POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

TOWARDS A REALISTIC COMPUTATIONAL MODEL OF PAIN HANDLING IN THE HUMAN BRAIN

Relatore: Prof. Andrea Bonarini
Correlatore: Prof. Hiroshi Ishiguro

Tesi di Laurea di:
Luca Piccolo, matricola 799283

Anno Accademico 2013-2014

Abstract

In this thesis, that is the first research effort in a long term research project, we develop a computational model to reproduce as faithfully as possible the mechanisms that in human brain are involved in pain handling: pain prediction and learning of meaningful actions for pain avoidance, with particular attention to their integration and interaction modes.

We propose a design for the three main behavioral modules identified in humans from psychologists and neuroscientists and we verify their good performance, then we design their integration mode and we show how it allows to reproduce interaction phenomena that we would expect from nature, then finally we provide some preliminary results that show how our model seems suitable to explain a psychological pathology related to pain handling. We showed all of this in a test scenario in which there is an arena with some obstacles and inside it a robot predator that chases a robot prey: the prey feels pain when the predator is close, so we could control the prey using our model and test our theories.

The novelty of our work resides in several aspects: we propose a clear and generalizable design for the three behavioral modules (strongly improving the generality of solutions presented in the literature), it is the first example of a computational version of the three modules working all together and we show the first results of a pathology explained through our model. Also, something that is very important is that the scenario that we adopted to test our model is significantly more complicated than typical scenarios used in previous works.

We show results that support the plausibility of the proposed architecture and we draw some conclusion about what of our model could be improved and in which direction the research project could be directed from now on.

Sommario

In questa tesi, che è il primo sforzo di ricerca all'interno di un progetto di ricerca a lungo termine, sviluppiamo un modello computazionale per riprodurre il più fedelmente possibile i meccanismi che nel cervello umano sono coinvolti nella gestione del dolore: la predizione del dolore e l'apprendimento di azioni che siano significative per evitarlo, con particolare attenzione alla loro integrazione e modalità di interazione.

Proponiamo un design per i tre principali moduli comportamentali identificati negli uomini da psicologi e neuroscienziati e verifichiamo che abbiano buone performance, poi definiamo la loro modalità di integrazione e mostriamo come permetta di riprodurre quei fenomeni di interazione che ci aspetteremmo dalla natura, infine forniamo dei risultati preliminari che mostrano come il nostro modello sembri adatto per mostrare una patologia psicologica collegata alla gestione del dolore. Abbiamo mostrato tutto ciò in uno scenario di test nel quale c'è un'arena contenente alcuni ostacoli e al suo interno un predatore robot che insegue una preda robot: la preda prova dolore quando il predatore è vicino, quindi possiamo controllare la preda usando il nostro modello e testare le nostre teorie.

La novità del nostro lavoro risiede in diversi aspetti: proponiamo un design chiaro e generalizzabile per i tre moduli comportamentali (migliorando significativamente la generalità delle soluzioni presentate in letteratura), è il primo esempio di una versione computazionale dei tre moduli che lavorano tutti insieme ed inoltre mostriamo i primi risultati di una patologia spiegata attraverso il nostro modello. Inoltre, una cosa molto importante è che lo scenario che abbiamo adottato per testare il nostro modello è significativamente più complesso dei tipici scenari usati nei lavori precedenti.

Mostriamo dei risultati che supportano la plausibilità dell'architettura proposta e traiamo alcune conclusioni su come il nostro modello possa essere migliorato e in quale direzione il progetto di ricerca possa essere condotto da qui in avanti.

Acknowledgments

First of all, I would like to thank the two people that followed me during this months of work: Dr. Fabio Dalla Libera and Prof. Andrea Bonarini.

Fabio, post-doc at Intelligent Robotics Laboratory, proved to be an exceptional supervisor, allowing me to test my ideas and theories, but being there for me when I got stuck or I was not sure about the best path to follow, taking the time to reason with me about issues and to explain me why one path would be better than another, not just imposing me his solutions; apart from this, he also helped me to bond with other members of the lab, one thing that you really appreciate when you just arrived by yourself on the other side of the world.

Professor Bonarini, my advisor for this thesis, proved to be an excellent advisor, giving me precious ideas to continue with my work and at the same time allowing me to pursue my own ideas in my first experience in research; also, he was the one to make all of this possible, believing in me from the beginning and giving me the contacts to come to this lab. Finally, even if he was not obliged to do so he helped me a lot with precious advice about issues that were not strictly related to this thesis, and growing up you really appreciate people giving to you their time.

Then I would like to thank Professor Hiroshi Ishiguro and Professor Yuichiro Yoshikawa for believing in me and accepting me in Intelligent Robotics Laboratory, and also for their precious ideas and indications, that allowed me to always keep a broader view on the whole research project even when I was concentrated on low level details. I also would like to thank Dr. Ben Seymour for the countless precious insights about the biological aspects of our work and for the precious ideas about how to shape the research. I want to thank all the people of the lab's staff, that helped me to organize all of this and helped me with the burden of bureaucracy.

I would like to say a big thank you to all the people of the lab, that welcomed me and allowed me to have a great time there, especially those with whom I spent most of my time. A special mention goes to James, my

travel partner and my official disturber while I was working (but I really appreciated that), and to Eduardo, for the countless technical stops (we will have the next one in Europe I guess) and the countless discussion that taught me a lot.

I would like to thank all the people from Campus Martinitt in Milano for the awesome time I spent there, especially those of the whatsapp group. I really enjoyed my time in Milano and it was thanks to you guys, I hope to meet you again somewhere, to share memories and create new ones.

Thanks to all my friends, we shared lots of memories and some of the best stories I have to tell involve you, and I think this is very important. Most of all, a huge thank you to my very few closest friends: you have been an irreplaceable support throughout these years, being there for me when I needed it and being ready for a beer whenever I wanted to chill out; if I arrived where I am and did what I did it depended a lot on you.

Finally, last but not the least, a big thank you to my whole family for the tireless support and for allowing me to study without thinking about money.

Contents

Chapter 1

Introduction

The work presented in this thesis has been developed during my stay in Intelligent Robotics Laboratory of Osaka University in Toyonaka (Osaka prefecture, Japan) and represents the very first research effort in the context of a long term research project. Dr. Hiroshi Ishiguro (the director of the Intelligent Robotics Laboratory) and Dr. Ben Seymour, well known names in their fields (respectively robotics and neuroscience), started this joint research effort trying create a computational model of the brain parts involved with pain processing that could mimic the way these parts work and interact as closely as possible and use it for the development of realistic life-like robots.

The project lies in the crossroad of these researchers' needs and fields of expertise, so that the problem can be tackled from different perspectives. Hiroshi Ishiguro started a long time ago to study through robotics the human nature and the essence of what makes us humans, taking inspiration and relating closely to the biological world; trying to model and reproduce humans and human behaviors as closely as possible, then simplifying and reducing to the very essence these models, he tries to identify the boundary between what one deems life-like and inanimate. Ben Seymour throughout his career invested a considerable effort in studying and understanding the way humans process pain, trying to tackle problems such as as chronic pain; in recent years he also started to study and analyze the possibility of designing a computational model that could reproduce what he observed in nature, because "what I cannot create, I do not understand".[1]

We can consider pain as a very important component for living creatures, that influences several aspects of their everyday life, like for instance the way

[1] The quote, originally from Richard Feynman (Nobel prize in physics in 1965), appears in the opening slide of a presentation that Ben Seymour gave at Osaka University.

they behave and interact with each other. Ishiguro's main purpose in this context is to study and reproduce as closely as possible the way human pain works, trying to understand which pain related mechanisms can actually lead to the realization of more life-like robots, while Seymour's main purpose is to model and test state of the art theories about how pain related mechanisms work in the human brain, trying with this process to understand more about how the human brain itself works.

As a first effort in this long lasting research project, the main target of this work was understanding and reproducing some basic pain related mechanisms, leaving a refinement of our work and an extension to more complex mechanisms as future work. Three main behavioral modules related to pain have been identified by psychologists and neuroscientists. The first one, Pavlovian control, comes from evolution and represents powerful but inflexible actions. Then we have instrumental learning, that is divided into goal-directed control and habitual control; in this case, the modules have learning capabilities and can shape the actions they generate to give the maximum possible benefit to the individual. goal-directed control consists in a model of the environment that is explored through computationally expensive tree searches, while habitual control stores synthetic information that allows the individual to generate actions at a lower computational cost.

Our very first goal was creating a computational equivalent that would reproduce what we know from the scientific literature: if for one of the modules (habitual control, or model-free) this had already been reached, we deem that the design we proposed for a second module (goal-directed control, or model-based) is closer with respect to previously proposed techniques to what science knows about the biological counterpart, while for the third module (Pavlovian) we could not find at all a clear computational definition.

If from a neuroscientifical perspective it is sufficient to test these modules in the simplest possible environment, as several works in the literature show, it was very important for Ishiguro to create a testing framework that could be applicable to a robotics context, so another important goal was to define such framework and to show that our modules can actually work and interact in this more complex environment. This made everything significantly more complex, but is a fundamental step towards the application of these theories to realistic environments.

Following this first goal, a second important issue to tackle was starting to study and define how these modules interact, to create what to the best of our knowledge is the first working example of the three modules working together. Also, we wanted to study the characteristics of these modules' interactions during the learning processes, trying to understand if it was

possible to design a procedure for their integration that allowed to have a biologically sound interaction that reproduced the mechanisms that we would expect from previous psychological studies. In particular, we wanted to find an integration mechanism that could show at least partially how the modules interacted with each other so that the strong points of each of them emerged, defining the evolutionary soundness of what we did.

Finally, we also wanted to start studying what happens when this equilibrium between modules breaks down, that is what happens in humans when they are affected by psychological pathologies. This also probably constitutes the most interesting part of this work and gives an idea of what kind of issues and longer term goals could be pursued within this research project, creating contents that could be interesting for both the robotics and neuroscientific worlds.

If we take a look at the research project with a broader perspective, there are several interesting issues that could be explored starting from here. For instance, recently some researchers used a robot that had writing problems to teach to children how to write: what if we asked to Obsessive-Compulsive Disorder patients to interact with robots presenting OCD symptoms? What if we used robots showing the symptoms of some pathology for the first part of training for professionals that have to deal with that pathology? What if we studied how pain relates to social activities and how behaviors such as altruism generate, so that we would have robots accepting to endure some unnecessary pain just to help other robots? This work is the very first milestone of a research project that we are confident in the future will allow us to tackle more and more interesting problems and scenarios.

In Chapter 2 we present and analyze the related scientific literature and we explain how all these mechanisms work in a human brain, as well as analyzing previous efforts from a computational perspective; in Chapter 3 we define our proposed solution to create a computational model of these mechanisms, defining and explaining how we designed the single modules and their interactions; in Chapter 4 we show and comment the main results obtained, we discuss some limits of our model and give some ideas about how one could continue from now on; finally, in Chapter 5 we perform a general wrap-up of our work and of the main achievements and limitations. We made the choice to introduce concepts only when they were necessary and to reduce information duplication as much as possible, so we warmly recommend to read the thesis cover to cover in the order it has been written for a much easier comprehension.

Chapter 2

Context and state of the art

In this chapter we outline the main theoretical foundations that are necessary to understand our work and we explore the related scientific literature. In Section 2.1, we explain the biological and psychological basics underlying our work, in Section 2.3 outline how the processes outlined in the previous section have a neuroscientific correspondence, in Section 2.4 we explore the main related works from the perspective of engineering and computational models and finally in Section 2.5 we explain the main motivations for our work and why we deemed it was an important contribution to the field.

2.1 General foundations

This section gives an overall view of the biological and psychological processes that constitute the foundations of our work. Care must be taken since not all the references specifically refer to pain signals: we assume that the reader will be able to adapt the contents to such context. Also, some of the cited works refer to animals and not to humans, but the underlying mechanisms to which we refer behave in a very similar way.

2.1.1 A metaphor to understand the different types of control

We decided to postpone to the next section the exploration of the literature in order to try to guide the reader's comprehension through a metaphor: it is probably not the most rigorous way to explain the concepts, but it supposedly is a very good way to present these concepts to one that does not have them in one's background, so that one will be able to bring back these concepts to one's everyday experience.

Imagine that you are driving a car for the first time. Probably, you will think a lot about what to do, through mental processes similar to this: "In order to change gear I need to lift my right foot from the gas pedal, press the clutch pedal with my left foot and hold it to the floor, then move the gear shift knob to the neutral position and to the desired gear, then finally lift my left foot from the clutch pedal and put a slight pressure with the right foot on the gas pedal". As you know, this process is quite complicated and requires you to perform a consistent mental effort, so we have been endowed with a different procedure to generate our actions. As time goes by, habits will start to settle and a lot of these action chains will become automatic, so that you will not have to think about all these issues. Eventually, you will probably end up having a perfectly automatized set of actions regulated by habits, so that you can drive a car without thinking too much about it; is it even likely that in that moment you will get a better and smoother driving since, as it is said, "practice makes perfect". What if during your driving lessons, especially the first ones, you risk to bump into another car or to go off the road? If you are lucky, your instincts will save you, allowing you to press the brake pedal or to steer: probably these will not be optimal actions, in the sense that they will probably be sudden and abrupt, but they will serve the purpose. The explanation we just gave sets the basics for what we will talk about in the following sections: the habits that the driver develops correspond to the habitual control, the activity of hard and thorough thinking corresponds to the goal-directed control and the instincts correspond to the Pavlovian control. The explanation should have also helped the reader to understand the relation that holds between the different modules and how they interact in order to give to the individual the possibility to perform meaningful actions.

Now, imagine that, once you have got used to a manual car you sit into a semi-automatic car. Of course, part of the knowledge you have acquired driving a manual car will still be applicable and useful, so you will not start from zero, but you will have to get used to all the different aspects of your new car. Especially at the beginning, you may still need your instincts to save you, for instance when you forget that such cars do not have a clutch pedal and you press heavily the break pedal with your left foot; also, you will need to rethink some of your actions during the first times you drive the car, for instance trying to adapt and get used to the different way of changing gears. However, if you stick with that car and drive it a lot of times, you will eventually get used to it. As you can observe, in the face of the first learning as well as in the face of change the different modules will cover different roles: Pavlovian control will provide instincts to save

the agent from a strongly negative outcome, even if through sub-optimal actions; goal-directed control will then allow the user to plan his actions in a new situation that does not allow him to exploit his habits, even if this planning will require a substantial mental effort; finally, habitual control will come into play and provide optimal actions that require a limited effort from the individual.

Now that the reader supposedly has a clear background about these issues, in the next section we discuss some publications that explore them.

2.1.2 The three modules

A good and thorough description of the modular structure of pain handling in humans can be found in *Seymour and Dayan (2009)* and *Seymour et al. (2007)*, that contain the description of the three most important modules involved in evaluating outcomes and choosing actions.

Goal-directed control (model-based values) is a way for an individual to make predictions about future rewards or punishments. It basically consists of a model of the world that encircles states, actions, action outcomes and expected utilities. "This sort of outcome-sensitive control is a close relative of human notions of 'cognitive' control, in which individuals explicitly consider the outcome of actions, and of subsequent actions, and use some form of tree-search to inform current actions"[1]. The advantage of this kind of models is that they can easily adapt to changes in the environment. However, using them to find the best possible action is typically computationally expensive and requires consistent amounts of memory, so that it can be used to find accurate values only in small environments.

Habitual control (model-free values) gives a way round the complexity by storing information that synthesize what would be the result of a tree search. We call this kind of control "model-free" because we can use the information in a way that is independent from the action outcomes. A way to do this is through temporal-difference learning, explained for example in *Sutton and Barto (1998)*, with value estimates: since estimates are bootstrapped from adjacent states, there is no need to represent action outcomes. These values are a direct representation of the desirability of an action in a particular state, so that there is no need to perform expensive tree computations. How-

[1] Quote from *Seymour and Dayan (2009)*.

ever, the drawback of these models is a lower flexibility in the face of changes.

Pavlovian control represents "powerful, pre-specified, but inflexible alternatives"[2]. The state-outcome pairing is used to produce anticipatory responses whenever the predictive state is encountered again. This kind of control, that in nature comes from evolution, has a lower flexibility with respect to the other two kinds of control, but removes the need of an expensive learning process. Apart from innate responses to stimuli, we also experience Pavlovian conditioning, that "is a process of behavior modification in which an innate response to a potent biological stimulus becomes expressed in response to a previously neutral stimulus"[3]. Pavlovian conditioning can in turn be of two different kinds: consummatory, when the form of the conditioned response is in the same form of the perceptual properties of the stimulus (e.g., when a conditioned stimulus is associated to the administration of food and as such increases salivation), and preparatory, when the responses are not specific to the stimulus but rather they are characteristic to the motivational class to which the stimulus belongs (e.g., anatomic responses, like increased heart rate, and behavioral responses, such as attempted escape). We basically only consider escaping actions, but it would be interesting to extend our work with other alternatives (see e.g., *Fanselow (1994)*).

An overall view can be found in *Figure 2.1.* The image is taken from *Seymour et al. (2007)*, but we modified it removing the "learning following observation" process, since we do not consider it in our work.

The main difference between Pavlovian learning and instrumental learning (i.e., habitual and goal-directed learning) is that in the latter one the individual learns to associate an action with its outcome (even though, as it was said beforehand, in habitual learning the outcome is not explicitly represented), so that one learns about the causal consequences of one's behaviour and can control them depending on one's needs. In *Seymour and Dayan (2009)* it is also outlined how the lack of a pain signal (i.e., the lack of a negative signal) can serve as a positive signal, so that the pain alone is sufficient for a learning process to take place.

[2]Quote from *Seymour and Dayan (2009)*.
[3]Quote from wikipedia.org.

2.1.3 Interactions between modules

How does the brain decide which one of the three mechanisms should be used to produce behavior? The modules are in competition and proper experiments can be designed to show their interactions. It may be especially tricky to understand the exact interactions between habitual and goal-directed control.

A discussion about their relation can be found in *Gillan et al. (2015)*, where it is shown how in healthy individuals there is a tendency towards habit formation and how model-based learning gives a higher sensitivity when facing outcome devaluation[4]. Nevertheless, it is also shown that typical behavior comes from a mixture of goal-directed and habitual behaviors. As a way to measure the relationship between goal-directed and habitual control, outcome devaluation consists in reducing the value of a given outcome, then measuring in extinction the responses of the individual to the stimulus associated with the outcome. In this context, "in extinction" means that the individual is informed about the change in outcome values, but does not have, e.g., a visual input during the experiment about the values: the

[4]Theoretically, "outcome devaluation" should refer only to an outcome whose value has been reduced, but in the literature it is used more generally to refer to outcomes whose values have changed.

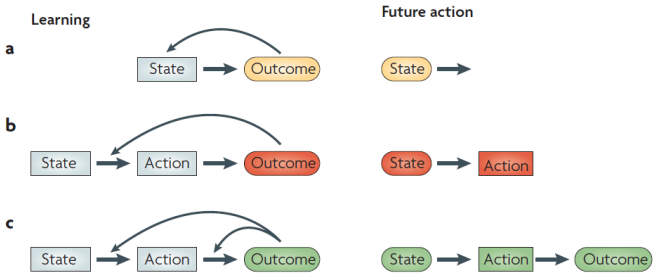

Figure 2.1: **Different mechanisms of learning and action**. **a**| Pavlovian responses: repeated pairing of a state or a cue with a certain outcome causes an innate response when the state is encountered again. **b**| Habitual learning: if an action is executed in a certain state and it brings a reward, the likelihood of performing that action in that state in the future will be increased. **c**| Goal-directed learning: if an action in a certain state brings a reward, then an explicit representation of the sequence is remembered for future action generation.

individual knows about the devaluation, so goal-directed control comes into play, but he receives no feedback about the values during the experiment, so that habitual control learning is slowed down. It is also interesting to notice (*Gillan and Robbins (2014)*) that the definitions of goal-directed and habitual control are somehow linked, since there is no current definition of the latter that does not rely on the absence of the former.

In *Balleine and Dickinson (1998)* it is discussed why a simple stimulus-response model is not sufficient to explain all the shades of behavior: basically, a simple stimulus-response mechanism cannot represent and encode the relationship between an action and a reward, so that stimulus-response models are only sensitive to the contingency of actions and reinforcements and not to the causal relationship between them.

In *Dayan et al. (2006)* some of these conflicts are outlined and it is discussed how in animals the Pavlovian control can sometimes even produce a behavior that is not consistent with the instrumental control.

2.2 Related pathologies

With respect to pain, the Obsessive-Compulsive Disorder (OCD) consists in a failure or anyway a slowdown in the extinction of avoidance actions (see *Gillan et al. (2014)*). Even if the instrumental learning of pain avoidance actions in unaffected, meaning that healthy subjects and OCD patients learn at the same speed how to avoid pain, among the latter the ability to to extinguish those actions is reduced: this means that, even if some learned actions will lose their meaning because they will cease to be associated to pain signals, OCD patients will keep performing those actions. It is outside the scope of this thesis to explain the psychological reason of why this happens, also because not all researchers agree on the same explanation.

It has been observed (*Gillan et al. (2011)* and *Gillan and Robbins (2014)*) how Obsessive–Compulsive Disorder (OCD) modifies the balance between goal-directed and habitual control. Even if healthy subjects and OCD patients show comparable ability in using external feedback for instrumental learning, the latter show a much more restricted sensitivity to outcome devaluation and a weaker knowledge of the associations between actions and outcome. Even if the original hypothesis was slightly different, the latest findings suggest that rather than an impairment in goal-directed control, OCD may be caused by an exaggerated mechanism of habit formation. To make it simple, we could say that while OCD patients and healthy individuals tend to perform instrumental learning in a similar way, when the former have acquired some habits they tend to stick with them. *Gillan et al. (2014)*

shows these mechanisms in an avoidance context, where healthy subjects and OCD patients need to learn to avoid shocks, but the latter tend to stick with avoidance actions even when the machine that performs the shocks has been unplugged and all subjects have been informed of this change.

2.3 Neuroscientific correspondences

As we have said, for our work the similarity with the biological world is very important. In the previous sections we described a three modules system, but we only justified it from a psychological perspective, that is somehow a behavioral perspective. It is very important in our opinion to show some examples of neuroscientific works that studied these same mechanisms and show how the considerations in previous chapters are grounded also from a neural perspective. There is evidence for the involvement of several brain areas in the activation of these mechanisms, even if most of the material refers to rewards and not to punishments.

For model-based control, there is evidence of the involvement of the dorsolateral prefrontal cortex, the ventromedial prefrontal cortex, the left parahippocampal gyrus and the middle frontal gyrus (see *Balleine and Dickinson (1998)*, *Carter et al. (2006)* and *Koechlin et al. (2003)*).

Regarding the habit-based system, for the reward prediction error there is evidence of the involvement of the striatum (mainly the bilateral ventral striatum and the left posterior putamen), the right anterior insula and the dopamine neurons of the ventral tegmental area and of the substantia nigra (see *Pessiglione et al. (2006)* and *Schultz et al. (1997)*).

Pavlovian control seems to be under the control of the periaqueductal grey (PAG) (and amygdala for the correlated emotional aspects), while autonomic[5] changes involve connections between the amygdala and the dorsal medullary nuclei (see *Fanselow (1994)* and *Fendt and Fanselow (1999)*).

2.4 Computational models

To the best of our knowledge, there is still no work showing the complete pain model working, with all of the three modules interacting and working together, so our work is intrinsically novel and unique. Anyway, we present and analyze here some publications that are somehow related, typically because they propose a computational alternative to a part of our model.

[5]The autonomic nervous system is the involuntary nervous system and regulates (mostly unconsciously) things like heart rate, digestion, respiratory rate and so on.

In literature, one can find articles (e.g., *Bonarini (1997)*) that show models that are able to shape behaviors suitable for complex environments with multiple agents interacting, also in a prey-predator setting. A lot of care must be taken when reading these articles and comparing them to the content of this thesis, since they are conceptually different from our work and for this reason they are not a suitable comparison. The main point on which the reader should focus on is that these models are probably the most efficient and optimized engineering solution to solve the prey-predator problem (i.e., to control the predator or the prey), since their purpose is exactly that of having a prey that escapes in an effective way or a predator that can learn meaningful movements to catch a prey, while our model's main purpose is that of trying to reproduce and mimic with the highest possible accuracy the largest possible number of mechanisms and phenomena that have been identified in the human brain with respect to pain signal processing. In this sense, it is really important to keep in mind this point while reading our work: for us the prey-predator setting is just a framework to test our model, so our purpose is not to design a controller that shows an optimal chase or escape behavior, but a controller that can faithfully mimic the internal and external behavior of the human brain as it is understood from current state of the art works.

Dayan et al. (2006) shows a computational model able to show the negative automaintenance phenomenon[6], but the approach adopted does not seem generalizable to our purposes; in fact, within a single mechanism of action choosing an action is selected as Pavolvian and favoured artificially, but it is not clear how such an approach could be extensible to an automatized learning setting. Also, since as mentioned beforehand most of the literature agrees on the fact that the three modules are biologically located in different areas of the brain, it seems more logical to keep them separate. Still, the underlying phenomenon outlined in the article is interesting and important, and we will come on this again later on.

In *Gillan et al. (2015)* (in the supplemental material) a different model is proposed for model-based and model-free control. Since in our model (see Chapter 3) we discount rewards during ϵ-greedy exploration and each row

[6]Negative automaintenance "uses an omission schedule to pit classical and instrumental conditioning against each other. [...] the pigeons are denied food on any trial in which they peck the lit key. In this case, the birds still peck the key (albeit to a reduced degree), thereby getting less food than they might. This persistence in pecking despite the instrumental contingency between withholding pecking and food shows that Pavlovian responding is formally independent from instrumental responding, since the Pavlovian peck is never reinforced". (quote from *Dayan et al. (2006)*)

of the table constitutes an approximate model of the transition probabilities, we deem that our model-based control is extremely similar to the one used in this work. However, the action generation policies are not directly comparable, since they are very different conceptually; while we first decide which module should act and then that module alone generates an action, in their work they generate actions mixing the action evaluations coming from the different modules. Also, they only use model-based and model-free control, so their model is not directly comparable to ours.

Schultz et al. (1997) proposed a model to reproduce temporal difference learning only, that would correspond to our habitual control. Since it is only one module and it tackles some more specific issues it does not make sense to make a comparison, since our work and their work have different purposes. However, the aspects that they claim their model is able to explain (e.g., blocking[7] and secondary conditioning[8]) could be investigated in future work: it would be interesting to understand if our model is able to explain those mechanisms, and in case it is not their work could be a starting point for an improvement.

Lee et al. (2014) introduces interesting ideas for the model-based module, but we decided to discard it for several reasons. First of all, because of the way they formulate the function $T(s, a, s')$ and keep it updated, their techniques does not seem to scale well to complex environments. Another issue is related to the future rewards: in their case due to the simplified environment they can assume that rewards are known from the first moment, but it does not apply for a complex environment where rewards need to be discovered through experience. Finally, given the shape of their equations we had doubts about the reactivity of such a technique after an extensive training session. Anyway, their main contribution from a computational point of view is in a quite articulated technique for model-based and model-free arbitration based on their relative uncertainties: we did not adopt it for simplicity and we left it as an interesting idea to test in future works, but there are several issues that one should solve before adopting it. First of all it adopts a computationally inefficient technique to combine the actions of different modules and this should be solved, but we do not know if changing that technique would still give good performance. Apart from this, their Bayesian approach could cause a loss in reactivity after extensive training sessions. Finally, this kind of integration does not consider

[7] If there is already a cue that predicts pain (e.g., a light turning on), a second cue (e.g., a buzzer) will fail to acquire an aversive conditioned response.

[8] First a subject learns to associate e.g., a light with food, then it learns to associate e.g., a buzzer with the light.

the Pavlovian module, but it would not be straightforward to extend this integration technique to consider it.

Following similar considerations, we claim that the models adopted in other literature works are not comparable with our work (e.g., see *Pessiglione et al. (2006)*). In general, the following reasoning holds: if we upgraded the other models to make them comparable to our model, for instance by extending them to three modules systems, with generalized reinforcement learning equations and so on, one could claim that depending on the way we chose to extend the models we would obtain different results; because of this, the only sound way to make a comparison would be that of downgrading our model to the same number of modules and the same simplified environment, but for the moment our main goal was that of obtaining a first version of the three modules model that could explain and show as many biological phenomena as possible. Still a comparison with proposed techniques for single modules design or single integration modes remains an important part for future work, trying to get the best possible overall system design.

2.5 Motivation for our work

There is more than one reason to say that our work is a relevant and important contribution to the field, namely:

- to the best of our knowledge, there is still no working example of the three modules working together, even if the three models have been identified and somehow separated in the psychological literature: we provide examples of the three modules working together and we deem that this already is an interesting point;

- we did not find a clear, meaningful and generalizable definition of a model for Pavlovian control: we provide a design and a way to use a Pavlovian control that is automatically learned and adapted to the context;

- all the examples that we have found that specifically design and test two of the pain modules together use an over simplistic environment, typically just a decision tree with a very limited number of possible states (i.e., between 3 and 7) and with rewards assigned only in the final states: we provide examples of the model working in a realistic environment (i.e., an arena with obstacles where a prey escapes from a predator) and with rewards distributed over time;

- an interesting detail linked to the previous point that is important to notice is that some models in the literature use over simplistic equations that for example ignore discount factors (i.e., are typical RL equations but with discount factor set equal to zero): we show a working example of a model that uses equations and a design suitable for a more general and generalizable approach, so that, for example, our model can take into account the fact that rewards may be distributed over long chains of temporal events;

- we could not find any example of a computational model of pain able to explain and show pain related pathologies: we achieved this and we deem this could be very interesting for scientists studying such pathologies, but it could also open new interesting scenarios (e.g., if we are studying human-robot interaction, how different it would be for a human to interact with a robot that shows OCD symptoms?).

Chapter 3

Logical design

In this chapter, we explain our proposed design for the pain system and the framework used to test it. In Section 3.1 we describe the framework that we used to test the model (i.e., the test scenario), in Sections 3.2, 3.3 and 3.4 we describe the design respectively of the model-free, model-based and Pavlovian modules, in Section 3.5 we explain how we integrated the model-based and model-free modules to obtain a meaningful behavior from their aggregation, in Section 3.6 we also integrate the Pavlovian module so that finally we have the design with the three modules working together, then finally in Section 3.7 we give some final notes about the different learning procedures that we adopted for instrumental learning (i.e., habitual and goal-directed learning) and Pavlovian learning.

3.1 Our framework

We needed a framework in which we could apply our model and check whether it complied or not with expectations coming from the literature analysed in Chapter 2, but we also wanted something that could be easily applicable to the world of robotics. Finally, it needs to be said that, even if in the future the model will probably be applied to complex and articulated robots (e.g., the Geminoid, see *Nishio et al. (2007)*, or for a much simpler design the Telenoid, see *Sumioka et al. (2014)*), at the current state of the work it was of particular interest to professor Ishiguro to apply the pain model to minimal design robots that are designed and produced in Intelligent Robotics Laboratory (see *Figure 3.1*), so we needed something that could be applied to such robots.

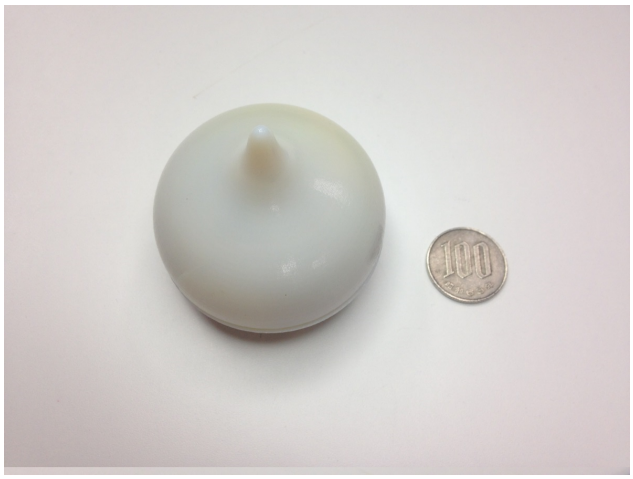

Figure 3.1: **The first robot that will be used to test the model**. It is an omni-drive robot. The external design and the number and typology of sensors onboard might change in the future.

3.1.1 The prey-predator setting

Given our needs, we decided to adopt a simple scenario in which a predator chases a prey inside an arena and when the predator is close to the prey the latter feels pain, a task that is similar to the one adopted in *Mobbs et al. (2009a)* and *Mobbs et al. (2009b)*. The arena we adopted is a squared arena and can contain an arbitrary number of obstacles of various shapes; this is important for the prey because, apart from escaping at a high distance, in order to avoid pain it can also simply hide behind an obstacle. For the sake of simplicity and since it would not have added any particularly interesting points to our work, we decided to stop the chase when the prey gets at a high distance from the predator or when it is hidden by an obstacle. The episode starts with both robots in random position in the arena and ends when the chase ends, as we have just explained. The predator is controlled by a very simple model, since we only needed a minimal behavior sufficient to test our model: basically, as soon as it sees the prey it turns in its direction and starts to chase it. Since wanted to allow the prey to easily escape once it learns the proper action policy and since it was not relevant for our purposes to push the prey-predator task to its limits, we decided to move the predator half of the time with respect to the prey (i.e. the predator is half as fast). The prey, instead, is of course entirely controlled by the model discussed in this thesis. As this was the starting point of this research, we decided to ignore for the

moment the dynamics of a robotics system and we adopted a discrete-space environment, obtained through an hexagonal tessellation of it. Given the hexagonal tile in which one of the robots currently is into, it will be able to move to any of the six hexagonal tiles that surround it. We also did not consider the concept of robot's direction: given a position in which a robot is into, that is everything that matters, we cannot say whether the robot is facing a certain direction or another. We chose an hexagonal tiling so that simulated robots could have higher freedom of movement while still having a constant distance between centers of contiguous tiles (while, for example, in a squared tiling we would have that distance between centers of tiles in a diagonal direction would be higher than the distance between centers of tiles in the vertical or horizontal direction). Since the robot that will be first used to test our model is an omnidirectional omni-drive robot, the hexagonal tiling is suitable, even though as we have said for now for simplicity we ignore the robot's direction. In order to test all our hypotheses and to see how our model performed in a robotics simulation, we wrote a simulator using C++ and OpenGL; a screenshot from the simulator (*Figure 3.2*) should give to the reader a clear idea about a typical setting.

It is of paramount importance to keep in mind that for us the prey-predator setting is just one of many possible frameworks and its purpose is just that of testing especially some particular theoretical properties of our model. As we have already discussed in Section 2.4, the performance of our model cannot be compared directly with that of standard approaches to the prey-predator problem: in those cases the purpose will just be that of escaping, while in our case we also care about other aspects (e.g., smooth interactions between modules and biologically inspired behaviors and learning curves), but these issues will be better explained later on in our work (especially in Chapter 4).

3.1.2 A note on the control system for the prey

Even though in this work the model has been designed and tested in a discrete space (i.e., the hexagonal tiling), the final purpose of this research effort will be that of showing all these results with real robots. A likely intermediate point will be testing our model on a continuous space simulation that also takes into account robots' dynamics. For this reason, there are some design choices that may be considered unnecessary in a discrete-space setting, but we claim that such choices will make the transition to a continuous-space setting much easier.

One of these choices is related to the control system for the prey and is

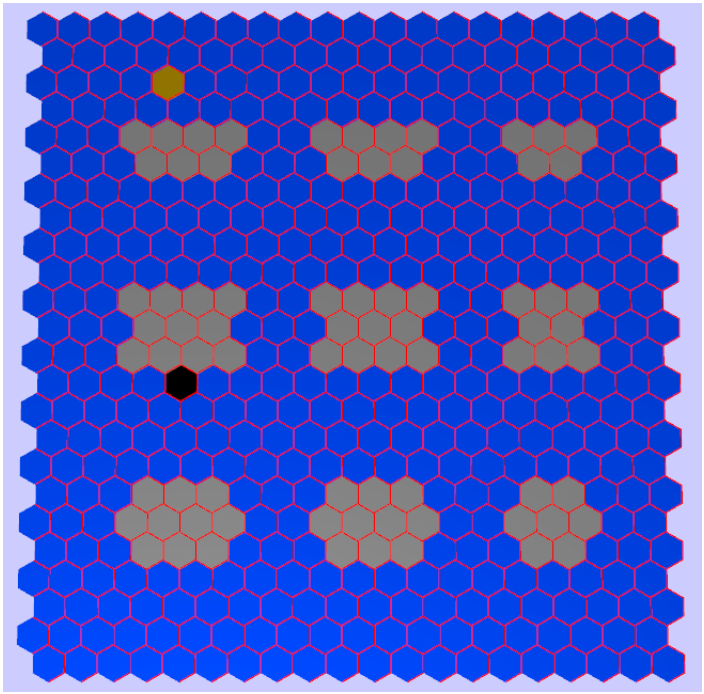

Figure 3.2: **First screenshot of the arena**. The blue tiles represent free terrain where robots can move, grey tiles represent space occupied by obstacles, the prey is the brown tile and the predator the black one.

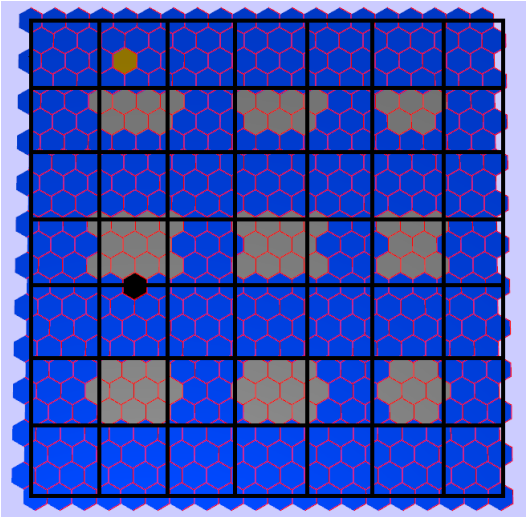

Figure 3.3: **The map with the grid drawn on it**. The grid has been drawn manually, but is very close to the grid actually used by the algorithm. The 7x7=49 squares generated by the black grid are the same squares that one can see in *Figure 3.4*.

very important to notice, since it is used in the algorithms explained in the following sections. Imagine that you are driving a car: you could say that at a high level you are following GPS points, while at a lower level you are trying to go from a GPS point to the next one without hitting other cars, going out of the road and so on. We apply an idea that is very similar; we divide the map in squared areas, drawing a grid on the map, and we take the squared area's centers as "GPS points". For every area, if more than a small percentage (in our case we considered 20%) is occupied by obstacles we consider it occupied and its center cannot be used as "GPS point", while if this does not apply the cell is considered free and we can use its center. Just to give the idea, the map shown in *Figure 3.2* on page 20 is shown with the squared grid drawn on it in *Figure 3.3* and is discretized like in *Figure 3.4*. From now on, we will refer with the term of "tiles" to the hexagons, while with the term "cells" to the squares of the grid.

Let us say that at a certain point in time we need to decide the new high level target: in order to do that, we first determine in which one of the cells the prey currently is into; then, the potential targets will be the center of the cell to which the prey currently belongs plus the centers of the eight cells that surround the current prey's cell. Once the next high level target has been chosen, the low level control will decide which low level

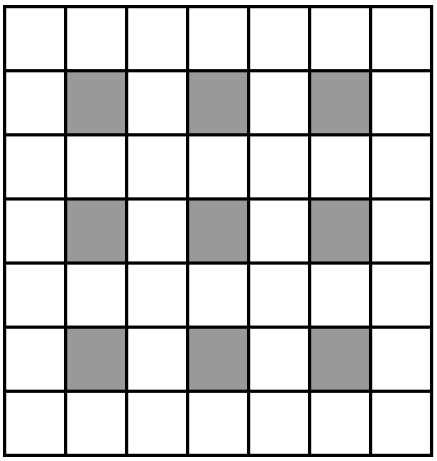

Figure 3.4: **The squared grid discretization based on *Figure 3.2***. The white squares represent cells considered free, while the grey squares represent cells considered occupied by obstacles.

moves to perform (i.e., which hexagonal tiles to get into) in order to get there. The learning algorithms will concentrate on picking the best possible cell center given by the squared grid, while the low level control that chooses movements on hexagonal tiles will be automatically controlled by a lower level controller: in our case, the latter controller will just choose the tile that with respect to the current high level target position has the smallest angle from the current position. Substantially, since the interaction with the hexagonal tessellation is handled by an independent and automatic low-level controller, to the eyes of our algorithm all that matters is the discretization shown in *Figure 3.4*.

What if the action generated by the algorithms tries to set the target in the center of a cell that is considered occupied? In this case a null action is generated: we consider the action non valid and we set the target to the center of the cell in which the prey is currently into; this means that the prey remains substantially still, since it is already in that cell. Since the predator is always chasing the prey, if the algorithms start to generate a big number of null actions they are penalized, because null actions slow down the prey since they do not push it towards other cells. This forces our algorithms to try to always generate valid actions.

It is important not to get confused by the two types of discretization used in our work: one thing is the hexagonal tessellation that we applied in order to have discrete-space, which regulates how robots can move at each

time step, whereas a different thing is the squared grid we draw on top of the hexagonal tessellation, which gives a series of high level set points that the low level control can then pursue. Another important application of the squared grid is the fact that it will be used to be part of the Reinforcement Learning state used by the prey, but again we postpone this discussion to Section 3.2.

Why then for the learning processes should we concentrate on a high level control based on the centers of the cells of the squared grid and not directly control in which hexagonal tile the prey wants to go? Probably one of the most natural steps towards a final implementation would be that of moving all our work towards a continuous-space simulation, that keeps into account a more realistic continuous space and robots' dynamics, and in this sense it should be easier for the reader to understand the reason for our choice: if the learning algorithms refer to the space defined in terms of the squared grid and not to the hexagons, a transition to continuous space should not pose any problems from this point of view. If one could argue that in a discrete space setting we could have attached the learning modules directly to the low level actions between hexagonal tiles, we deem that in continuous space it would be unsustainable to do that, requiring an excessive amount of time for learning, while this higher level layer allows for training in a reasonable time. We could then say that this particular design choice was not crucial in the current state of the work, but it will be as soon as the model will be extended to continuous space.

From now on, unless differently specified we will ignore the hexagonal tessellation that discretizes the map in tiles and the low level control, considering them transparent with respect to our learning algorithms and a sort of low level layer that depends from the particular application and it is modifiable in case we need to adapt our model to different application scenarios. We will just consider the discrete squared grid drawn upon the map that divides it in squared cells and the high level control that, given the cell in which the prey is currently into, decides in which one of the eight surrounding cells plus the current cell the prey should aim at. This higher level control is in fact our model and the content of this thesis.

3.2 Model-free module

First of all, we introduce the model-free module, that needs to meet the requirements specified in Section 2.1.2 on page 7. In order to obtain such model, we decided to use a RL technique called actor-critic (see *Witten (1977)* and *Sutton and Barto (1998)*, or see *Barto et al. (1983)* for a version

with also eligibility traces, where the actor is called "associative search element" and the critic is called "adaptive critic element"). Actually only the actor represents the model-free module, while the critic is Pavlovian, but we will come on this again later on (this has also been observed in *Barto et al. (1983)*). Let us start to understand how the actor-critic works, then we will explain why we chose this technique.

3.2.1 Theoretical basics

Actor-critic is a technique that can be used to find an optimal action selection policy in a Markov Decision Process (MDP); we then need to understand what an MDP is and what does it mean to find an optimal action selection policy.

An MDP is a mathematical framework that is useful to study situations in which there is an agent interacting with an environment and the outcomes of his actions depend both on the agent's choices and on the environment. Formally, an MDP can be defined by the following tuple:

$$\{S, A, P_{a \in A}(s \in S, s' \in S), r(s \in S)\}$$

Let us analyze the elements of the tuple:

- the set S represent the finite set of all possible states in which the agent can be;
- the set A is the set of all the actions that the agent can perform;
- $P_{a \in A}(s \in S, s' \in S)$ represent the probability that, starting from state s and performing action a, the agent will arrive in state s';
- $r(s \in S)$ represents the expected immediate reward that the agent gets when he reaches state s.

An important property of MDPs is that its state transitions satisfy the Markov property: this means that, given a state s and an action a, the future state s' depends only on s and a and not on previous states or actions. Basically, this means that a state is representative of the past history and this is coherent with the task that we are facing: given a certain configuration of the two robots, we can pick the best action no matter how we arrived in that situation.

Figure 3.5 should help the reader to understand the typical structure of an MDP. Let us imagine that the agent is currently in state s_0: he can choose between action a_1 (that will bring him to state s_1 with probability

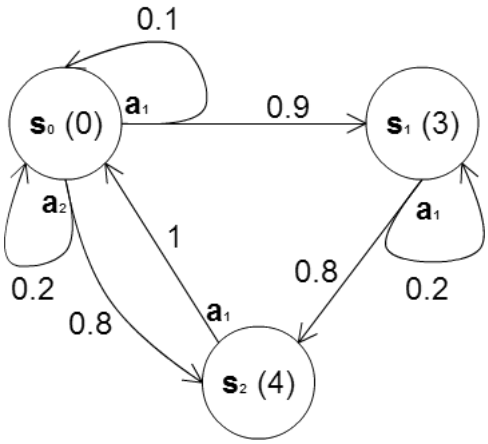

Figure 3.5: **An example of an MDP**. States are represented by circles, while transitions between states are represented by arcs. In every state, the reward associated to the state is reported in the parenthesis. On the arcs, apart from the name of the associated action, are reported the probabilities of each transition.

0.9 and leave him in the state s_0 with probability 0.1) and action a_2 (that will bring him to state s_2 with probability 0.8 and leave him in the state s_0 with probability 0.2). Suppose the agent chooses a_1 and arrives in state s_1: he will receive a reward of value 3 and he will only be able to choose action a_1, that will bring him to state s_2 with probability 0.8 and leave him in state s_1 with probability 0.2. Repeating this process over and over, we can understand how the agent explores the environment and in doing so he collects rewards from the environment itself. One thing that it is important to notice is that the process is not deterministic: when the agent is in state s_0 and performs action a_1 he does not know for sure if he will transition to state s_1 or if he will stay in state s_0. *Figure 3.6*[1] shows a conceptual view of the process. As it can be seen, at each time step the agent perceives the state s_t and receives a reward r_t from the environment, then he chooses an action a_t that will influence the environment, and so on. It is very important to keep *Figure 3.6* in mind because the concept of agent is very generic: in this section the agent will be the actor-critic system, but later on the whole model will play the part of the agent.

Now that we have found an answer to the first question, we should try to find an answer to the second one. In a first approximation, we could say

[1]Figure from *Sutton and Barto (1998)*

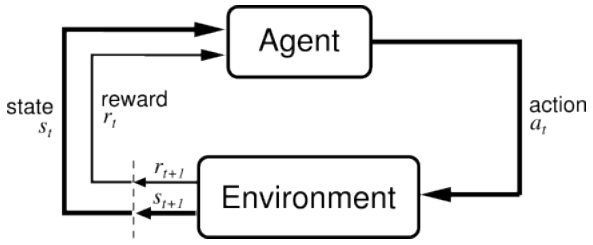

Figure 3.6: **The agent-environment interface**. It is important to notice that the concept of agent is very generic. Also, notice how the notation $r_t(s_t)$ has been dropped for a simpler r_t, since the associated state is implicit and should be clear to the reader.

that the goal of learning is that of maximizing the sum of expected rewards that the agent will get, defined by:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T.$$

However, it is easy to see that if the experience is not divided in episodes, we may have $T = \infty$, so that we would have an infinite sum. Then we introduce the concept of discounting, so that the formula becomes:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{3.1}$$

where $0 \leq \gamma \leq 1$ is called the discount rate. Basically, the discount says that rewards that are closer in the future are more important than the ones that are further. Apart from the mathematical purpose, adding γ will have another important effect: the prey will prefer to receive the same reward sooner than later, so that it will try to perform the task that we assigned it through the rewards as soon as possible.

Actually, given our setting and the purpose of our work saying that our goal is only that of collecting high rewards is over simplistic, but for the sake of simplicity for now we will just concentrate on this. Let us see how the actor-critic technique solves the problem.

3.2.2 The actor-critic technique

First of all, actor-critic's approach to solving the problem is that of discretizing perceptions into discrete states (showed as s_t in *Figure 3.6*), that basically represent "situations" in which the agent can be and needs to decide which action to perform. We will not enter now in the details of how

we defined and discretized the states of the system, but we will do that in Section 3.2.3; for now, let us just imagine that there is a set of numbered states in which the system can be and that the agent is able to recognize them.

The first constituting element is called critic, and basically holds a representation of how desirable it can be for the agent to be in every possible state. Formally, the critic contains a value function that is defined for every $s \in S$ as follows:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s\right\} \tag{3.2}$$

where it is easy to see that we are using the definition of future discounted reward given in equation 3.1. For now we will ignore the π symbol in the notation, but we will come on this in a short while.

How does the agent know which values to assign to the function $V(s)$? An interesting aspect of this technique (and in general of RL techniques) is that the agent learns through experience of interaction with the environment. Let us say that the agent is in state s_t, performs an action and arrives in state s_{t+1} getting reward r_{t+1}; then, it will compute the prediction error like:

$$\delta_t = r_{r+1} + \gamma V(s_{t+1}) - V(s_t) \tag{3.3}$$

Now the agent will be able to update the value of $V(s_t)$ with:

$$V(s_t) \leftarrow V(s_t) + \alpha * \delta_t \tag{3.4}$$

where α is a coefficient that regulates the speed of the learning process. In essence, we could then say that the critic contains a representation of how good is it to be in a certain state (equation 3.2), that is updated looking at how reality meets expectations (equation 3.4) and that expectations are generated bootstrapping from close states (equation 3.3).

However, up to now we ignored an element of equation 3.2, that is the symbol π: it basically means "following the policy indicated by π", but what does this mean? Basically, in order to decide what action it would be better to perform in each situation, we define a function $\pi(s_t, a_t)$ that associates to a pair of state s_t and action a_t a value that represents "how good" is to perform action a_t in state s_t; given a certain situation (i.e., a certain state s_t), the agent will typically want to pursue the action a_t that gives the maximum value of the function $\pi(s_t, a_t)$. It is intuitive that the rewards that an agent collects during time depend strongly on the action policy that

he follows (i.e., on the values of the function $\pi(s_t, a_t)$) and this is why we added the symbol π in equation 3.2.

Actually, there is still something that needs to be punctuated, since saying that our system always chooses the action that maximizes the function π would not be completely correct. A well known issue in RL is that of the exploration-exploitation trade-off: we would like to both exploit the best of the knowledge we extracted until now (in terms of state values knowledge and action policies) and to explore new alternatives, that we currently regard as sub-optimal, but that could lead to new unexpected ways to explore the environment and subsequently to better action policies. Basically, what we are saying is that we would like to follow most of the times the action policy that we currently deem as the best one, but we would like to try different actions from time to time to see if we can find something better. This is done through an ϵ-greedy action policy; given a value of ϵ where $0 < \epsilon < 1$ and tipically small, every time the actor needs to choose an action it will extract a random number between 0 and 1: if the number is above ϵ (i.e., most of the times) the actor will follow the action that maximizes the π function (i.e., the action currently deemed as the best one), while if the number is below ϵ the actor will choose a random action.

Again, we are facing the problem of how to set π values, but again we will let the system learn automatically. The actor module will get the δ_t signal from the critic and will update its internal function with:

$$\pi(s_t, a_t) \leftarrow \pi(s_t, a_t) + \beta * \delta_t \tag{3.5}$$

The meaning of equation 3.5 is: "if I performed a certain action in a certain situation and it led me to results better than the results I expected to get, then when I am in this situation I should try to perform this same action". This will bring to a competition between actions and the ones that will be performed in the end (i.e., the ones with the highest values associated) will be the ones that more often brought to particularly good outcomes. *Figure 3.7* will probably help the reader in understanding the structure of what we have described until now.

The nature of actions has already been introduced in Section 3.1.2: given the squared cell to which the prey currently belongs, an action will consist in setting the high level goal on the centre of the same cell or on the centre of one of the eight surrounding cells. It should also be clear to the reader that it will be an automatic low level control to decide in which hexagonal cells to move in order to reach the high level goal set by the RL algorithm. It could be useful for the understanding to point out that in case the state changes before the high level goal is reached the RL control will be asked to

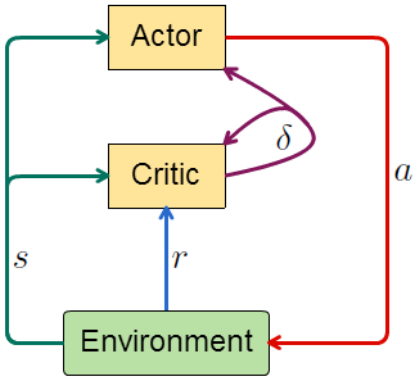

Figure 3.7: **The actor-critic module**. The notation is the following: s is the state perceived, r is the reward received from the environment, δ is the value computed by equation 3.3 and a is the action performed by the agent.

produce a new action based on the new state; high level goals should then be thought of as a sort of directional goal rather than a point to reach, even though if the state does not change the prey will be able to reach such point.

There are still two aspects that we need to define, namely how we code the states and how we define rewards, but before discussing them we would like to point out an aspect of this solution. As it is also discussed in *Barto et al. (1983)*, the action selection process is only biased towards the action that has the highest π value, but not completely defined by that action, so we can claim that actions are not mere responses directly and invariably connected to the input state, a form of action selection that would be appropriate for Pavlovian control and not for habitual control. Also, even more relevant is the fact that the values of both the $V(s)$ function and the $\pi(s, a)$ function can be learned and modified through an extensive learning process, so we believe that our solution is actually appropriate for an habitual control module.

3.2.3 The state coding

Our state coding is composed of three elements. The first one is the squared cell to which the prey currently belongs, encoded in a single number. Starting from *Figure 3.3*, imagine to number the cells by rows, from 0 to 48: the number corresponding to the cell in which the prey is will be the first component of the state. The number 7 is just an heuristic value and proved to work well with the dimension of the maps and the number of hexagonal tiles used in our work, so we did not investigate this parameter thoroughly.

The two other components of the state basically regard the interaction between prey and predator. One component is the angle of the predator with respect to the prey; remember that there is no concept of direction in these simple discrete world, meaning that we do not consider that given a position the robot might be facing different directions, so an angle of zero only means that in the map the predator is positioned at the same height but more shifted to the right with respect to the prey. This measure has been heuristically discretized in 8 different values, so that we will have a number from 0 to 7 indicating from which discrete direction the predator is approaching.

The other component is the distance between the predator and the prey, that again has been heuristically discretized in 3 values (we could think about them as "close" for a distance within 1/10 of the map size, "normal" for a distance between 1/10 and 1/4 of the map size and "far" for higher distances).

At this point, the three components are combined through a cross product of their domains, meaning that an example of a state in natural language could be: "the prey is in cell 15 and the predator is at a normal distance above the prey". It should be clear that basically the prey is learning how to behave in the map regardless of its structure, meaning that the prey will learn what to do e.g., in cell 15 and its choices may depend on the obstacles that surround cell 15, but it has no explicit representation of such obstacles; we could then say that the prey is learning the map. Someone could argue that this is a high level ability, but since we are trying to reproduce a human pain model we claim that humans possess such ability and so our model is biologically sound. Also, the reader should remember that we regard the prey-predator just as a framework to test our model: this means that we did not concentrate on the task itself, but mostly on the structure of the modules and on their interactions, trying to make sure that they were as biologically accurate as possible, so the "learning the map" task is just one of the possible tasks to test whether our model learns an MDP in a way similar to the one you would expect from a human.

3.2.4 The rewards

The only reward given to the prey was negative: a pain signal based on the position with respect to the predator. As already pointed out in Section 2.1.2, since the absence of pain can serve as a prize, the prey basically learns to avoid pain. How do we give pain to the prey? We chose a very simple approach, since again we were just interested in testing our model.

We assigned a negative reward to the prey based solely on the linear distance from the predator: when the predator was extremely close to the prey we assigned the maximum possible negative reward to the prey, then, if the distance from the predator increased, we linearly reduced the pain signal until we came to zero, that means the total absence of pain (so, basically, the best that our system can aim to). In order to simplify the problem and the definition of episodes, we decided to set the distance where the prey gets zero pain equal to the distance where the prey is considered too far from the predator and the episode ends; remember that, apart from hiding behind an obstacle, the other way the prey has to conclude an episode is to leave the predator behind.

Besides this aspect, it needs to be said that we used discrete rewards, so that there is a linear gradient of pain towards the predator but in discrete steps. This was meant in order to reduce the effects of slightly different distances between different hexagonal tiles, but we deem that the only really important matter to consider is the actual gradient, so that this discretization will not affect our results.

How is this pain signal generated is still an open issue. In our case, as we have just said, we simply assumed to have a certain amount of pain that was perceived by the prey as decreasing with the distance from the predator (i.e. a higher distance corresponds to a reduced amount of pain), but this should be investigated more thoroughly in the future. When implemented in a real robot, it could be interesting to pose more attention on the module that translates sensors readings in a pain signal for the robot, exploring the literature in search of a solution that more closely resembles what happens in a human body. Now that we have explored all the elements of the model-free module, it is interesting to understand why we actually chose the actor-critic technique.

3.2.5 Why actor-critic?

A first reason why we decided to use the actor-critic technique is linked to the ease of extension of such a technique for future work. The actor component basically resembles the characteristics of habitual control discussed in Section 2.1.2, but for this sole purpose we could have used some other technique like Q-learning (see *Watkins (1989)* or *Sutton and Barto (1998)*). However, apart from an action policy to reflect habitual control, actor-critic gives us also a representation of the perceived desirability to be in certain states, which can be found in the critic. A phenomenon like that of Pavlo-

vian conditioning[2], that we did not explore in our work, would be easier to integrate when a critic is present. In this case, a possible design would be that of setting some thresholds on state values (i.e., values of $V(s)$) and associating to values over such thresholds the actions that represent conditioned responses. Anyway, the exploration of this phenomenon and of a possible implementation is outside the scope of this thesis, but we wanted to point out how our choice makes an extension easier. To wrap it up, we could say that the actor is the one actually linked to habitual control, while the critic is Pavlovian, meaning that it puts the basics for an extension with Pavlovian conditioning.

Another important point that guided our choice is the signal δ, that can be seen in equation 3.3 and in *Figure 3.7*. That signal represents the difference between what the individual expects and how reality is and this can be a useful element for an action arbitrator between different modules. Basically, δ represents the surprise element for the agent, so this solution that allows us to generate it before an action is chosen is way better for modules integration: in this way we can generate δ and use it to decide which module should produce the final action, using such surprise element as a factor in the choice. This issue will be further clarified in the following sections, where it is explained how we integrated the modules.

3.3 Model-based module

The model-based module, that has to meet certain characteristics (see Section 2.1.2 on page 7). We wanted to have a design that reflected both the behavioral aspects and the internal processes of the analogous module in humans (of course, for what concerns the second issue our reference was what in the literature is believed to be the internal mental process). Since we did not find in the literature any model that reflects all these criteria, we designed a new model from scratch. Let us start by looking at how the model is actually represented in our system.

3.3.1 The model

What we internally keep as a model can be easily thought as a table, indexed by pairs of state and action. Every time we are in a certain state and we

[2]from wikipedia.org: "Pavlovian conditioning is a process of behavior modification in which an innate response to a potent biological stimulus becomes expressed in response to a previously neutral stimulus; this is achieved by repeated pairings of the neutral stimulus and the potent biological stimulus that elicits the desired response."

perform a certain action, we update the model registering the outcome of the action in terms of reward obtained and state reached at the following time step. The table is initially empty, meaning that at the beginning we do not know anything about the environment's structure, then it is filled in as soon as we proceed with the exploration. However, as it has been pointed out in Section 2.1, we need the model-based module to be reactive in the face of change: how do we reach such a goal? We set a maximum length L_{max} for every table row (i.e., a maximum number of pairs of future state and reward registered given a state and action pair), and in case the number is exceeded we only keep the most recent L_{max} pairs. This means that for every state and action pair the agent will only keep in memory the most recent L_{max} pairs of states and rewards that the action led to. The fact that we only keep the most recent pairs gives to the model the property of being reactive in the face of changes, so that if a change in the environment means a change in the model, as soon as the rows of the table affected by the change are updated the model (on which the model-based module is based) will be in line with the new environment.

(s;a)	**(s';r)**			
(1;UP)	(2;-3)	(4;-2)		
...	...	...	...	...
(2;UP)	(1;-3)	(3;-4)	(5;-3)	(1;-5)
(2;LEFT)	(3;-2)	(4;-1)	(5;-1)	(3;-2)
(2;DOWN)	(3;-3)	(5;-3)	(12;-3)	(1;-3)
(2;RIGHT)	(3;-2)	(1;-5)	(3;-5)	
(3;UP)	(5;-1)	(7;-1)		
(3;LEFT)	(1;-2)	(2;-2)	(7;-2)	
(3;DOWN)	(2;-5)			
(3;RIGHT)	(6;-5)	(8;-4)	(1;-1)	
...	...	...	...	...

Figure 3.8: **An example of the model**. The leftmost column holds a list of all the possible pairs of state s and action a. For every pair, the right part of the table keeps the most recent pairs of future state s' and reward r where the action a performed in state s led. If an action in a certain state has not been performed enough times, table rows may be empty or partially empty.

Figure 3.8 should foster the reader's comprehension. Even if it is not

exactly what we did in our work, for the sake of simplicity we decided to present the model with only four actions (UP, $LEFT$, $DOWN$, and $RIGHT$) and L_{max} equal to 4. We actually used L_{max} equal to 4 in our work, but the difference is that we used 9 actions that are exactly the same actions available to the model-free module and that we have explained in previous sections; we believe anyway that the extension from the 4 actions example presented here and the version with 9 actions that we used in our work is straightforward.

Basically, the model can be thought of as a representation of the underlying MDP built through samples. Let us take another look at *Figure 3.5* on page 25: the reader could say that, performing action a_1 from state s_0, the agent would end up 9/10 times in state s_1 and 1/10 times in s_0. In the same way, looking at *Figure 3.8* we could say that, performing action $RIGHT$ from state 2, the agent would end up 2/3 times in state 3 and 1/3 times in state 1. Of course the table contains just an approximate version of the underlying MDP since it is based on a limited number of samples for every transition, but it is important to understand the deep meaning of the model.

A final note that may be interesting for the reader is related to an important parameter of the model, that is the maximum number of elements per row. Even if we expected the model to need higher values of the parameter to give good performance, we found that the required performance (i.e., accurate action policy while keeping reactivity) was obtained for values around 4.

3.3.2 How to choose an action using the model

How do we choose an action given the model? The process is quite simple and basically is the following: first of all we evaluate every action that we can perform from the current state (remember that actions in our case are always 9, one for every cell), then we choose in an ϵ-greedy manner, in the same way that has been explained for the actor-critic technique in Section 3.2.2 on page 26. Notice that we are still using the letter ϵ for convenience and, even if in our work the ϵ parameters in the model-free and model-based modules had the same value, they could actually have two different values. Also notice that we still did not define how we evaluate an action: we will explain this in Section 3.3.3. The pseudocode of the algorithm is shown in box *Algorithm 1.*

Apart from line 3, that will be better explained in Section 3.3.3, the procedure here adopted should be clear to the reader (remember that $0 <$

Algorithm 1 Action chooser for model-based module

```
1: initialize actionEvaluations; //the size of this array equals the number of actions
2: for all possible actions a_i ∈ A that can be performed in state s_k do
3:     actionEvaluations[i] ← evaluateAction(s_k, a_i);
4: end for
5: extract random number r ∈ [0; 1];
6: if r > ε then
7:     choose action a_i with maximum evaluation in actionEvaluations;
8: else
9:     choose a random action a_i;
10: end if
```

$\epsilon < 1$ and that ϵ is typically a small number). So whenever the agent is in a certain state $s \in S$ and wants to use the model-based module to pick an action, it will use the procedure that we just illustrated to do it. For instance, referring to *Figure 3.8*, if the agent is in state 2 it will evaluate the actions (UP, $LEFT$, $DOWN$, and $RIGHT$), and choose one in an ϵ-greedy fashion. But how does it evaluate the value of every action?

3.3.3 How to evaluate an action using the model

Given a state s_k, an action a_i is evaluated averaging between a certain number N of σ-greedy explorations: every single σ-greedy exploration will give an estimate of the goodness of a_i when performed in state s_k, then we average between the N values estimated by the N σ-greedy exploration. We will soon enter into details, but for now it should be enough for the reader to know that a σ-greedy exploration has a certain random component, so we need the average to make sure that the final evaluation of action a_i in state s_k is not biased by some random circumstance. In box *algorithm 2* is shown the pseudocode of this part (for now ignore the $MAXD$ parameter, it will be explained later on).

If we ignored the function $getFutureEval$ and the quantity $futureEval$, we could basically say that the code computes the average reward that we could get performing action a_i in state s_k. As the careful reader could have already guessed, in order to be more precise we would also like to add to our estimate some component that takes into account some information about the future reward and we do that through the $getFutureEval$ function, that basically implements the concept of the σ-greedy exploration.

What is a σ-greedy exploration? Basically, it is an exploration through

Algorithm 2 The evaluateAction function

```
1: function EVALUATEACTION(s_k, a_i)
2:     average ← 0;
3:     for j that goes from 1 to N do
4:         randomElement ← random element on the table row indexed
           by s_k and a_i;
5:         randomState ← randomElement{state};
6:         randomReward ← randomElement{reward};
7:         futureEval ← getFutureEval(randomState, MAXD − 1);
8:         curEpisodeSum ← randomReward + futureEval;
9:         average ← average + curEpisodeSum;
10:    end for
11:    average ← average/N;
12:    return average;
13: end function
```

the table up to a certain depth, trying to figure out what the agent should expect from the future. The role of the σ parameter is somehow similar to the role of the ϵ parameter explained beforehand, but let us do one step at a time, using the box *algorithm 3* as a reference.

First of all remember that we are starting from a state s_k and we want to understand something more about its future value. Given all the actions a_j that can be performed starting from s_k, using the pair $(s_k; a_j)$ we can index one row of the model table, so we can start by computing the average reward saved in the table for the row $(s_k; a_j)$; this substantially represent a rough estimate of the immediate goodness of performing action a_j in state s_k. At this point, we choose the action a_{chosen} in a σ-greedy manner that is similar to what we have done in model-free for ϵ-greedy choices: we extract a random number, than if it is greater than σ we choose the action associated to the maximum average reward, while if it is smaller we choose a random action (remember that σ is typically small). Now we are in s_k and we have decided an action a_{chosen}, so that with the pair $(s_k; a_{chosen})$ we can access a row of the table that typically contains more than one pair: where should we move? The answer is that we choose a random couple from the row of reward *randomReward* and future state *randomState*. At this point, using *randomState* with the role that s_k had at the beginning of *algorithm 3*, we can repeat this process over and over, going deeper with the exploration.

How deep should we go? It is quite complicated to answer to this question, since the answer may depend on several factors such as the cells' di-

Algorithm 3 The getFutureEval function

```
1: function getFutureEval(s_k, curDepth)
2:     if curDepth = 0 then
3:         return 0;
4:     end if
5:     initialize averageRewards; //the size of this array equals the
       number of actions
6:     for j that goes from 1 to numberOfActions do
7:         if table row indexed by s_k and a_j is empty then
8:             evaluation ← 0;
9:         else
10:            evaluation ← average reward on the table row indexed by
               s_k and a_j;
11:        end if
12:        averageRewards[j] ← evaluation;
13:    end for
14:    extract random number r ∈ [0; 1];
15:    if r > σ then
16:        a_chosen ← action with maximum value in averageRewards;
17:    else
18:        a_chosen ← random action;
19:    end if
20:    if table row indexed by s_k and a_chosen is empty then
21:        return 0;
22:    else
23:        randomElement ← random element on the table row indexed
           by s_k and a_chosen;
24:        randomState ← randomElement{state};
25:        randomReward ← randomElement{reward};
26:        futureEval ← getFutureEval(randomState, curDepth − 1);
27:        curEval ← randomReward + futureEval;
28:        return γ ∗ curEval;
29:    end if
30: end function
```

mension and the map structure, but in our experiments we have found that values in the range $[2; 6]$ gave the best performance.

Another thing that the reader may be wondering is the meaning of the γ at line 28 in *algorithm 3* and why we are multiplying the rewards that we collect during the exploration of the table for γ. At this point in time is useful to take another look at equation 3.1 on page 26: we are basically trying to reproduce a structurally similar equation up to a certain depth. Again, for simplicity we are using the same letter γ that we used in equation 3.3 on page 27 because the parameter in our work assumed the same value used for the actor-critic technique, but of course we could use for γ a different value than the one used in equation 3.3. Notice that when we encounter an empty table row during the exploration or when we reach the maximum depth we return 0, that basically means truncating equation 3.1. This is the meaning of the parameter $MAXD$ in box *algorithm 2*: it is an indication of the maximum depth that we want to reach with our exploration.

Notice the difference between the criterion used to pick actions during the σ-greedy exploration and the rewards that we collect and use to compose the action evaluation. When choosing between an action and the other, we choose in a σ-greedy way comparing the average instantaneous reward that the actions gave us in the recent past; however, once we have decided an action, we pick a random element and it is that element that defines both the reward that we collect during this σ-greedy exploration and the future state that we will reach in this exploration.

We have introduced the parameter $numberOfActions$ that keeps the number of available actions (again, we remind the reader that in our case this number is always equal to 9, one action for each cell). Since this algorithm is not a standard one and we designed it, we want to make a numerical example to make sure that the way it works is clear to the reader.

3.3.4 An example of a σ-greedy exploration

The starting point for our example is *Figure 3.8* on page 33. let us imagine we are currently in state 1 and we have to decide an action using the model-based module; as *algorithm 2* says, for every action we will have to perform N σ-greedy explorations and average their values. let us do one single σ-greedy exploration for action UP step by step: it should be then easy for the reader to understand how all the rest of the algorithm works.

First of all, we need to pick a random element from the table row $(1; UP)$: let us say that we pick $(2; -3)$. This means that we will need to move to the next step to state 2 and that in the meantime we collect a reward of -3

(we will explain later on in this section how we compose collected rewards). Now, for state 2 we have four possible actions and we need to compute the average reward for every action. For the action UP performed in state 2 we will have:

$$averageRewards_{(2;UP)} = \frac{-3-4-3-5}{4} = \frac{-15}{4} = -3.75$$

In the same way, we would have -1.5 for action $LEFT$, -3 for action $DOWN$, and -4 for action $RIGHT$.

Now we pick a random number, let us say 0.6, and we decide that our σ value was 0.1: this means that we have to go for the greedy action, that is the action $LEFT$, that with -1.5 had the highest average reward. At this point we choose a random element from the row indexed by $(2; LEFT)$, let us say $(3; -2)$ (there are two pairs, but since they are identical one is worth the other); this means that we will need to move to state 3 and collect a reward of -2.

Now, again, we need to compute the average reward of every action performed in state 3. For the action UP we have:

$$averageRewards_{(3;UP)} = \frac{-1-1}{2} = \frac{-2}{2} = -1$$

while for the other actions we have: -2 for $LEFT$, -5 for $DOWN$, and -3.33 for $RIGHT$. The winning action would be UP, with an average reward of -1, but let us say we extract a random number of 0.05 and we pick $DOWN$ as random action. At this point there is only one element in the row, so we will get the pair $(2; -5)$, that means going to state 2 and collecting reward -5. let us imagine for simplicity that when we arrive to state 2 this time we have a *curDepth* of 0 and we need to stop.

Now we have done a σ-greedy exploration for the action UP performed in state 1 and we need to assign a "score" to it, that will be averaged with the scores of the other $(N-1)$ σ-greedy explorations starting from action UP in state 1, so that the average will be the final score that we assign to action UP performed in state 1. we have collected three rewards, namely -3, -2 and -5, but as you remember we need to discount them (it is done automatically in the line 28 of *algorithm 3*). If we suppose to have $\gamma = 0.99$, we will have:

$$curEpisodeSum = -3 + (-2) * 0.99 + (-5) * 0.99^2 = -3 - 1.98 - 4.9 = -9.88$$

This is the value that the current exploration gave to action UP performed in state 1: we do it N times, we average the results and we get the final

value for action UP in state 1. Remember that this time when we started from row $(1; UP)$ we picked the pair $(2; -3)$, but since each of the N runs starts with a random choice then each run could start with a different choice of this initial pair, and so on.

After we have done this for all the actions we will have an evaluation for any action, that takes the same conceptual meaning of the evaluation that the actor keeps in function $\pi(s; a)$, as shown in equation 3.5 on page 28. In the same way explained for the actor-critic technique in Section 3.2.2 on page 26, we will choose here a move in an ϵ-greedy manner and this will be the move chosen by the model-based module.

3.3.5 The tree structure of σ-greedy explorations

As discussed in Section 2.1.2, it is thought that model-based inferences use some sort of tree-based search, so we deemed important to underline how our model-based module reproduces this process. let us take again the example used in Section 3.3.4 and let us analyze it side by side with *Figure 3.9*. Of course in the figure it is shown only a portion of the tree, the one involved in the example, also because *Figure 3.8* contains only the information necessary for the example, but with a complete model-example one should be able to build the full tree and to understand how the model-based module explores it. It should also be clear that the fact that we are doing σ-greedy explorations and not simply greedy explorations gives to the process a degree of uncertainty, so that each exploration should go through the tree in a different way, at least probabilistically; this resembles the mind process of evaluating different alternatives.

3.3.6 States, rewards and updates in model-based module

In order to be as clear as possible, we would like to specify something about state coding and rewards in this module. The state coding used in the model-based module is the same used for model-free and explained in Section 3.2.3 on page 29, so that at a given time step the model-based and model-free modules always perceive the same state. Also, the same applies for the rewards, so that both modules always receive in input the same reward given a certain situation. This could be obvious after *Figure 3.6* on page 26, since the perceived state and the reward depend on the environment, but since for the Pavlovian module we will have to do something slightly different we thought that it was a good idea to clarify this.

Also, it may be better to anticipate something about model-based and model-free modules working together. The reader may be wondering what

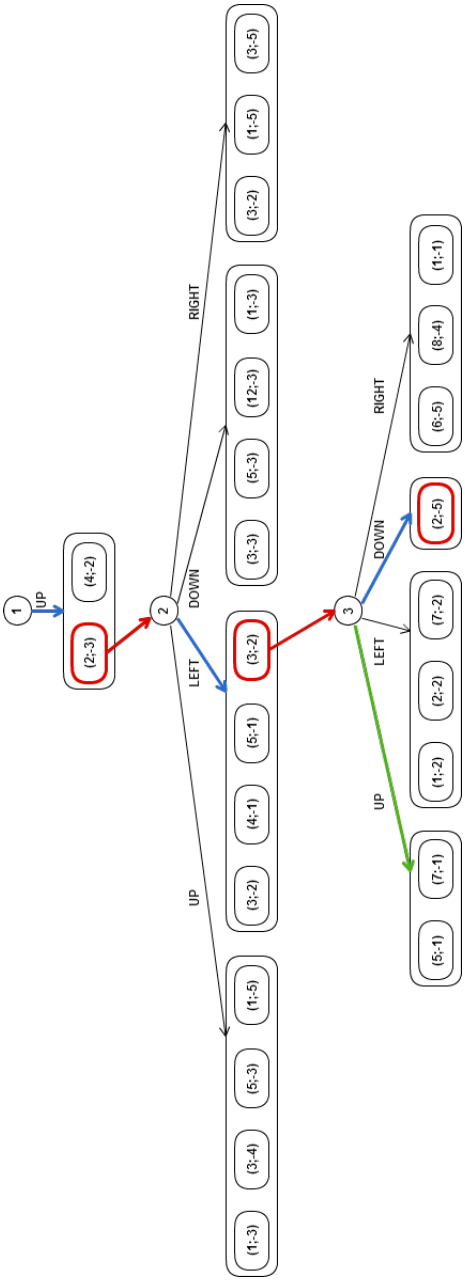

Figure 3.9: **The tree structure of model-based reasoning**. We highlighted in blue the chosen actions, in green the action that was the best action but was not picked because of the σ-greedy process and in red the choices within each table row with respect to the example of Section 3.3.4.

happens to one of the two modules when the executed action has been chosen using the other module and vice versa, in a situation in which we are using both modules. The answer is simply that, independently on the module that picks the action, both of the modules are updated. This means that every time a transition happens, we add the state and reward pair to the proper table row and we update the model-free module (using equations 3.4 on page 27 and 3.5 on page 28); afterwards, we will choose the action using the proper module, but every time there is a state transition both modules will be updated.

3.3.7 Why not other techniques?

Until now we have just said that we were not satisfied with previously designed techniques to use for the model-based module so we designed a new technique, but we actually never explained why other techniques were not suitable for our purposes. In this section we make some examples of techniques that we took in consideration and we discarded and we explain why.

Since one of the main requirements for the model-based module is that it should be more reactive in the face of change with respect to the model-free module, the very first technique one could come up with is called $Q(\lambda)$ (see *Watkins (1989)*). One could claim that, due to the eligibility traces adopted in that technique, changes in the underlying MDP are propagated faster in the data structures of the algorithm. Anyway, even if the sheer output of the technique could be reasonable, the biological faithfulness of it is not acceptable. One of the main reasons is that with this technique the computational load is distributed among all time steps because of traces updates and the final action generation comes at the price of a table lookup, while we would like the computational cost to be only in the action generation process. Also, there is no trace of a model and of a tree search in this algorithm, so we decided that it was too far from what the literature says.

Other algorithms of the so called Dyna architecture were introduced in *Sutton (1990)* and *Sutton (1991)*, namely Dyna-PI and Dyna-Q. For the former of the two, the reason why we discarded it is very simple: the author itself noted how it lacks reactivity in the face of changes in the environment (he calls the problems "blocking problem" and "shortcut problem") and this property was fundamental for our needs. For the latter of the two techniques, one issue is that the author assumes a deterministic model for the environment, but this is not suitable for complex tasks such as the ones to which we aim to apply our model, both in this thesis and in future work. Another issue with Dyna architectures is again the distributed workload: as

the author says, "execution is fully reactive in the sense that no planning intervenes between perception and action"[3], but we have already explained why this solution is not suitable for our purposes.

Another example of an algorithm that one could use for the model-based module has been introduced in *Lee et al. (2014)* (in the supplemental methods) and in *Glascher et al. (2010)*, but even if in their simple environment it works perfectly fine there are several issues related to the generalization to our more complicated setting. One first reason is that the function $T(s, a, s')$ does not seem to scale well because the dimensionality is $|states|^2$x$|actions|$, so one should try to understand if it is possible to use a sparse representation to reach good performance; also, since it represents probabilities it should be kept normalized and this is likely to add a non negligible computational overhead. Another issue is related to the equation that computes values for $Q_{MB}(s, a)$: in their case they give future rewards $r(s')$ as given because of the simple task, but we would need to save them from previous experience and it is not completely straightforward how many of them to save and how to use them (just saving one reward for every state looks over simplistic and not able to capture the nondeterministic nature of the underlying process). One last issue is related to the shape of the equations that update the function $T(s, a, s')$: since the equations are somehow similar to temporal difference equations, it may be that after extensive training their model partially lacks reactivity; for this last issue one would just need to implement the algorithm and perform a comparative study, but given the previously illustrated limits we decided that it was not worth it.

3.4 Pavlovian module

For the Pavlovian module, in the same way of the previous modules, there are some biological and psychological characteristic to follow (see Section 2.1.2 on page 7). Even in this case, we could not find a solid and generalizable model for Pavlovian behavior in the literature, so we designed one. Since the whole process is not completely straightforward, we will proceed one step at a time. First of all, we decided to use an artificial neural network (ANN) to get Pavlovian responses under the form of actions, so in the following section we give the necessary basics for ANNs.

[3] Quote from the abstract of *Sutton (1991)*.

3.4.1 Basics of Artificial Neural Networks (ANNs)

For our purposes, we can think about ANNs simply as mathematical instruments that, given a set of input values, gives a set of output values; we will use *Figure 3.10*[4] as a support for the explanation.

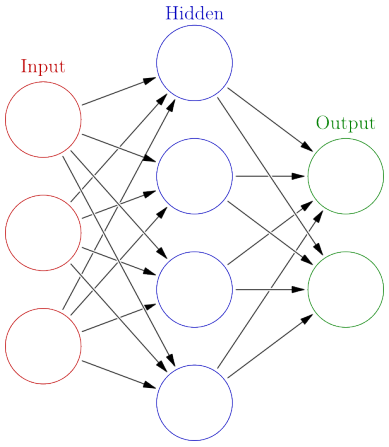

Figure 3.10: **An Artificial Neural Network (ANN)**.

In order to get an output from the ANN, the first thing that needs to be done is setting the input: imagine that to each one of the red nodes is associated a real value. After this, imagine that from every input node the input is sent through all the arcs going out from its node, towards the proper blue hidden nodes. To every arc we associate a weight and when the input goes from an input node to a hidden node through that arc it is multiplied by the weight associated to the arc; the weights are the crucial part of the process, but for now we will just assume that somebody has set them in the proper way. When all the inputs arrive to the hidden each hidden node (i.e., all the inputs coming from input nodes connected to the hidden node with arcs), the inputs are summed and pass through a sigmoid function. In mathematical terms, we would have that the output of the hidden node j is given by:

$$y_j = sigm(\sum_{i=1}^{I} x_i * w_{ij}), \tag{3.6}$$

where I is the total number of input neurons, J is the total number of hidden neurons, x_i is the input associated to input neuron i, w_{ij} is the weight of the arc that goes from input node i to hidden node j and $sigm$ is the sigmoid

[4]Figure from wikimedia.org

function defined by:

$$sigm(x) = \frac{1}{1 + e^{-x}}. \tag{3.7}$$

After this, the same process is repeated from the hidden nodes layer to the output nodes layer; written with a formula, we would have:

$$y_k = sigm(\sum_{j=1}^{J} y_j * w_{jk}), \tag{3.8}$$

where y_j is the output of hidden neurons given by equation 3.6, $sigm$ is the sigmoid function defined by equation 3.7, K is the number of output neurons and w_{jk} represents the weight of the arc that goes from hidden node j to output node k. If we put everything together, the relation that binds inputs x_i to every output y_k is:

$$y_k = sigm\left(\sum_{j=1}^{J} sigm\Big(\sum_{i=1}^{I} x_i * w_{ij}\Big) * w_{jk}\right)$$

At this point, it is better to clarify some issues, in order to make this explanation consistent with what can be found in the literature. The nodes in *Figure 3.10* are actually artificial simplified models of biological neurons (the justification of this is outside the scope of this thesis), so typically the term "artificial neuron" or "neuron" is used to call such nodes. Also, as has already been introduced in the previous paragraphs, typically the term "layer" is used to describe a homogeneous set of neurons with the same conceptual role: in *Figure 3.10* every layer has been highlighted with a different color. Finally, it needs to be said that the ANN represented in the figure is called "feed forward", referring to the fact that inputs move from left to right without any sort of loops and without ever returning to a previous layer. Thus, the ANN represented in the figure is actually an artificial feed forward network with one hidden layer. A last thing that can be useful to notice is that the sigmoid is a non-linear function, so that this ANN actually introduces a non-linear mapping between input values and output values.

As we have already said and as can be guessed from equations 3.6 and 3.8, the crucial part of the input-output mapping is the set of weights w_{ij} and w_{jk}, that defines how inputs are mapped to outputs, but how can we set these weights? The answer is postponed to Section 3.4.3 on page 50, because in order to answer the question we need to introduce the basics of genetic algorithms.

3.4.2 Basics of genetic algorithms

Now we need to talk about Genetic Algorithms; a lot could be said also about this technique, but we will keep the explanation to the very minimum required to understand what we did in our work (also in this case, the biological foundations of GAs are outside the scope of this work). let us proceed one step at a time in their understanding.

The first element of GAs is the individual: in our case an individual is simply represented by an array of floating point numbers (we will explain later on the meaning of this kind of individual); we can refer to the array also with the term chromosome, since somehow the array contains the characteristics of the individual. The next concept we need is the population, that is basically just a set of N individuals with N reasonably big (in our case, N=500). Of course, in a population all the individuals can and should be different from each other, or in other words if we take two arrays and compare them they will not probably contain the same numbers; this is an important aspect of the algorithm because it regulates the variety in the population, but it will be more clear later on.

At this point, starting from the first population, we want to generate a second population that is somehow related to the first one; we are now going to explain how it is related, while in order to understand the reason for this the reader needs to arrive until the end of this section. In order to generate the second population, we repeat several times the following operation: we select two individuals (for now we assume completely at random) from the first population (let us call them ind_A and ind_B), we somehow combine their content to get two new individuals (let us call them ind_C and ind_D) and finally we add the new individuals ind_C and ind_D to the second population. Obviously, if we repeat this operation several times we can obtain a second population that has the same number of individuals as the first one (for N=500, we need to repeat the operation 250 times). Of course, the second population is related to the first one, since its individuals are obtained through a recombination of individuals of the first population. Once we have obtained the second population, we can actually trash the first one; we can repeat this process over and over, obtaining every time a population the depends on the previous one. The reason why we should do this and how this happens will be clear later on.

Now let us analyze in more detail the operation of selection and recombination of two individuals. The very first thing that needs to be done is selecting two individuals from the first population and for now we will just assumed that they are picked randomly: let us call them ind_A and ind_B.

Now we want to combine somehow ind_A and ind_B to generate other two individuals ind_C and ind_D, in a way that is inspired to what happens in a normal biological reproduction.

The operation that we need to perform now is called crossover, and it happens with a probability $CROSSOVER\_PROB$ for each pair of selected individuals; in case it does not happen, for now we just copy the chromosome of ind_A into ind_C and the chromosome of ind_B into ind_D. In case we need to perform crossover, we select a random point P along the chromosome: we fill in ind_C with the first half (i.e., from the beginning to point P) of ind_A and with the second half (i.e., from point P to the end) of ind_B, then we do the opposite for ind_D (first the first half of ind_B, then the second half of ind_A). Basically, we are doing the same that happens in absence of crossover until point P, but then we switch individuals and from point P on we start copying ind_B into ind_C and ind_A into ind_D. *Figure 3.11* should be useful for the comprehension of crossover.

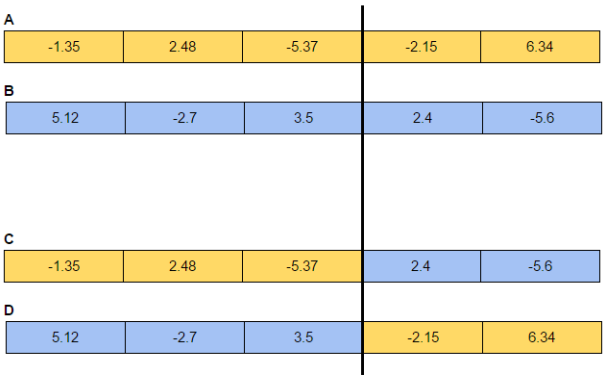

Figure 3.11: **Crossover in GAs**. The dark line represents the point P chosen for crossover.

At this point we need to perform another operation called mutation; it is applied to both individuals ind_C and ind_D in the same way, so we will just explain it as applied to a general chromosome, that is in our case an array of real numbers. In order to mutate a chromosome, we go through all its loci (where with the word locus we indicate in this case a single cell of the array); for each locus, with probability $MUTATION\_PROB$ we mutate the single locus, meaning that we add a small real offset to the value of the locus. It is important to notice that we are talking about an offset that can be also negative, so that if a locus is mutated its value could increase but also decrease. *Figure 3.12* shows an example of mutation operator applied

to an individual.

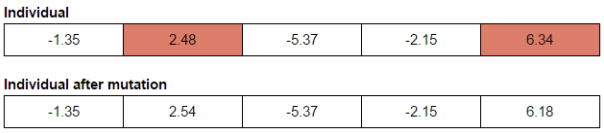

Figure 3.12: **Mutation in GAs**. The red loci indicate the ones that have been selected for mutation.

Now ind_C and ind_D are ready to be put into the second population: we have selected two individuals from the first population, we have combined them through crossover, we have mutated both of them and we have inserted the results in the second population, calling the new individuals ind_C and ind_D. Repeating this operation over and over, we can generate the second population; we can see in *Figure 3.13* a flowchart that summarizes the whole process. But why should we do this?

In order to understand the meaning of this, we are still missing a crucial element of GAs: the fitness function. To every individual we associate a fitness value, that basically states "how good" that individual is (the explanation of how this value is determined is postponed to Section 3.4.3). Once we have defined this, we would like to propagate through the various generations characteristics coming from individuals with higher fitness: a way to obtain this is to bias the selection process, picking with a higher probability individuals with higher fit value. This intuitively leads the whole process of generations to populations constituted by individuals with higher fitness values, in the same way that the survival of the fittest led biological creature to more evolved forms.

In order to obtain this result we used a process called roulette wheel selection, that selects each individual i with probability:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j},$$

where f_i is the fitness value of individual i and the j index iterates through all individuals. The explanation of how this is algorithmically obtained is outside the scope of this work.

There is one very last aspect of the algorithm we adopted that may be useful to point out, that is the elitism technique. Trying to push even more towards the preservation of the fittest individuals, at the very beginning of every new generation process (i.e., the process that from one population generates a new population through selection, crossover and mutation of the

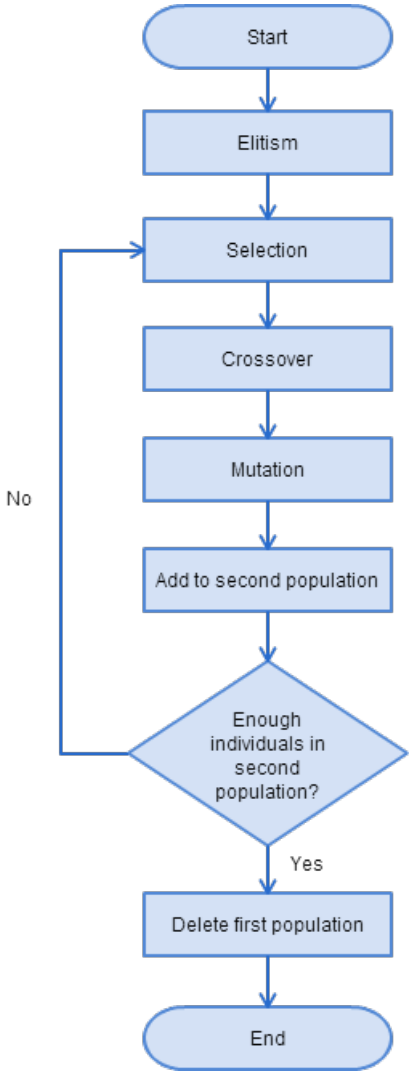

Figure 3.13: **The flowchart of one GA generation.**

individuals) we copy a certain small number of the fittest individuals directly in the second population. This means that we take the M individuals (with M small, for instance for us $M = 4$) from the first population that have the highest fitness values and copy them as they are in the second population. It is true that the fittest individuals have a higher probability of being selected by the aforementioned process, but they are likely to be modified by crossover and mutation before arriving to the new population; this process tries to preserve as they are the most promising individuals.

Up to now, we substantially defined a process that allows to start with a set of arrays of real numbers and, through a series of generations based on some fitness values we still did not explore in depth, arrives to a final set of arrays of real numbers that is likely to have higher fitness values. But how are these fitness values determined? And why do we need this process?

3.4.3 How to encode an ANN?

One issue that was left open in Section 3.4.1 on page 44 is how to set the weights for the ANN in order to get a good input-output performance and now we are ready to explain that. The set of weights of an ANN is actually a set of real numbers, so that if we code it into an array we could say that, once the mapping from the ANN to the array has been fixed, there is a biunivocal association between an individual (i.e., an array of real numbers) and an ANN. Since in an instance of our model we fix the architecture and the number of neurons for each layer of the ANN, there are no dimensional problems since all the individuals will have the same length, defined by:

$$array\_length = n_i * n_h + n_h * n_o,$$

where n_i is the number of neurons in the input layer (red in *Figure 3.10* on page 44), n_h the number of neurons in the hidden layer (blue in the figure) and n_o the number of neurons in the output layer (green in the figure).

A side note should be made regarding the coding of an ANN in an array, even though it may not be straightforward to understand for a reader not experienced with ANNs. If we have a promising individual, we would like to preserve as much as possible the feature that its ANN recognizes, at most splitting one through crossover (plus slight modifications of all features due to mutation, of course). For this reason, we basically code the ANN so that we encode one feature at a time, followed by its output activation pattern. This means that in the array we first place all the weights of the arcs going into the first hidden neuron, then all the weights of the arcs going out from that same neurons, then arcs going in the second hidden neuron, then

arcs going outside that same second hidden neuron and so on and so forth. Anyway, we understand that this note could be not so straightforward to understand and is probably not crucial for the success of the algorithm, so we will not insist too much on this.

Now we know about ANNs, we know how to code the network's weights in an array and so how to evolve a good ANN that has a high fitness value. As we may expect from what we have said until now, the whole purpose of this process is to generate a "good" set of weights that provides realistic and useful reflexes. But how do we define what is good in this context? And what is the input and the output of the ANN?

3.4.4 The input and output of the ANN and the locality of the Pavlovian module

In Section 3.1.2 on page 19 and in Section 3.2.3 on page 29 we introduced the idea of a grid laid on the map that generates a certain number of squared cells (in our case 49) in which the map is divided (again, the reader should pay attention to the fact that we are not talking about the hexagonal tessellation of the map). In the case of model-based and model-free modules, as has been said in Sections 3.2.3 on page 29 and 3.3.6 on page 40, the prey considers the absolute position in the map to compose the state, since it composes the state with also a number in $[0; 48]$ that indicates in which discrete squared cell the prey is. In the case of the Pavlovian module it seems more compliant with what is required by the underlying biological phenomenon to have a more local view of the map, without any sort of global view or higher level planning functionality. For this reason, we decided to use something similar to a local view of the map, but with a slight modification.

Let us imagine that at a certain point in time we need to use the Pavlovian module to generate an action. First of all, we determine the squared cell in which the prey currently is: let us call it C. For the Pavlovian, we will only consider this cell and the other 8 cells around C (it is approximately like drawing a square around the current prey's position and dividing it in 3x3 squared cells). The 9 cells at this point in time will be of three kinds: $EMPTY$, $OCCUPIED$, and $PREDATOR$. For $EMPTY$ and $OCCUPIED$ we apply what has been said in Section 3.1.2 on page 19; additionally, we say that the squared cell that contains the predator is in state $PREDATOR$. Finally, if the prey is close to the borders of the arena, in order to get the 9 cells we assume that the squared grid extends even outside the arena and that all the squares that cover areas outside the arena are in state $OCCUPIED$. An example should foster the comprehension: if

in the map represented in *Figure 3.2* on page 20 and in *Figure 3.4* on page 22 the prey is in the upper and leftmost corner of the arena and the predator is at its right (precisely in the squared cell just at the right of the prey's squared cell), the resulting minimap would be the one shown in *Figure 3.14*.

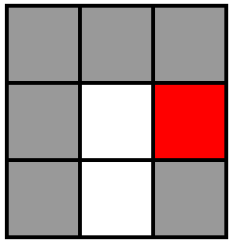

Figure 3.14: **Example of Pavlovian map**. White cells represent the $EMPTY$ state, grey cells the $OCCUPIED$ state and the red cell is in $PREDATOR$ state. The prey is of course in the central cell.

Now we would like to use the minimap represented in the figure as input for the ANN and get the chosen action as output. One simple coding is the following one: for each one of the 9 cells we define 3 input neurons (so that we will have $3*9 = 27$ input neurons), where each one of the 3 input neurons associated to a cell represents one of the states of the cell, then we set to 1 the input corresponding to the state in which the cell actually is and to 0 the other two. If the states are coded in input neurons in alphabetical order, for *Figure 3.14* we would have: $0-1-0-0-1-0-0-1-0-0-1-0$ for the first four cells (read by lines) to indicate that they are occupied, then $1-0-0$ for the empty cell, $0-0-1$ for the predator and $0-1-0-1-0-0-0-1-0$ for the last line. So, the final 27 inputs that translate the figure would be: $0-1-0-0-1-0-0-1-0-0-1-0-1-0-0-0-0-1-0-1-0-1-0-0-0-1-0$. Additionally, there will be the bias input neuron always set to 1, so that the actual input neurons will be 28, but this is not very relevant for the understanding of our technique.

For the output, as it has been explained in Section 3.1.2 on page 19 we need 9 actions: one will mean "try to get to the center of your current cell", while the other actions will push the prey towards the center of one of the surrounding cells. In order to obtain this from the ANN we define 9 output neurons, each one associated to one action, and we apply a winner-take-all approach. Given the inputs (all 0s and 1s) and the real weights, each one of the output neurons will assume a real value: at this point, the one with

the biggest value indicates which action will be executed, namely the action associated to that particular output neuron.

As it can be seen, the Pavlovian module takes as input very simple information: only a local part of the map and discretized and simplified in squared cells. An interesting property associated to this abstraction is that, while the other two modules somehow learn the map, the Pavlovian module has generalization properties across maps that share some sort of patterns in the structure. However, a drawback of this (that still makes sense biologically) is that this module has sub-optimal performance with respect to the other two modules: the reason is that given a certain pattern of inputs like the one shown in *Figure 3.14* we can associate to such pattern only one action, even if it could appear in very different parts of the map that would require different actions to get optimal performance. Also, as it is expected from a Pavlovian system, the local view gives to the module the property of being short-sighted, so that a purely Pavlovian control will easily get stuck in corners and impasses.

At this point we know ANNs, we know how encode them in arrays and how to evolve them through GAs and we know how to use ANNs to generate actions for the prey. But how can we evaluate a set of reflexes so that the GA will actually evolve a meaningful and useful set of reflexes to use in our individuals?

3.4.5 How to evolve a good ANN?

In order to get a very good set of reflexes we do the following: we generate a population of ANNs, where each one represents a set of reflexes (of course, they are encoded in an array), then we perform a very big (for us 10.000) number of population generations to improve the overall quality of the population in terms of fitness values, then finally we take the fittest element of the final population and that individual will constitute the agent's set of reflexes. However, we still have not defined the procedure that we use to evaluate a certain set of reflexes.

Currently, to evaluate a set of reflexes R coded in an array we do the following. First of all, we take a single map, let us call it M, and we manually define some fixed initialization points both for the prey and the predator. This makes us lose flexibility, since we need to manually define the points, but allows us to consistently reduce inequalities in the procedure that follows, since all individuals will be evaluated starting from the same initialization points. It would be of course an interesting future work to investigate whether this manual procedure is necessary or if it could be

automatized with random initializations, maybe raising consistently their number to balance random effects (even if this would cause a not negligible overhead in terms of computational expense).

After this, we take R and we create an ANN with the weights contained in R: let us call it N. Now we initialize prey and predator in all the initialization cases that we have manually set and we run a not too short episode (we used 50 state changes) to simulate what happens starting from that point and using only the Pavlovian module to generate actions. As we have said the Pavlovian control is not optimal, so we do not expect to get good performance especially during the first generations, so we needed to stop episodes. Even if the prey and predator lose contact, because the prey leaves behind the predator or is hidden by an obstacle, we stop the episode. When we are done, we sum all the rewards collected in all the episodes and we divide by the total number of rewards collected, obtaining in this way the evaluation of individual R (i.e., the individual represented by the set of reflexes R).

Basically the evaluation of an individual, i.e., its fitness value, is the average reward that the individual gets for these episodes starting from all of the initialization points in turn. A more interesting approach would be that of repeating the process for a certain number of maps, each one with its manually set initialization cases, and averaging the results over the various maps, but at this point in time we still did not try this road. Also, it may be that the same patterns in different maps would require different actions, so it is not even guaranteed that this alternative method would give better results.

Before closing this explanation, it is better to point out a tricky aspect of this training. At the beginning, we used much shorter episodes (10 state changes), but we always got a solution we wanted to avoid. Basically, with short episodes the solution that we always obtained from the GA consisted in a set of reflexes that always pointed in a direction: for instance, a full run of GA generations could provide a solution that always pointed towards the upper leftmost corner of the arena and remained stuck there. We then realized that this was probably due to the procedure we used to evaluate a set of reflexes: an ANN that always points in one direction will result after some time in an escape from most of the initialization points, so that a lot of good performances will hide the episodes in which the prey gets stuck because of the over simplistic action policy. Using long episodes solved the problem, because this evaluation technique penalizes strongly action policies that generate lots of null actions or block the prey in some corner (because while the prey is stuck it will probably receive lots of strongly negative

rewards).

3.4.6 A note on the number of hidden layers

Even if we did not investigate thoroughly the optimal number of hidden neurons in the ANN, we found out experimentally that a number of around 30 hidden neurons was sufficient to capture the complexity of the problem and to generate good performance, thing that we assessed heuristically observing the behavior of the prey when controlled only by the Pavlovian module.

Even if we do not report detailed results in the present thesis, we also investigated the effect of removing the hidden layer and the effect of having two hidden layers one after the other; in both cases it should be intuitive to understand how to modify the presented ANN to obtain these new two configurations, but in case it is not we deem that there are a lot of works in the literature that present the concept (we will not present these modifications here not to burden the discussion with something we did not use in the end). Removing the hidden layer did not allow us to obtain a satisfactory performance, because we always obtained action policies with lots of null actions and typically strongly biased towards one direction as happened with the training technique with short episodes; we believe that the hidden layer is necessary to represent the complexity of the problem. Conversely, adding a second hidden layer did not bring to an improvement in performance, probably because the performance with one hidden layer is already quite good; because of this, we decided to keep the model as simple as possible and we removed the second hidden layer.

3.5 Integrating model-based and model-free modules

A this point we have a complete design for the three modules, but we would like them to work together as they actually to in a human brain. For the integration of the modules, we decided to choose which one of the modules had to generate the action and then that module alone would pick an action to perform. An alternative would be that of always asking to all the three modules to evaluate all the actions for each time step, then somehow blend their evaluations to get a sort of overall evaluation for each action, then finally apply again a winner-take-all procedure. One relevant drawback of this second technique though, that is an important reason why we discarded it, is that the model-based module should generate an evaluation for all

actions at each time step even in advanced parts of the learning and this is known to be too expensive from a computational point of view.

We begin with the integration of the model-based and model-free modules. There are some works in the literature that claim that the relative uncertainty of model-based and model-free modules might be involved in their arbitration (see for instance *Glascher et al. (2010)* and *Lee et al. (2014)*). *Glascher et al. (2010)* actually computes the uncertainty of the model-based module (that they call SPE), but they do not use it for the arbitration, that is left to over simplistic and non realistic techniques (i.e. an automatic exponential decay, that would need an external reset signal if the environment changed). In *Lee et al. (2014)* there is the only really promising computational technique for arbitration based on relative uncertainty: for now we did not consider it for a matter of simplicity, but it would be of course really interesting to study it in the future to try to solve the integrations system's problems that we just outlined. However, even for this technique there are several problems to solve. First of all, they still adopt the approach that evaluates actions with all modules and then weights them based on their uncertainties, but we have already explained that this is not reasonable from a computational point of view. Then, their Bayesian approach to compute distributions of uncertainty might generate sharply peaked distributions after an extensive training and this might cause the system to lose reactivity after long periods of training. Finally, they do not consider the Pavlovian module, but in our case we should find a way to take also this module in consideration for the arbitration.

As it has been explained in Section 2.1.2 on page 7, we would like to use the model-based module whenever there is some novelty for the prey, so for the integration we decided to use the δ signal, that is defined by equation 3.3 on page 27 and shown in *Figure 3.7* on page 29. As we said before, such signal represents somehow the surprise of the prey with respect to expectations, so it looks reasonable to use it to arbitrate between model-free and model-based modules. let us proceed one step at a time.

First of all, we separate positive and negative δ values and we pass them through two exponentially decaying moving average filters, defined by:

$$\begin{aligned} if\ \ \delta \geq 0: &\quad \begin{cases} \bar{\delta}_{pos} \leftarrow (1-\rho) * \bar{\delta}_{pos} + \rho * \delta \\ \bar{\delta}_{neg} \leftarrow (1-\rho) * \bar{\delta}_{neg} \end{cases} \\ if\ \ \delta < 0: &\quad \begin{cases} \bar{\delta}_{pos} \leftarrow (1-\rho) * \bar{\delta}_{pos} \\ \bar{\delta}_{neg} \leftarrow (1-\rho) * \bar{\delta}_{neg} + \rho * \delta \end{cases} \end{aligned} \tag{3.9}$$

The equation might look rather confusing, but it is actually quite simple. ρ is is a parameter (typically small) that regulates how fast the filter changes

its value, $\bar{\delta}_{pos}$ filters the positive deltas and $\bar{\delta}_{neg}$ filters the negative ones. Whenever a new δ comes, it is filtered in the proper filter (either $\bar{\delta}_{pos}$ or $\bar{\delta}_{neg}$), while the other filter just decays exponentially. This is coherent with what happens biologically: if something happens that is not coherent with our expectations, we will slowly forget about it.

Basically, $\bar{\delta}_{pos}$ represents the amount of positive expectation in a certain moment (i.e., how much reality was better than we expected), while $\bar{\delta}_{neg}$ represent negative expectations (i.e., how much reality was worse than expected); now we just need to pass both of them through a threshold. In the case of negative deltas, if $\bar{\delta}_{neg}$ is less than a certain amount we will use the model-based module to generate the action, because the prey is very surprised, while if it is closer to zero we will used the model-free module. The same applies for $\bar{\delta}_{pos}$, but of course in the other way, so that we use model-based if $\bar{\delta}_{pos}$ is above another threshold. Actually, it is enough that one of the two filters surpasses the threshold to use model-based, so that if we want to formalize this we will have:

if $(\bar{\delta}_{neg} < model\_thresh_{neg}$ **OR** $\bar{\delta}_{pos} > model\_thresh_{pos})$
  $use\ model\_based$
else
  $use\ model\_free$

All of this should look fairly simple and logic based on what we have done until now and based on *Figure 3.15*, that shows a typical pattern of deltas that the prey receives during a training, in this case on the map shown in *Figure 3.2* on page 20. *Figure 3.16* enriches *Figure 3.7* on page 29 with the added module. But the reader could have one doubt: why should we use a filter before applying the threshold? Why cannot we apply the threshold directly on the values of δ? The answer lies in the fact that we are actually arbitrating between modules that generate actions independently one from each other. Since δ values change rapidly, if we did not pass them through a filter we would continuously jump from one module to the other to generate actions and we would obtain an inconsistent set of actions along time.

Let us make a very simple example to explain this. Let us imagine that in a certain situation both "escape to the right" and "escape to the left" are reasonable actions; let us also imagine that, for whatever reason, the model-based module would like to apply one of the two and the model-free module would like to perform the other one. Now, if the control quickly alternated between the two modules we would get contradictory actions that would result in the prey staying still or anyway moving in a strongly suboptimal way. If instead we filter deltas, it is likely that one of the two

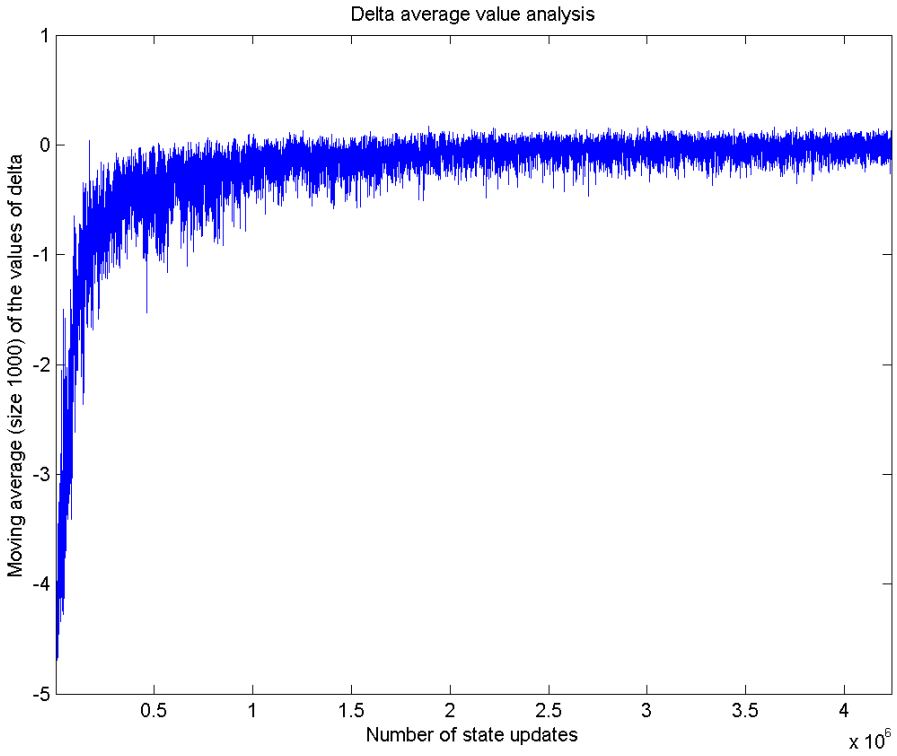

Figure 3.15: **Example of delta values during a training**. Notice that, in order to be analyzable, the values have been filtered with a moving average filter of size 1000. The delta values have been registered during a training of the model-free module alone on the map shown in *Figure 3.2*.

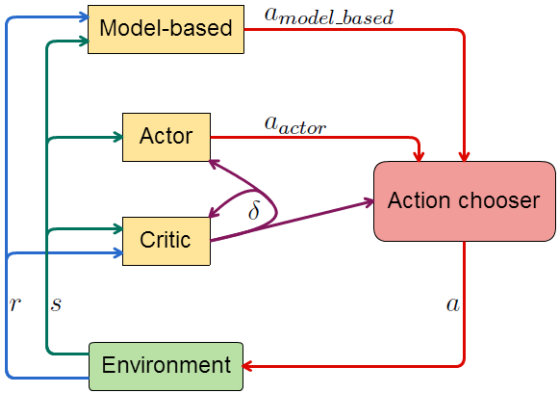

Figure 3.16: **Two modules integration**. The figure shows how to add to *Figure 3.7* the model-based module. The "*Action chooser*" block behaves as described in Section 3.5.

modules (which one depends on the situation) will take the control, allowing the prey to effectively escape in one of the two directions.

3.6 Adding the Pavlovian module: three modules integration

In order to integrate the Pavlovian module with the other two modules we apply similar principles to those explained in the previous section. In particular, we still consider $\bar{\delta}_{neg}$ (so we are still filtering δ values for the same reasons explained in the previous section), but this time we filter through a threshold that is negative and bigger in module than the one used for model-based. Up to now, then, we are basically applying ideas very similar to those aforementioned.

However, for the Pavlovian module we also had to apply another type of filtering, in fact it makes sense to apply reflexes only if there is a consistent level of pain (i.e., the pain signal is strongly negative). We noticed that sometimes deltas were big in module because of long chains of updates, but they were not always associated to highly negative reward signals: this means that, even if in that precise moment the experienced pain was limited, we were activating reflexes because of the function values of the surrounding environment, that were reflected in δ values. For this reason, we also applied a filter based on pain, in conjunction with the filter explained beforehand. Formalizing this and putting it together with what we have said in the previous section we get:

if $(\bar{\delta}_{neg} < pavl\_thresh_{neg}$ **AND** $last\_reward < reward\_thresh)$
  $use\ Pavlovian$
else if $(\bar{\delta}_{neg} < model\_thresh_{neg}$ **OR** $\bar{\delta}_{pos} > model\_thresh_{pos})$
  $use\ model\_based$
else
  $use\ model\_free$

where of course $pavl\_thresh_{neg} < model\_thresh_{neg}$.

The reader should notice that, while model-based is activated for both a positive and a negative surprise, Pavlovian is only activated for a negative surprise: in fact, the reflexive behavior as it is meant in our work is intended only as a protective layer from strongly negative rewards. Also, one could refer to one's everyday life to understand this: we all have the instinct of retracting our hand from a boiling pan, but if we touch a pan that we expected to be hot and it is actually cold we do not have the reflex of touching it. *Figure 3.17* enriches *Figure 3.16* with this last modification.

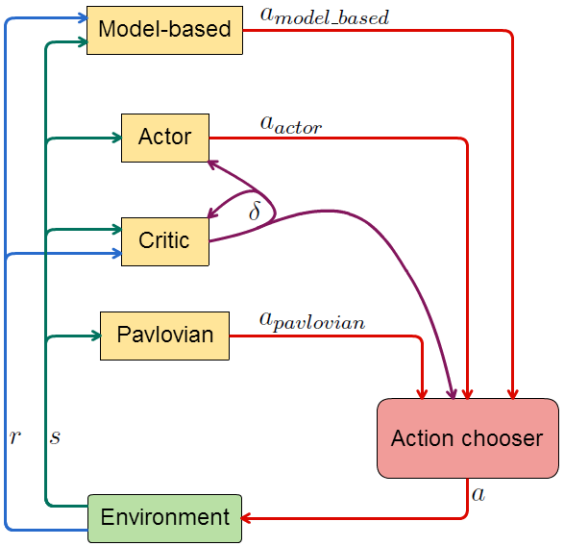

Figure 3.17: **Three modules integration**. The figure shows how to add to *Figure 3.16* the Pavlovian module. The "*Action chooser*" block behaves as described in Section 3.6.

A final thing that might be interesting to notice is that if the prey is exploring an unknown environment the Pavlovian module is somehow the one with the highest priority because it has the biggest threshold in module, followed by the model-based module and finally by the model-free module.

3.7 A note on the processes of learning

At this point, the reader could have some doubts about the relation between the evolutionary process realized through GAs and explained in Section 3.4.5 on page 53 and the training of the other two modules (model-based and model-free). The approach that we adopted is the following: first of all, we evolve a good set of weights in the way that has been explained beforehand especially in Section 3.4.5 (using of course only the Pavlovian module for the control during the fitness evaluations), then we fix the agent's weights to the weights of the best individual resulting from the GA and we start the learning for the other modules.

The process can be biologically thought of in this way: we assume that an individual is born with hard-coded reflexes that evolution gave him to protect him and then modifies his behavior through higher forms of learning, here represented by model-based and model-free modules. It needs to be said

that, in order to have complete adherence with the biological process, the fitness evaluation in the GA should be performed on a way longer period of time using all the three modules for the control and not only on few initialization cases and only using Pavlovian control. Anyway, given the typical population size and number of generations required for a complete training process this is currently computationally prohibitive; it would be interesting, though, during the future developments of the current research to investigate some way to adopt this different training process, maybe at least through a hybrid approach.

Chapter 4

Results

In this chapter we present the results obtained applying our model to the framework that we described in Section 3.1. We will try to keep results and parameter studies to a minimum, not trying to show all the results we obtained, but rather focusing on the ones that can give more interesting insights into our model. We would also like to point out something about the results, especially the convergence graphs: since when we needed to compare two learning curves we plotted them on the same figure and since in many cases it was necessary to point out certain details of the graphs, we often decided to allow Matlab to automatically resize the axes; we deem that rather than affecting the understandability of the material this will enhance it because of the aforementioned reasons, but in case for some reason the reader wants to compare results reported in different figures some extra attention should be paid.

In Section 4.1 we talk about the maps used for tests, in Section 4.2 we empirically prove the convergence of the techniques we adopted for goal-directed and habitual control (even though model-free module uses a well known technique, we report it for completeness), in Section 4.3 we show the typical results of a GA training to produce a set of reflexes, in Section 4.4 we show how all combinations of modules can actually reach convergence, in Section 4.5 we analyze in more detail the learning process trying to find out how many properties that are expected from the biological background are actually shown by our model, then in Section 4.6 we propose a new experimental procedure useful for some final analyses on data and we show some more experimental results, in Section 4.7 we show results related to the OCD pathology, then finally in Section 4.8 we explain some current limitations to our work and at the same time we give some ideas that may solve the issues.

4.1 The maps used for tests

One map that we used for our tests is the one shown in *Figure 3.2* on page 20 and the associated squared grid discretization is shown in *Figure 3.4* on page 22. Another map used for tests is shown in *Figure 4.1* and its

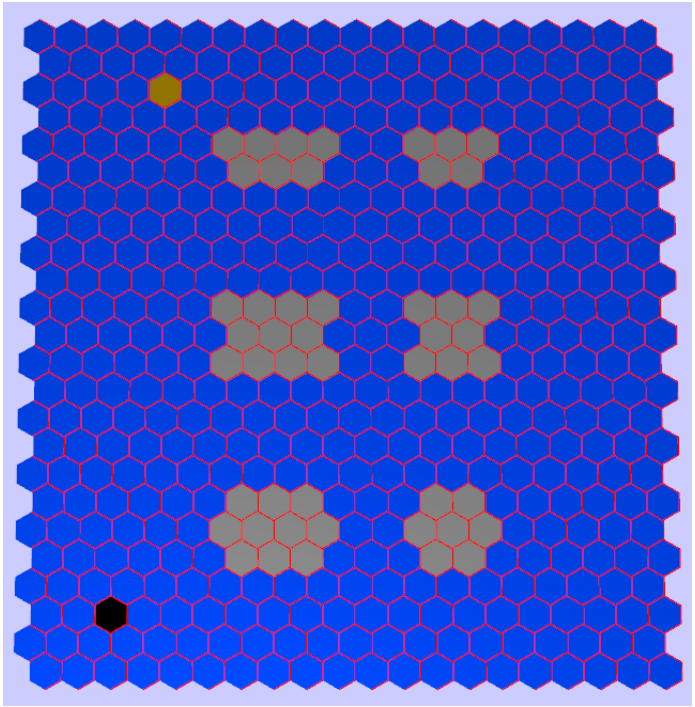

Figure 4.1: **A second map used for tests**.

associated squared grid discretization is shown in *Figure 4.2*. Where it is not differently specified, the results refer to the first of these two maps; also, in all the results reported in this chapter we used a set of reflexes trained in the first map.

It is better to spend some extra words here about the two maps. They look quite similar, because a certain number of patterns repeat, but one should not get confused by this. From the eyes of the Pavlovian module, that somehow sees the world as is shown in *Figures 3.4* and 4.2, it is true that some patterns are common, but not all of them (e.g., look at the sides and the four corners). For the instrumental learning this should be even more evident: since model-free and model-based modules basically "learn the map" and do not learn relative patterns, the two maps look very different from these modules' perspective. They will need to learn that grey squared cells in *Figure 3.4* can be crossed if the reference map is the one of *Figure 4.2*

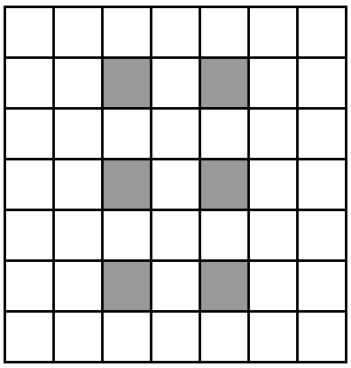

Figure 4.2: **The squared grid discretization based on *Figure 4.1***. The white squares represent cells considered free, while the grey squares represent cells considered occupied by obstacles.

and vice versa for cells that were considered empty in the first map and are occupied in the second one.

In this sense, this is one of the aspects of our work where the fact that our focus was on the biological similitude of the model and not on the prey-predator problem made a big difference in our design choices. If we had tried to solve the prey-predator problem probably we would have chosen some solution that could give also to the instrumental control better generalization properties, but since our focus was on the model this difference was actually useful to us: we just needed to change the number and position of the obstacles in order to generate a significant change in the underlying MDP, thus allowing us to study with a minimum redesign effort how our model reacted to different situations.

4.2 Convergence of model-based and model-free modules alone

First of all, we show the convergence of model-free (see *Figure 4.3*) alone through a sample run. It may be useful to spend some words describing how we obtained these graph, since the procedure is somehow common for most of the results shown in this chapter. First of all, we ran a learning process consisting in several episodes and we registered the average rewards (i.e., punishments) that the prey received during the chasing part of each episode; after this, what we plot is that list of average rewards, but passing it through a moving average filter of size 1000 (this last part is necessary because

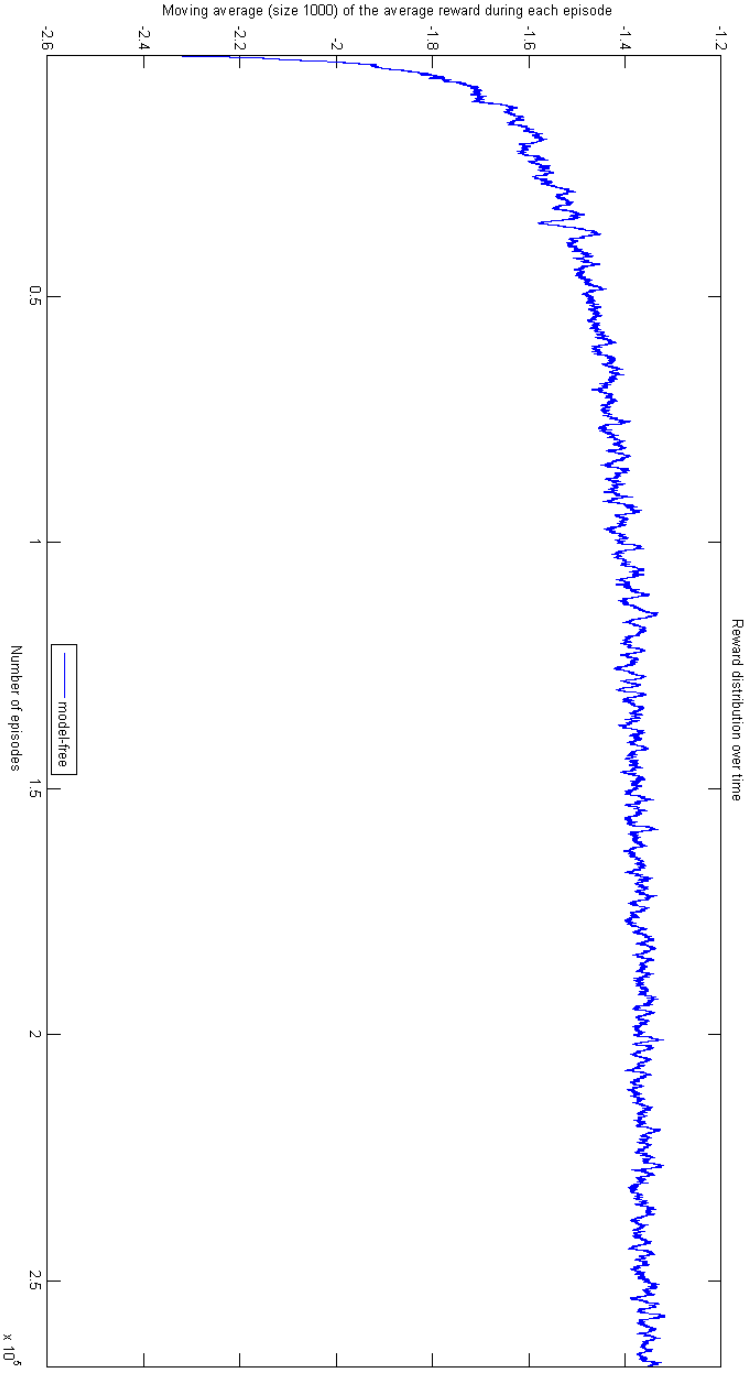

Figure 4.3: **Convergence of the model-free module alone on the map of *Figure 3.2*.**

the rewards oscillate too much to be directly analyzable graphically). For completeness, it also needs to be said that we exclude from the graphs the rewards of the first 1000 episodes because of the moving average filter (wherever this possibly had an impact on the results' meaning we analyzed the data to make sure that what we represent is conceptually sound even without the first episodes).

For further completeness, we also decided to show an example of a certain situation seen from the eyes of the model-free module (see *Figure 4.4*). In the case of the model-free module this is quite interesting because we can show side by side the evaluation of states given by the critic and the action policies that were subsequently defined from the actor. From the cell analysis we can see how of course the worst area is the one strictly around the predator and it becomes increasingly better as we go far from it. In the policy analysis, the actions that are interesting to observe are the ones that are in the few cells around the predator (for us the action policy adopted when prey and predator are not in sight[1] are not important, because there is no pain involved). We can see how, depending on the distance, the chosen actions well represent high level actions such as "try to leave the predator behind" or "hide behind an obstacle" (remember that the predator is slower than the prey, so leaving it behind becomes a viable option).

At this point, it may also be useful to review *Figure 3.15* on page 58. As it can be seen, δ values proceed on average from strongly negative values towards values closer to zero as the module moves towards convergence. This is one of the main reasons why the integration with a threshold on filtered values of δ that was explained in Sections 3.5 and 3.6 works and makes sense. A second reason is the deep meaning of δ values that was explained in this thesis).

Another reason why the integration technique works can be seen taking a closer look at the δ values that bring to convergence performing a histogram analysis, shown in *Figure 4.5*. The results in the figure show that at the beginning of the training δ values are unimodal; we performed the same analysis for the following intervals of episodes and we obtained the same result, with the mode of the distribution that was moving towards zero and with a more peaked distribution (we do not report them here for a matter of space). Another detailed analysis of these values is reported in Section 4.8.

Finally, we report also a sample run of the model-based module up to convergence (see *Figure 4.6*) . It is left as future work the task of designing

[1] We remind the reader that this means that the prey is hidden behind an obstacle or prey and predator are too far from each other, where the limit distance is roughly half of the dimension of one side of the arena.

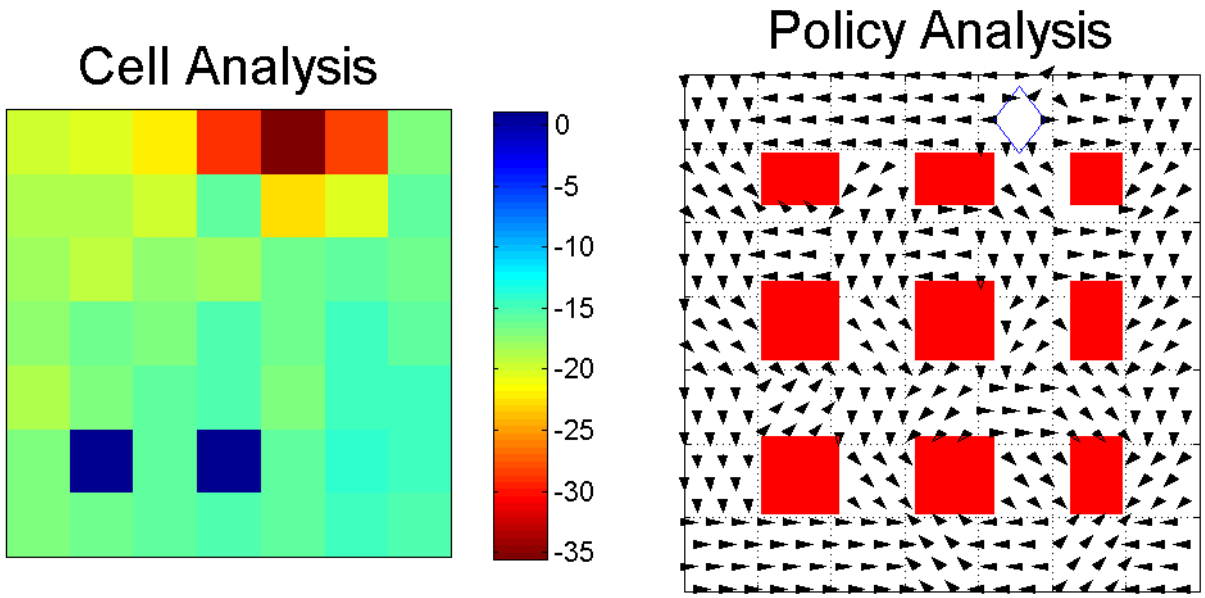

Figure 4.4: **Cell and policy analysis for the model-free module**. The position of the predator is fixed at the top of the map (darkest cell in the cell analysis, blue diamond in the policy analysis), then we move around the prey and observe how the actor and the critic behave. In the left part (cell analysis) we show the average state value for each squared discretized cell when the predator is in the aforementioned position and the prey is moving around. The two dark blue cells remained with the same value that they had at the very beginning because they coincide with obstacles, so since the prey can never go there their values will not be updated. In the right part (policy analysis) we fix the predator as we have said before, we place the prey in all possible hexagonal tiles, we check which action the actor would pick and we represent it with an arrow. The red squares represent the obstacles, or more precisely the exact obstacles that we defined even before the hexagonal tessellation was performed (of course, due to the tessellation the final shape of the obstacles will be slightly different).

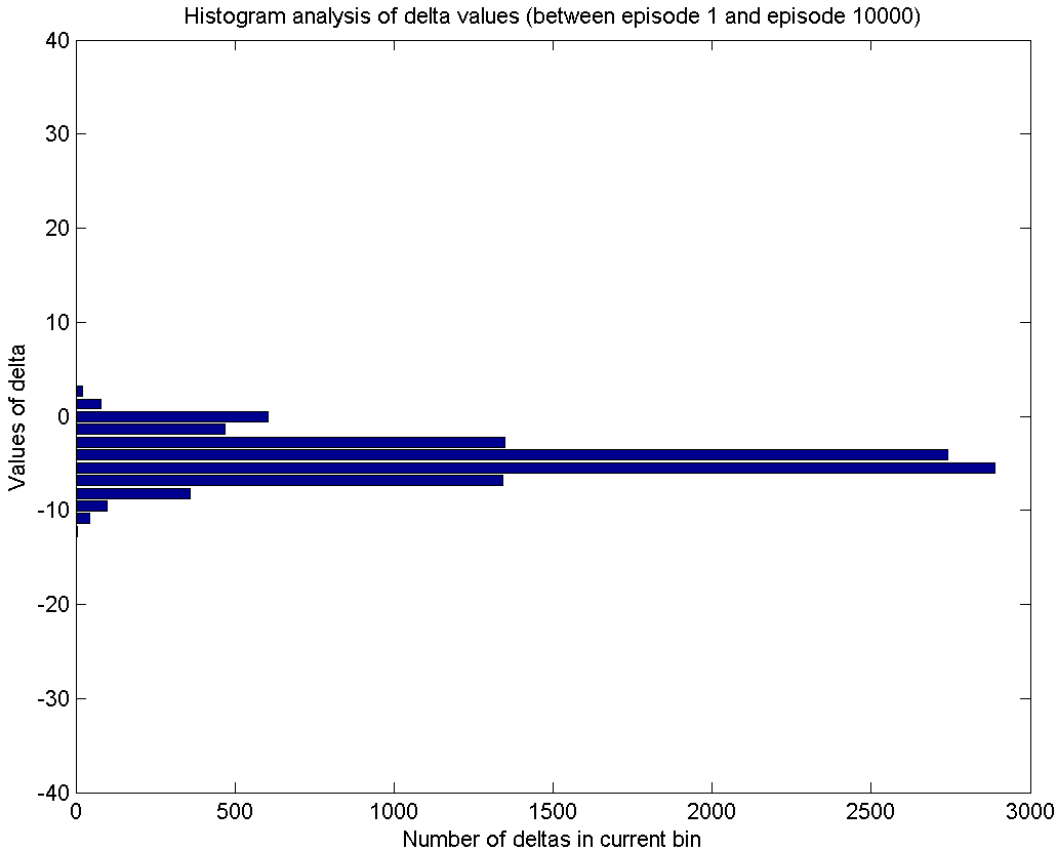

Figure 4.5: **Histogram analysis of δ values**. We took δ values of the first 10000 episodes, we divided the whole range of deltas obtained during the training in 50 bins and we counted how many δ values we had in each bin for the first 10000 episodes.

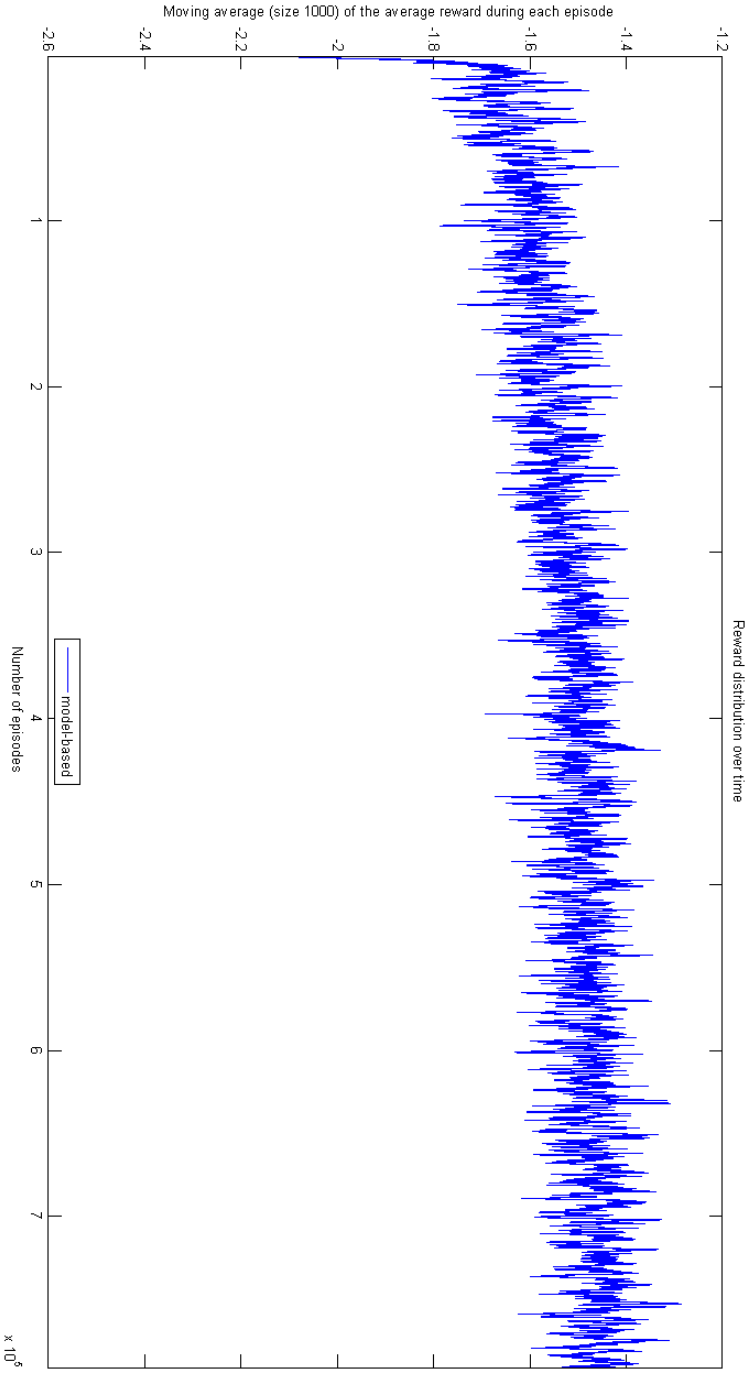

Figure 4.6: **Convergence of the model-based module alone on the map of *Figure 3.2*.**

new procedures to better visualize and analyze what is happening during this module's convergence, but for our current purposes the convergence graph was sufficient. It is anyway an important result because the technique that we are using for goal-driven control is novel, so the fact itself of empirically proving its convergence is an important achievement.

4.3 Training of Pavlovian module

As we have said in Section 3.7, before integrating the modules we need to train a set of reflexes using GAs. We show in *Figure 4.7* the GA run that

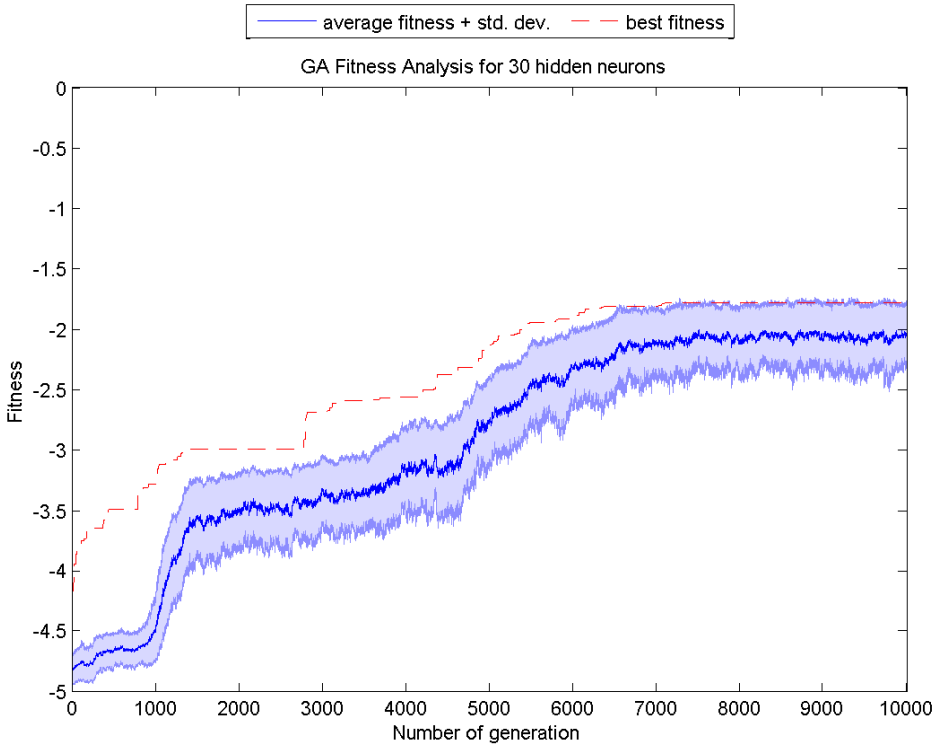

Figure 4.7: **Evolution of reflexes through a GA**. The dark blue line represents the average fitness during each episode, the light blue shade around it represents the standard deviation of individuals' fitness values and the red line is the fitness value of the best individual in each generation.

generated the reflexes that we use in this chapter. Since we already explained

how we obtained the fitness values in Section 3.4.5, we will not spend other words about it; the only thing that we would like to remind the reader is that, even if fitness values are rewards, they are not comparable to the effective rewards of the other convergence graphs (the averaging of rewards and the initialization of episodes to produce fitness values are different).

We want to point out to the reader that, given the nature of reflexes, we cannot make a convergence study similar to those shown for the other modules. There are several reasons behind this, but the main one is that Pavlovian control only works at a reduced distance between prey and predator (i.e., for substantial amounts of pain), so a purely Pavlovian control would probably seldom lead to infinite time episodes, because it would push the prey to escape from the predator, but often not enough to leave it behind or to hide behind an obstacle (of course, this also depends on the map, on the predator's speed and so on). We heuristically validated the final outcome of the GA: we set up a prey with a purely Pavlovian control with the best reflexes produced by the GA, we observed the resulting behavior and we verified that it was as expected. Again, since even in this case we were not sure about the performance of this technique, the fact that we could use it to obtain a good output behavior is in itself an important achievement.

4.4 Convergence of the aggregation of modules

Now we show the convergence of the various aggregations of modules that, as aforementioned, it is by itself a very important result because it shows that all combinations can learn and reach convergence. We report in *Figure 4.8* the convergence of the aggregation of model-free and Pavlovian modules, in *Figure 4.9* the convergence of the aggregation of model-based and model-free modules, then in *Figure 4.10* the aggregation of the three modules.

Even if the pair of model-based and Pavlovian modules is not very relevant from a biological point of view, we report anyway for completeness a study of its convergence in *Figure 4.11*. Anyway, because of this reason we will not explore it further.

One thing that needs to be said is that, depending on the tuning of the parameters, the aggregation of different modules gives a different output performance and in the graphs that we presented in this section we did not use any particular combination of parameters. This means that these graphs were only meant to show the convergence of the various combinations and are not suitable for a comparison: we postpone this activity to the following sections.

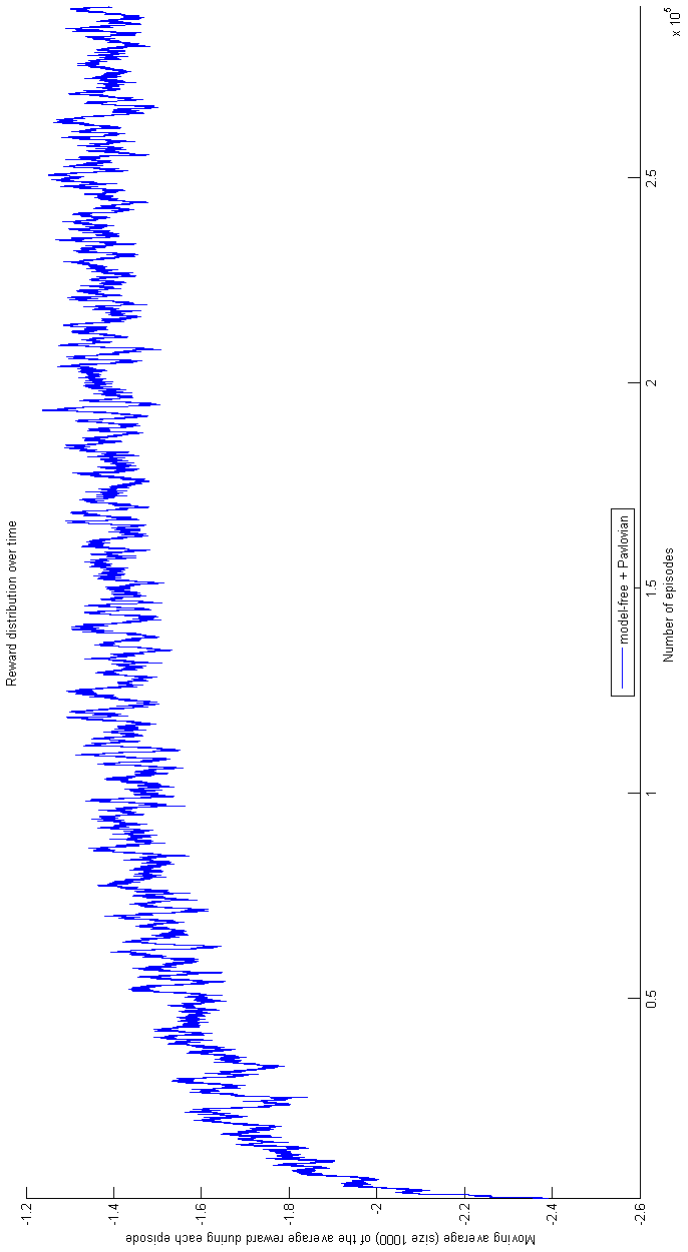

Figure 4.8: **Convergence of the integration of model-free and Pavlovian modules.**

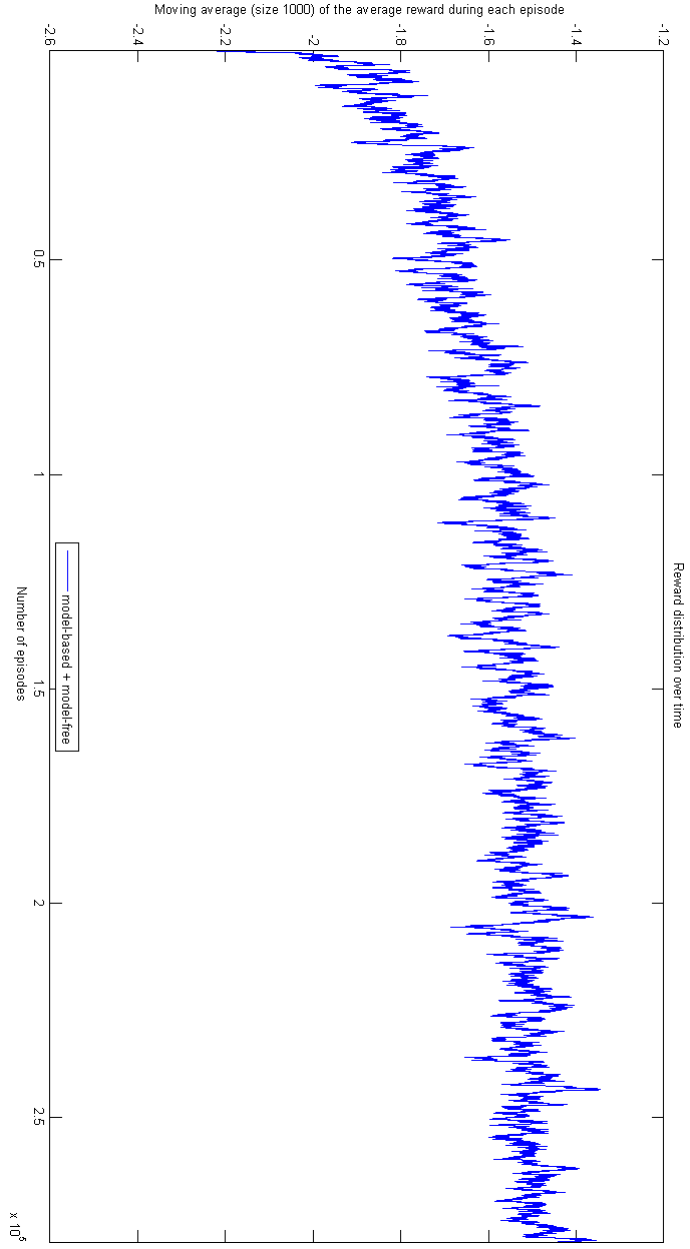

Figure 4.9: **Convergence of the integration of model-based and model-free modules.**

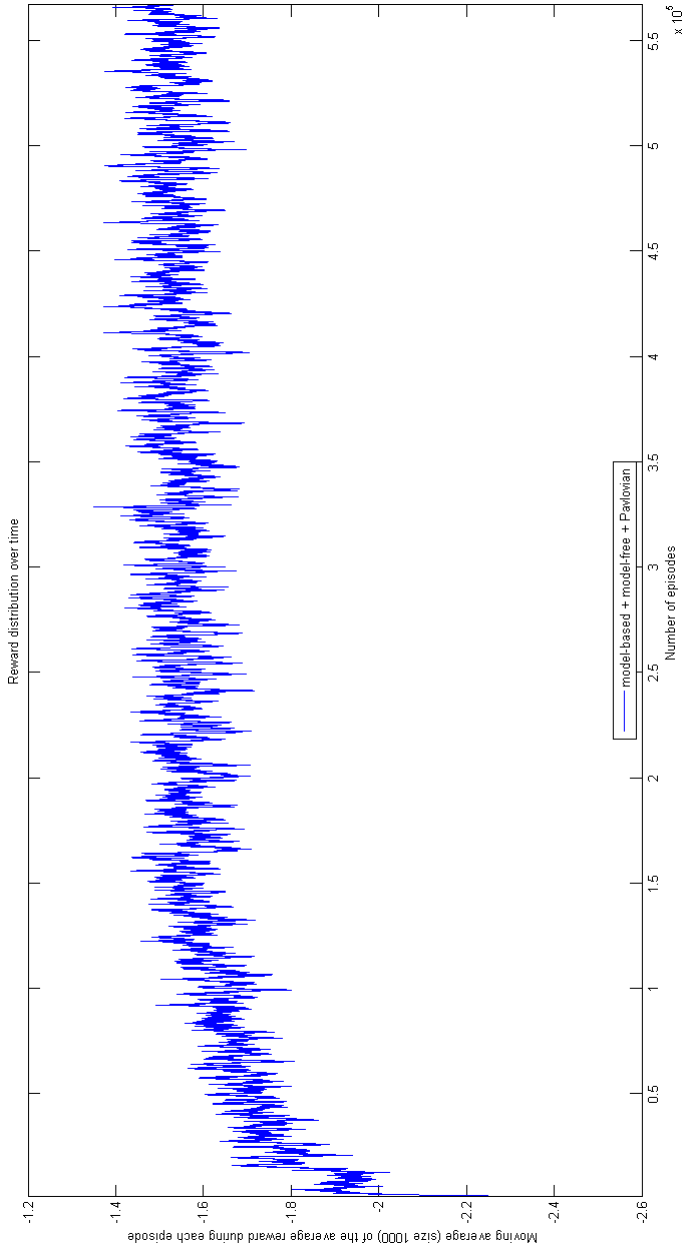

Figure 4.10: **Convergence of the integration of the three modules (model-based, model-free and Pavlovian).**

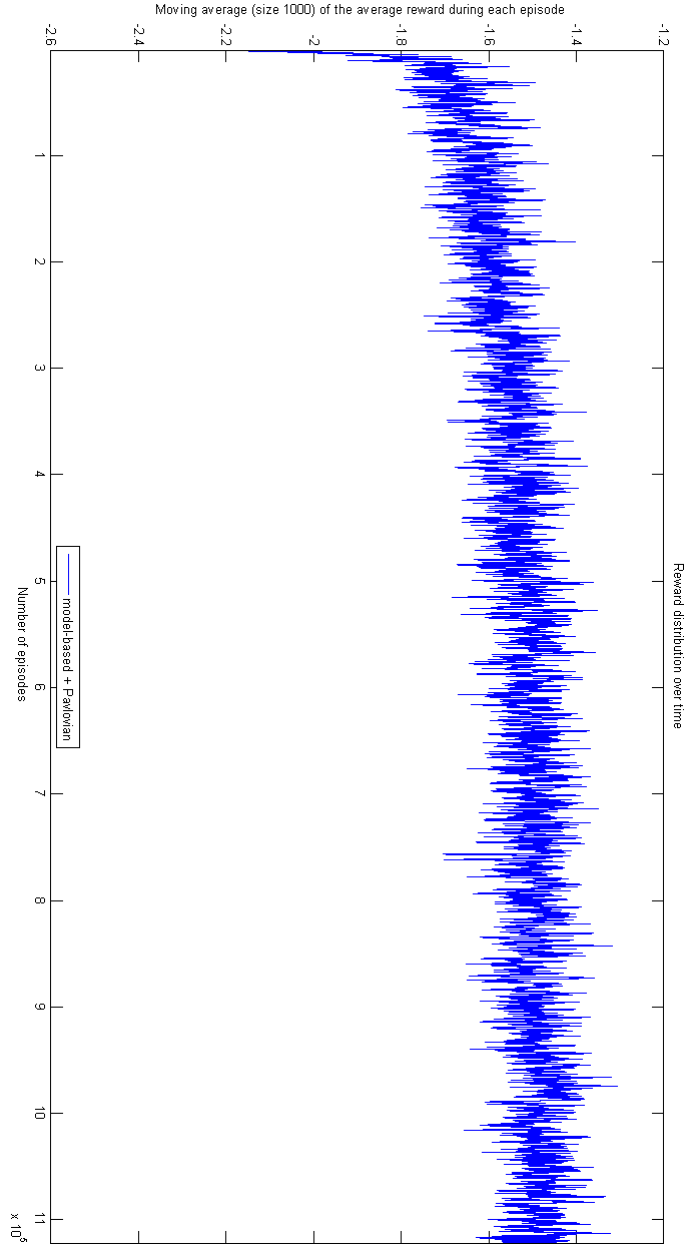

Figure 4.11: **Convergence of the integration of model-based and Pavlovian modules.**

Even if we are not going to show this analysis for every single pair because we believe that there is no important information added, for the three modules integration we report also an analysis of the activation frequencies of the three modules in *Figure 4.12*. As it was expected, if at the beginning

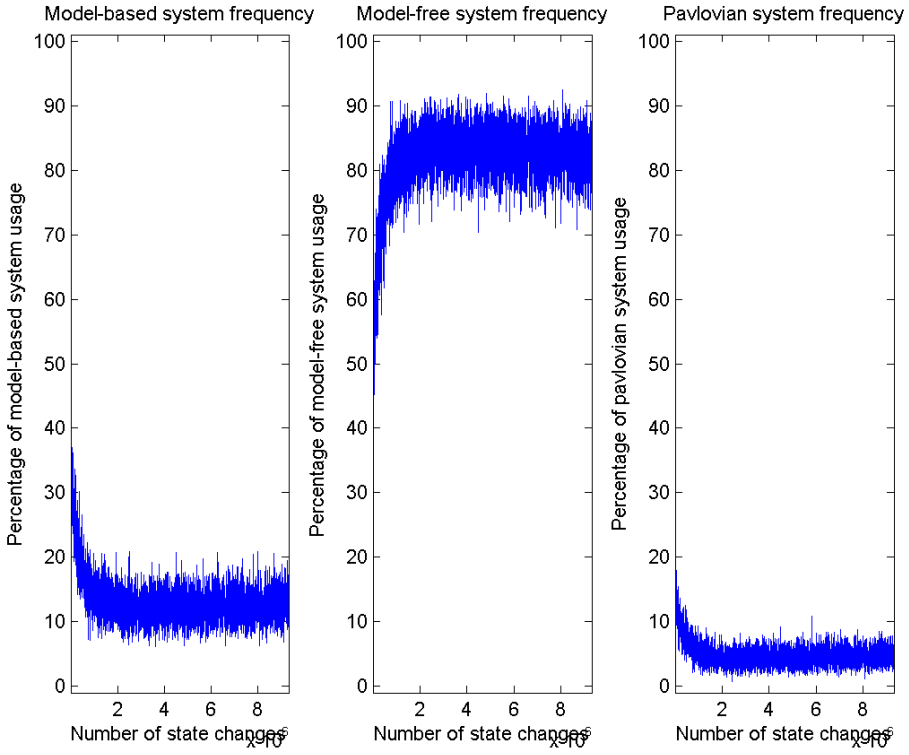

Figure 4.12: **Frequencies analysis of the three modules integration**. From the left to the right we have the percentages of activation during a sample run of the three modules model-based, model-free and Pavlovian. The percentages were obtained passing through a moving average filter of size 5000 the three vectors containing the integer 1 when each module was activated and 0 when it was not, then multiplying the results for 100.

the most active modules are model-based and Pavlovian, that are better at handling novelty, as time goes by model-free slowly takes control of most of the situations. Also, as it is reasonable to imagine, we do not use (apart from the very beginning) an extensive amounts of reflexes, since they are typically sub-optimal actions and are mainly useful at the beginning of an exploration.

Up to now, we could say that everything behaves as we expected, showing

that it is possible to mix these modules' actions to produce meaningful behaviors. Actually, one could notice some small flaws in the integration technique, but we will discuss them in Section 4.8.

4.5 Comparison of integration of modules

After showing the convergence of all possible combinations of modules, we started to explore in more detail the convergence process, trying to understand if it was possible to handle the arbitration process between modules in such a way that would allow to get the best of all of them. As we have already shown in *Figure 4.12*, when convergence is reached the model-free modules picks the great majority of actions, meaning that we can have good performance at a very low computational cost (actually, there's something more that needs to be said and we will do it later on in this section).

Following this, we concentrated on the beginning of the convergence process, that is of high interest since most of the interesting environment for future applications are likely to be dynamic. The first thing we did was showing how through some parameter tuning model-based and Pavlovian modules could bring benefits to the sole model-free module.

First of all, we compared the model-free module alone with the integration of model-free and Pavlovian modules on the map shown in *Figure 3.2* and we report the results in *Figure 4.13*, with a detail on the first episodes shown in *Figure 4.14*. For time reasons we decided to cut the length of the training, because anyway the results reported in the previous section guarantee us convergence and because our main focus here was on the first part of the training. However, to have more reliable results we decided to launch 10 runs for both combinations and average the results.

Since in the graphs we cut out the first 1000 episodes, we decided in this case to perform one further analysis to make sure that even for those episodes the performance is as we expect. What we did was first of all taking the two vectors of rewards averaged over 10 runs (basically the vectors that we show in *Figure 4.13* after filtering them with the moving average filter of size 1000) and then comparing them during the first 1000 episodes. In particular, if we call $avgModRew$ the vector that has the reward performance of the model-free module averaged over 10 runs and $avgLowRew$ the same vector for the integration of model-free and Pavlovian modules, we used the following formula for the comparison:

$$comp(x) = mean(avgModRew(1:x) - avgLowRew(1:x)). \tag{4.1}$$

The interpretation of the equation is the following: let us say that we want to

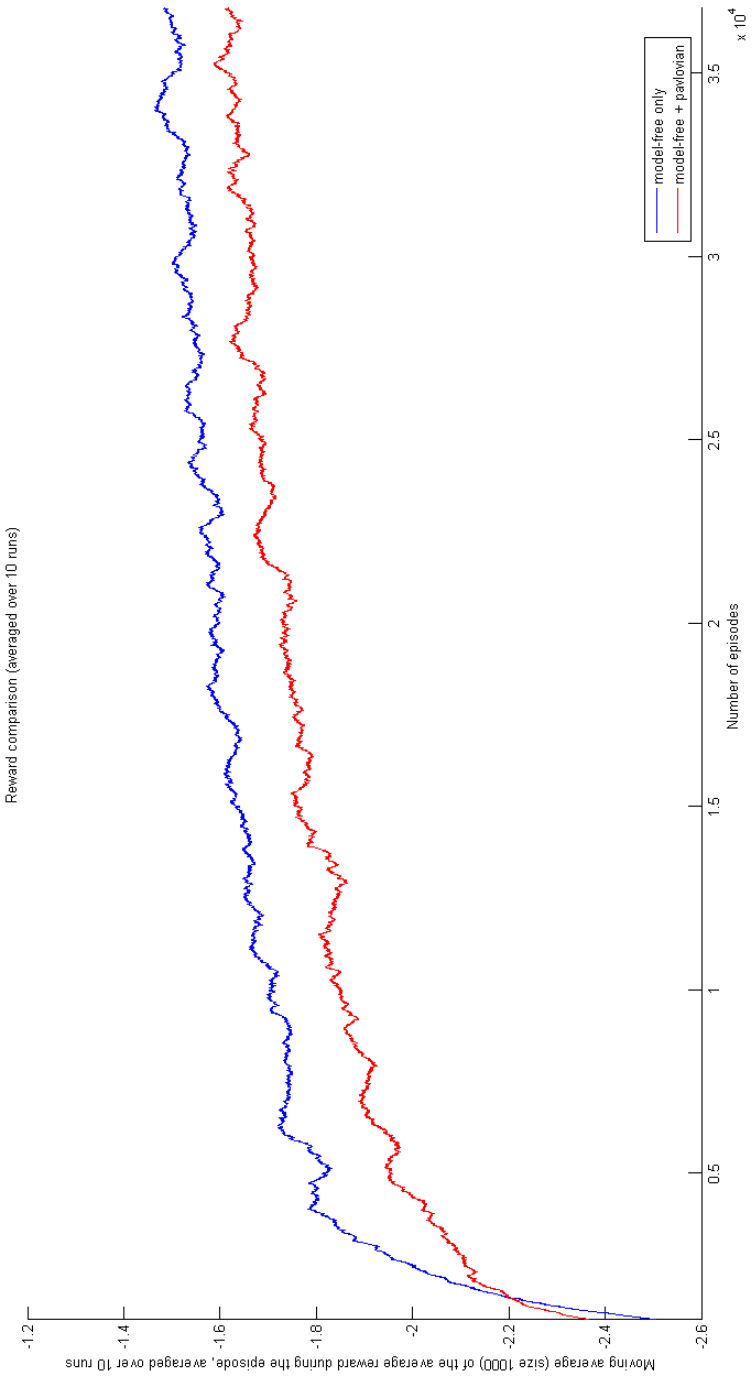

Figure 4.13: **Comparison of model-free alone and integration of model-free and Pavlovian**. The model-free module's performance is shown in blue, while the integration's performance is shown in red. Both of them actually represent the average performance over 10 runs, in order for them to be more reliable. In *Figure 4.13* we can see a detail of the first episodes.

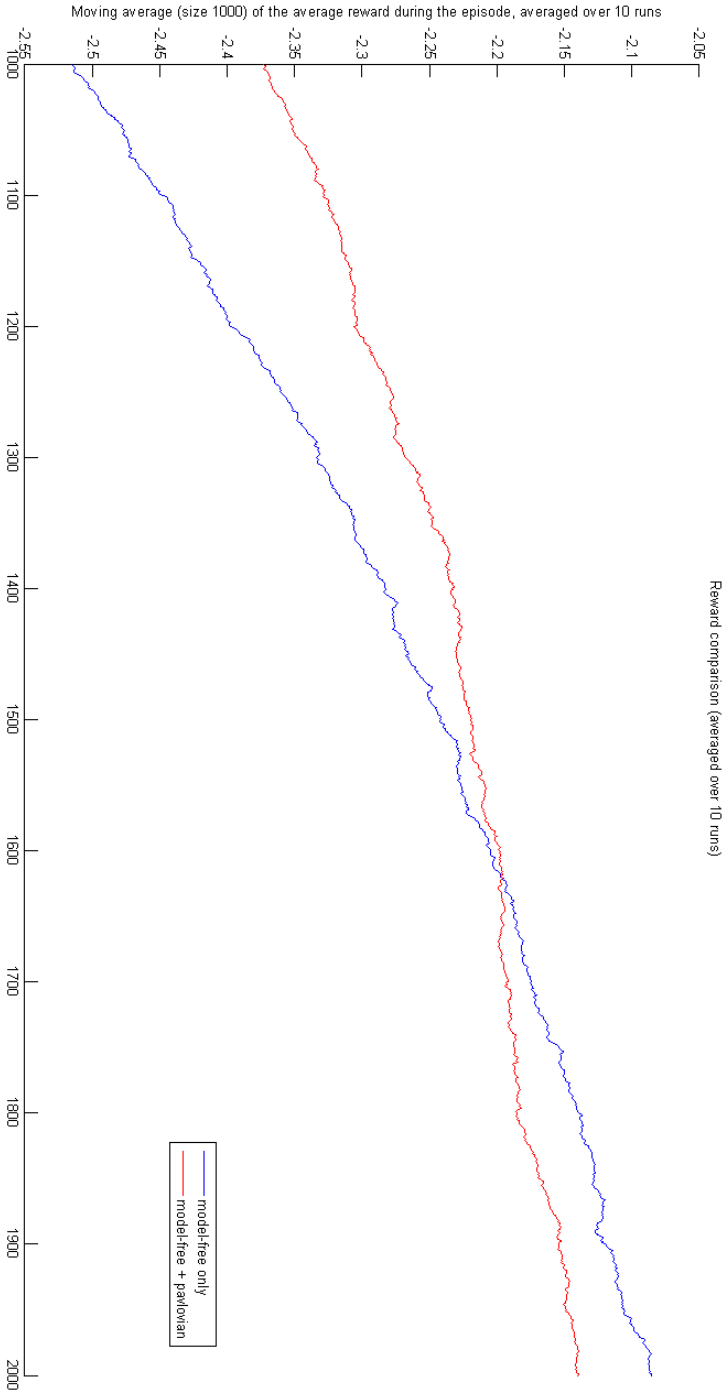

Figure 4.14: **Detail of *Figure 4.13*.** Here we show episodes between the 1000th and the 2000th.

compare the first x episodes, then we will take the first x cells of both vectors (as the notation "$1 : x$" suggests), compute the element-wise difference[2] obtaining as a result a single vector of size x, then eventually compute the average of that vector's values. If one thinks carefully about it, a negative value means that the model-free module alone has worse performance than the integration and vice versa. Results for these vectors are reported in *Table 4.1.*

x	**comp(x)**
100	-0.2422
200	-0.2370
300	-0.2483
500	-0.2280
750	-0.1928
1000	-0.1420

Table 4.1: **Comparison for the first 1000 episodes, not represented in *Figure 4.14*.**

Basically, what we can see here is that we actually can get an improvement in the registered performance at the beginning of the learning process with respect to model-free module alone, even if there is a tradeoff in the long term convergence process. The reader needs to be careful anyway: due to the high derivative in the first part of the learning process, it might be easy to underestimate the gain in the first episodes. Basically, we could say that mixing these two modules one can obtain an improvement in performance at the beginning of the convergence process, giving up some of the long term precision (we will come again on this in Section 4.8). Precise statistical analyses of this tradeoff remain an important part for future work. For the moment, what we believe is that this tradeoff is biologically sound: for an individual it may not be a big deal a difference in pain levels if we are talking about small amounts, while it may make a serious problem if the involved amounts of pain are considerable.

We then tried to use the same set of reflexes (that, as we want to remind to the reader, has been trained on the map of *Figure 3.2* on page 20), to obtain the same result on the map represented in *Figure 3.4* and a detail of

[2] An example of element-wise difference with vectors of size 2 is the following:

$$\begin{bmatrix}3\\5\end{bmatrix} - \begin{bmatrix}2\\-3\end{bmatrix} = \begin{bmatrix}3-2\\5-(-3)\end{bmatrix} = \begin{bmatrix}1\\8\end{bmatrix}$$

the results is reported in *Figure 4.15* (the long term convergence is longer but similar, so we will not show it here).We also repeated the analysis with Equation 4.1 and the results are reported in *Table 4.2*. The first thing that

x	**comp(x)**
100	-0.1326
200	-0.0410
300	-0.0586
500	-0.0618
750	-0.0767
1000	-0.0598

Table 4.2: **Comparison for the first 1000 episodes, not represented in *Figure 4.15*.**

we need to say is that, even if we used the same set of reflexes that we used on the other map, we needed to use a different threshold when we changed the map; anyway, we will spend more time on this in Section 4.8. The positive aspect of this result is that the Pavlovian actually showed the generalization properties that we wanted to see. It remains as future work to check how different the map can be to still get generalization and to improve the generalization aspects of the Pavlovian module.

After this, we passed to the integration between model-based and model-free modules, again through some parameter tuning. The overall results are shown in *Figure 4.16*, but again due to the time scale and to the high derivative at the beginning of the learning curve it is not easy to see the first part of the graph, so we provide the more detailed *Figure 4.17*. Again, we realized an analysis with an equation similar to Equation 4.1, where negative values meant that the integration of modules was better than model-free alone, and the results are reported in *Table 4.3*. Basically, the same reasoning that we applied for the former comparison applies here, so we will not repeat it.

At this point in time the last thing to try was starting with three modules and then deactivating one at a time and show that the same advantages that we have shown until now could be transferred to the three modules system. However, this was not the case as it can be observed in *Figure 4.18*. Basically, what we observed was that we did not lose the advantage that was given by the model-based or the Pavlovian modules to the model-free module, but that having them together did not give any additional benefits. Even in this case, however, there is a tradeoff that makes the performance worse on the long run. Our explanation, that should be further explored

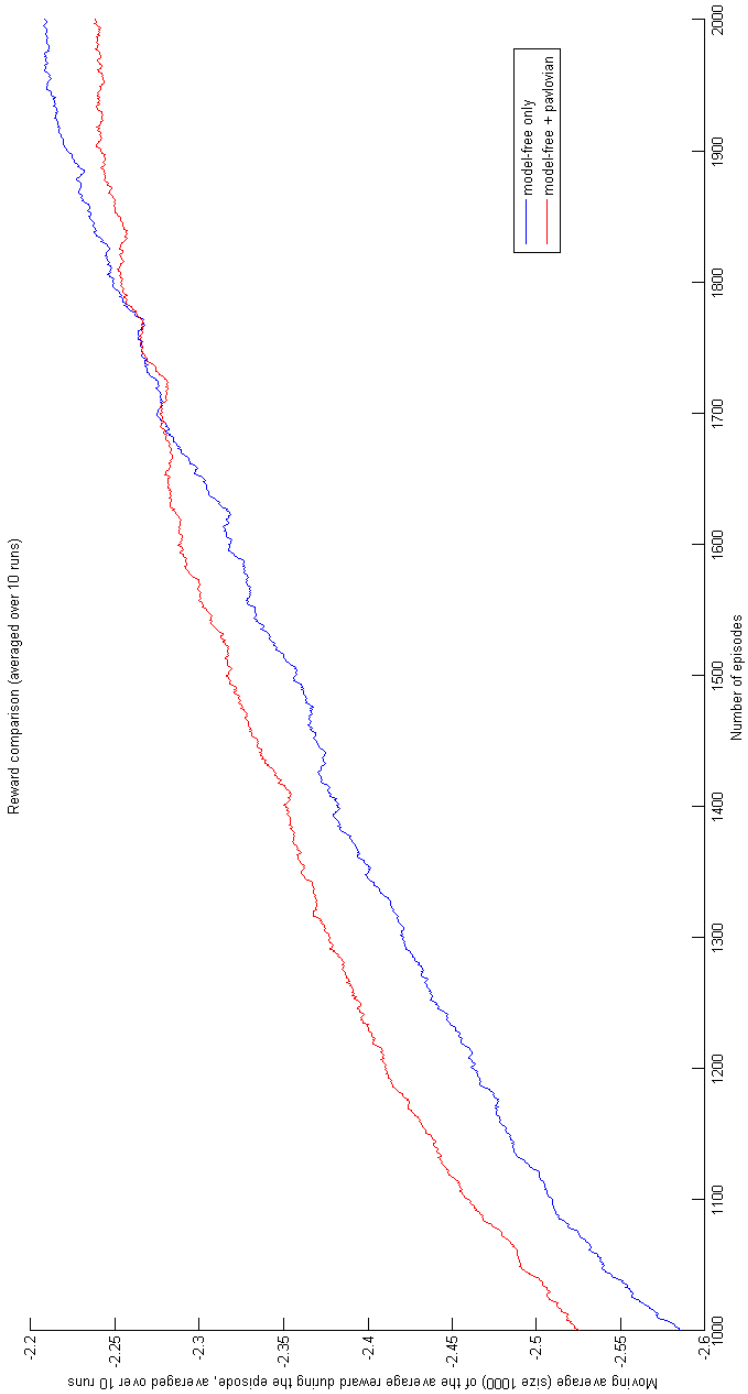

Figure 4.15: **Detail of a second comparison**. In this case we use reflexes trained on the map of *Figure 3.2* and we ran 10 runs for both combinations on the map of *Figure 3.4*. Again we show the episodes between the 1000th and the 2000th.

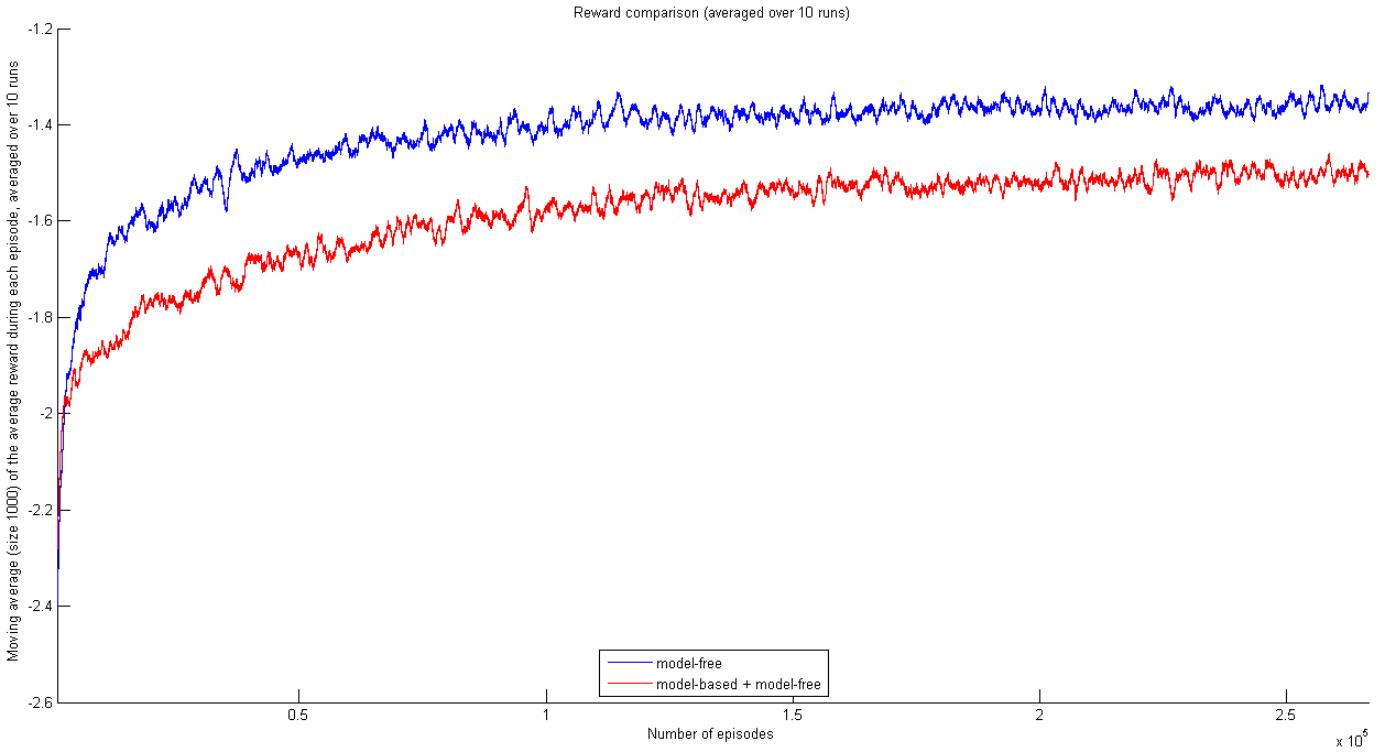

Figure 4.16: **Comparison of the model-free module alone and of the integration of model-based and model-free modules**. The model-free module's performance is shown in blue, while the integration's performance is shown in red. Both of them actually represent the average performance over 10 runs, in order for them to be more reliable. In *Figure 4.17* we can see a detail of the first episodes.

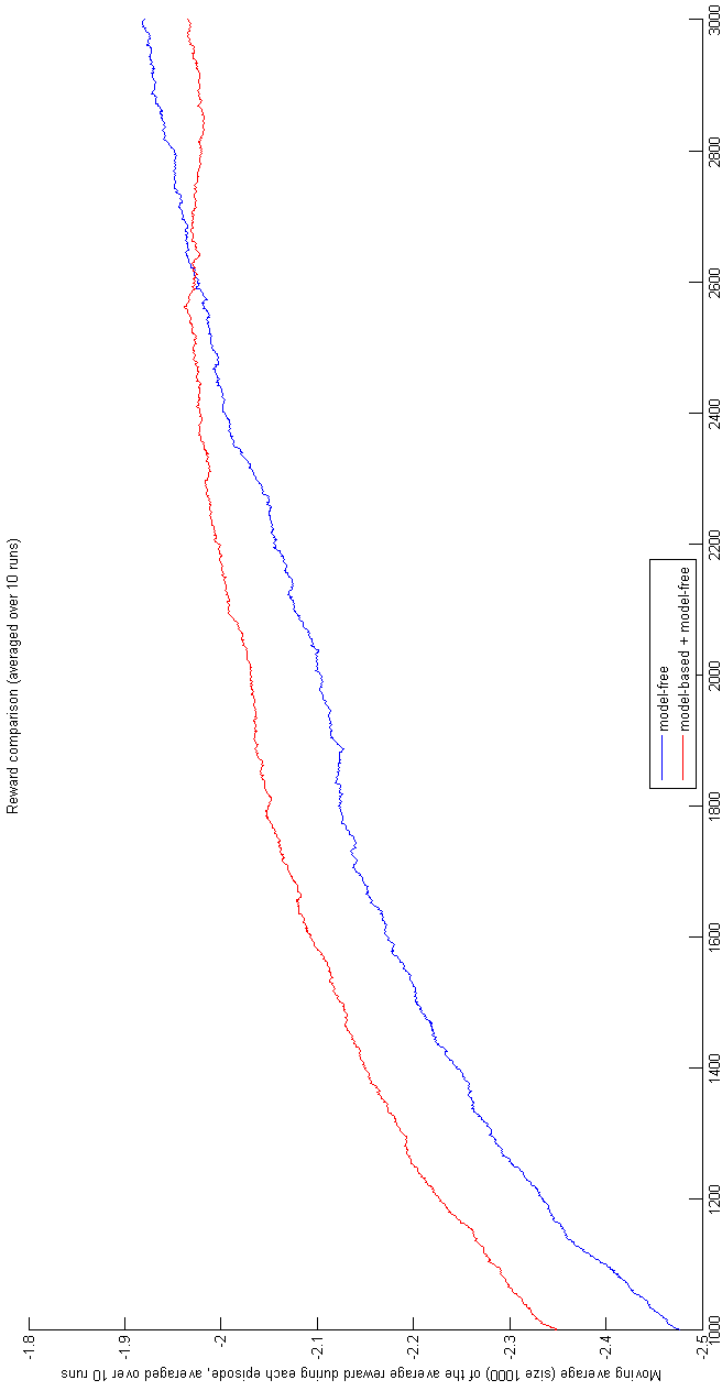

Figure 4.17: **Detail of *Figure 4.16***. Here we show episodes between the 1000th and the 3000th.

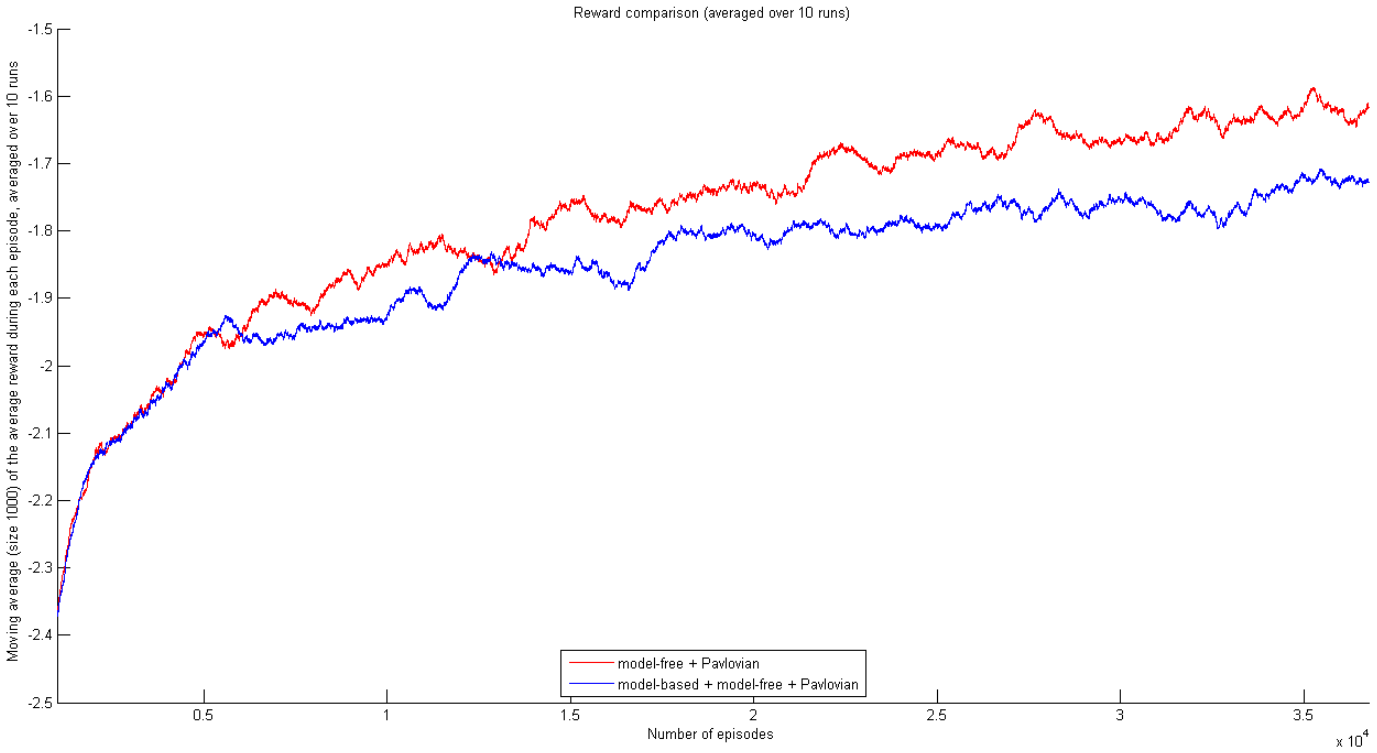

Figure 4.18: **A reward comparison between the system complete of the three modules and the same system with the model-based module disabled**. The system with three modules activated is represented by the blue line, while the one in which we shut down the model-based module is represented by the red line.

x	comp(x)
100	-0.1375
200	-0.1723
300	-0.1891
500	-0.1772
750	-0.1565
1000	-0.1283

Table 4.3: **Comparison for the first 1000 episodes, not represented in *Figure 4.16*.**

in the future, is that basically when it comes to learning avoidance actions the model-based and the Pavlovian module give to the model-free module a conceptually somehow similar kind of benefit, so that once one of them has covered the gap adding also the other does not bring additional benefits.

To wrap up what we have seen until now, first of all we showed the good performance of the three single modules (especially the two that were newly designed), then we showed that mixing the Pavlovian or the model-based modules with the model-free module makes it faster without losing its quality of being computationally cheap on the long run, then finally we showed that we could not obtain an advantage in terms of rewards when mixing three modules together (and we gave our hypothesis about why we could not do this). However, what we analysed until now is the avoidance learning, that is the learning of how to choose actions to avoid pain, but there is another concept that is very important: action extinction. How does our model behave when something that was a pain source stops being that?

4.6 An analysis with inverted rewards

Another kind of learning that is important to analyse is that of action extinction, that as has been explained in the previous chapters is a common test mode in psychology (see e.g. *Gillan et al. (2014)*). It is actually tricky to design a procedure that can be considered equivalent to the psychological counterpart, because it is not clear to define what would be an exact equivalent of an action extinction test in our context.

Assigning a constant zero reward when prey and predator can see each other is not feasible: since a reward equal to zero is also what the prey gets when the predator is not in sight, this solution would basically always assign

to the prey rewards equal to zero and this would create a flat surface in the space of policy evaluations, so that every policy would be optimal and we cannot predict the resultant behavior.

Given this first considerations, we evaluated a simple procedure that is the following: we assign to the prey a reward of zero when it cannot see the predator and a constant positive reward of value C when prey and predator are in contact. The problem in this case is that, since in our analyses we only consider the rewards that the prey gets when it is in contact with the predator, what we would see from the analysis would just be a constant C value for the rewards, independent on the progress of the learning process.

The final procedure that we designed and that we use in this section consists basically into inverting the sign of the reward procedure used in previous sections. Of course, again if prey and predator are not in contact (i.e. they are distant or separated by an obstacle) the assigned reward is zero. However, now if they are very close (i.e. enough to assign the maximum negative reward with the standard procedure) we assign the maximum positive reward, that is equal in module and opposite in sign to the previously used maximum negative reward; then, as the distance increases, we keep decreasing the assigned rewards, again using values that are equal in module but opposite in sign with rewards that were assigned in the normal procedure. This creates a gradient in rewards opposite to the one we had with the other experiments, pushing the prey to stay as close as possible to the predator. One can find e.g. in *Dickinson and Balleine (2002)* a biological justification of this procedure.

It needs to be said that, since we did not change the predator's policy (it does not affect the meaning of the experiment), the likely outcome was having "contact situations" (i.e. situations in which prey and predator can see each other) of duration increasing during time, because the prey would learn episode after episode to stay close to the predator. For this reason and in order to keep the task as episodic, we decided to manually interrupt the episodes when during an episode we reached a certain number of consecutive state changes in which the prey was in constant contact with the predator (in our case, we used 80 state changes). For the standard procedure an episode was interrupted when, after getting into contact, prey and predator did not see each other anymore (because of distance or obstacles); to this, we add the state changes limit.

The final test procedure that we adopted is the following: first of all, we launched a full convergence of the three modules together with negative rewards (previously shown in *Figure 4.10*), then we used the final point of the convergence as a starting point, we inverted the reward assignment pro-

cedure (i.e. started to assign positive rewards) as we have just explained and we launched several different runs. In particular, we launched 10 runs with again the three modules altogether and positive rewards and also we launched 10 runs with positive rewards and with only model-free and Pavlovian modules (i.e. shutting down the model-based module). The purpose of this study was to determine whether the model-based module could bring any benefit in the action extinction context or not. We point out to the reader the fact that in the inverted rewards context we are using positive rewards and in the current model configuration the Pavlovian can be activated only if rewards are below a certain negative threshold, so in the positive reward context there will be no reflexive responses (as we would expect from a biological point of view).

The experiment is shown in *Figure 4.19*. The single run with all the three modules working and with negative rewards is shown at the beginning of the graph in black; notice that we cut out several episodes because they were not interesting in this analysis. From there, we started with positive rewards 10 runs of the three modules again (the average reward is shown in red) and 10 runs of only model-free and Pavlovian (average reward in blue). We do not show the detail of the transition, but we verified that the red curve is always above the blue one.

As it is possible to see from the figure, having three modules gives a considerable benefit in an action extinction context and this is consistent with what we know from the literature; in fact, it is possible in action extinction to isolate the effects of model-based and model-free in humans (see also the reported material about OCD), so that we knew that the result shown here was fundamental since it has been shown clearly also in humans.

4.7 OCD

For the OCD pathology, the reference work was *Gillan et al. (2014)*: we will try to summarize its core here. Test subjects have both hands connected to electrodes that can give them a painful electrical shock. In front of the subjects there are two lights, one red and one blue; finally, on the floor there are two pedals. If the test subject does not perform any action, after some time one of the lights is turned on a shock is administered: if the light is the blue one, the shock will be on the right hand, while if it is the red light the shock will be on the left hand. However, subjects are given a way to avoid the shock: if just after a light is turned on they press the associated pedal, they can avoid the shock (i.e. they need to press the right pedal just after the blue light and vice versa to avoid the shock). This setting is shown in

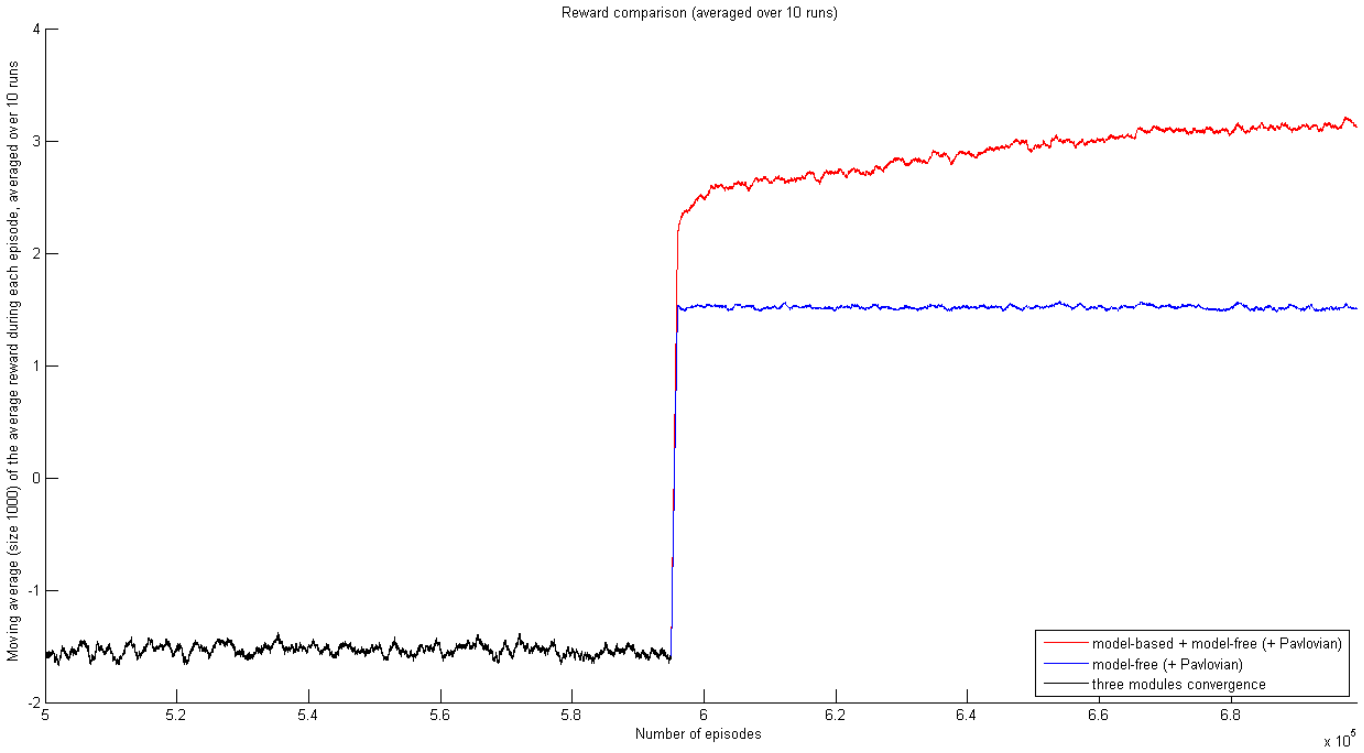

Figure 4.19: **An experiment with inverted rewards**. At the beginning, the black line represents the convergence with negative rewards; we cut several episodes to have a graph that could be more clear, since we already showed the normal convergence. After that point we invert the rewards and we observe the behavior with three modules working (in red) and with only model-free and Pavlovian (in blue). We do not show a detail of the transition, but it can be observed that the red curve is always above the blue one.

Figure 4.20[3].

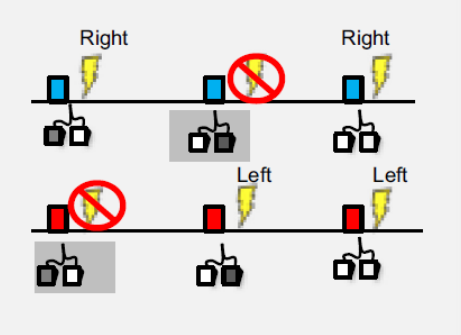

Figure 4.20: **The experimental setting for OCD testing**. If just after the blue light is turned on the subject presses the right pedal, he can avoid the shock; the same applies for the red light and the left pedal. If the wrong pedal is pressed, or no pedal is pressed or both of them are, the subject receives the shock.

The first part of the experiment consists in a training session in the context that we have just described; the subjects will then learn to associate a visual cue (i.e. the light) to an action (i.e. pressing the pedal). After this, we disconnect the electrode from one of the two hands, we tell the subject about this so that he knows that he cannot receive any more pain on that hand and we continue the experiment. The hand with the disconnected electrode is now devalued, meaning that it cannot receive pain and the subject knows about that, so we intuitively expect him to slowly stop pressing the associated pedal. This experiment actually outlines and brings out the difference between the model-based and model-free control systems: if the former reacts to the change and suggests to stop pressing the pedal associated with the disconnected hand, the latter tends to stick with the same actions because habits are more difficult to change.

Since model-based and model-free have these different characteristics and OCD disrupts their balance, this experiment allows us to study this pathology. However, there is one tricky aspect of the researchers' results: during the first part of the training (i.e. when both hands are connected) healthy subjects and OCD patients basically learn at the same speed, while in the second part (i.e. when one hand is disconnected) OCD patients tend to rely more on model-free actions and tend to press more the pedal that is

[3]Figure from *Gillan et al. (2014)*.

now useless. Their results are summarized in *Figure 4.21*[4].

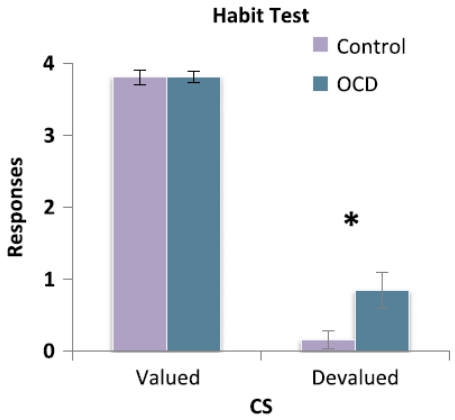

Figure 4.21: **The histogram for the results of the shock experiment**. What this graph basically shows is that, even if when the hand is attached to the electrode both healthy individuals and OCD patients learn at the same rate to press the pedal, when the hand is disconnected the latter tend to stick more with habits and press more the pedal even if it is useless at that point.

The testing scenario that we adopted is substantially the same that we used in the previous section with the analysis for inverted rewards: we trained the individual with negative rewards (i.e. with pain), then we inverted the rewards and we observed what happened. We anticipate here the core of this results: all the shades between a healthy subject and an OCD patient can be covered modifying the value of the positive threshold that activates the model-based module (that we called $model\_thresh_{pos}$ in Section 3.6), while the negative threshold remains the same for both of them.

The reason why this happens is basically the fact that when after a normal training we invert the rewards also δ values are inverted (even if with slightly different values), so that the part of training with positive rewards is basically regulated only by $model\_thresh_{pos}$. This is also linked to an important finding: even if there are some oscillations, the two thresholds $model\_thresh_{neg}$ and $model\_thresh_{pos}$ showed to be able to regulate the two parts of the training quite independently (the former regulates the part with negative rewards and the latter the part with positive ones). From these observations we could derive this conclusion: having the same value for

[4]Figure from *Gillan et al. (2014)*.

$model\_thresh_{neg}$ gives to both healthy subjects and OCD patients the same capabilities during the training with negative rewards, while giving them two different values for $model\_thresh_{pos}$ allows us to create the difference in performance that we expect between the two categories.

Unfortunately, for a matter of time in this case we can only show some preliminary results, that are still quite promising. We could not do the averaging between 10 runs, that would give to our results some more strength, but we deem that the regularity of our results should give the reader an intuitive proof of reliability.

One last premise that we need to do before showing the results is the evaluation technique that we used. Since in this case we are dealing with a psychological pathology, it looks more reasonable to observe the same quantities that one would observe in a psychological experiment; for this reason, instead of looking at the rewards we observed the time necessary to escape (i.e. leave the predator behind or hide behind an obstacle), measured in terms of number of state changes for every episode.

First of all, given the independence of the effects of the two thresholds, we just studied the second part of the training (i.e. the one with positive rewards) trying to show how moving the positive threshold could actually show the various shades of learning speed and we show the results in *Figure 4.22*. It can be seen how for increasing values of the positive threshold the individual tends to stick more and more with the previously acquired habits, since the model-based module tends to be activated less and this makes the prey less reactive. Basically, if one could observe this from the outside (as psychologists do) would see the prey still trying to escape from the predator even after it cannot receive pain, because the number of state changes remains much lower than it could be if the prey used the model-based module more frequently.

After this, we decided to repeat the experiment for one healthy subject and one OCD patient, but this time with separate convergences also in the part with negative rewards, just to show that the two threshold values control in a substantially independent way the two parts of the learning; results are shown in *Figure 4.23*. If one compares these results with the histogram from *Gillan et al. (2014)* (shown in *Figure 4.21*) one should be able to see a common pattern. These are just preliminary results and more work should be done in this sense in order to verify more clearly whether these results are reliable or not (even if we do believe so). Also, one should conduct more precise study in order to figure out the correct amount of impairment that a typical OCD patient has.

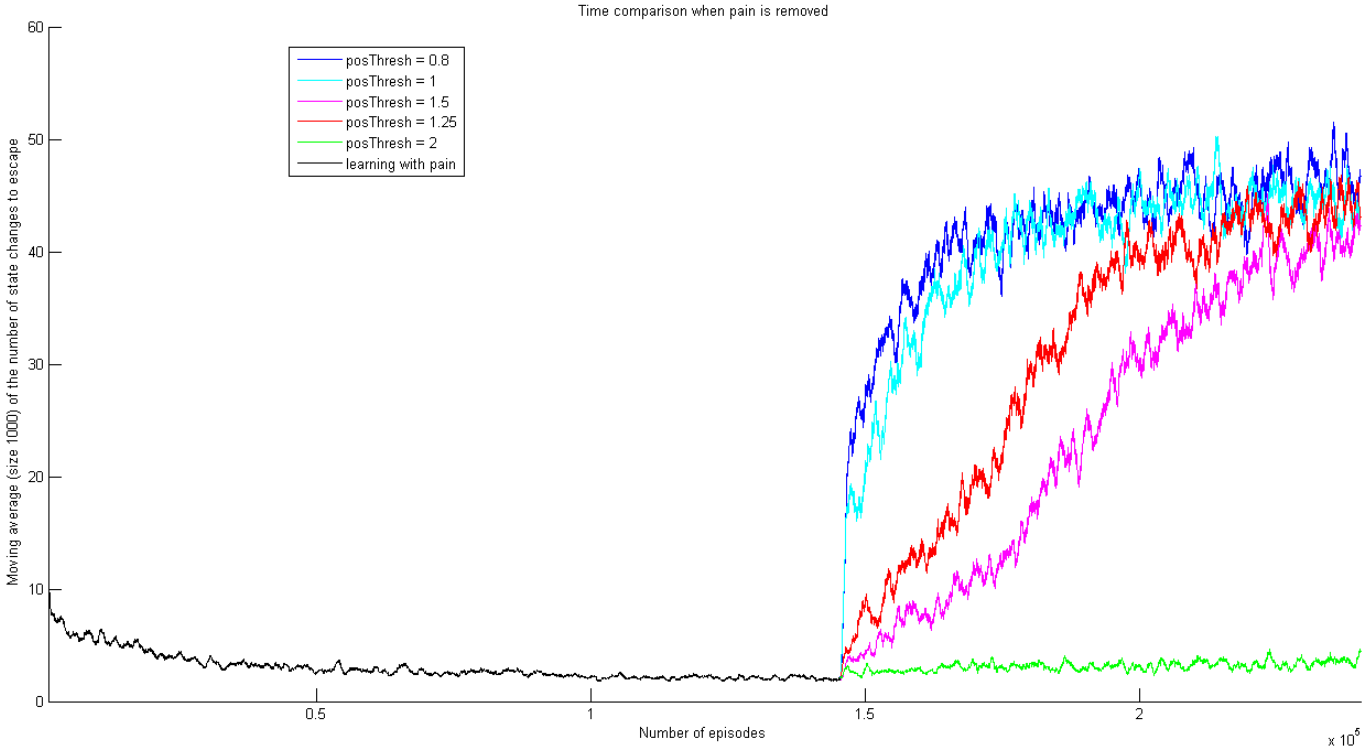

Figure 4.22: **A parameter study for OCD**. One can observe how for small values of the positive threshold the model-based module is activated often, so that it brings benefits to the learning process and an external observer would see a prey that immediately starts to stay closer and closer to the prey. Instead, for big values of the threshold the model-based module is often activated and the observer would see a prey that, even if it is not receiving pain anymore, keeps escaping from the predator (the number of state changes to escape remains low even after we invert the rewards).

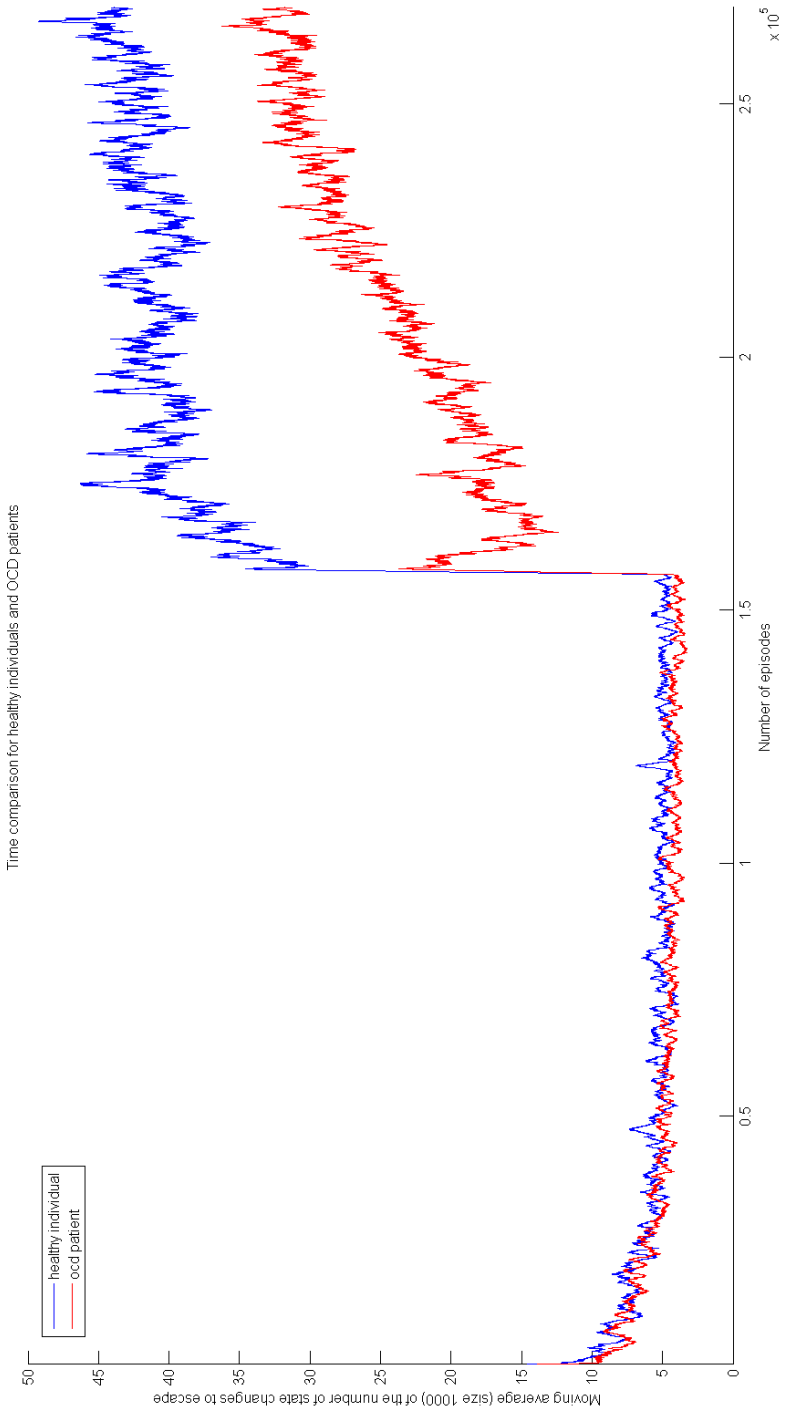

Figure 4.23: **Comparison between healthy subject and OCD patient.** Performances are measured in time, in terms of the number of state changes necessary for the prey to escape from the predator (we start to count in the moment they can see each other). One should compare these results with the histogram in Figure 4.21.

4.8 Limitations of our work

In this section we try to point out some limitations that our work has and that we still did not mention in the previous sections. We try to analyze our work with a slightly broader perspective and, when we identify some (even potential) flaws in our model or in our test procedure, we propose some ideas about how to solve these issues.

First of all, as we started to discuss in the previous sections, one issue of the model is that the Pavlovian module does not have complete generalization capabilities. This means that, even if most of its benefits can be generalized across maps that are quite similar, like the ones used in this chapter, if reflexes are trained on a substantially different map (e.g., a map with a big, cross-shaped obstacle in the middle of the arena), they do not perform well on these maps.

One way to try to obtain cross map generalization capabilities would be that of repeating the training process explained in Section 3.4.5 for different maps, each one with its initialization points, and then computing a fitness value that takes into account the overall performance on different maps. Anyway, it may also be the case that this procedure produces some average reflexes that are decent on all maps but not good enough to give the expected benefits on any of the maps. It is left for future work to assess in the detail the generalization capabilities of the reflexes and to design a procedure to improve their performance.

One hypothesis that we have about the somehow limited generalization capabilities of the Pavlovian module trained on one single map, but that we do not have tested at all, is that the main issue in generalization is related to null actions. As we have explained in the previous chapters, if the generated action would set the virtual target in a squared cell that is considered occupied (where this could also mean that it is just outside the arena), the target is automatically set to the center of the cell in which the prey currently is. Of course, if there is a considerable amount of such actions the prey is somehow impaired in its capacity to escape and it will get higher pain levels. What we think is that, more than the quality of the valid actions proposed, when reflexes are trained on a map and brought on another one the main issue is that a lot of null actions are generated. Anyway, this is just an hypothesis and in the future some tests should be conducted to disprove it or to substantiate it.

Another more important limit to our current model is that each map requires different values for the thresholds for the integration, explained in Section 3.6, and this limits heavily the generalization properties of the

results. This does not mean that using the same parameters on two different maps gives divergence, because the system showed on average good stability properties, but in order to obtain the benefits discussed in previous sections one needs to tune the parameters on the specific map that one wants to use with our model.

It is still unclear how to obtain an equivalent performance in different maps using the same threshold values, or equivalently how to have adaptive threshold values. This is of course one of the biggest and most important challenges for future work, because in order to apply our model to real world cases it needs to be able to automatically generalize in front of a change in the environment. We believe anyway that this will require a consistent effort, because one will need to test different solutions with several simulations ran on different maps to make sure that the model's properties are actually extended to different maps.

Another future work that needs to be done is an accurate statistical analysis of the gain that the module integration brings at the beginning of the learning and of the loss in the following part. Our current claim is that, since most of the environments in which we expect to use the model are dynamic, the short term gain in front of novelty is much more valuable than a comparable loss in the long term, but a statistical analysis is needed to analyze this difference quantitatively.

One last issue that we found out is about the integration technique and is easier to see taking into consideration *Figure 4.24* and *Figure 4.25* (that is the same analysis, but with a different size for the groups of values used for box-plots). As it can be observed from the box-plot analysis of the registered δ values, at the beginning of the learning process there is an expansion of the values (see especially *Figure 4.24*) before the later compression starts to happen (see *Figure 4.25*) as δ get closer to zero. This does not fit very well with the threshold integration mode, because at the very beginning of the learning we would like to use especially the Pavlovian and model-based modules, but this requires going over the threshold from the very first episodes and this aspect of δ values is not optimal in this sense.

We did not have problems using reduced values for the thresholds (i.e., values within the maximum pain values), but when we tried to set for the thresholds some values that were bigger in module we found strange warps at the beginning of the activation frequencies graphs, due to this issue. An important part of future work will be the investigation of this aspect, to understand whether it can be ignored because it does not affect significantly the performance or if it needs to be eliminated somehow. Anyway, we believe that this aspect and the fact that threshold values depend on the map should

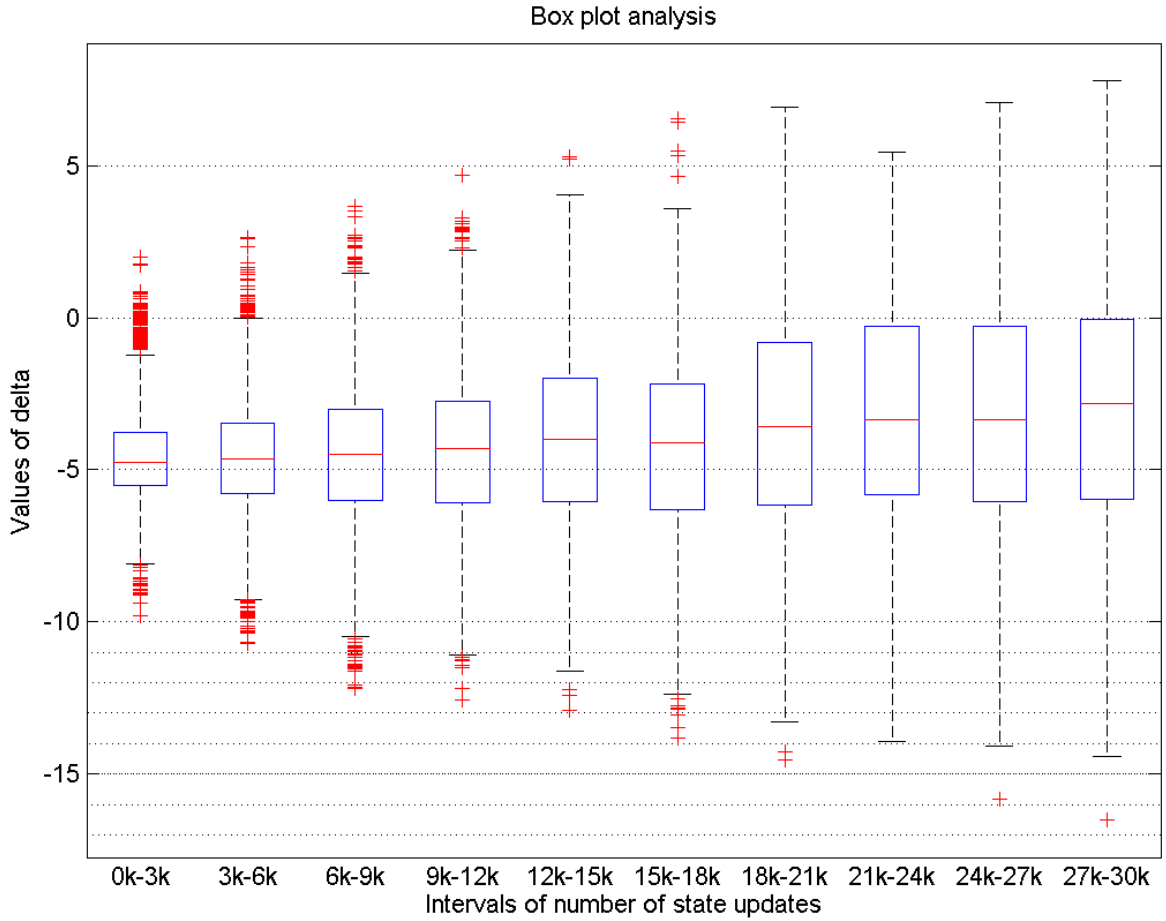

Figure 4.24: **A box-plot analysis of δ values**. The first 30.000 values of δ have been grouped in 10 groups of 3.000 values each (of course in chronological order), then we drew a box-plot for each group of values.

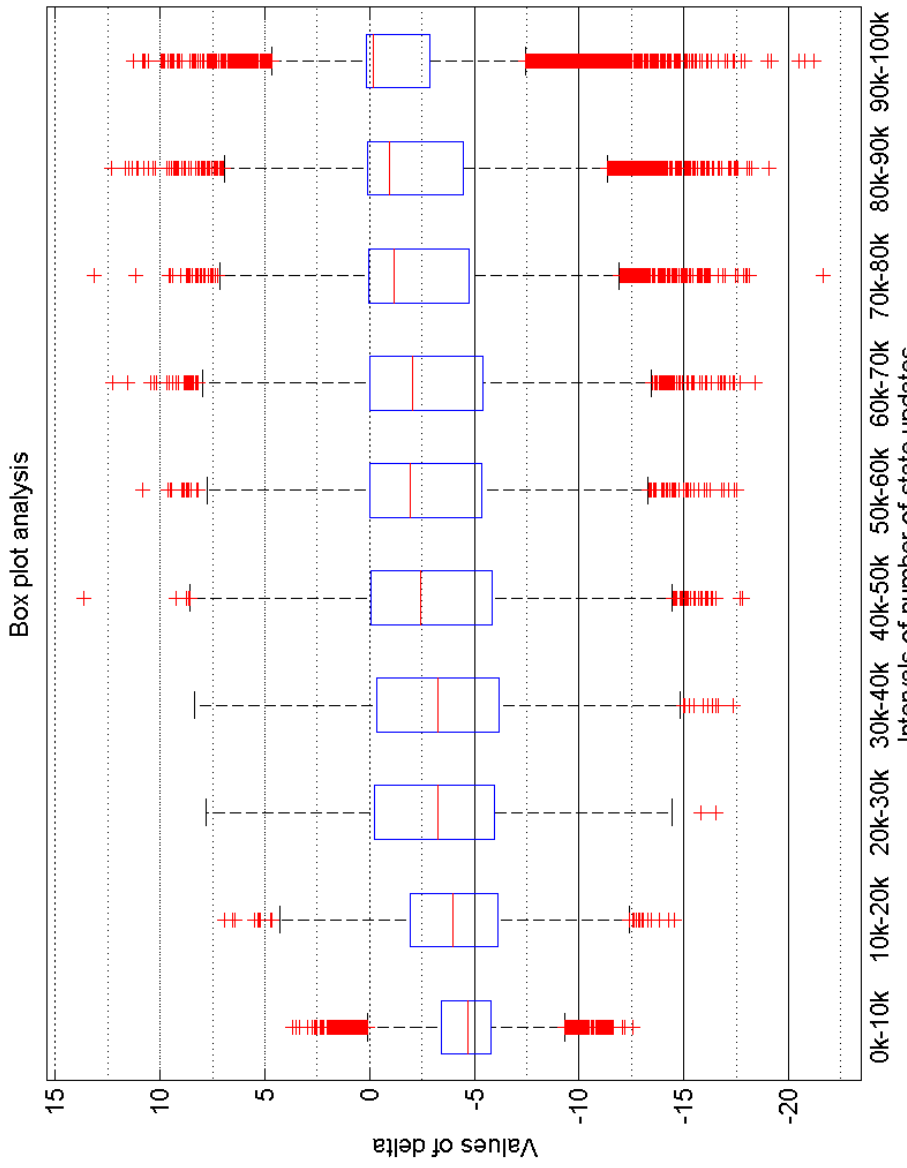

Figure 4.25: **A box-plot analysis of δ values.** The first 100.000 values of δ have been grouped in 10 groups of 10.000 values each (of course in chronological order), then we drew a box-plot for each group of values.

be investigated at the same time, because we expect many common patterns and problems.

Chapter 5

Conclusions

What we have presented in this thesis is the first research effort in a long term project just started at Intelligent Robotics Laboratory of Osaka University. The high level target of the project is to reproduce in a realistic and life-like way the interaction modes of living creatures, in particular humans, with pain. The goal of our work was to study the main building blocks of pain handling mechanisms in humans and to design and test a computational model that replicated them and mimicked their interaction modes as closely as possible.

First of all, we chose and defined the details of the prey and predator task, so that we could use it to test our work and so that in the future we will be able to transition our work towards robotics. The chosen task is significantly more complicated than the typical task chosen in computational neuroscience works, so we believe that results obtained in this context are very interesting for researchers working in this field.

After this, we chose the most suitable algorithm for the model-free module and we proposed two newly designed algorithms for the model-based and Pavlovian modules, so that the three algorithms all faithfully reproduce what has been observed in nature. We showed how all of them could reach good performance, so that we had three modules defined and working.

Subsequently, we defined an integration procedure that could replicate what has been observed in nature and that could show as many natural phenomena as possible. This allowed us to show how all possible combinations of these modules can reach good performance and even the integration of the three modules and this is by itself an unprecedented result in this field.

Then we performed some comparative analyses and we showed how the three modular system can give benefits in the first episodes and in an action extinction context, even if this comes at the price of worse performance on

the long run. This is still an open issue and one of the major goals for future work should be that of trying to keep the best possible performance even during long term convergence. This kind of analysis had already been performed in the past, but of course not on a three modular system since our work is the first proposal of such a model.

Finally, we showed some preliminary but promising results that outline how our model is able to easily explain a pathology such as OCD. Again, to the best of our knowledge this has never been done before in a computational way, so we believe this to be an important result that opens new application scenarios.

Probably, the part of our project that requires more attention in the future is the integration system: as we have explained in previous sections, at the very beginning of the training and during the phases of long term convergence it does not allow to have the best possible blend of the models and this limits the performance that we can obtain. Also, another very important limitation of our model is that threshold values depend on the map, because this at the current state limits the generalization properties and the applicability of our work. We believe that from now on a considerable effort should be invested in trying to investigate new ways to integrate the modules, that could keep and improve the benefits of the three modular system that we have shown previously and allow our model to be generalizable across different maps.

Another issue that we did not consider at all is the overall speed of our system, since at this stage our main concern was the design, the behavior and the interaction between modules. Anyway, given the particular purpose of this research project and the involved researchers' aims, the best solution is not the one that favors performance regardless of natural limits: following efforts should try to define a speed that is reasonable for living creatures and try to mimic that as faithfully as possible. This because we believe that the main purpose of this research project is to be the compliant with biological phenomena and not the sheer performance of the system.

An obvious prosecution of our work would be of course removing the hexagonal tiling and simulating the same processes with continuous space, then finally realizing all of this with real robots. One thing that will probably be fundamental in this transition is adding the robot dynamics to the state, since with real robots choices may depend also on the relative speeds and rotations of prey and predator.

After this, there are several things that may be pursued: the integration and testing of more sophisticated mechanisms, testing in different scenarios, new real world applications, pairing these computational experiments with

psychological experiments and so on. If we look at the big picture there are several things that can be done within this research project, but we believe the first steps we moved brought important contributions to the field and a considerable improvement with respect to previous works.

Bibliography

Balleine, B. W., Dickinson, A., April 1998. Goal-directed instrumental action: contingency and incentive learning and their cortical substrates. Neuropharmacology 37 (4-5), 407–419.

Barto, A. G., Sutton, R. S., Anderson, C. W., September/October 1983. Neuronlike adaptive elements that can solve difficult learning control problems. IEEE transactions on systems, man, and cybernetics 13 (5), 835–846.

Bonarini, A., Winter/Spring 1997. Anytime learning and adaptation of structured fuzzy behaviors. Adaptive Behavior Journal - Special issue on environment structure and behavior 5 (3-4), 281–315.

Carter, R. M., O'Doherty, J. P., Seymour, B., Koch, C., Dolan, R. J., February 2006. Contingency awareness in human aversive conditioning involves the middle frontal gyrus. NeuroImage 29 (3), 1007–1012.

Dayan, P., Niv, Y., Seymour, B., Daw, N. D., October 2006. The misbehavior of value and the discipline of the will. Neural Networks 19 (8), 1153–1160.

Dickinson, A., Balleine, B. W., July 2002. Stevens' handbook of experimental psychology (chapter title: The role of learning in the operation of motivational ssystem). John Wiley & Sons, Inc., Ch. 12, pp. 497–533.

Fanselow, M. S., December 1994. Neural organization of the defensive behavior system responsible for fear. Psychonomic Bulletin & Review 1 (4), 429–438.

Fendt, M., Fanselow, M. S., May 1999. The neuroanatomical and neurochemical basis of conditioned fear. Neuroscience and Biobehavioral Reviews 23 (5), 743–760.

Gillan, C. M., Morein-Zamir, S., Urcelay, G. P., Sule, A., Voon, V., Apergis-Schoute, A. M., Fineberg, N. A., Sahakian, B. J., Robbins, T. W., April 2014. Enhanced avoidance habits in obsessive–compulsive disorder. Biological Psychiatry 75 (8), 631–638.

Gillan, C. M., Otto, A. R., Phelps, E. A., Daw, N. D., March 2015. Model-based learning protects against forming habits. (Submitted to: Cognitive, Affective, & Behavioral Neuroscience).

Gillan, C. M., Papmeyer, M., Morein-Zamir, S., Sahakian, B. J., Fineberg, N. A., Robbins, T. W., de Wit, S., July 2011. Disruption in the balance between goal-directed behavior and habit learning in obsessive–compulsive disorder. The American Journal of Psychiatry 168 (7), 718–726.

Gillan, C. M., Robbins, T. W., November 2014. Goal-directed learning and obsessive–compulsive disorder. Philosophical Transactions of the Royal Society of London B: Biological Sciences 369 (1655).

Glascher, J., Daw, N. D., Dayan, P., O'Doherty, J. P., May 2010. States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. Neuron 66 (4), 585–595.

Koechlin, E., Ody, C., Kouneiher, F., November 2003. The architecture of cognitive control in the human prefrontal cortex. Science 302 (5648), 1181–1185.

Lee, S. W., Shimojo, S., O'Doherty, J. P., February 2014. Neural computations underlying arbitration between model-based and model-free learning. Neuron 81 (3), 687–699.

Mobbs, D., Hassabis, D., Seymour, B., Marchant, J. L., Weiskopf, N., Dolan, R. J., Frith, C. D., August 2009a. Choking on the money: reward-based performance decrements are associated with midbrain activity. Psychological Science 20 (8), 955–962.

Mobbs, D., Marchant, J. L., Hassabis, D., Seymour, B., Tan, G., Gray, M., Petrovic, P., Dolan, R. J., Frith, C. D., September 2009b. From threat to fear: the neural organization of defensive fear systems in humans. The journal of neuroscience: the official journal of the Society for Neuroscience 29 (39), 12236–12243.

Nishio, S., Ishiguro, H., Hagita, N., June 2007. Humanoid robots, new developments (chapter title: Geminoid: teleoperated android of an existing person). I-Tech Education and Publishing, Ch. 20, pp. 343–352.

Pessiglione, M., Seymour, B., Flandin, G., Dolan, R. J., Frith, C. D., August 2006. Dopamine-dependent prediction errors underpin reward-seeking behaviour in humans. Nature 442 (7106), 1042–1045.

Schultz, W., Dayan, P., Montague, P. R., March 1997. A neural substrate of prediction and reward. Science 275 (5306), 1593–1599.

Seymour, B., Dayan, P., 2009. Neuroeconomics: Decision-making and the brain (chapter title: Values and actions in aversion). New York, NY: Academic Press, Ch. 12, pp. 175–192.

Seymour, B., Singer, T., Dolan, R., April 2007. The neurobiology of punishment. Nature Reviews Neuroscience 8 (4), 300–311.

Sumioka, H., Nishio, S., Minato, T., Yamazaki, R., Ishiguro, H., December 2014. Minimal human design approach for sonzai-kan media: investigation of a feeling of human presence. Cognitive computation 6 (4), 760–774.

Sutton, R. S., 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings of the seventh international conference (1990) on machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 216–224.

Sutton, R. S., July 1991. Dyna, an integrated architecture for learning, planning, and reacting. ACM SIGART Bulletin 2 (4), 160–163.

Sutton, R. S., Barto, A. G., March 1998. Reinforcement learning: an introduction. Adaptive Computation and Machine Learning series. A Bradford Book.

Watkins, C. J. C. H., 1989. Learning from delayed rewards. Ph.D. thesis, King's College.

Witten, I. H., August 1977. An adaptive optimal controller for discrete-time markov environments. Information and Control 34 (4), 286–295.

List of Figures

List of Tables