Entertainment*, pages 77–84. Springer.

Cowan, B., Kapralos, B., and Collins, K. (2020). Realistic audio ai: Spatial sound modelling to provide npcs with sound perception. In *2020 AES International Conference on Audio for Virtual and Augmented Reality, AVAR 2020.*

Cowan, B. B. (2020). *A graph-based real-time spatial sound framework.* PhD thesis, University of Ontario Institute of Technology (Canada).

d'Eon, E., Luebke, D., and Enderton, E. (2007). Efficient rendering of human skin. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 147–157. Citeseer.

Drewnowski, A. (1987). Sweetness and obesity. In *Sweetness*, pages 177–192. Springer.

Drewnowski, A. (1997). Taste preferences and food intake. *Annual review of nutrition*, 17:237.

Drewnowski, A., Henderson, S. A., Driscoll, A., and Rolls, B. J. (1996). Salt taste perceptions and preferences are unrelated to sodium consumption in healthy older adults. *Journal of the American Dietetic Association*, 96(5):471–474.

Drewnowski, A., Henderson, S. A., and Shore, A. B. (1997). Genetic sensitivity to 6-n-propylthiouracil (prop) and hedonic responses to bitter and sweet tastes. *Chemical senses*, 22(1):27–37.

Drewnowski, A., Mennella, J. A., Johnson, S. L., and Bellisle, F. (2012). Sweetness and food preference. *The Journal of nutrition*, 142(6):1142S–1148S.

Fan, J., Wang, Z., Xie, Y., and Yang, Z. (2020). A theoretical analysis of deep q-learning. In *Learning for Dynamics and Control*, pages 486–489. PMLR.

Fang, Z., Cai, L., and Wang, G. (2021). Metahuman creator the starting point of the metaverse. In *2021 International Symposium on Computer Technology and Information Science (ISCTIS)*, pages 154–157. IEEE.

Galatzer-Levy, I. R., Ruggles, K. V., and Chen, Z. (2018). Data science in the research domain criteria era: relevance of machine learning to the study of stress pathology, recovery, and resilience. *Chronic Stress*, 2:2470547017747553.

Garcés, M. and Finkel, L. (2019). Emotional theory of rationality. *Frontiers in Integrative Neuroscience*, 13.

Gilbert, S. B. (2016). Perceived realism of virtual environments depends on authenticity. *Presence*, 25(4):322–324.

Glendinning, J. I. (1994). Is the bitter rejection response always adaptive? *Physiology & behavior*, 56(6):1217–1227.

Hardy, J. D. (1956). The nature of pain. *Journal of Chronic Diseases*, 4(1):22–51.

Haro, A., Guenter, B., and Essa, I. (2001). Real-time, photo-realistic, physically based rendering of fine scale human skin structure. In *Eurographics Workshop on Rendering Techniques*, pages 53–62. Springer.

Hejlsberg, A., Torgersen, M., Wiltamuth, S., and Golde, P. (2008). *The C# programming language*. Pearson Education.

Hubble, A., Moorin, J., and Khuman, A. S. (2021). Artificial intelligence in fps games: Npc difficulty effects on gameplay. In *Fuzzy Logic*, pages 165–190. Springer.

Hussain, T. S. and Vidaver, G. (2006). Flexible and purposeful npc behaviors using real-time genetic control. In *2006 IEEE international conference on evolutionary computation*, pages 785–792. IEEE.

Jones, L. (2009). Thermal touch. *Scholarpedia*, 4(5):7955.

Jones, L. A. and Berris, M. (2002). The psychophysics of temperature perception and thermal-interface design. In *Proceedings 10th symposium on haptic interfaces for virtual environment and teleoperator systems. HAPTICS 2002*, pages 137–142. IEEE.

Kalita, L. (2014). Socket programming. *International Journal of Computer Science and Information Technologies*, 5(3):4802–4807.

Karunanayaka, K., Johari, N., Hariri, S., Camelia, H., Bielawski, K. S., and Cheok, A. D. (2018). New thermal taste actuation technology for future multisensory virtual reality and internet. *IEEE transactions on visualization and computer graphics*, 24(4):1496–1505.

Kim, S.-H., Hur, Y.-K., and Choi, J.-K. (2005). Suprathreshold taste intensities for sucrose, nacl, citric acid, and quinine hcl in young koreans and the influence of sex, taste preference, and smoking. *Journal of Oral Medicine and Pain*, 30(2):149–162.

Kocher, E. C. and Fisher, G. L. (1969). Subjective intensity and taste preference. *Perceptual and Motor Skills*, 28(3):735–740.

Kuiper, D. M. and Wenkstern, R. Z. (2015). Agent vision in multi-agent based simulation systems. *Autonomous agents and multi-agent systems*, 29(2):161–191.

Lin, Y., Liu, Y., Sun, Y., Zhu, X., Lai, J., and Heynderickx, I. (2014). Model predicting discomfort glare caused by led road lights. *Optics express*, 22(15):18056–18071.

Lin, Y.-L., Chou, T.-Y., Lieo, Y.-C., Huang, Y.-C., and Han, P.-H. (2018). Transfork: Using olfactory device for augmented tasting experience with video see-through head-mounted display. In *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*, pages 1–2.

Löchtefeld, M., Lappalainen, T., Väyrynen, J., Colley, A., and Häkkilä, J. (2017). Comparing thermal and haptic feedback mechanisms for game controllers. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1829–1836.

Lucchi, M., Zindler, F., Mühlbacher-Karrer, S., and Pichler, H. (2020). robo-gym–an open source toolkit for distributed deep reinforcement learning on real and simulated robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5364–5371. IEEE.

Maslow, A. H. (1958). A dynamic theory of human motivation.

May, K. W., Walsh, J., Smith, R. T., Gu, N., and Thomas, B. H. (2020). Vrglare: A virtual reality lighting performance simulator for real-time three-dimensional glare simulation and analysis. In *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, volume 37, pages 32–39. IAARC Publications.

Maynes-Aminzade, D. (2005). Edible bits: Seamless interfaces between people, data and food. In *Conference on Human Factors in Computing Systems (CHI'05)-Extended Abstracts*, pages 2207–2210. Citeseer.

Mcburney, D. H., Smith, D. V., and Shick, T. R. (1972). Gustatory cross adaptation: sourness and bitterness. *Perception & Psychophysics*, 11(3):228–232.

Mennella, J. A., Finkbeiner, S., Lipchock, S. V., Hwang, L.-D., and Reed, D. R. (2014). Preferences for salty and sweet tastes are elevated and related to each other during childhood. *PloS one*, 9(3):e92201.

Merrick, K. and Maher, M. L. (2006). Motivated reinforcement learning for non-player characters in persistent computer game worlds. In *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, pages 3–es.

Messina, E. and Evans, J. (2006). Standards for visual acuity. *National Institute for Standards and Technology.*

Miyashita, H. (2020). Taste display that reproduces tastes measured by a taste sensor. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, pages 1085–1093.

Moskowitz, H. R., Kluter, R. A., Westerling, J., and Jacobs, H. L. (1974). Sugar sweetness and pleasantness: Evidence for different psychological laws. *Science*, 184(4136):583–585.

Murphy, C. and Withee, J. (1986). Age-related differences in the pleasantness of chemosensory stimuli. *Psychology and aging*, 1(4):312.

Narumi, T., Nishizaka, S., Kajinami, T., Tanikawa, T., and Hirose, M. (2011). Meta cookie+: an illusion-based gustatory display. In *International Conference on Virtual and Mixed Reality*, pages 260–269. Springer.

O'Shaughnessy, B. (1989). The sense of touch. *Australasian journal of philosophy*, 67(1):37–58.

Pangborn, R. M. (1970). Individual variation in affective responses to taste stimuli. *Psychonomic Science*, 21(2):125–126.

Patrick, A., Gittens, C., and Katchabaw, M. (2015). The virtual little albert experiment: Creating conditioned emotion response in virtual agents. In *2015 IEEE Games Entertainment Media Conference (GEM)*, pages 1–8. IEEE.

Peiris, R. L., Feng, Y.-L., Chan, L., and Minamizawa, K. (2019). Thermalbracelet: Exploring thermal haptic feedback around the wrist. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–11.

Perret, J. and Vander Poorten, E. (2018). Touching virtual reality: a review of haptic gloves. In *ACTUATOR 2018; 16th International Conference on New Actuators*, pages 1–5. VDE.

Plack, C. J. (2018). *The sense of hearing*. Routledge.

Postel, J. (1981). Transmission control protocol. Technical report.

Ranasinghe, N., Cheok, A. D., Fernando, O. N. N., Nii, H., and Ponnampalam, G. (2011). Electronic taste stimulation. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 561–562.

Ranasinghe, N., Tolley, D., Nguyen, T. N. T., Yan, L., Chew, B., and Do, E. Y.-L. (2019). Augmented flavours: Modulation of flavour experiences through electric taste augmentation. *Food Research International*, 117:60–68.

Reed, D. R. and Knaapila, A. (2010). Genetics of taste and smell: poisons and pleasures. *Progress in molecular biology and translational science*, 94:213–240.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., et al. (2022). A generalist agent. *arXiv preprint arXiv:2205.06175*.

Ripamonti, L. A., Gratani, S., Maggiorini, D., Gadia, D., and Bujari, A. (2017). Believable group behaviours for npcs in fps games. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 12–17. IEEE.

Rodin, J., Moskowitz, H. R., and Bray, G. A. (1976). Relationship between obesity, weight loss, and taste responsiveness. *Physiology & behavior*, 17(4):591–597.

Rozin, P. and Vollmecke, T. A. (1986). Food likes and dislikes. *Annual review of nutrition*, 6(1):433–456.

Sardo, J. D., Pereira, J. A., Veiga, R. J., Semião, J., Cardoso, P. J., and Rodrigues, J. M. (2018). Multisensorial portable device for augmented reality experiences in museums. *International Journal of Education and Learning Systems*, 3.

Sawicki, D. and Wolska, A. (2016). The unified semantic glare scale for gr and ugr indexes. In *2016 IEEE Lighting Conference of the Visegrad Countries (Lumen V4)*, pages 1–6. IEEE.

Sawicki, D. and Wolska, A. (2019). Objective assessment of glare at outdoor workplaces. *Building and Environment*, 149:537–545.

Sessler, D. I. (1993). Perianesthetic thermoregulation and heat balance in humans. *The FASEB journal*, 7(8):638–644.

Shevlin, H., Vold, K., Crosby, M., and Halina, M. (2019). The limits of machine intelligence. *EMBO reports*, 20(10):e49177.

Silva, A., Restivo, M. T., and Gabriel, J. (2013). Haptic device demo using temperature feedback. In *2013 2nd Experiment@ International Conference (exp. at'13)*, pages 172–173. IEEE.

Stoll, A. M. and Greene, L. C. (1959). Relationship between pain and tissue damage due to thermal radiation. *Journal of applied physiology*, 14(3):373–382.

Strasburger, H., Rentschler, I., and Jüttner, M. (2011). Peripheral vision and pattern recognition: A review. *Journal of vision*, 11(5):13–13.

Strigo, I. A., Carli, F., and Bushnell, M. C. (2000). Effect of ambient temperature on human pain and temperature perception. *The Journal of the American Society of Anesthesiologists*, 92(3):699–707.

Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103.

Thompson, D. A., Moskowitz, H. R., and Campbell, R. G. (1976). Effects of body weight and food intake on pleasantness ratings for a sweet stimulus. *Journal of applied physiology*, 41(1):77–83.

Vieira, R., Dembogurski, B., Alvim, L., and Braida, F. (2018). A cognitive architecture for agent-based artificial life simulation. In *International Conference on Computational Science and Its Applications*, pages 197–213. Springer.

Wang, H., Gao, Y., and Chen, X. (2009). Rl-dot: A reinforcement learning npc team for playing domination games. *IEEE Transactions on Computational intelligence and AI in Games*, 2(1):17–26.

Weiffenbach, J. M., Baum, B. J., and Burghauser, R. (1982). Taste thresholds: quality specific variation with human aging. *Journal of Gerontology*, 37(3):372–377.

Witherly, S., Pangborn, R. M., and Stern, J. S. (1980). Gustatory responses and eating duration of obese and lean adults. *Appetite*, 1(1):53–63.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Yoshinaka, M., Ikebe, K., Uota, M., Ogawa, T., Okada, T., Inomata, C., Takeshita, H., Mihara, Y., Gondo, Y., Masui, Y., et al. (2016). Age and sex differences in the taste sensitivity of young adult, young-old and old-old j apanese. *Geriatrics & gerontology international*, 16(12):1281–1288.

Yost, W. A. and Killion, M. C. (1997). Hearing thresholds. *Encyclopedia of acoustics*, 3:1545–1554.

Yudkowsky, E. (2007). Levels of organization in general intelligence. In *Artificial general intelligence*, pages 389–501. Springer.

Zhang, B. and Sra, M. (2021). Pneumod: A modular haptic device with localized pressure and thermal feedback. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*, pages 1–7.

Ziat, M., Rolison, T., Shirtz, A., Wilbern, D., and Balcer, C. A. (2014). Enhancing virtual immersion through tactile feedback. In *Proceedings of the adjunct publication of the 27th annual ACM symposium on User interface software and technology*, pages 65–66.

.1 Curves tables

Duration per day, hours	Sound level dBA slow response
8	90
6	92
4	95
3	97
2	100
1.5	102
1	105
0.5	100
0.25 or less	115

Table 1: Hearing loss for occupational noise - Table from the United States Department of Labor

Duration per day, hours	Sound level dBA slow response, Non-Occupational exposure
16	70
8	75
6	77
4	80
3	82
2	85
1.5	87
1	90
0.5	95
0.25	100
0.125	105
0.0625	110
0.03125 or less	115

Table 2: Hearing loss from non-occupational noise - Table from "Sociocusis"—hearing loss from non-occupational noise exposure" Cohen et al. (1971)

de Boer Scale	Glare Rating
Unbereable	90
2	80
Disturbing	70
4	60
Just Acceptable	50
6	40
Satisfactory	430
8	20
Imperceptible	10

Table 3: Table for Glare Curve, in relation with de Boer Scale

Time exposed (s)	Temperature (℃)
25200	45
360	50
20	55
5	60
1	65
¡ 1	70

Table 4: Table for Temperature Curve. Time exposed to a specific temperature (℃) before irreversible tissue damage

.2 Taste sense

Human Taste

The taste system function is one of the guides to detect toxicity and nutrients in food, helping us to decide what should we eat. Taste is combined with two more senses, odour and tactile, to classify the food between two: familiar or novel. If it's familiar, the body can recognize the consequences of eating this food. If it's not, we can use the sensory data to try to predict the outcome of eating the food. If we detect that the outcome is positive, the taste will send pleasure and reward, combining the quality of the taste itself and the toxicity that it has (Breslin (2013)). Taste is differentiated into five basic categories: sweet, salty, sour, bitter and umami. Simple carbohydrates are experienced as sweet, the amino acids glutamate, aspartate and selected ribonucleic acids are experienced as savoury (or umami), sodium salts, and salts of a few other cations, are experienced as salty, acids are experienced as sour, and many toxic compounds are experienced as bitter (Breslin (2013)). Some substances are accepted or rejected primarily because of the anticipated consequences of ingestion. These could be rapid effects, such as nausea or cramps, or the pleasant feeling of satiation. More delayed effects involve beliefs or attitudes about the health value of substances (Rozin and Vollmecke (1986)).

The animals and humans can recognise if the food is toxic of not depending mostly by bitter flavours (Reed and Knaapila (2010), Glendinning (1994)), and could be improved is if it's taken into account the texture and odors.

Children's food preferences are often guided by taste alone. In contrast, food choices of adults also tend to be influenced by nutritional beliefs and attitudes toward weight and dieting. (Drewnowski (1997)). Because our agent is not going to be influenced by nutritional beliefs or societal impacts, the taste developed is going to be child alike

.2.1 Human taste replication

The human sense is less investigated. It should be distinguished from the flavour, which is influenced by the other senses. The digital taste remains in the research state.

Investigations that have been made are more focused for being use with Augmented Reality applications (Sardo et al. (2018)). One option is to use chemical stimulation, using an extra device that can be applied in your lips or tongue as presented in "Transfork: Using olfactory device for augmented tasting experience with video see-through head-mounted display" (Lin et al. (2018)) or through a screen interface like the one is presented in TasteFood Maynes-Aminzade (2005), where the chemicals are released in the screen and the user has to lick it to taste it. Another one is to add digital gustation to utensils like chopsticks or a soup bowl using electronic taste stimulation (Ranasinghe et al. (2011), Ranasinghe et al. (2019)). Other interesting approach is to use the temperature to produce and maximise different tastes (Karunanayaka et al. (2018)).

In "Meta cookie+: an illusion-based gustatory display" (Narumi et al. (2011)), instead of reproduce the food taste using the taste sense, they want to transmit the taste of the cookie through the sight sense and the olfactory sense.

Related to how the taste is being codified, the chemical approach uses the specific taste of the food to reproduce it. So, if a strawberry is being tasted, the chemical released would be the strawberry one. This would define the strawberry taste as "strawberry" and couldn't generate a general rule to define the intrinsic variables related to taste. Electric and thermal stimulation does not define the taste of the food but try to maximize the taste by applying other stimuli.

But electric stimulation was applied in (Miyashita (2020)), to finally reproduce the taste of the food by dividing it by the intensities of every basic taste (salty, sweet, sour, bitter and umami). These basic taste intensities could be recorded to be able to reproduce the taste later. As can be seen, in Figure 1, a complex taste is divided into 5 simple tastes intensity. This shows that even the final taste is not the sum of the intensity of the basic tastes detected in complex tastes. A taste could be reproduced by its final basic taste intensity felt by the tongue.

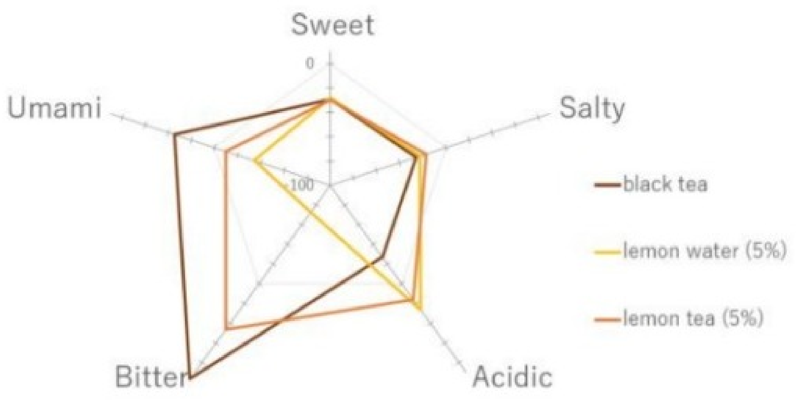


Figure 17: Complex component divided in basic testes by Miyashita (2020)

.2.2 Taste curves

For every flavour, different aspects have been studied historically: sensitivity, ability to taste, consistency response, taste identification and taste preferences. To return valances, taste preferences was the one chose and two more sensitivty is one used to add different characteristics between agents created and add realism.

Salty

Salty figure show at Figure 18 was extracted unifying the following sources: Beauchamp et al. (1983) Bolhuis et al. (2010) Drewnowski et al. (1996) and Baharuddin and Sharifudin (2015) Murphy and Withee (1986).

Sweet

Sweetness graph has differences depending on the research that was done Kocher and Fisher (1969), as it's shown in Figure 19, created by the different curve types (Baharuddin and Sharifudin (2015) Moskowitz et al. (1974) Drewnowski (1987) Rodin et al. (1976) Thompson et al. (1976) Bartoshuk et al. (1986)). It is not clear why this curve is so

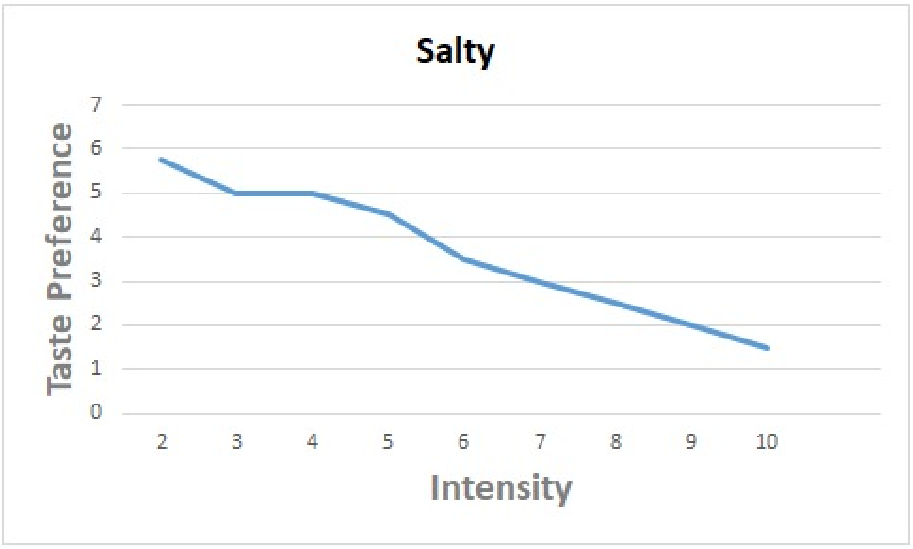


Figure 18: Salty graph point to point

different depending the research. Some studies believe that cultural and childhood could be one of the causes (Mennella et al. (2014), Drewnowski et al. (2012)), and despite the cultural thinking is that weight could be a consequence of this, this is refuted in another studies like (Rodin et al. (1976)). To follow the similar line, it was only taken the studies where the glucose compound was the sugar used with stilled water and that's why some studies like (Pangborn (1970)), which mixes sugar with monosodum glutamate (associated to umami) or (Witherly et al. (1980)) that mixes glucose solution with lemonade were discarded.

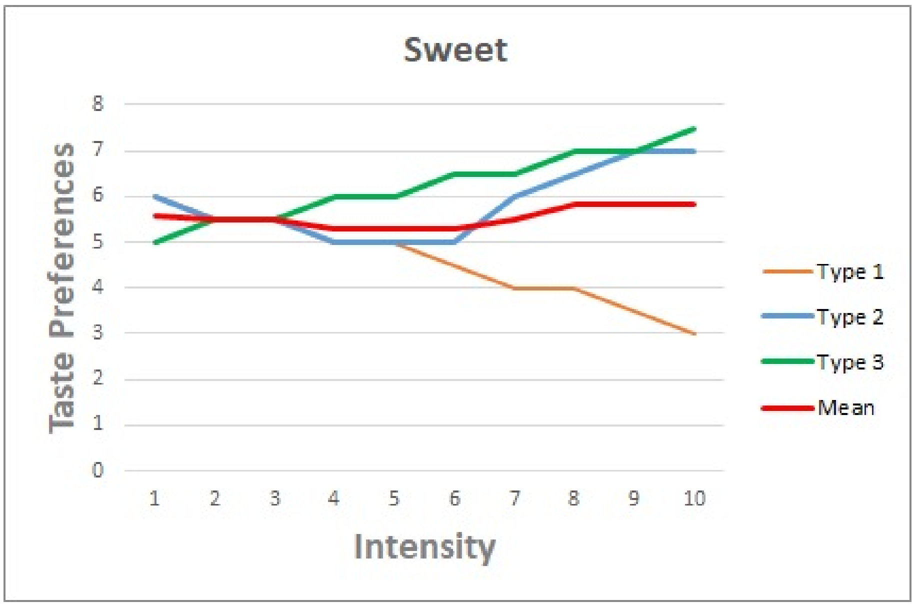


Figure 19: Different types of sweetness taste preference

Bitter

Figure 20 links is extracted from (Drewnowski et al. (1997)) that links bitterness liking with intensity. Bitterness taste preferences are also influenced by toxicity, but with the taste preferences bitter graph we are taking into account the toxicity prevention (Glendinning (1994)).

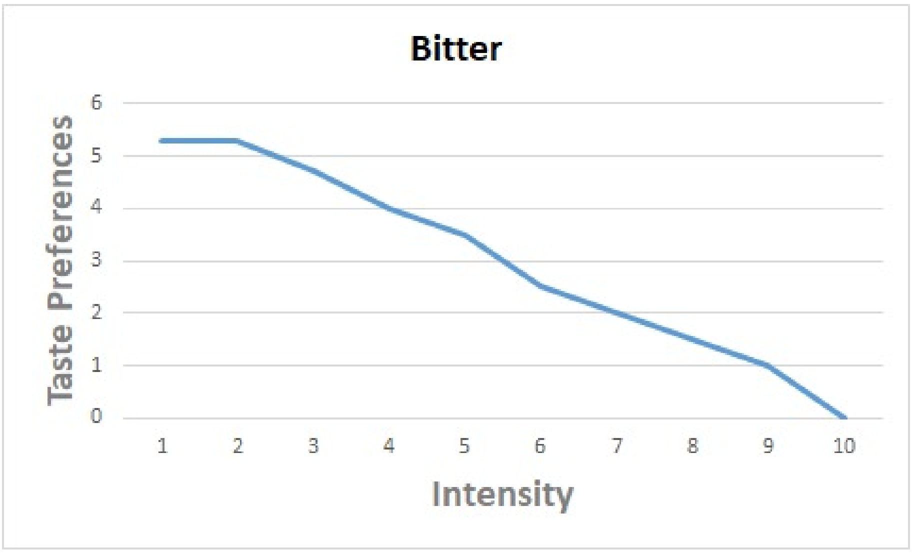


Figure 20: Bitter graph point to point

Sour

Sourness Figure 27 was extracted unifying the following sources: (Kocher and Fisher (1969), Baharuddin and Sharifudin (2015) Kim et al. (2005) Mcburney et al. (1972))

Umami

The overview of umami studies in (Beauchamp (2009)), highlight that sodium salt monosodium glutamate(MSG), which is the usual form of umami, enhances the patability of most of the foods that is added. Also, refers to other studies that confirmed that umami substances added increase liking to the food. This is the reason that in this study, umami is going to be used as a highlight of the good flavours. It's not going to be a curve like the other flavours, because it doesn't work like them.

Sensitivity threshold

Detection threshold is the lowest concentration at which a compound can be detected, and subjects often perceive this as only a hint of "something"—just enough to discriminate

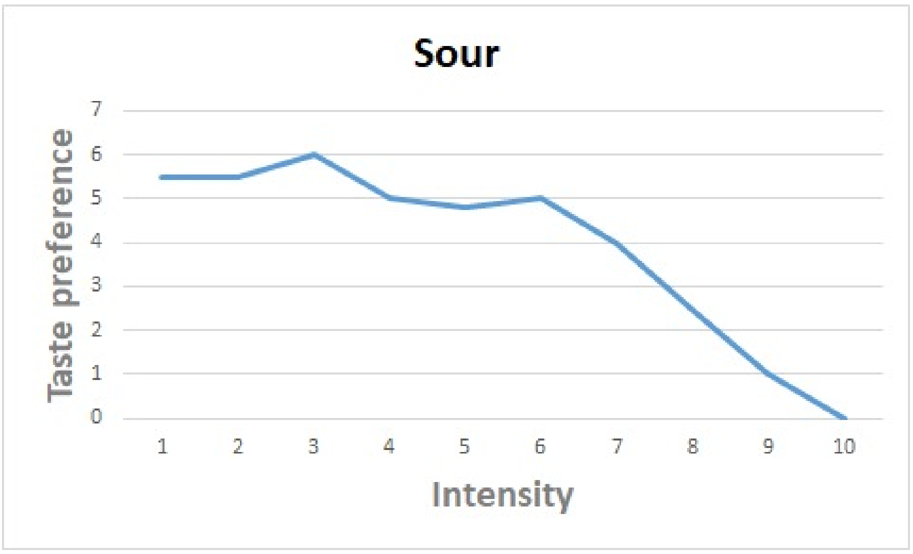


Figure 21: Sour graph point to point

the stimulus from a blank, but not necessarily enough to recognize its type or quality (e.g., sweet). The lowest concentration at which a stimulus can be named for its quality is called the recognition threshold (Yoshinaka et al. (2016), Weiffenbach et al. (1982)).

Preference variable for every taste

In the different papers that was used to create the graphs above, the final point for every intensity was the mean of every tester that was part of the research. Because of this, the plot should not be mark the final result. In order to add difference between the agents, an attribute called "Preference *taste* variable" that will be between can be added.

Final taste calculation

In order to be able to use the taste sense, the food will be defined by the intensity scale of each flavour. Knowing the intensity of each flavour, we can apply it in the taste preferences graphs and if we use the *sensitivyThreshold* We take the result of the substraction between

$$intensity - sensitivityThreshold = finalintensityflavour. \tag{10}$$

and use the result for the function that define the taste preference for that taste. Next step, would be to apply the random taste preference, if used, for the taste being calculated. With that we have the general "liking rating" for the specific taste.

If we do that for each taste, and sum all of them, we obtain the general rating, and if positive, umami value (between 0 and 5), will be added, being 10 the maximum general rating.

.3 Unity - PyCharm Communication

In order to mix Unity and PyCharm, a TCP Communication via Sockets was made. None reliable reference was found to do that, so it was made from completely from scratch. Two classes were created to have the control of the communication, one for Unity Engine and another one for PyCharm project, both called *CommunicationScript.* Before init the conneciton, the same IP adress and connection port were defined.

.3.1 Server and Client connection

Unity project, as a server, had to initialize the connection creating a new Thread, and started using the *Start()* method. The thread was running all the time, while getting the info from the client.

To get the initial information from the environment, the thread was initialized with a class function called *GetInfo(WorldClass Board, Vector3, avatarPosition, Vector3 avatarForward)*,

```
Thread threat = new Thread(() => GetInfo(board, avatarPosition, a
threat.Start();
```

To be able to establish the communication, a new TcpListener was created and with the method *Start()*, the server started listening for incoming connection requests.

After that, the next thing will be to capture the client. The TcpListener called the method *AcceptTcpClient()* which accepted a pending connection request.

Now, it becomes important the *NetworkClass*, which provides the underlying stream of data for network access. The NetworkStream class was initalized usig the TcpClient and calling to Start(), which returns the NetworkStream used to send and receive data. Until receiving the NetworkStream, the client would be waiting all the time.

In next step is when the information between server and client will be shared., but first take a look to the *GetInfo()* function:

```
localAdd = IPAddress.Parse(connectionIP);
listener = new TcpListener(localAdd, connectionPort);
listener.Start();

client = listener.AcceptTcpClient();
running = true;
networkStream = client.GetStream();
while (running)
```

```
{
    shareInfo()

}
listener.Stop();
```

On the other hand, the connection from the client to the server is easy to programm. Using the socket class and initializing it with *socket(family, type)* method with he family and type by default, it would be already prepared to be used. After that using the method *connect(host, Port)*, specifying the IP Adress and the port, the connection will be stablished.

Now, the *shareInfo()* has to be created. This function will be in charge of sending the initialize states of the environment to the client. After that, the diagram shown in 11 starts.

But the key question here is: how can the information be sent and received?

.3.2 Sharing info

Regarding the server, the two key methods used from *NetworkStream* class were: *Read(byte[] array, int32 offset, numOfBytes int32)* for reading and *Write(byte[] array, int32 offset, numOfBytes int32)* for writing. Concerning the client, it was using *socket.recv(int bytes)* and *socket.send(int bytes)* for receiving and send information.

In order to share informatio, it had o be established the buffer size, for writing and for reading. In both scripts two functions where used to establish and detect the size of the buffer received: *receiveInfo()* and *sendInfo()*. In both functions, two possibles data to be sent were specified: "number" and "string". For each possible data, the length of the buffer size was set, being 16 for both. This size was established knowing that the data sent would never exceed 16 bytes, but it could be changed if that is not the case.

If the string sent does not have the size of 16 bytes, symbol '-' would be include after the message, and when the client or server receives the messages, a loop is used to save he information sent only until the symbol '-'.
Same strategy was followed for the "number" option. First, it has to be in mind that every message that it is received will be save in a bytes array and converted to string. The number '0' as a string was added before write the all number in the buffer. Writing '0' before the number allows the program to use whateverTypeOfNumberYouWant.Parse(string number) directly after the reception. For example, if the server is waiting for a float, it will only have to do: float.Parse(numberAsString) in C# language or (float)numberAsString in Python programing language. With these two functions, it was able to receive every

type of information that it was wanted. In the case of a list, the length of the list had to be sent before, to know how many numbers were going to be sent.

Noted that this modification, specifying exactly the number of bytes that every type of info is going to have, had to be done because when it was tried to used the default client size, 8192 bytes, errors occurred when loops were implemented, and info was interleaved and mixed between them.

Here it is the code for C#

```
private void sendInfo(string info, string infoType)
{
    if (infoType == "number")
    {
        //We limit each number to 16 bytes
        maxBytes = 16;
        if (Encoding.ASCII.GetBytes(info).Length < maxBytes)
        {
            int difference = maxBytes - info.Length;

            for (int i = 0; i < difference; i++)
            {
                info = info + '␣';
            }
        }
    }

    else if (infoType == "string")
    {
        //We limit each message to 16 bytes
        maxBytes = 16;
        if (Encoding.ASCII.GetBytes(info).Length < maxBytes)
        {

            int difference = maxBytes - info.Length;
            //Debug.Log("name␣difference␣=␣" + difference.ToString())
            for (int i = 0; i < difference; i++)
            {
                info = info + "-";
            }
        }
    }

    networkStream.Write(Encoding.ASCII.GetBytes(info), 0, Encoding.AS
}

private string receiveInfo(string infoType)
{

    else if (infoType == "number")
    {
        //The size for a single number will be 16
        size = 16;
    }
    else if (infoType == "string")
    {
        size = 16;
    }

    byte[] buffer = new byte[size];
    //Receiving Data from the Host
    int bytesRead = networkStream.Read(buffer, 0, size);
    return Encoding.UTF8.GetString(buffer, 0, bytesRead);



}
```

And here the code for Python:

```
    def receiveInfo(self, messageType):
        if(messageType=="number"):
            info = self.sock.recv(16).decode("UTF-8")
            try:
                info = int(info)
            except:
                info = float(info)
        else: \#string case
            firstInfo = self.sock.recv(64).decode("UTF-8")
            found = False
            info=""
            count = 0
            while(found == False):
                if(firstInfo[count] == '-'):
                    found = True
                else:
                    info = info+firstInfo[count]
                    count = count +1


        return info

    def sendInfo(self, info, infoType):

        if(infoType=="string"):
            maxBytes = 16
            while(len(info) < maxBytes):
                info= str(info) + "0"

        else: \#number case
            maxBytes = 16
            while(len(str(info)) < maxBytes):
                info = "0" + str(info)
            #print("float sent: ", info)
        self.sock.send(info.encode("UTF-8"))
```

.4 Ablation Study for Reinfcement Learning

To decide which hyperparemters were better for the exeperiment, two different values were tried for the following hyperparameters:

- Epsilon ϵ : 0.2 and 0.5. These two values were chose because the initialization of the Q-table was optimistic. That means that the initialization done encourage to explore the world. If all of the pair (s,a) are initialized as 0, but if in all the states the agent receives a reward below 0, as can be seen in Figure 29, because of the nature of the Q-Learning equation, the agent will visit first the states not yet visited.

- Learning Rate, α: 0.75 and 0.94. The learning rate is how quickly the agent abandons the previous Q-value. In the experiment done, the agent knows exactly why the reward is being received, because it is directly related with the position and the rotation. Based on that, if the agent receives always the same reward in every state, and it controls why, it was decided to use large numbers

- Discount factor, γ: 0.75 and 0.99. The idea in the investigation was to encourage the agent to find the food placed in the board. If eating the food means to terminate the episode, that means that the most further episode is to eat the apple. With a high discount factor, the agent will be aimed to search for the apple.

The results show the mean reward and number of steps for every 5 episodes. Checking the results, it was clearly seen that the steps plots do not help. This is because the food is being randomly placed after being eaten or after not being found. On the other hand, the agent is not being restarted after not completing the episode. It can not be known how far away will be the agent at the starting point of the episode.
After checking the rewards plot, it was decided to use $\epsilon = 0.5$, $\alpha = 0.75$, $\gamma = 0.99$. First, other alpha and gamma were discarded because, even almost all of them were achieving the better results, that hyperparemeters led to the algorithm to have more regularly achieved states terminated.

.4.1 Results for Experiment 2

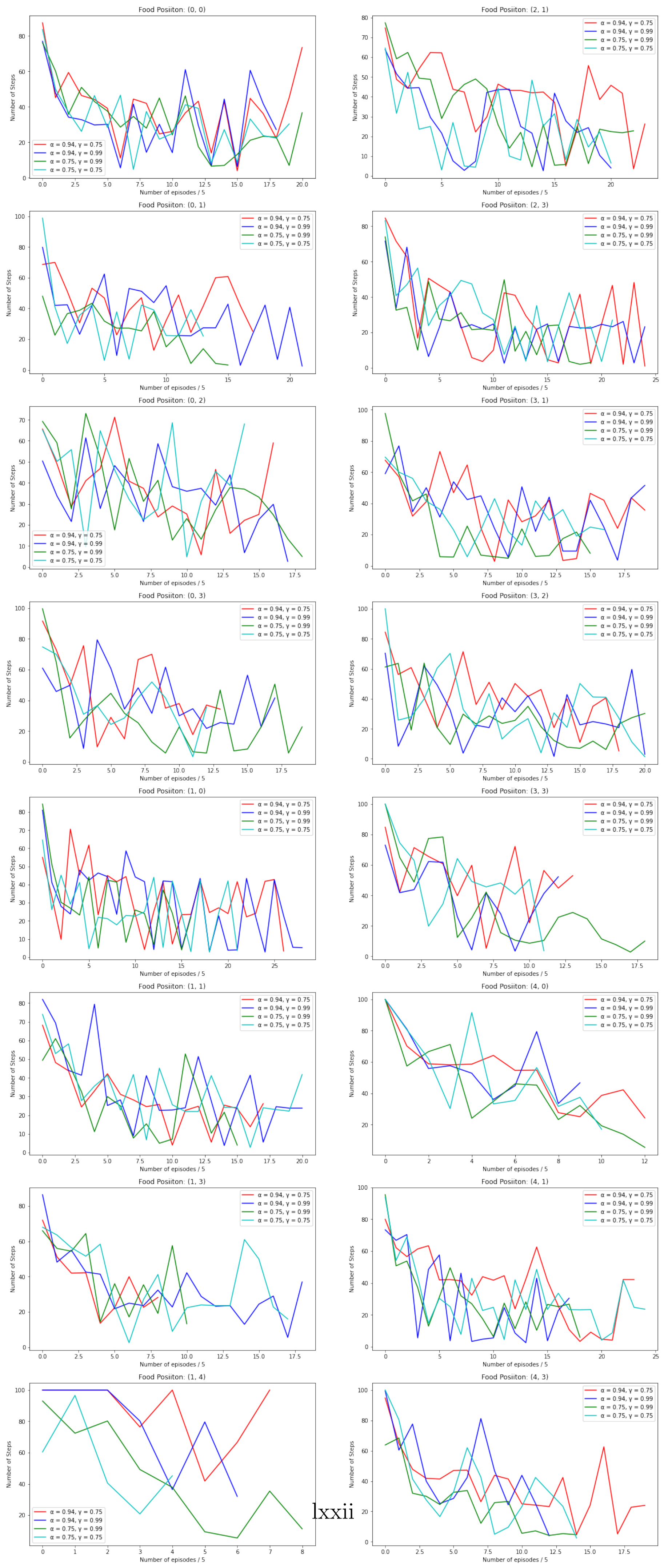


Figure 22: Ablation study: Number of steps, $\epsilon = 0.2$

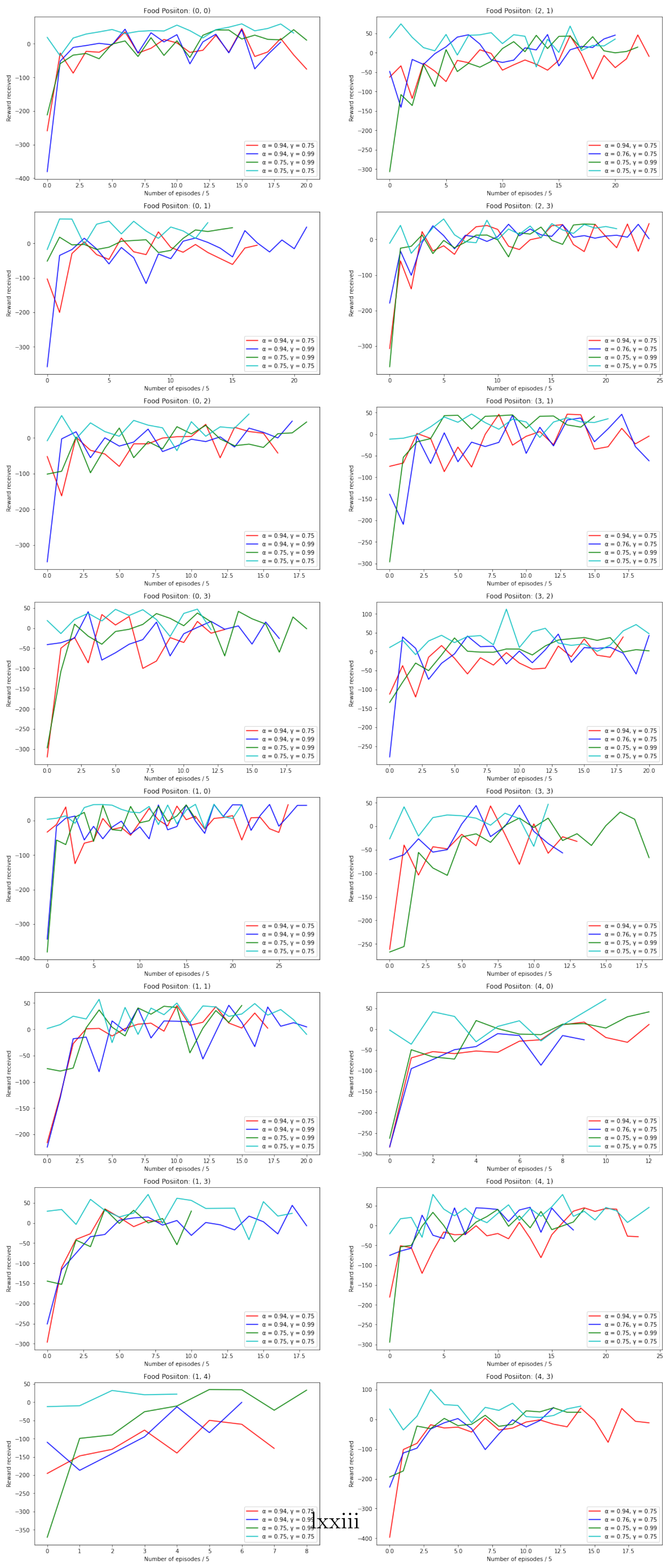


Figure 23: Ablation study: Number of rewards, $\epsilon = 0.2$

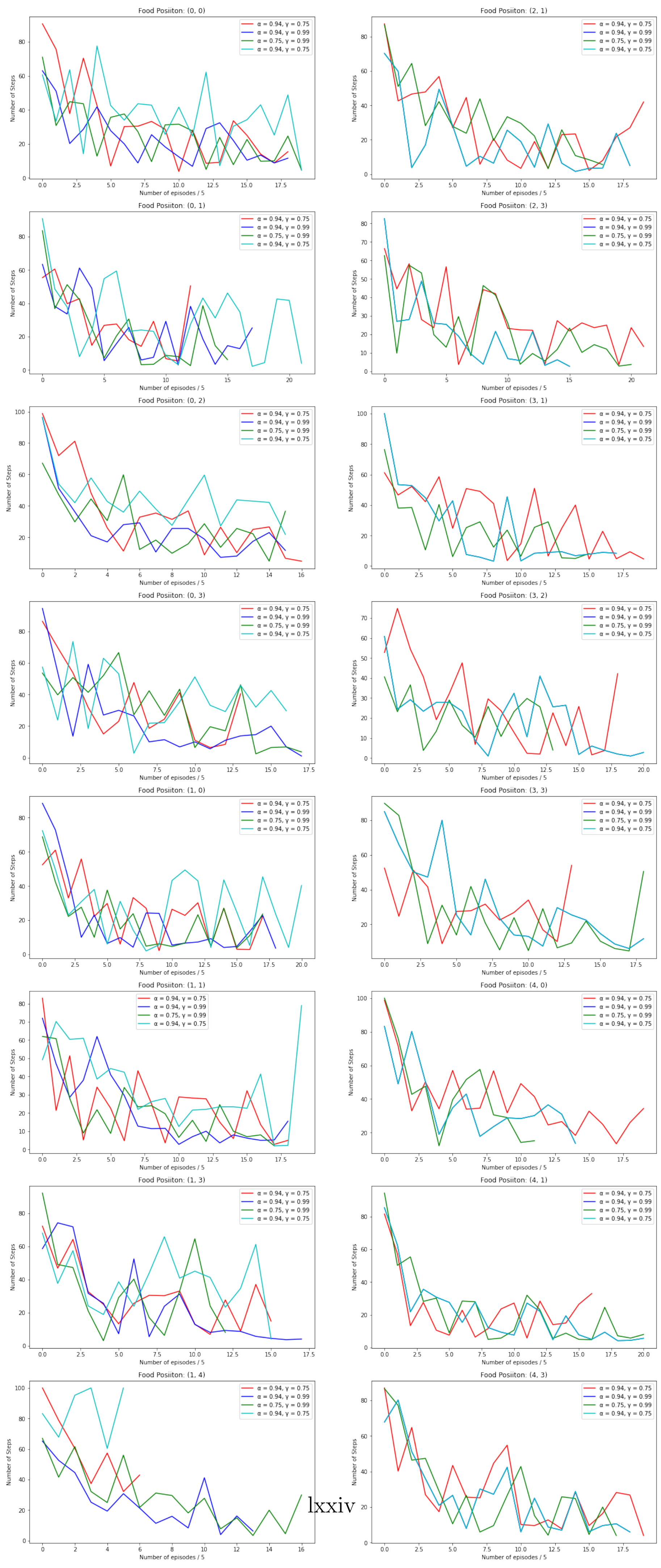


Figure 24: Ablation study: Number of steps, $\epsilon = 0.5$

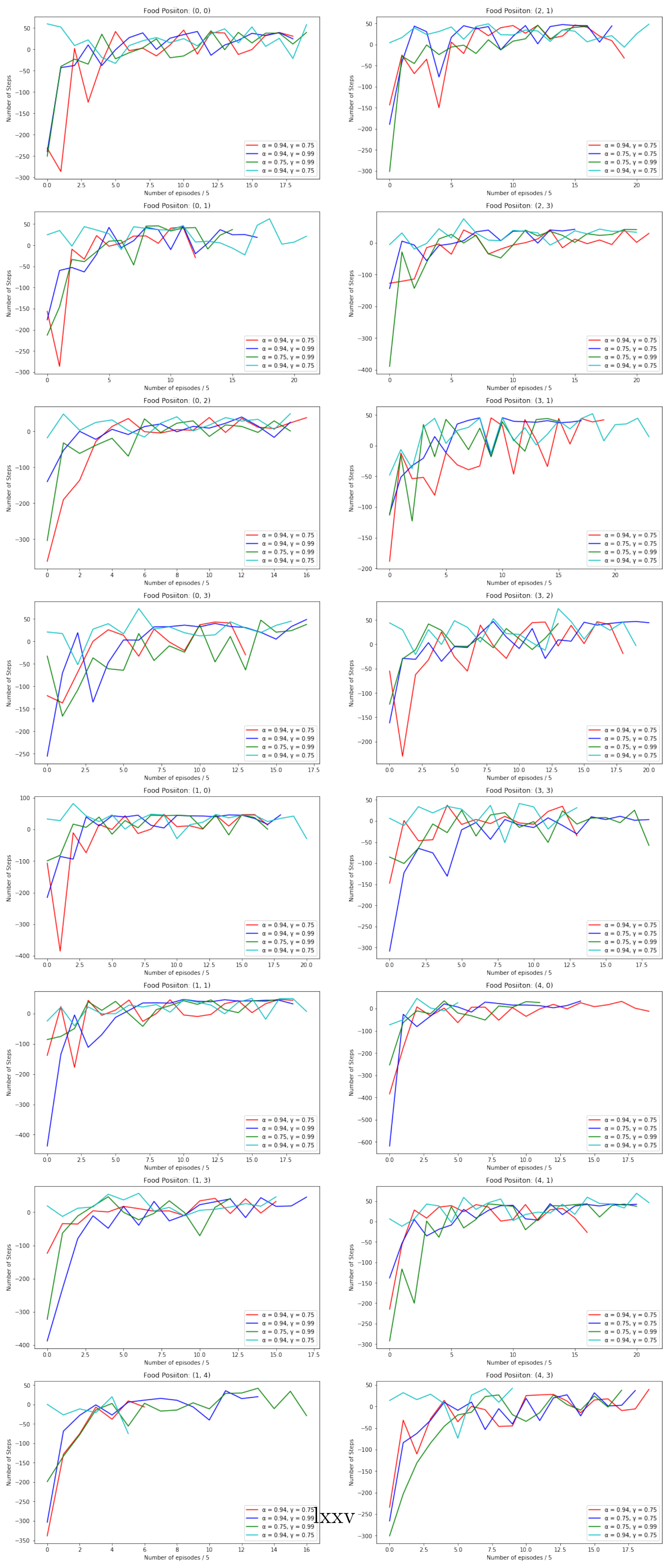


Figure 25: Ablation study: Number of rewards, $\epsilon = 0.5$

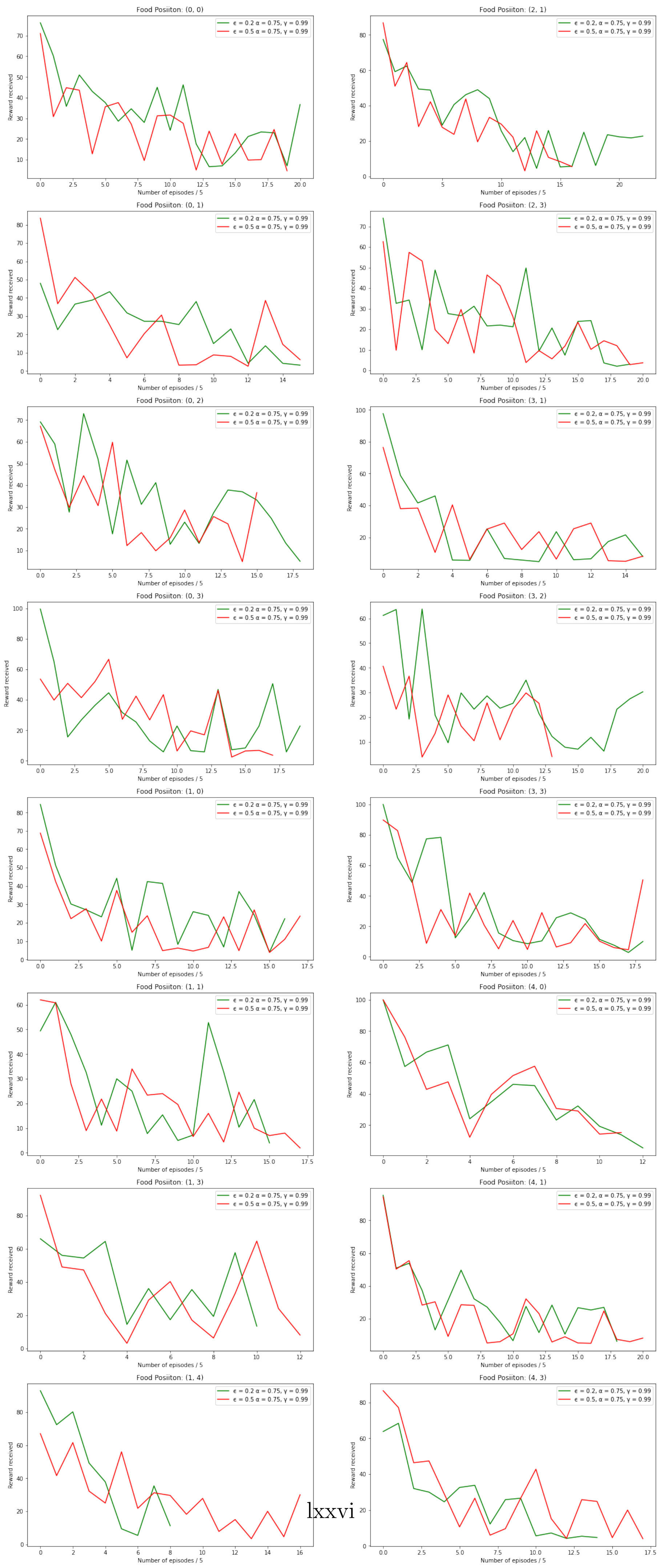


Figure 26: Ablation study: Comparing epsilon, steps

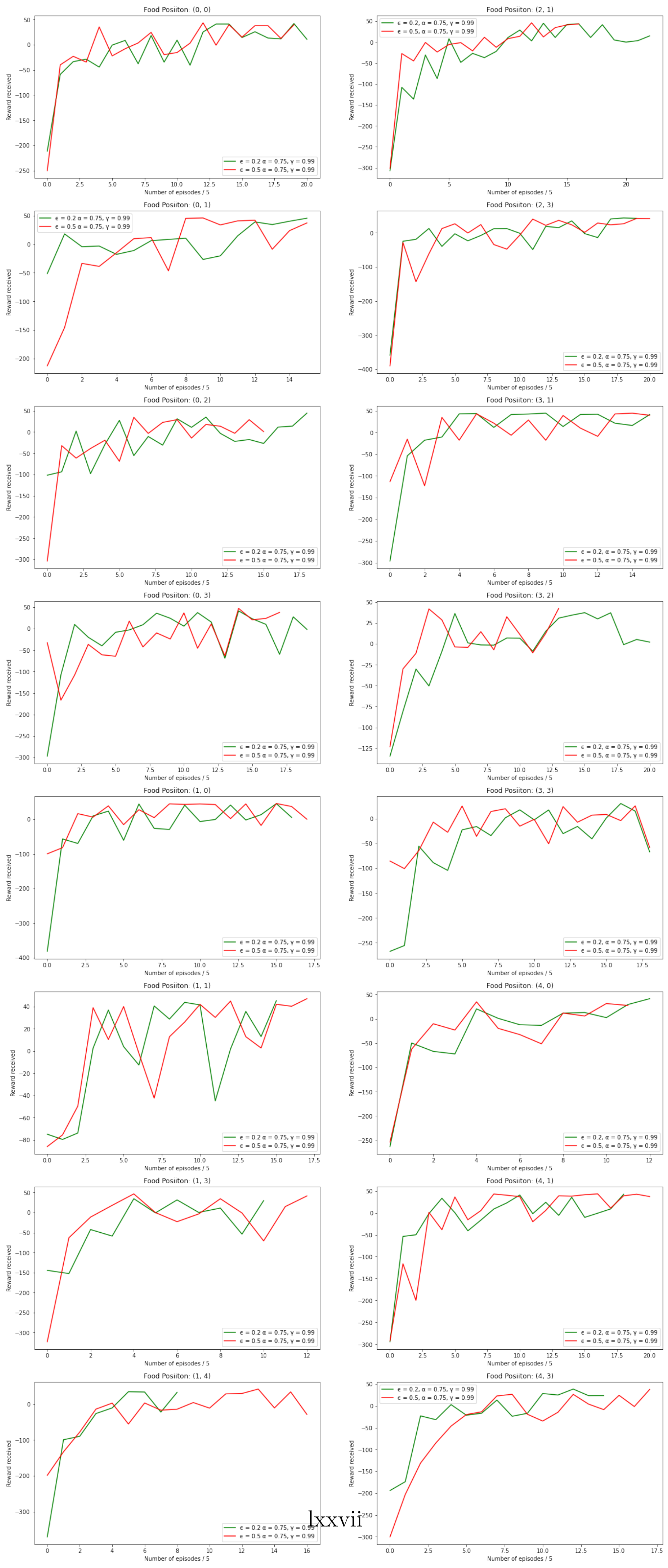
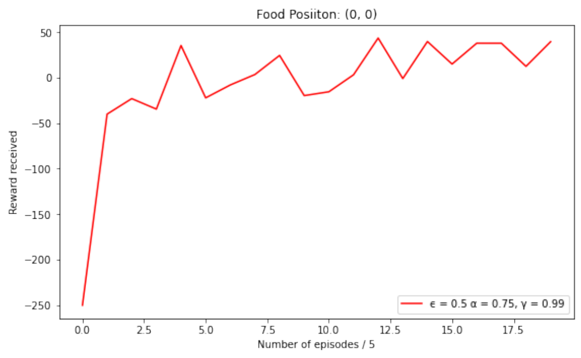
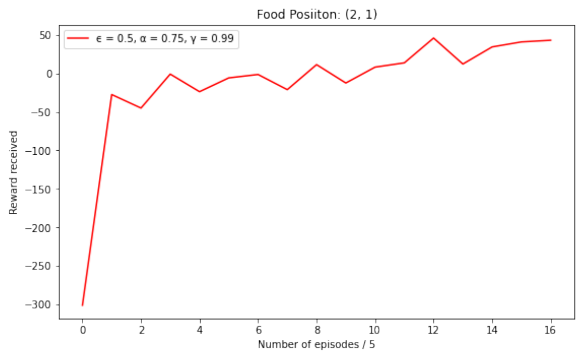
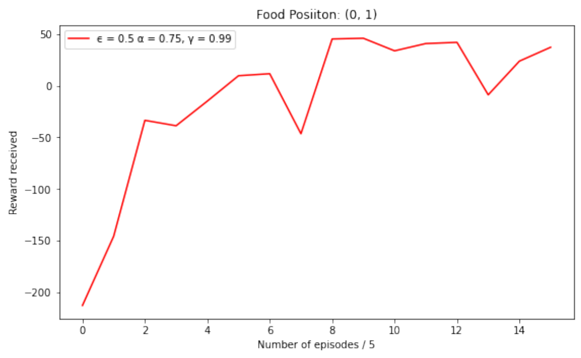
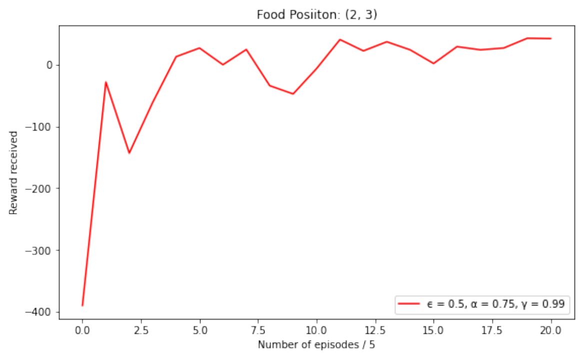
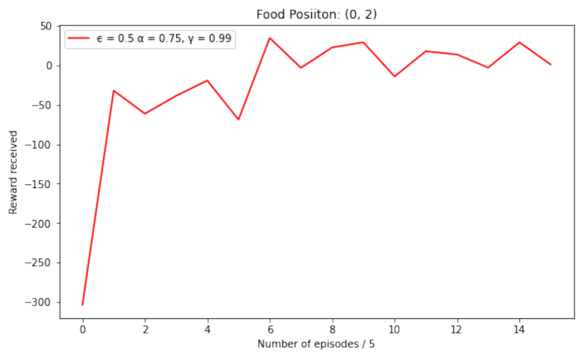
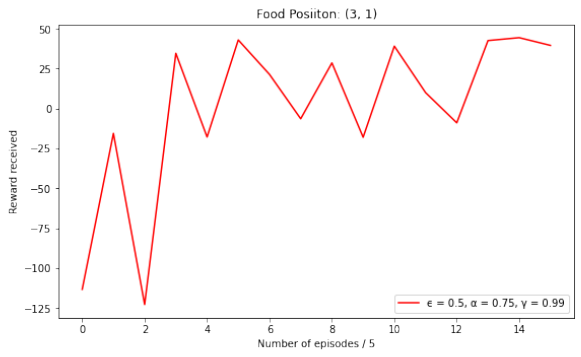
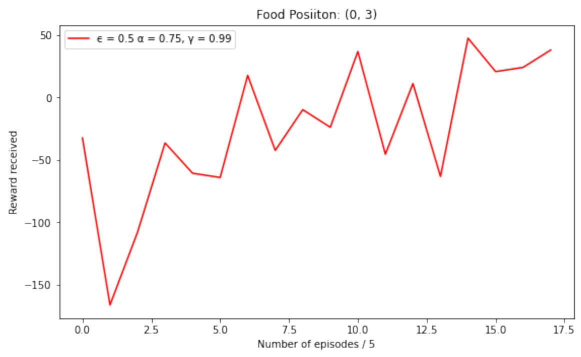
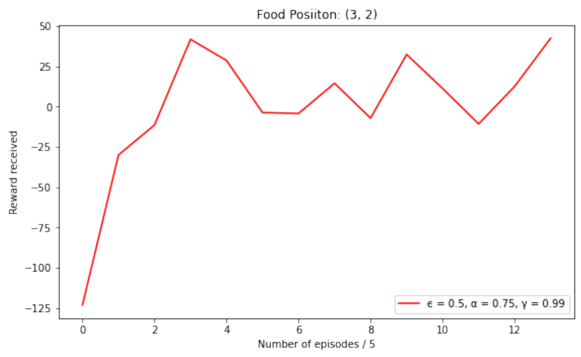
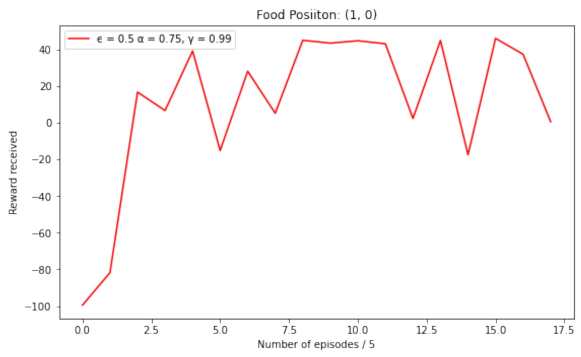
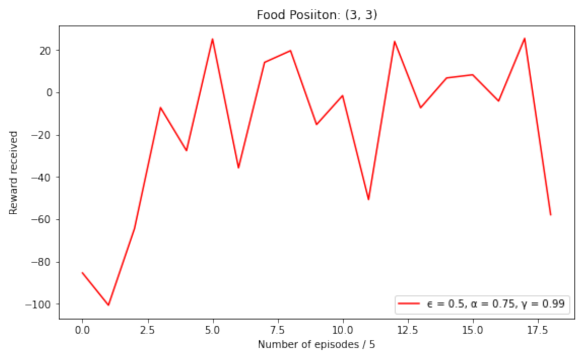
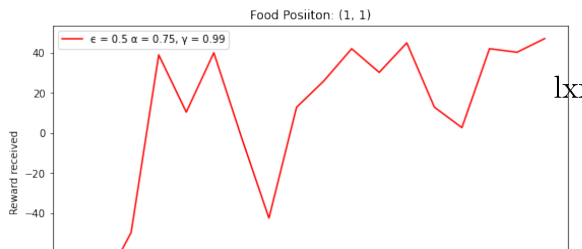
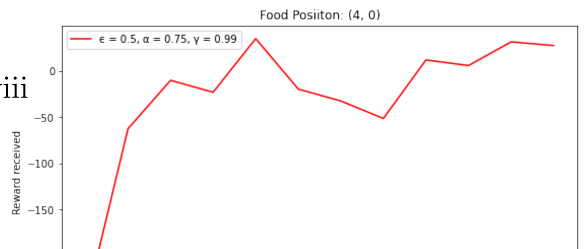
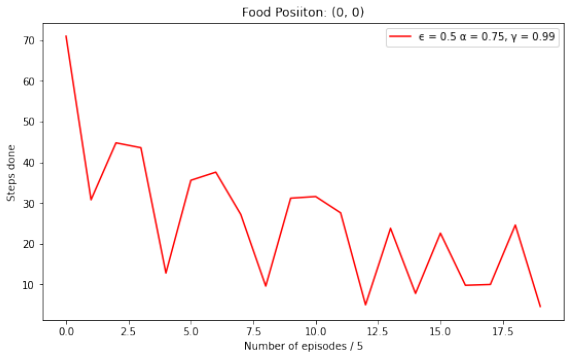
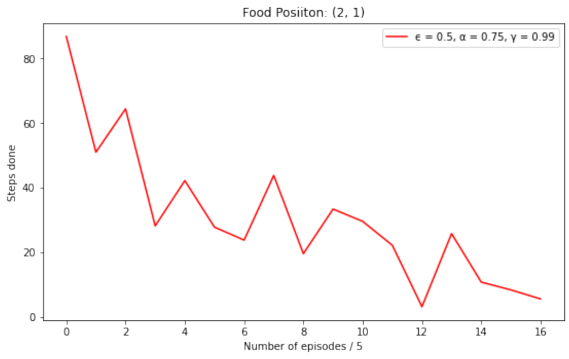
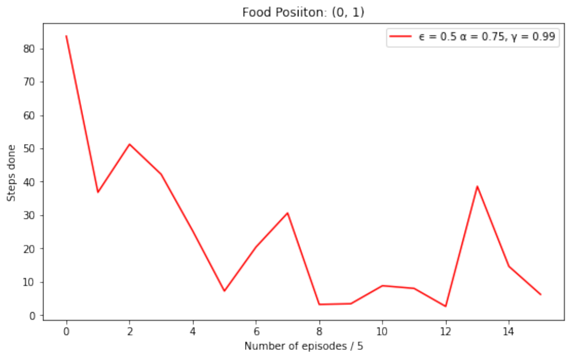


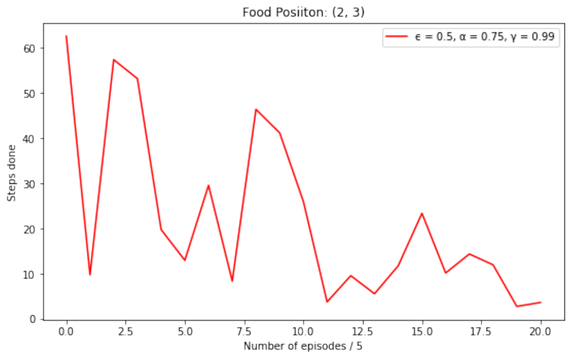
Figure 27: Ablation study: Comparing epsilon, rewards

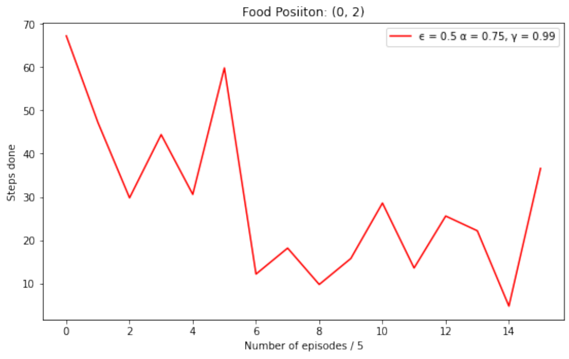

Food Posiiton: (0, 0)
ϵ = 0.5 α = 0.75, γ = 0.99
Reward received
Number of episodes / 5

Food Posiiton: (2, 1)
ϵ = 0.5, α = 0.75, γ = 0.99
Reward received
Number of episodes / 5

Food Posiiton: (0, 1)
ϵ = 0.5 α = 0.75, γ = 0.99
Reward received
Number of episodes / 5

Food Posiiton: (2, 3)
ϵ = 0.5, α = 0.75, γ = 0.99
Reward received
Number of episodes / 5

Food Posiiton: (0, 2)
ϵ = 0.5 α = 0.75, γ = 0.99
Reward received
Number of episodes / 5

Food Posiiton: (3, 1)
ϵ = 0.5, α = 0.75, γ = 0.99
Reward received
Number of episodes / 5

Food Posiiton: (0, 3)
ϵ = 0.5 α = 0.75, γ = 0.99
Reward received
Number of episodes / 5

Food Posiiton: (3, 2)
ϵ = 0.5, α = 0.75, γ = 0.99
Reward received
Number of episodes / 5

Food Posiiton: (1, 0)
ϵ = 0.5 α = 0.75, γ = 0.99
Reward received
Number of episodes / 5

Food Posiiton: (3, 3)
ϵ = 0.5, α = 0.75, γ = 0.99
Reward received
Number of episodes / 5

Food Posiiton: (1, 1)
ϵ = 0.5 α = 0.75, γ = 0.99
Reward received

Food Posiiton: (4, 0)
ϵ = 0.5, α = 0.75, γ = 0.99
Reward received

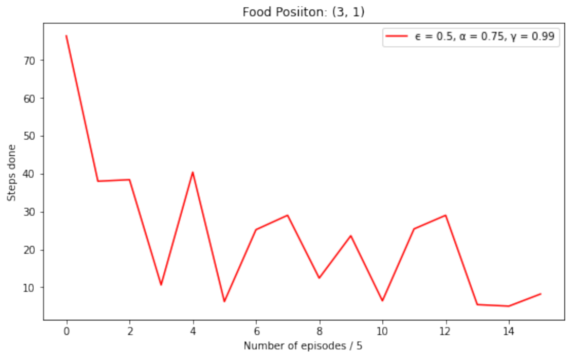

Food Posiiton: (0, 0)
ϵ = 0.5 α = 0.75, γ = 0.99
Steps done
Number of episodes / 5

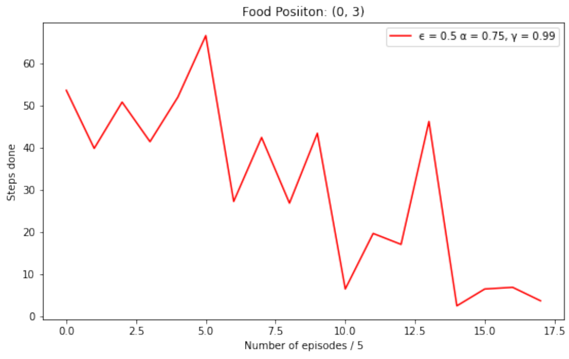

Food Posiiton: (2, 1)
ϵ = 0.5, α = 0.75, γ = 0.99
Steps done
Number of episodes / 5

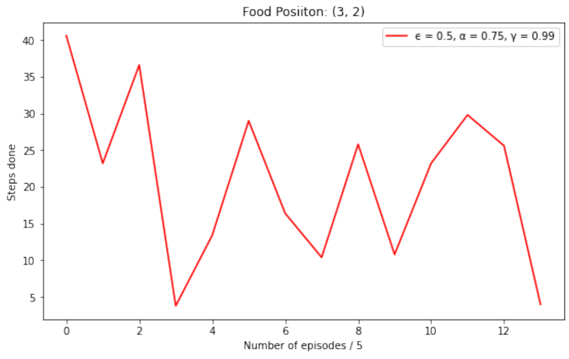

Food Posiiton: (0, 1)
ϵ = 0.5 α = 0.75, γ = 0.99
Steps done
Number of episodes / 5

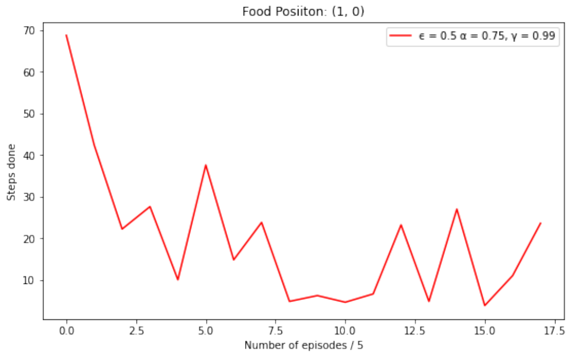

Food Posiiton: (2, 3)
ϵ = 0.5, α = 0.75, γ = 0.99
Steps done
Number of episodes / 5

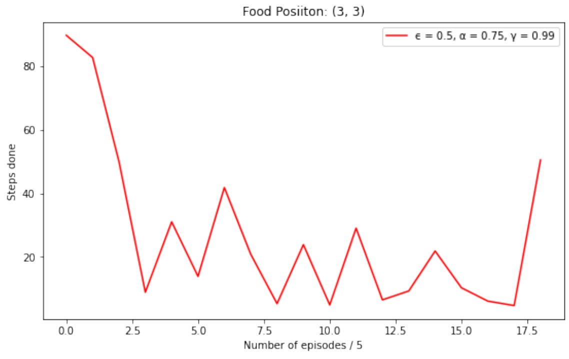

Food Posiiton: (0, 2)
ϵ = 0.5 α = 0.75, γ = 0.99
Steps done
Number of episodes / 5

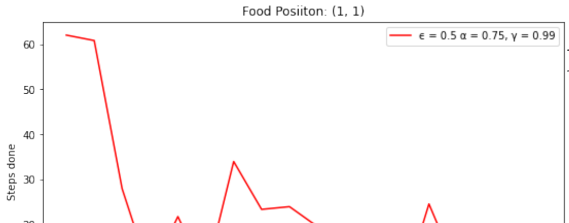

Food Posiiton: (3, 1)
ϵ = 0.5, α = 0.75, γ = 0.99
Steps done
Number of episodes / 5

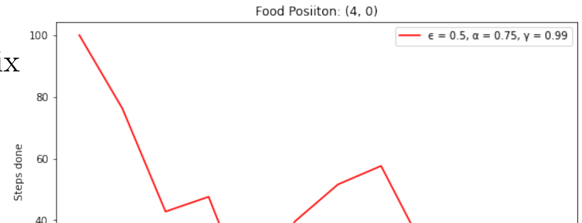

Food Posiiton: (0, 3)
ϵ = 0.5 α = 0.75, γ = 0.99
Steps done
Number of episodes / 5


Food Posiiton: (3, 2)
ϵ = 0.5, α = 0.75, γ = 0.99
Steps done
Number of episodes / 5


Food Posiiton: (1, 0)
ϵ = 0.5 α = 0.75, γ = 0.99
Steps done
Number of episodes / 5


Food Posiiton: (3, 3)
ϵ = 0.5, α = 0.75, γ = 0.99
Steps done
Number of episodes / 5


Food Posiiton: (1, 1)
ϵ = 0.5 α = 0.75, γ = 0.99
Steps done


Food Posiiton: (4, 0)
ϵ = 0.5, α = 0.75, γ = 0.99
Steps done