CAIO LUCAS DOS SANTOS SOUZA

VIRTUAL INTELLIGENT AGENTS

IMPA - VISGRAF LAB

PHD THESIS, IMPA - VISGRAF LAB

ADVISOR: Luiz Velho

Abstract

Virtual Intelligent Agents

by Caio Lucas dos Santos Souza

In recent years, advances in computer graphics have brought an abundance of applications with superb virtual characters that are closing the visual gap of the uncanny valley. Machine learning has also experienced a breakthrough with new deep learning techniques. With today's computational power allowing graphics and artificial intelligence in real-time, virtual intelligent agents combining these two features are already at the front doorstep. Although intelligence can be perceived in many distinct ways, here we address the behavioral animation of these intelligent agents. Moreover, we approach this problem using a hierarchical solution benefiting from traditional artificial intelligence and newer deep learning. In turn, we bring to life agents intended to get closer to Artificial Life instead of 'wooden' automation agents that do not take human perception into account.

Keywords: intelligent agents, behavioral animation, reinforcement learning

Contents

List of Figures

List of Tables

1
Introduction

Since the beginning, one of science's goals is to understand or explain phenomena of many kinds, natural, historical, social, and many others. The road to understanding, analyzing, and explaining phenomena through formulas also paved the way for synthesis. These formulas, allied with big calculators, the computers, could become algorithms that can simulate fluids, physics, lighting, and various other complex dynamics.

The synthesis of these complex dynamics led to advancements in many fields, improving the process of production, design, and safety in general. Yet, there was still the desire to synthesize intelligence or intelligent behaviors. Through time, *Artificial Intelligence* (AI) has handled this task in various ways. Traditional techniques require expertise and explicit formulations and thus have become a bottleneck in achieving anything close to human intelligence.

In this scenario, the advancements in neural networks and *Deep Learning* (DL) that allow them to learn and represent these unknown formulations are shifting the paradigm of analysis and synthesis into new unthinkable ways. *Reinforcement Learning* (RL) is one of the fields that found a revival with such new techniques allowing learning controllers capable of emulating intelligent decision behaviors (in limited contexts) with super-human performance.

While simulating complete "digital twins" of living beings in all their full glory is still far away, especially when we consider in terms of conscience, a significant amount of effort is being put into synthesizing these autonomous intelligent agents in narrow scopes. Here, we approach this problem by hierarchically conceptualizing intelligence and also exploiting the perceptual aspects of intelligence in an autonomous agent.

In the next sections, we delve deeper into detailing and motivating this problem and our perspective on designing these agents with today's technology.

1.1 Autonomous agents and artificial life

One of the critical take-ups on artificial life is how to simulate living beings' behavioral animation and intelligence traits. Such entities must show autonomy in their decisions and interactivity, responding

to stimuli on time.

In artificial intelligence, these capable entities are enclosed by autonomous agents who can sense or take inputs from the environment and perform output actions. Interestingly, this abstract definition can fit a variety of automated tools. In general, these automation agents use the same techniques as the intelligent agents mimicking life. Their main difference is our perception of intelligence, which is biased by their adaptability, specific behavioral details, and perceived naturalness.

These similarities in techniques of both automation and said intelligent agents reflect how the design choices of these agents can shift the user perception between an automated robot and an intelligent agent. While these design choices involve many areas, our focus is on the behavioral animation that gives 'life' to an agent.

Here, we approach the development of these agents in a virtual environment hierarchically. Figure 1.1 shows our three-level hierarchy abstraction, a few traditional AI and optimization techniques used to solve them, and the current trend of deep learning that can find a place in every level.

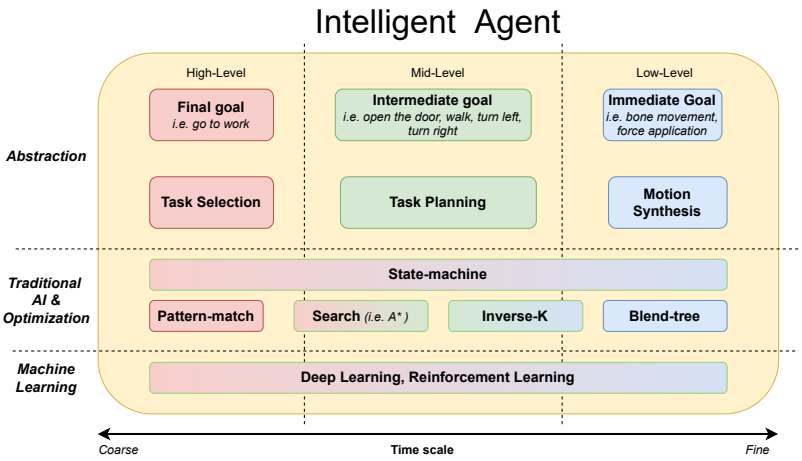

Figure 1.1: Diagram of an intelligent agent relative to its decision-making time scale. Initially, there is the abstraction of what kind of decisions are in each time scale; then, a few examples of commonly employed techniques of artificial intelligence and optimization. Lastly, machine learning can be used in different ways on any granularity.

Our hierarchy reflects the decision's time scale of the agent. In the lower level, we have the *Motion Synthesis* which is the motor skills controlling which animations or actions our agent can execute and manage in a finer time scale. Next, the mid-level *Task Planning* governs the medium time scale decisions sending a sequence of commands to the motion synthesis controller. These commands or decisions are translated into animated action to achieve a given goal. Finally, the top-level *Task Selection* abstracts our agent's will in selecting its long-term tasks and responses.

This modeling decomposes complex behavior dynamics into comprehensive pieces of movement set, decision-making, and long-term goals. Each piece has its caveats that are treated individually and then composed to achieve intelligent agents simulating artificial life.

1.2 Motivation

Despite the usage of virtual agents not being new, their past applications without or with limited machine learning techniques have often led to mechanical agents that could struggle with adaptability and fail their tasks with minimal sign of variation in their inputs [1].

Today, with the advancements in deep learning, graphics, and computational power, these agents can perform on a (super)human level in specific tasks. While its capability for automation is well known, if combined with designs that consider the human perception of intelligence and autonomous behavior, these agents could go much further and change our communication methods [2].

As a matter of fact, the human-machine interface has been relying on the keyboard and mouse as input interfaces for a long time. However, recently touch devices became commonplace (and brought new interface designs), and, nowadays, voice commands are spreading out with capable personal assistants. Yet, skilled agents with complex behaviors who interacts with the user would personify the interfaces on a new level. This shift in the human-machine communication interface would be an unprecedented breakthrough bringing qualitative changes.

Although there is still a long way to achieve intelligent agents performing well on multiple tasks, we believe that the hierarchical design grants these agents broader possibilities. Incorporating machine learning and hand-crafted controllers can circumvent various limitations of back-to-back learning on artistic control and fine-tuning, leading to more perceptually impactful agents.

Moreover, now is the right moment to explore these agents more profoundly. Provided with superb computer graphics, machine learning, and *Virtual Reality* (VR), such agents can significantly impact many areas, such as, entertainment, medical, and services, to cite a few. Not by accident, this area is recently receiving more and more attention from researchers and companies (*i.e.*, Meta, Nvidia, Epic Games, Unity 3D, and others).

1.3 Contribution

We highlight the contribution of this thesis on three points. The first is our proposed hierarchy for intelligent agents. While the idea of control levels is not new, we approach it in a specific way by decoupling motion synthesis from task planning, which we believe is a crucial step to give fine control over both animation and behavior of the agent. Additionally, the top level of the hierarchy (task selection) can get the best of both worlds from expert knowledge and machine learning without interference from the previous levels. These benefits express how such division allows for better control of each level.

Our second contribution is introducing and analyzing behavior types (command triggered, self-initiated, and interactive) that take into account their perceptual role and implementation details for

[1] Those cases can be well illustrated by virtual attendants on the telephone, that usually give birth to many jokes on their struggle.

[2] Here communication includes our means of interacting with machines and other humans mediated by machines.

agents mimicking artificial life. Understanding their impact on the user's perception is essential in achieving believable agents capable of bringing the qualitative changes we mentioned earlier.

Lastly, we deliver a proof-of-concept implementation using the hierarchy levels and behavior type abstraction of a dog agent in a virtual reality environment. Admittedly, the choice of a dog is not by chance; pets are much more inviting for a new user's experience and also allows us to focus on the behavioral animation such as the agent's expressiveness.

In fact, this playable scene is a glimpse of what these agents can accomplish, and more than that, it can work as a framework and starting point for further development. Although we implement every aspect discussed in this thesis, every one of them can go above and beyond, including better graphics, more and more complex behaviors, better motion synthesis, and many additions depending on the intended application.

With these contributions, we cover how intelligent agents can be approached with today's technology and substantiate on ideas and concepts that may also remain useful for future technologies.

1.4 Organization

The organization of this thesis is the following: Chapter 2 contains related works split into two parts, the first reporting on background-related works and the second describing specific works starting with motion synthesis and going up to complete systems abstracting artificial life agents.

Next, Chapter 3 contains the background for virtual characters and intelligent agents with special sections for (deep) reinforcement learning, which is the computing base for our agent.

Afterward, Chapter 4 is our system overview, going in-depth on the hierarchical control and design and also covering the general training procedures. The details of each type of behavior and their realizations on our implementation are in the subsequent Chapter 5.

Finally, Chapter 6 summarizes our development progress and presents our proof-of-concept application using the Unity 3D game engine where the user can interact with our dog agent in virtual reality.

We conclude with a brief overview of our vision for intelligent agents, followed by remarks on the work of this thesis and possible next steps, as usual.

2
Related Work

Developing agents capable of achieving defined goals have presented many challenges in both conceptual and practical aspects. Wooldridge und Jennings [1] tackled various abstract elements of intelligent agents and multiple approaches for implementing such agents with the techniques of their time.

With the advent of Deep Reinforcement Learning, the coverage and scope of applications using these agents significantly increased, especially in automation and robotics (Pierson und Gashler, 2017) [2]. This same framework can also significantly impact the field of (intelligent) virtual characters moving on to entertainment, education, and medical areas, to cite a few. Unlike automation, these applications incur additional challenges since they are subordinate to human perception. (Cavazza u. a., 2002; Vinayagamoorthy u. a., 2006; Zell u. a., 2019) [3] have studied how we perceive various aspects of virtual characters (specifically human characters), including appearance, animation, expression and how their careful development can influence our response for interactions.

For narrative context, exploiting these responses, such as emotions and empathy, is crucial for delivering the intended message and enhancing the user experience. In our case, of a dog character, most of its expression comes from its animation and behavior.

On the next sections, we divide the related works on *general background* and *application specific*. The first shortly covers the background techniques. The latter is split into three levels: works on motion synthesis for physics and animation-based agents, then works involving task planning, and finally, works with more complete agents covering complex or systematic behavior together with other features.

[1] Wooldridge und Jennings 1995 Wooldridge, Michael ; Jennings, Nicholas R.: Intelligent agents: Theory and practice. In: *The knowledge engineering review* 10 (1995), Nr. 2, S. 115–152

[2] Pierson und Gashler 2017 Pierson, Harry A. ; Gashler, Michael S.: Deep learning in robotics: a review of recent research. In: *Advanced Robotics* 31 (2017), Nr. 16, S. 821–835

[3] Cavazza u. a. 2002 Cavazza, Marc ; Charles, Fred ; Mead, Steven J.: Interacting with virtual characters in interactive storytelling. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, 2002, S. 318–325; Vinayagamoorthy u. a. 2006 Vinayagamoorthy, Vinoba ; Gillies, Marco ; Steed, Anthony ; Tanguy, Emmanuel ; Pan, Xueni ; Loscos, Céline ; Slater, Mel: Building expression into virtual characters. (2006); and Zell u. a. 2019 Zell, Eduard ; Zibrek, Katja ; McDonnell, Rachel: Perception of virtual characters. In: *ACM Siggraph 2019 Courses*. 2019, S. 1–17

2.1 *Background related*

In this section, we want to address the conceptual aspects of our background section, which handles its practical aspects. For that task, we believe the work *Is imitation learning the route to humanoid ?* [4] offers an interesting conceptualization of the various frameworks needed when developing AI that mimics partial abilities of living beings. Although their review focuses on humanoid robots, it can be very well abstracted for more general cases, such as our dog subject and

[4] Schaal 1999 Schaal, Stefan: Is imitation learning the route to humanoid robots? In: *Trends in cognitive sciences* 3 (1999), Nr. 6, S. 233–242

virtual environments that benefit from simplified dynamics compared to the real-world environments.

While (Schaal, 1999) dates from 1999, when current Deep Reinforcement Learning techniques did not exist, we can draw the links between his ideas to the present methods. The first point in this line would be the traditional reinforcement learning ideas and their complexity that is developed in-depth in (Sutton und Barto, 2018) [5]. Next, their definition of what imitation is and other valuable ideas; for instance, the "action level imitation" relates to the *Behavioral Cloning* technique[6]; the more far-fetched idea of facilitating the assimilation of a new behavior from previous learned (motor) skills, which could be seen very well as a curriculum[7] [8]. Finally, their discussion regarding learning representation, understanding task goals, and the processes involved to make imitation effective are especially demonstrated on the *Generative Adversarial Imitation Learning* (GAIL)[9] technique. Indeed, such a method would be unthinkable at that time, but despite the breakthrough of generative models, the concepts for effective imitation fit this approach very well. On another front, GAIL could also be seen as a proper way of approaching the curse of dimensionality when searching for behavior policies. It works as a way of "focusing on learning the interesting part of the state-action space" (similar to the exploration problem described by (Nair u. a., 2018) [10]), which also makes complicated problems more tractable by bootstrapping with imitation).

Now, compared with our current work, one could consider our hierarchy with separated motion synthesis and task planning as a realization of their earlier concepts for "movement primitive" and "task-level learning". Continuing in the hierarchy topic, (Simon, 1991) "Architecture of Complexity" speculates on complex systems organization. In short, they believe such arrangements, combining smaller stable subsystems, incur faster evolution than directly achieving a single complex stable system. Not by chance, a direct consequence of it is the amount of natural hierarchical systems, reinforcing our choice to split intelligent agents into *stable* sub-problems.

[5] Sutton und Barto 2018 SUTTON, Richard S. ; BARTO, Andrew G.: *Reinforcement learning: An introduction*. MIT press, 2018

[6] Torabi u. a. 2018 TORABI, Faraz ; WARNELL, Garrett ; STONE, Peter: Behavioral cloning from observation. In: *arXiv preprint arXiv:1805.01954* (2018)

[7] Bengio u. a. 2009 BENGIO, Yoshua ; LOURADOUR, Jérôme ; COLLOBERT, Ronan ; WESTON, Jason: Curriculum learning. In: *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, S. 41–48

[8] Also, with the same far-fetched thinking, one can imagine the Proximal Policy Optimization algorithm as a form of facilitation, with the clipped gradient updates not only solving stability issues but also as a way of adapting learned skills to new behaviors.

[9] Ho und Ermon 2016 HO, Jonathan ; ERMON, Stefano: Generative adversarial imitation learning. In: *Advances in neural information processing systems*, 2016, S. 4565–4573

[10] Nair u. a. 2018 NAIR, Ashvin ; MCGREW, Bob ; ANDRYCHOWICZ, Marcin ; ZAREMBA, Wojciech ; ABBEEL, Pieter: Overcoming exploration in reinforcement learning with demonstrations. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)* IEEE (Veranst.), 2018, S. 6292–6299

Applications and frameworks

With the temporal linking of ideas complete, it is vital to trace the relationship with the current scene on applications and frameworks that are growing in interest.

One area with increased interest is the *Collaborative Virtual Environments*, pushing the technological development of AI and graphics. The game engine Unreal Engine with its *MetaHumans* is a step-up on virtual humans that can be used both as *Non-Playable Characters* (NPCs) and avatars. Also, the Unity3D Engine, with its *ML-Agents* [11] package for creating intelligent agents is, likewise, bringing the ease of integrating machine learning with a game engine.

With other developments in *Natural Language Processing*, these factors, graphics and intelligent agents culminate in applications such as

[11] The Unity Machine Learning Agents Toolkit.

the metaverse (for instance, the *NVIDIA Omniverse*), intending to be a copy of the real world. Their possibilities are immeasurable, it can be used for simulating complex industrial problems (such as logistics of robots in a warehouse), collaborative development of 3D designs, interactive socio-educational environments, and entertainment purposes.

In this context, intelligent agents such as our dog can find their way to entertainment and socio-educational applications. For instance, pets on VR platforms such as *VR Chat* would be great content for their collections. Another facet is the usage of virtual agents for education[12] and therapy[13], offering the advantages of a safe and controlled environment and tuning their behaviors for specific purposes[14].

2.2 *Specific works*

After laying grounds on our perspective for the background and possible applications, we get back to the more usual form of related work with specific developments and breakthroughs in the field.

Early on, (Lin, 1992) [15] investigated RL usage with artificial neural networks for planning but was uncertain if it could scale for more complex tasks. The answer came with the initial works employing deep reinforcement learning in scale "Playing atari with deep reinforcement learning" (Mnih u. a., 2013) [16] and its development (Mnih u. a., 2015)[17]. It brought great attention to deep reinforcement learning by showing the possibilities of achieving human-level control on tasks with complex goals and taking high-dimensional raw inputs [18]. Various new learning algorithms and applications came to life following this breakthrough in the possibilities of reinforcement learning combined with deep learning. To cite a few, we have (Hessel u. a., 2018) as an attempt to combine various improvements into a single algorithm and with (Baker u. a., 2019) a multi-agent environment playing hide-and-seek.

In the next sections, we will address the works related to motion synthesis, task planning, and systematic behaviors in a bottom-up fashion. Starting with low-level controllers handling physics-based motion synthesis, up to works with systematic behaviors and agents emulating various aspects of living beings that are not only related to planning but also perception and interactivity.

Locomotion and Deep RL

Deep Reinforcement Learning (DRL) found great applicability on motion synthesis, on real-world application, such as end-to-end robotics (Levine u. a., 2016) [19] with raw visual inputs; and in virtual spaces for physics-based motion synthesis (Peng u. a., 2017), (Yu u. a., 2018), (Lee u. a., 2018) [20]. While these examples demonstrate how complex movements can be learned from trial-and-error, varying the gait types, terrain adaptation, and dexterity, they are mostly unsuitable for artistic purposes. The output movements lack the naturalness of

[13] (Rothbaum u. a., 1997),(North und North, 2016),(Emmelkamp und Meyerbröker, 2021)

[12] (Hedberg und Alexander, 1994), (Kavanagh u. a., 2017)

[14] (Velho und Alevato, 2022) does excellent technical coverage of *Digital Humans* including both realistic and stylized avatars and how they are being employed on different applications.

[15] Lin 1992 Lin, Long-Ji: Self-improving reactive agents based on reinforcement learning, planning and teaching. In: *Machine learning* 8 (1992), Nr. 3-4, S. 293–321

[16] Mnih u. a. 2013 Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Graves, Alex ; Antonoglou, Ioannis ; Wierstra, Daan ; Riedmiller, Martin: Playing atari with deep reinforcement learning. In: *arXiv preprint arXiv:1312.5602* (2013)

[17] Mnih u. a. 2015 Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Rusu, Andrei A. ; Veness, Joel ; Bellemare, Marc G. ; Graves, Alex ; Riedmiller, Martin ; Fidjeland, Andreas K. ; Ostrovski, Georg u. a.: Human-level control through deep reinforcement learning. In: *nature* 518 (2015), Nr. 7540, S. 529–533

[18] We believe that these successful applications are a great way of bringing attention and development to the field. It also motivates our own choice of having a concept application.

[19] Levine u. a. 2016 Levine, Sergey ; Finn, Chelsea ; Darrell, Trevor ; Abbeel, Pieter: End-to-end training of deep visuomotor policies. In: *The Journal of Machine Learning Research* 17 (2016), Nr. 1, S. 1334–1373

[20] Peng u. a. 2017 Peng, Xue B. ; Berseth, Glen ; Yin, KangKang ; Van De Panne, Michiel: Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. In: *ACM Transactions on Graphics (TOG)* 36 (2017), Nr. 4, S. 1–13; Yu u. a. 2018 Yu, Wenhao ; Turk, Greg ; Liu, C K.: Learning symmetric and low-energy locomotion. In: *ACM Transactions on Graphics (TOG)* 37 (2018), Nr. 4, S. 1–12; and Lee u. a. 2018 Lee, Seunghwan ; Yu, Ri ; Park, Jungnam ; Aanjaneya, Mridul ; Sifakis, Eftychios ; Lee, Jehee: Dexterous manipulation and control with volumetric muscles. In: *ACM Transactions on Graphics (TOG)* 37 (2018), Nr. 4, S. 1–13

the living beings they mimic. This issue arises from the simplified joint-torque models employed to simulate, for example, the human body. More complex models, for instance, (Nakada u. a., 2018), can achieve more natural movements at the cost of much more complex simulations involving replicas of bones and muscles. Developing such refined models for many different biotypes and animals for real-time and commodity hardware would be impracticable.

Another perspective relative to our hierarchical approach (separating motion synthesis from task planning) is that most of the works mentioned early make no distinction between task planning and motion synthesis, learning both together. This type of end-to-end learning is interesting for a specific task; however, for more general agents comprising many tasks, learning the motion synthesis every time for each task is not optimal.

Animation

In the previous subsection, we touched on the requirement of naturalness of the synthesized motions for media and entertainment applications. There is a plethora of works addressing this need, among them we can cite (Holden u. a., 2016), (Holden u. a., 2017), (Liu u. a., 2016) and (Zhang u. a., 2018) [21] that takes advantage of deep learning and motion capture to generate high-quality animations.

Their approach focuses solely on solving the motion synthesis part; such low-level controllers could be used for NPCs, agents, or playable characters in virtual environments. These controllers fit our hierarchy of control very well, and indeed, they could be used by our agent with the proper adaptations on the controlling interface and move-set. These high-quality animations would influence the user perception and immersion, with the only downside needing motion capture data when learning these low-level controllers.

The recent work of (Luo u. a., 2022) [22] approaches both motion synthesis and rendering of realistic animals using motion capture data, convolutional neural opacity radiance fields[23], and other newly developed techniques for motion synthesis. Interestingly, they also offer a VR demo of their animation and rendering, which could initially resemble our concept scene, however, missing the task planning component. Nevertheless, it could also be a choice for a low-level controller for motion synthesis integrated with realistic rendering inside our proposed hierarchy.

Works that make distinction between motion synthesis and task planning (controller)

Task planning is essential in many fields; traditionally, it has been solved with searching and optimization algorithms or hand-crafted heuristics. Today, it is a commonplace to find the usage of machine learning on heuristic controllers. In this subsection, we look at a few works using traditional algorithms and machine learning for task

[21] Holden u. a. 2016 HOLDEN, Daniel ; SAITO, Jun ; KOMURA, Taku: A deep learning framework for character motion synthesis and editing. In: *ACM Transactions on Graphics (TOG)* 35 (2016), Nr. 4, S. 1–11; Holden u. a. 2017 HOLDEN, Daniel ; KOMURA, Taku ; SAITO, Jun: Phase-functioned neural networks for character control. In: *ACM Transactions on Graphics (TOG)* 36 (2017), Nr. 4, S. 1–13; Liu u. a. 2016 LIU, Libin ; PANNE, Michiel Van D. ; YIN, KangKang: Guided learning of control graphs for physics-based characters. In: *ACM Transactions on Graphics (TOG)* 35 (2016), Nr. 3, S. 1–14; and Zhang u. a. 2018 ZHANG, He ; STARKE, Sebastian ; KOMURA, Taku ; SAITO, Jun: Mode-adaptive neural networks for quadruped motion control. In: *ACM Transactions on Graphics (TOG)* 37 (2018), Nr. 4, S. 1–11

[22] Luo u. a. 2022 LUO, Haimin ; XU, Teng ; JIANG, Yuheng ; ZHOU, Chenglin ; QIU, QIwei ; ZHANG, Yingliang ; YANG, Wei ; XU, Lan ; YU, Jingyi: Artemis: Articulated Neural Pets with Appearance and Motion Synthesis. In: *arXiv preprint arXiv:2202.05628* (2022)

[23] Luo u. a. 2021 LUO, H. ; CHEN, A. ; ZHANG, Q. ; PANG, B. ; WU, M. ; XU, L. ; YU, J.: Convolutional Neural Opacity Radiance Fields. In: *2021 IEEE International Conference on Computational Photography (ICCP)*. Los Alamitos, CA, USA : IEEE Computer Society, may 2021, S. 1–12. – URL https://doi.ieeecomputersociety.org/10.1109/ICCP51581.2021.9466273

planning. More specifically, we report on works related to motion synthesis and task planning that acknowledge their separation in any sense.

The first of them is (Levine u. a., 2011), involving path planning with a variation of A^* algorithm; next (Agrawal und van de Panne, 2016) and (Naderi u. a., 2017) use optimization for footstep and climbing planning, both synchronized with their motion synthesis modules. Lastly, the more recent (Ling u. a., 2020) uses machine learning for its motion variational autoencoder controller, with the exciting fact of classifying its approach as a "model-then-control" as a form of separating the motion synthesis from task planning. With such distinction, it is easier to understand the particular challenges of animation and planning, even though their solution for each problem works together and has some dependencies.

Here, we approach both problems similarly. While our motion synthesis module is more straightforward and based on traditional animation techniques, the planning module explores recent advances in deep reinforcement learning on multiple tasks.

Works focusing on complex/systemic behavior

The last piece of the related works puzzle is the broader view of artificial life and intelligence involving everything from motion synthesis to systemic behavior, learning, and perception.

The work of (Terzopoulos, 1999) presents an interesting pyramid for modeling artificial life. Begining with the *geometry*, *kinematic*, *physical*, *behavioral* and ending with the *cognitive* level. Here, our agent modeling touches on a few aspects from kinematics to the behavioral level using today's techniques. Nevertheless, it is crucial to understand how these parts, together with many others, function on a broader scale. (Terzopoulos, 1999) does an excellent job reviewing various approaches for simulating life, including plants, animals, and humans; and how its behavior (as individual and group), expressiveness, and autonomous properties are interdisciplinary and, in his words *"offers a wealth of provocative research problems and great commercial potential"*.

We certainly are already grasping this potential, as we have seen in the previous section 2.1. Yet, even with all the fantastic today's applications and technology, there is still a long path to conquer the *cognitive* top of the pyramid.

One work that approaches the entire stack of artificial life in the form of a virtual (human) agent is (Kuffner Jr, 2000). While the techniques employed are not similar to ours, we can draw similarities to our concepts in the big picture. The first of these resemblances is the idea of different levels of control with high-level planning and motion synthesis. Even though their split point is very different from ours, we believe this type of understanding is vital for developing complex agents. His work uses the concept of task-level animation, with the motion synthesis also containing path planning (using search and

optimization). On the other hand, his high level is scripted using concepts such as "moveTo" or "getObject". Differently, we use motion synthesis as movement primitives that the task planner controls to achieve the required goal. Our task planner would include both the path planning and the high-level concepts learned in a per-task fashion. Although our motion synthesis is constrained to general primitives, they are guaranteed to have desired naturalness and artistic expression. On the planner side, the effort of scripting a behavior is converted to developing learning environments, gaining the real-time reactiveness of one-step planning [24]. Lastly, (Kuffner Jr, 2000) investigates on the agent sensing aspect. His focus is on synthetic visual perception with an unlit scene approach that resembles the idea of visual segmentation, where each object has a specific color. Our ray-cast sensor works similarly in practice, segmenting objects by tag instead of color. Their visual sensory system works together with a memory system for navigation purposes. Although our agent navigates well for its tasks without memory, it would be interesting to explore its usage in future works, such as remembering past interactions with players or creating an emotional memory. These steps would effectively move toward the cognition level we cited before.

As a concluding remark for this section, while these works, (Terzopoulos, 1999) and (Kuffner Jr, 2000), do not contemplate the new machine learning techniques for intelligent agents, they are great for visualizing the complete stack for such agents, also highlighting the modeling similarities and adaptation for new designs.

[24] Interestingly, (Kuffner Jr, 2000) also has an autonomy versus interactivity graph, where the type of agents for VR and games (which we call here virtual agents for media) need to achieve high levels in both requirements.

3
Background

In this chapter, we present the fundamental concepts behind intelligent agents. First, we begin with an introduction of *virtual characters*. Tthe next section 3.2 cast virtual characters as a specialized type of intelligent agent which avoids the handling of real-world sensing and acting, followed by the mathematical framework for these agents, reinforcement learning, in section 3.3. Finally, the last section 3.4 goes deeper into the trending topics of *deep reinforcement learning* and its application in *locomotion*, which are the basis for various state-of-the-art agents.

3.1 Virtual characters, applications and requisites

Virtual characters can be imagined as non-material entities that may mimic behaviors of living beings or any kind of creature [1] . Interestingly, the focus of these entities lies in their behavior. Their presentation or appearance depends on the context they are inserted and is more related to the expressiveness and emphatic aspects of a virtual character. Virtual characters differ significantly from *avatars* [2] , where the behavior depends on who is controlling the it.

[1] Although most characters present them-selves in the form of humans, animals, or plants, in a virtual environment a character can take any form, limited only by creativity.

[2] A representation of a human in a virtual environment, be it a personalized character, a nickname, or anything that represents a person in such environment.

The application and intend of such characters can vary significantly; they can play different roles from storytelling contexts (Interactive media) to automating tasks (attendants and assistants). One way of analyzing the characteristics of a virtual character is about their *level of interaction* and *presentation*.

The level of interaction can be imagined as the complexity of their behavior; for example, a simple virtual guide could interact in a fixed way giving information about a place or monument. These characters can look attractive in their first interaction, but their static behavior will certainly seem boring and repetitive on a second try. On the other hand, a personal assistant has a complex behavior of trying to answer any question. Even if the question is the same or the set of possible questions is limited, it cannot have a fixed answer to, for example, "tell me a joke" or "what's the weather today". Here we could compare the guide example to a static linking place to information. At the same time, the personal assistant does a dynamic linking of a question to many possible answers based on some pre-defined reasoning.

Next, we call *presentation* the form these characters take and their

means of interaction. Their form is related to their features, such as their visual appearance or voice, and other more subtle features such as language and movement style. It is also noteworthy that artistic applications can use features as means of expression, for example, the movement style in dance.

The mix of features and means of interaction is directly related to the intention and requisites. Automation applications (*i.e.*, assistants) would cherish objectiveness and clarity, while artistic and entertainment applications would thrive for expressiveness. In the first case, if it were a voice assistant, it would require an easier to understand voice and direct and precise answers. Conversely, an artistic approach would account for the voice accent, tone, and more-fetched language. Those examples show how the final goal is the most important for the first one while the process is fundamental for other applications.

The reasoning of these virtual characters is often done by artificial intelligence, varying from traditional reactive techniques to modern machine learning tools. In this field, they are a subset of a broad abstraction: *intelligent agents* (which is the topic of the next section).

3.2 *Intelligent agents and machine learning*

Intelligent agents are one of the central points of study in the AI field. Although both ideas of "intelligence" and "agent" are not unique, we define an agent as an autonomous entity living in an environment taking sensory information and performing output actions. Wooldridge [3] gives a simple example of a thermostat, which can turn on or off the heating based on its sensors. This example correlates well with the ideas mentioned earlier, with the real world and how the agent is autonomously changing the *state* of its environment [4] through its sensory observations.

[3] Wooldridge 1999 WOOLDRIDGE, Michael: Intelligent agents. In: *Multiagent systems* 6 (1999)

[4] The environment of an agent, whether it is located in the physical space or virtual space of a computer, is defined by the extent of its sensors and actuators.

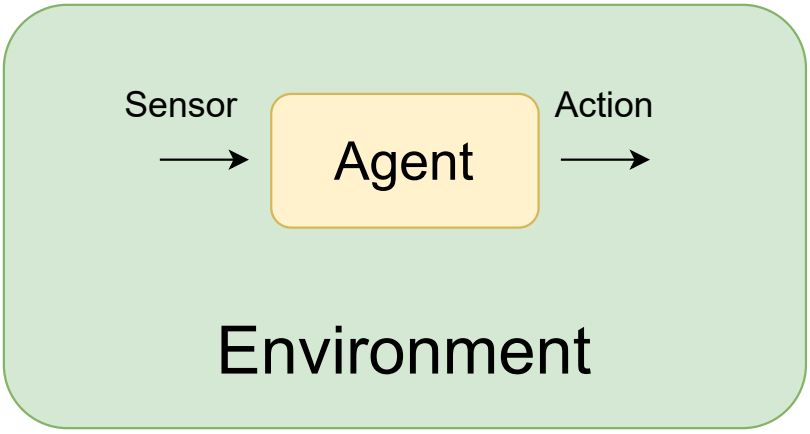

Figure 3.1: Agent abstraction

The agent abstraction can fit a variety of real and virtual examples, from simple alarms to complex robots, personal assistants, and virtual characters. Nevertheless, the idea of "intelligence" can be associated with various factors such as retaining knowledge, reasoning based

on both current and previous information, acting towards a goal, etc. The previous example of a thermostat could fit all of those requisites (retain the knowledge of what temperature is low/high, reason whether it should turn on/off based on information from its sensor, and act towards the goal of keeping the temperature stable). However, it would be hard to convince a person that a thermostat is an intelligent actor.

Using those concepts to classify an agent as intelligent or not can be a lot more tricky than it seems. Let's take the example of a vacuum cleaner robot. It has the same properties aforementioned, but it can also map its environment, remember where it has already been cleaned, and adapt if someone moves it to another place. This *flexibility* is usually perceived as evidence of intelligence. It is also fascinating how perception can be fooled by exploiting the context and protocol of specific interactions. For example, when booking a hotel room or ordering food, obeying a fixed protocol of introducing yourself, informing the options, and confirming the order would undoubtedly meet the expectation of many clients. In contrast, any slight deviation of the expected protocol would completely break the interaction.

While exploring only the context and protocol may not be enough to make an intelligent agent, a flexible agent with such knowledge can be much more impactful or simpler to model. The lack of exploring those *domain knowledge* when developing an agent can significantly increase the computational power and difficulty of implementing the agent.

Successfully traditional AI approaches, such as Chess Deep Blue [5] for example, make extensive use of domain knowledge and search techniques matching and surpassing human level. Still, many problems lack that explicit knowledge or are not computationally viable to search for a solution [6] [7]. , or decision (reasoning) in case of intelligent agents, motivating the usage of *Machine Learning*.

An early definition of machine learning from *Tom M. Mitchel* is:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

Today, using ML has become a trend in all areas, given its power of intrinsically learning the "intellectual mechanisms" needed for solving many tasks. This recent growth has been driven mainly by three factors: The advancements in artificial neural networks with DL[8] and the increase in both data availability and computational power.

One task that significantly improved with the advances of DL is image classification [9] . Traditional computer vision would use *SIFT*[10] to extract features from images while DL learns how to extract the features with the convolutional layers of an artificial neural network.

While ML may seem like a possible solution for all tasks, it still relies on human expertise [11]. Also, other challenges are present in the

[5] Hsu 1999 Hsu, Feng-hsiung: IBM's deep blue chess grandmaster chips. In: *IEEE micro* 19 (1999), Nr. 2, S. 70–81

[6] McCarthy says *"Whenever people do better than computers on some task or computers use a lot of computation to do as well as people, this demonstrates that the program designers lack understanding of the intellectual mechanisms required to do the task efficiently."* Which is an interesting statement (even if not valid for all tasks), In many fields, our understanding of these mechanisms is limited, leading to solutions using brute-force approaches and extensive searches.

[7] An example of a problem that is practically impossible to be solved only by search is the 3×3 Rubik's Cube

[8] Bengio u. a. 2007 Bengio, Yoshua ; LeCun, Yann u. a.: Scaling learning algorithms towards AI. In: *Large-scale kernel machines* 34 (2007), Nr. 5, S. 1–41; and LeCun u. a. 2015 LeCun, Yann ; Bengio, Yoshua ; Hinton, Geoffrey: Deep learning. In: *Nature* 521 (2015), Nr. 7553, S. 436–444

[9] The yearly competition on image classification using *ImageNet* dataset (Deng u. a., 2009) had a score of 50% in 2011 using SIFT, jumped to 63% in 2013 with *AlexNet* (Krizhevsky u. a., 2012), and today (2021) sits at 90% (Dai u. a., 2021)

[10] Lowe 1999 Lowe, David G.: Object recognition from local scale-invariant features. In: *Proceedings of the seventh IEEE international conference on computer vision* Bd. 2 Ieee (Veranst.), 1999, S. 1150–1157; and Lowe 2004 Lowe, David G.: Distinctive image features from scale-invariant keypoints. In: *International journal of computer vision* 60 (2004), Nr. 2, S. 91–110

[11] For example, the understanding of convolutional layers working as learned spatially invariant filters is the reason for its success on images.

field: The learning progress is often much slower than a human and requires a tremendous amount of data (for example, ImageNet has about 14 million images); Overfitting (the inability to generalize), *i.e.*, when trying to classify cat images, the model may work correctly for the images used when learning but not for new images. One could argue that it didn't grasp the concept of what a cat looks like but memorized the cat images seen; measuring progress for some tasks may not be straightforward, etc.

Bringing the learning to the agent context, it makes a lot of sense by looping through sensing of the environment, choosing an action, evaluating and improving its decision-making process based on the outcome. However, for agents acting based on a goal arises the problem of quantifying each action's contribution to the goal. For example, it is easy to detect who won a chess game, but assigning individual values for each move that led to winning is not easy.

Reinforcement Learning tackles this fundamental problem known as *the credit assignment problem* and is the main focus of the next section.

3.3 *Reinforcement Learning*

Reinforcement learning is one of the three paradigms of machine learning[12] and models the problem of goal-directed agents.

Russell und Norvig[13] describes it interestingly, as:

"Imagine playing a new game whose rules you don't know; after a hundred or so moves, your opponent announces, 'You lose'. This is reinforcement learning in a nutshell."

We can link this example to the main concepts of reinforcement learning. First, you want to learn a *policy*, a way of behaving, that best achieves the goal; then you have the *reward signal*, which is a response to the agent's performance; for example, it could be $+1$ for winning and 0 otherwise.

Here the concept of the *best policy* is related to the sequence of actions maximizing the expected reward. Note that the expectation implies a dynamic environment where taking the same action on a given state can lead to different rewards and next-state. The concept of a *delayed reward* problem is also evident: while each action contributes to the end goal, one may not know the effects of the actual action until later stages.

In practice, a common approach is *online* learning (know as on-policy), beginning with an initial policy to interact with the environment and improving it after each new step through a *value function*. This function $V(s)$ represents the expected reward beginning from a state s and is learned by propagating the experienced rewards across the states. In the same way, the policy chooses an action based on the values $V(s)$, for example, greedily taking the action that leads to the state with the higher $V(s)$. A challenge of such approaches is related to the dilemma of exploring and exploiting. Exploiting your knowledge and taking only the known best action can lead to the agent being stuck on sub-optimal behaviors, while exploring too

[12] The other two are supervised and unsupervised learning. Supervised learning works on labeled data for pairing tasks (*i.e.*, classification), usually with the output pair labeled by an expert. In contrast, unsupervised learning tries to find hidden structures on unlabeled data, with a simple example being clustering tasks.

[13] Russell und Norvig 2002 RUSSELL, Stuart ; NORVIG, Peter: Artificial intelligence: a modern approach. (2002)

much may never converge to an optimal.

Together with the application requisites, the challenges mentioned above can significantly hinder learning. In this regard, virtual agents have many advantages; they don't need to handle typical real-world signals; their experience is simulated in a computer, which is usually faster, cheaper, and safer; their sensing can use information that is not available in the real-world, etc. In contrast, simulating real-world agents or transferring the learning from a virtual copy of an agent to its real twin is very hard, comprising a full research area.

Next we will introduce the mathematical framework for agents and learning.

Markov Decision Process

Markov Decision Process (MDP) mathematically models the problem of reinforcement learning. It is expressed as a four-tuple (S, A, T, R), where:

- S - *State* - is the set of states
- A - *Action* - is the set of actions
- R - *Reward* - is the set of rewards, $r \in R \subset \mathbb{R}$.
- T - *Transition*- is a function $T : S \times A \times R \times S \to [0, 1]$, that returns $T(s, a, r, s')$ the probability of transitioning to state s' with reward r given that the action a was taken on state s.

Usually the dynamics of an agent and its environment happens in the following way; in timestep t_i the environment is in state s_i, the agent takes the action a_i and in the next time t_{i+1} the agent receives the reward r_{i+1} and the environment transits to state s_{i+1} according with to distribution T.

This framework for representing the reinforcement learning problem has various interesting properties. First, as the name suggests, it holds the Markov property of depending only on the previous pair (s_t, a_t), implying that all information available in s is sufficient for the process [14] . Next, while t denotes sequential events in time, its intervals have no limitations.

Lastly, the elements of S, A, and R are certainly represented as tuples in a computer. Still, the tricky part is how to encode their information and meaning in a good way for the learning, which is currently more of an art than the result of a mathematical analysis, affecting the agent's performance and computational viability.

The reward representation is also crucial, as it is the way of telling the agent's goal, but it is not always easy to define a good reward. A few examples of such difficulty are:

- When the reward is *sparse*, the agent hardly ever may achieve a reward, and its learning gets stuck. However, adding rewards that are not precisely the goal can lead to unpredictable side effects,

[14] Note that the state information can be anything used for decision making; for example, when navigating in a room, it may be essential to know the distance to the obstacles, but not necessarily the color or appearance of the obstacles. Other than that, it could be helpful to have a room map, but it may not be available.

a problem known as *reward exploitation* as the learning happens blindly based on maximizing the reward.

- When the reward tries to shape how to do a task; Take as an example an agent with the goal of moving from point A to B. Now imagine that you want the agent to complete its goal without colliding with walls. Adding a negative reward each time agent hits a wall seems reasonable, but in fact, you are giving two goals "go from A to B" and "avoid walls". They can act against each other depending on the weight of each reward and the given environment complexity. The learned behavior may completely ignore the "avoid walls" because reaching point "B" alone maximizes the reward; or doing the opposite, not going to B but avoiding walls at any cost. It is not straightforward to balance many goals to shape how a task is done.

Policy, experience and learning

We saw earlier that MDP models the reinforcement learning problem, but not yet how to solve and represent its solution. Here we begin with the agent's behavior abstraction: the *policy*.

A policy $\pi : S \times A \to [0, 1]$ is, for each fixed state, a probability distribution defined on the set of action, thus probability of taking action a in the state s is $\pi(s, a)$. Collecting experience is commonly done by following a policy π, for example a uniform policy, leading to the *trajectory* τ recording the sequence of states, actions and rewards $(s_0, a_0, r_1, s_1, a_1, r_2, s_2, \cdots, s_n, a_n, r_{n+1}, s_{n+1})$, that are used to learn an optimal policy π_* maximizing the expected reward [15] .

[15] Learning the optimal policy is the ideal objective, but in practice what is learned is a "good enough" policy.

The subsequent concepts are needed to set up the learning process (note that we will change our lowercase notation of the *set element* to the uppercase *random variable*):

1. **Discounted total reward** - G_t - The total reward of a trajectory τ starting from time t, using the discount factor $\gamma \in [0, 1]$, is

$$G_t = \sum_{i=0}^{|\tau|-t} \gamma^i R_{t+i+1}$$

   Here, the discount factor weights the importance of future rewards. When $\gamma < 1$, the further in time the reward is, the less important it becomes [16] . This parameter balances if the agent should seek instant or late rewards when learning.

2. **Value function** - $V(s)$ - The value function expresses the expected outcome of the discounted total reward when starting from state s and following a policy π [17] , and is given by:

$$V(s) = \mathbb{E}\left[G_t | S_t = s\right]$$

   thus,

$$V(s) = \mathbb{E}\left[R_{t+1} + \gamma V(S_{t+1}) | S_t = s\right]$$

[16] Interestingly, when $\gamma < 1$, the infinite summation of the γ^i's is a geometric series equal to $\frac{1}{1-\gamma}$ resulting in $G_t \leq \max R_i$

[17] It can also be written as $V_\pi(s)$, but we prefer to omit it when not referring to a specific policy.

where S_{t+1} is entirely defined by the transition function T,

$$V(s) = \mathbb{E}[R_{t+1}] + \gamma \sum_{s' \in S} \hat{T}_{s,s'} V(s')$$

with $\hat{T}_{s,s'} = \sum_{r \in R} \sum_{a \in A} \pi(s,a) T(s,a,r,s')$.

3. **Action-value function** - $Q(s,a)$ - The action-value function is similar to the value function but indicates the expected outcome when choosing a specific action a in the state s. It is given by:

$$Q(s,a) = \mathbb{E}\left[G_t | S_t = s, A_t = a\right]$$

thus, using $V(s)$

$$Q(s,a) = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right] + \sum_{s' \in S} \sum_{r \in R} T(s,a,r,s') V(s')$$

Unsurprisingly, $V(s) = \sum_{a' \in A} \pi(s,a') Q(s,a')$, which in turn, allows Q to be written as:

$$Q(s,a) = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right] + \sum_{s' \in S} \sum_{r \in R} T(s,a,r,s') \sum_{a' \in A} \pi(s',a') Q(s',a')$$

.

It is interesting that both V and Q satisfy recursive relations and depend on the explicit knowledge of the environment dynamics: the T function. When the environment dynamics is known, the optimal V, the one that maximizes G_t, can be learned through *value iteration* [18] , with its update rule given by:

[18] Although V is not a policy, it generates a deterministic policy by looking at the possible next-state from s (determined by T) and weighting the action with the best outcome $Q(s,a)$.

$$\begin{aligned} V(s) &\leftarrow \max_{a \in A} \left(\mathbb{E}\left[R_{t+1} + \gamma V(S_{t+1}) | S_t = s, A_t = a\right]\right) \\ &\leftarrow \max_{a \in A} \left(\sum_{r \in R} \sum_{s' \in S} T(s,a,r,s') \left(r + \gamma V(s')\right) \right) \end{aligned}$$

When in the form of equality, this update rule is known as the *Bellman equation*. A few other remarks are noteworthy; first, recursively using $V(s')$ to update $V(s)$ is a technique called *Dynamic Programming*, which is very useful and faster than common search techniques. Still, it must process all states many times to converge, which may not be doable for many problems. The drawbacks of the need for the transition function T [19] limits the scope of methods such as value iteration.

[19] Methods that needs the transition function explicitly are known as *model dependent*.

Temporal Differences

The last subsection showed how the value iteration could learn when the environment transition function T is known. However, in many cases T is not known, and *model-free* methods that can learn directly from experience are needed [20] . Figure 3.2 shows a diagram of the idea for such learning, which resembles the human notion of learning: experiencing and adapting iteratively to new outcomes and situations.

[20] A few methods try to approximate the transition function and them do the learning afterwards. (**?**) surveys these *model-base* approaches.

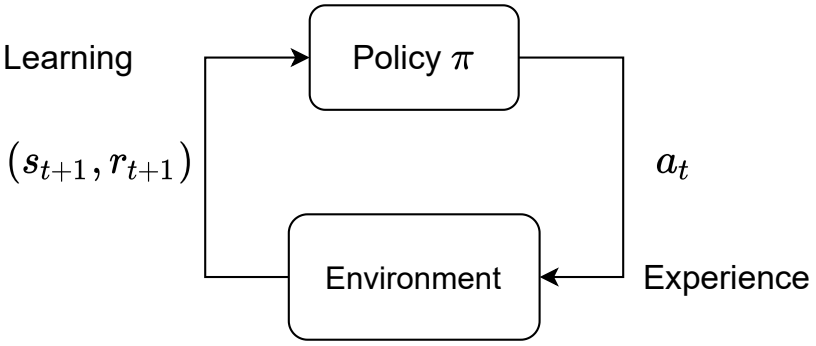

Figure 3.2: Agent learning dynamics.

The breakthrough for learning the value function directly from experiences came with *Temporal Differences*[21]. The learning happens at each new timestep based on the difference between the experiences acquired and the current knowledge of the state's value. The update rule for V can be written as:

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha(r_s + \gamma V_\pi(s') - V_\pi(s))$$

where r_s is the experienced reward acquired from a policy π, and α is the learning rate, usually $\alpha \gg 1$. Here, it is essential to note that the value function learned is relative to the policy used to acquire r_s, which may not be the optimal. Also, α is a parameter of choice mixing the old knowledge with the new experiences; one wants it to be the greater value possible for which this process converges [22] . Finally, what we want to achieve is the *optimal policy* π_*. V_π does not follow the optimal policy, and even if it did, it would not be possible to generate a policy from V without knowing T. Hence, other methods are necessary.

[21] Sutton 1988 SUTTON, Richard S.: Learning to predict by the methods of temporal differences. In: *Machine learning* 3 (1988), Nr. 1, S. 9–44

[22] If α is too big, convergence may not be achieved; conversely, if it is too small, the learning will be slower than necessary.

Q-Learning method

One of the methods addressing the learning of the optimal policy is the *Q-Learning*[23]. It uses the same idea of the temporal differences, but for the action-value with a small addition: it supposes that the best action was taken, leading to the optimal policy [24] . Its update rule is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s + \gamma \max_{a' \in A} Q(s',a') - Q(s,a))$$

Directly approximating Q allows for efficiently generating a policy, for example:

$$\pi(s,a) = \begin{cases} 1, & \text{if} \quad a = \arg\max_{a' in A} Q(s,a') \\ 0, & \text{otherwise.} \end{cases}$$

It is also noteworthy that the usage of the maximum Q in the formulation not only makes it converge to the optimal but also makes the Q-Learning an *off-policy* method not depending on the policy used for generating the experiences.

[23] Watkins 1989 WATKINS, Christopher John Cornish H.: Learning from delayed rewards. (1989)

[24] Of course, a few assumptions are required to guarantee the optimal property; α must be sufficiently small to converge. Also, in the limit, it must visit and update infinitely times the action-state pairs.

3.4 *Parametric functions & Deep Reinforcement Learning*

Until now, we delayed a significant point of reinforcement learning on purpose: How does one represent the learned knowledge of Q and V? For simple examples, with few states, it is not uncommon to use tables to store the values of Q or V, but doing so for problems with a massive amount of states or even infinitely many states is impractical.

In this regard, parametric functions are an excellent solution for approximating state-action functions using much fewer parameters than the number of states and actions. Next, we present a common approach to finding such approximation using least-squares error and gradient descent optimization.

Suppose that θ is the vector of parameters and

$$Q_\theta(s,a) = f(\theta, s, a)$$

First, we define the least squares error, with $q_{s,a}$ the target value for Q_θ

$$err_\theta = \frac{(q_{s,a} - Q_\theta(s,a))^2}{2}$$

Then, iteratively update θ with gradient descent rule

$$\theta \leftarrow \theta - \alpha \nabla_\theta err_\theta$$

where α is the learning rate of the gradient descent. Using the same machinery, a parametric version of the Q-Learning for each i-th parameter θ_i would be:

$$\theta_i \leftarrow \theta_i - \alpha(r_s + \gamma \max_{a' \in A} Q_\theta(s', a') - Q_\theta(s,a)) \frac{\partial Q_\theta(s,a)}{\partial \theta_i}$$

While the usage of parametric functions with reinforcement learning are not new, two of the main difficulties of using these models are the representation and generalization capacity:

- How good a family $f_\theta : \Omega_X \to \Omega_Y$ can approximate the pair samples (x_i, y_i), with $x_i \in X \subset \Omega_X$ and $y_i \in Y \subset \Omega_Y$, used in the fitting process, or the *representation capacity* of f_θ.
- How good a previously given f_θ that approximates (x_i, y_i) performs when approximating unseen samples (x'_i, y'_i), with $x'_i \in X' \subset \Omega_X$ if $y'_i \in Y' \subset \Omega_Y$, and $X \cap X' = \emptyset$ and $Y \cap Y' = \emptyset$, know as *generalization capacity* of f_θ.

In general, these two points are conflicting; too much representation capacity can negatively affect a models generalization capacity. Also, the lack of prior knowledge of which kind of model would meet both demands limited the success of early parametric models with Reinforcement Learning; only recently its applications experienced remarkable growth from leveraging Deep Learning[25] techniques [26]. Most of DL's success lies in improving the general capacity and generalization by using artificial neural networks parametric models (ANN,

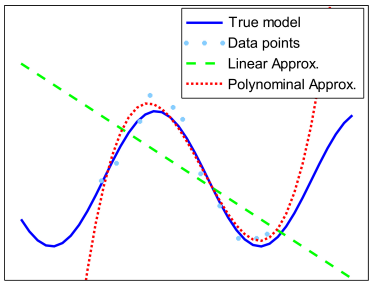

Figure 3.3: An excellent example for understanding capacity and generalization is comparing it with simple function fitting. Here the true model is the blue curve ($w_1 \sin(w_2 x)$); the light blue points are the available measures of the actual model. Trying to approximate such points with a linear model, in green ($w_1 x + w_2$), exemplifies the lack of *capacity* as no choice of $\bar{w}$'s can approximate it well. Conversely, the larger model in red ($w_1 x^3 + w_2 x^2 + w_3 x + w_4$) matches well the model inside the range of the sample points but fails to *generalize* for outside their bounds. Note that the quality and amount of samples also play a crucial role in achieving a good fit.

[25] Bengio u. a. 2007 Bengio, Yoshua ; LeCun, Yann u. a.: Scaling learning algorithms towards AI. In: *Large-scale kernel machines* 34 (2007), Nr. 5, S. 1–41; and LeCun u. a. 2015 LeCun, Yann ; Bengio, Yoshua ; Hinton, Geoffrey: Deep learning. In: *Nature* 521 (2015), Nr. 7553, S. 436–444

[26] On a side note, the θ's fitting for Q_θ (or any parametric function) is a supervised learning task, and hence, can also enjoy the gains and advancements in that field.

also known as *universal function approximators*). One of the first applications in reinforcement learning taking advantage of DL was *Playing atari with deep reinforcement learning*[27] from raw images using Deep Q-Learning and attaining superhuman performance [28].

[27] Mnih u. a. 2013 MNIH, Volodymyr ; KAVUKCUOGLU, Koray ; SILVER, David ; GRAVES, Alex ; ANTONOGLOU, Ioannis ; WIERSTRA, Daan ; RIEDMILLER, Martin: Playing atari with deep reinforcement learning. In: *arXiv preprint arXiv:1312.5602* (2013)

[28] Certainly, the choice of using images as input were influenced by the performance of DL and convolutional layers for image classification tasks (Krizhevsky u. a., 2012)

Policy Approximation

We have seen previously that the goal of reinforcement learning is learning an optimal policy. Yet, up to this point, we only presented methods that learn state-action values and afterward generate a policy from them.

A reasonable question, equipped with the parametric function approximation, is: why not directly learn a policy? This question leads to methods known as *policy approximation* that directly learns a parametric policy π_θ. For its formulation, we define the object we want to maximize:

The exact individual values of $Q(s,a)$ and $V(s)$ are not important for a policy, but how its values ranks against each other, for example, when the policy is given by taking the $\arg\max_{a' \in A} Q(s,a')$.

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[R(\tau)\right]$$

which is the total reward of trajectory τ, $R(\tau)$, where τ is a realization of the distribution π_θ. Using the gradient ascent, we can write the update for θ as:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

with a few manipulation tricks is possible to show that:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{i=0}^{|\tau|} \nabla_\theta \log \pi_\theta(s_i, a_i) R(\tau)\right]$$

In practical terms, $R(\tau)$ can be approximated from sampling and averaging, for example, using G_t, with the effect of added variance. Several policy gradient methods have the form:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{i=0}^{|\tau|} \nabla_\theta \log \pi_\theta(s_i, a_i) \Psi_i\right]$$

Where Ψ_t can be:

1. $R(\tau) = \sum_{i=0}^{|\tau|} r_i$ - the total reward of trajectory τ
2. $R_t(\tau) = \sum_{i=t}^{|\tau|} r_i$ - the total reward after action a_t has been taken
3. $\sum_{i=t}^{|\tau|} r_i - b(s_i)$ - the reward with baseline
4. $Q(s_t, a_t)$ - the state-action value function
5. $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$ - the advantage function
6. $r_t + V(s_{t+1}) - V(s_t)$ - the temporal differences residual

The usage of the temporal differences residual is part of a class of methods named *actor-critic*. Such methods base their updates on previous knowledge which is signaled on temporal differences by $r_i + V(s_{i+1})$ which approximates the target for $V(s_i)$. It is also noteworthy that temporal differences enables updates on a per-action basis.

Their common ground is that Ψ_t always signals how good is a given trajectory, or individual timestep in the case of temporal differences, intending to reduce variance and better assign the influence of each action on the expected reward.

Finally, the advantages of using policy approximation methods, even those that depend on value functions, are their convergence and smoothness. Each update of θ makes minor changes to π_θ, while policies based on Q or V can change abruptly. Directly learning π_θ also enables easier ways of handling continuous state-action spaces and non-deterministic policies. Nevertheless, given the nature of those methods, they can get trapped in local maxima depending on how the experiences are acquired.

In this way, we conclude our background on the basics of reinforcement learning. Indeed it is much deeper and has many more methods and subtle details than the presented here. Hence, for the curious, the standard reference in the field is [29] with extensive coverage of all topics in reinforcement learning. In the next section, we make some final remarks and introduce ideas concerning today's RL challenges.

[29] Sutton und Barto 2018 Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement learning: An introduction*. MIT press, 2018

Final Remarks and Ideas

The methods presented earlier are the basis for RL, yet, they comprise the tip of the iceberg in this area. A significant challenge lies in designing suitable learning environments and rewards. We dealt with many facets of those challenges while developing our application. While a good insight into the problem is fundamental, achieving success undoubtedly requires a good amount of trial and error. In short, in the following paragraphs, we cover some ideas we deemed interesting and were used for our application.

Curriculum learning[30] - A key concept of human teaching is starting from simple things and increasing complexity. Our entire education is organized in the way of leveraging previous knowledge. Curriculum learning is an abstraction of this practice for machine learning. Some of the complex tasks can be broken into more straightforward tasks, and as the agent progresses, the task difficulty also increases. The concept of difficulty is related to entropy and how hard it is to achieve a significant reward signal. Sadly, developing a curriculum demands expertise that may be unavailable for some problems.

[30] Bengio u. a. 2009 Bengio, Yoshua ; Louradour, Jérôme ; Collobert, Ronan ; Weston, Jason: Curriculum learning. In: *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, S. 41–48

Generalized advantage estimation[31] - As seen in the policy approximation methods, there are many possibilities for the Ψ_t signaling the quality of trajectory actions. GAE is a class of formal generalizations of such signals intended to reduce the variance and accelerate the learning process.

[31] Schulman u. a. 2015b Schulman, John ; Moritz, Philipp ; Levine, Sergey ; Jordan, Michael ; Abbeel, Pieter: High-dimensional continuous control using generalized advantage estimation. In: *arXiv preprint arXiv:1506.02438* (2015)

Curiosity[32] - Learning an optimal policy can be challenging for various reasons. Not sufficiently exploring the environment (state-action space) is one of the easiest ways of becoming trapped in a sub-optimal. Curiosity is a form of intrinsic reward intended to encourage the exploitation of actions for which the agent poorly predicts the outcome.

[32] Pathak u. a. 2017 Pathak, Deepak ; Agrawal, Pulkit ; Efros, Alexei A. ; Darrell, Trevor: Curiosity-driven exploration by self-supervised prediction. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, S. 16–17

Generative Adversarial Imitation Learning[33] - Embedding expert knowledge of a field into the solution method of a problem is always desirable. One way to do it for agents is learning from an expert, more

[33] Ho und Ermon 2016 Ho, Jonathan ; Ermon, Stefano: Generative adversarial imitation learning. In: *Advances in neural information processing systems*, 2016, S. 4565–4573

precisely, the actions an expert would choose. The challenge for *Imitation Learning*, is that the rewards for the expert are not known, and usually, the amount of data from the expert is limited. GAIL solves this problem using ideas from generative adversarial networks[34]. Interestingly, imitation learning can be used to *bootstrap* the learning and afterward continue the process with RL; This approach can be highly successful for complex tasks.

Experience replay and priority[35] - While a human can learn from a few experiences, RL using artificial neural networks require many orders of magnitude more experiences. The sample efficiency of learning is a concept directly related to the needed time, computational power, and general tractability. Re-using meaningful past experiences (for example, experiences with significant deviation from the expected outcome) can significantly improve the learning process by decreasing the number of experiences (simulation steps) needed.

Domain randomization[36] - The motivation to use ANN's rests on its ability to learn patterns from experience. Still, it is impossible to exactly control which patterns are learned. It is not uncommon to overfit the experiences and lack generalization because the network "focused on the wrong patterns". In this setup, randomizing the environment properties and configurations helps to take away the focus from undesired patterns and focus on general abstractions. This simple yet powerful idea significantly improves the generalization and enables the transition of an agent to different environments and conditions.

[34] Goodfellow u. a. 2014a GOODFELLOW, Ian ; POUGET-ABADIE, Jean ; MIRZA, Mehdi ; XU, Bing ; WARDE-FARLEY, David ; OZAIR, Sherjil ; COURVILLE, Aaron ; BENGIO, Yoshua: Generative Adversarial Nets. In: GHAHRAMANI, Z. (Hrsg.) ; WELLING, M. (Hrsg.) ; CORTES, C. (Hrsg.) ; LAWRENCE, N. (Hrsg.) ; WEINBERGER, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 27, Curran Associates, Inc., 2014. – URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[35] Schaul u. a. 2015 SCHAUL, Tom ; QUAN, John ; ANTONOGLOU, Ioannis ; SILVER, David: Prioritized experience replay. In: *arXiv preprint arXiv:1511.05952* (2015)

[36] Tobin u. a. 2017 TOBIN, Josh ; FONG, Rachel ; RAY, Alex ; SCHNEIDER, Jonas ; ZAREMBA, Wojciech ; ABBEEL, Pieter: Domain randomization for transferring deep neural networks from simulation to the real world. In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)* IEEE (Veranst.), 2017, S. 23–30

4
System overview

We have seen in Chapter 2 how various works handle the problem of task-directed controlling and animating a character jointly, and in chapter 3, the basics of learning from experience. Yet, the other half of developing concrete agents requires a good amount of effort on designing a proper learning environment, reward system, agent's sensing, and acting mechanisms. Here, we will present our approach and insight into such topics in a relevant and general way for our application (which we will go deeper in detail in the Chapter 6).

Our overview is split into two parts: First, our hierarchical modeling and implementation of DogBot (our virtual dog), and later the training procedures and development. It is important to note that, while many modeling options were tried for locomotion training, we will stick to their final choices. Our previous work [1] includes the study's details on modeling choices and their performance for locomotion.

[1] Souza und Velho 2021 Souza, Caio ; Velho, Luiz: Deep Reinforcement Learning for Task Planning of Virtual Characters. In: *Intelligent Computing*. Springer, 2021, S. 694–711

4.1 Hierarchical Control

Developing an agent for interactive media requires more than learning to achieve a goal. How the agent acts, independently of the task, defines its artistic expression, directly affecting the user experience. Most characters can construct part of their expression relying on language, both spoken and written, but pets in specific depend significantly on their body expression, which in our context translates to animations.

In Chapter 2, we have seen a few approaches to handling goal-directed tasks and characters' animation. One of those works is (Peng u. a., 2017) [2] which uses physical simulation for generating animations. This approach ends up with animations that are unnatural or unsuitable for interactive media. More sophisticated methods, i.e., (Zhang u. a., 2018)[3], uses data-driven techniques to generate better animations but usually couples the learning of a specific task with the animation.

Here we treat the creation of agents as a hierarchical problem decoupling the task planning from the motion synthesis. This abstraction enables treating separately the requirements of animation expressiveness and task planning, which is essential for our dog

[2] Peng u. a. 2017 Peng, Xue B. ; Berseth, Glen ; Yin, KangKang ; Van De Panne, Michiel: Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. In: *ACM Transactions on Graphics (TOG)* 36 (2017), Nr. 4, S. 1–13

[3] Zhang u. a. 2018 Zhang, He ; Starke, Sebastian ; Komura, Taku ; Saito, Jun: Mode-adaptive neural networks for quadruped motion control. In: *ACM Transactions on Graphics (TOG)* 37 (2018), Nr. 4, S. 1–11

application and simplifies the development of complex behaviors.

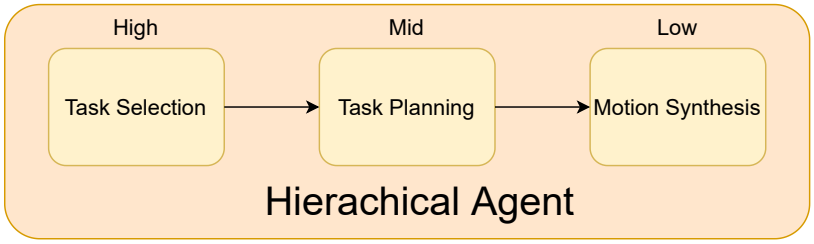

Figure 4.1: Three-level hierarchical agent abstraction. Each level has its time granularity scale and complexity working on different inputs.

Our agent hierarchy comprises three levels, *Task Selection*, *Task Planning* and *Motion Synthesis*, as shown in Figure 4.1. First, the task selection controls the high-level decisions on a coarse time-scale, such as setting objectives and selecting the appropriate behavior, based on global observations of the environment. Next, the mid-level task planner uses local observation to decide on the agent steps to achieve the set goal, for example, moving forward. Finally, the motion synthesis is the lower level, taking the previous said actions and synthesizing them into animations according to the agent's internal state (for instance, its pose and velocity) and actuators.

Here we focus on applying reinforcement learning for the mid-level controller while mixing different classic approaches for the other levels. Yet, it is interesting to note that machine learning can be employed in various forms for all levels with the appropriate modifications. A good example would be using more complex motion synthesis modules like the ones presented in the related work (*i.e.*, Zhang u. a.).

While our hierarchy appeals to the decision's time scale, another exciting way to look at such hierarchy is about how globally the agent observations and reasoning are. Taking a regular workday, on a higher level, one may think about how to organize its task, *i.e.*, dress up, eat breakfast, go to work, etc. On a mid-level, given a set goal, for example, "going to work", one would plan its steps to choose the means of transport, route, etc. Lastly, the lower level would take care of one's actions such as walking, opening doors, etc. It's also important to note that while humans may make plans in advance, agents (and humans as well) can better handle adversity in dynamic environments by planning only the immediate next step.

Agent & Environment

Our instance of hierarchical agent takes the form of a Virtual Reality application in the Unity3D with a space with variable obstacles/walls where the player can interact with the DogBot agent (Figures 4.2 4.3). The agent has a collection of behaviors that are conceptually split into three categories:

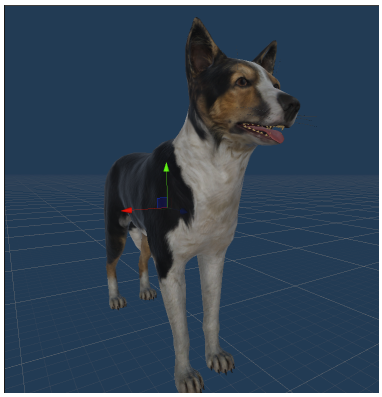

Figure 4.2: DogBot agent 3D model.

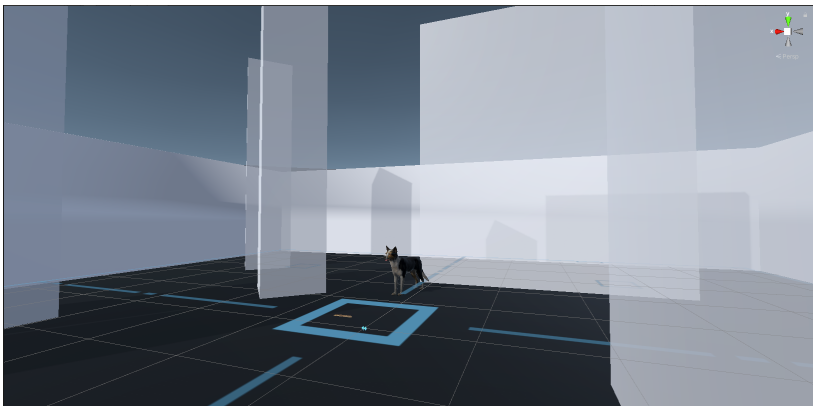

Figure 4.3: View of an instance of the playable area where the player and dog agent can interact in VR. Its size and obstacles can be variable. This instance uses a 10×10m size.

- **Command triggered** - Behaviors that are explicitly initiated by a player's command. This class represents the goal-directed functionalities and includes: calling the dog, playing fetch, jumping the hulahoop.
- **Self-initiated** - Behaviors that do not depend on the player; they account for the perceived autonomy and liveliness of the agent and includes: the dog wandering and looking at things of its choice.
- **Player-Agent interactive** - Behaviors without well-defined goals modeling interactions between the player and the dog, for example, petting or playing around. This class also influences the perceived autonomy and liveliness of the agent, but it plays a vital role in developing empathy.

Combined, these three kinds of behaviors produce a complete experience for the perceived intelligence of the dog agent. The construction of each specific behavior depends on various components or modules from all hierarchic control, which are overview next, and further developed into their respective Sections 5.1, 5.2, 5.3.

Figure 4.4, shows a detailed diagram of the modules of our agent and their flow. The high-level is similar to conventional game logic, processing user input and environment state obeying our game rules. These rules define our application's small world and intended experience, hence could differs for a different narrative.

In turn, the mid-level contains the task planners divided per functionality instead of per specific behavior, appealing more to the implementation than the earlier abstraction of the agent [4]. This implementation allows one to use the most straightforward method that suits the module and achieves complex behaviors by mixing their functionalities without the need of employing machine learning or any specific method on all levels.

[4] This approach is also a way of imbuing prior knowledge in a behavior instead of trying, for example, to shape the learning goal with multiple objectives.

The first controller split comprises the ones using *reinforcement learning*, in this case, locomotion and dexterous tasks that would be otherwise hard or computationally costly to solve using search or optimization. Next, there is the *stochastic controllers*, which in our application are used for the agent self-initiated behaviors when idle. Note that any controller can also communicate and use functionalities from other controllers below them.

Lastly, at the lower level of the hierarchy, the character controller and animator are responsible for the motion synthesis. Together they encode and control physical properties of the agent's movement, such as forward velocity, jump power, turn speed, and animation blending. This controller is similar to a usual playable character in a game.

The *condition check* (shared by all modules in Figure 4.4) is part of the environment dynamics influencing the decision and planning of all behaviors. It is also part of the game logic playing a role in how the interactions can happen.

We also note that the acyclic nature of our diagram implies in the independence of each module and allows for quickly identifying the hierarchy structure.

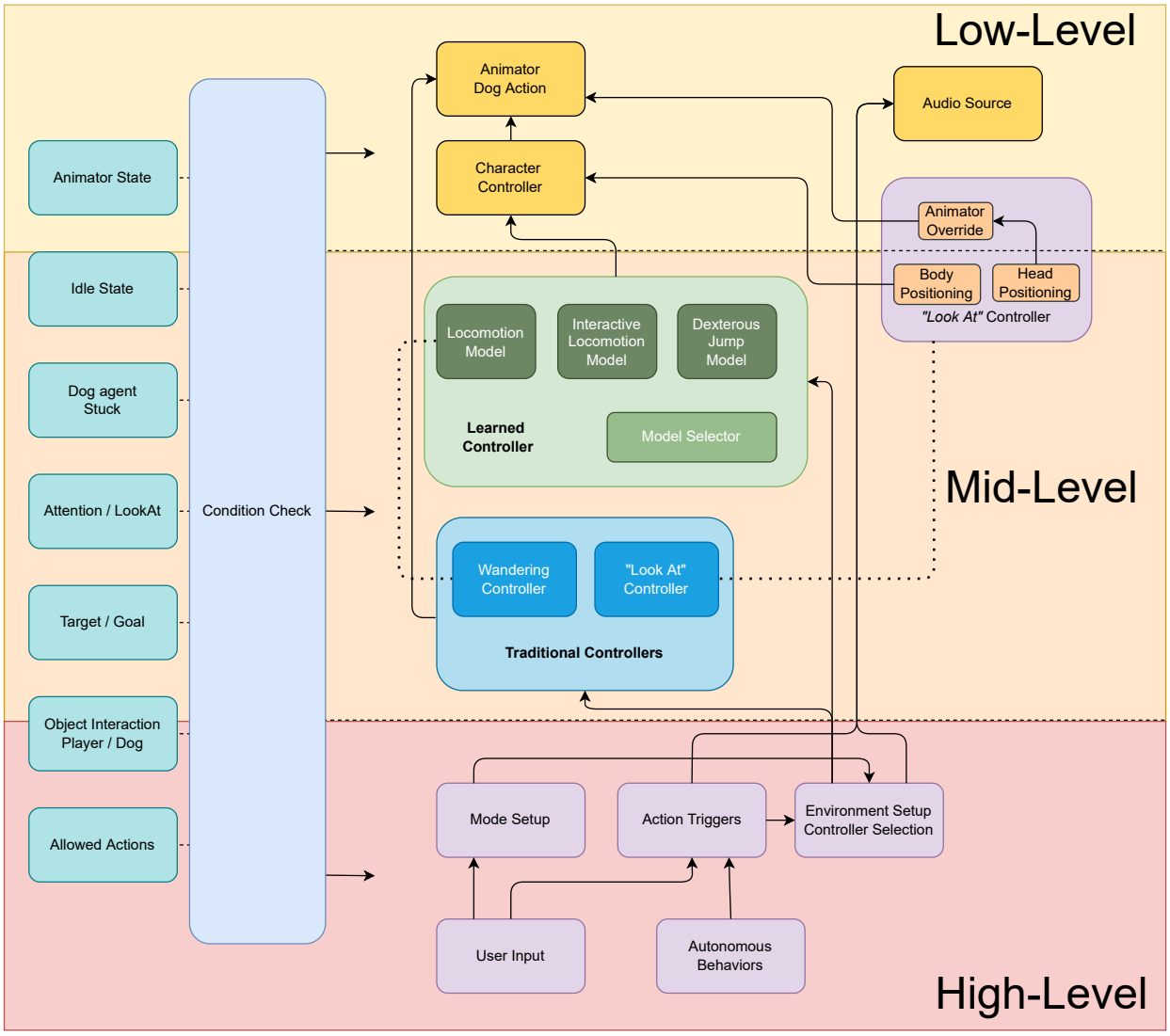

Figure 4.4: Detailed diagram of our environment organization and modules. On the left, the *condition check* symbolizes the state of the environment serving for our game logic and as observation for different controllers. Next, at the high level, the *user input* and *autonomous behaviors rules* flow through the modules setting the environment conditions, goals, and triggering actions. The mid-level controllers receive these actions and plan the tasks given the set goal. Finally, the lower level translates the planner steps into the agent animation. Each level/controller works on its own time scale, and the final agent's behavior results from the combinations of the planners.

We present details of our motion synthesis and task planners in the subsequent sections, leaving the higher level of task selection description for the Chapter 6. We believe it can be best understood together with the general application functioning, as its modeling is more related to our game-logic choices.

Motion Synthesis

Synthesizing motion is crucial in graphics applications and it is present in every game engine. Here, we use two approaches for generating the movement of our dog character. The first is through an ordinary heuristic character controller based on animation clips blended together and played according to a state machine. The method, as mentioned above, is one of the standard ways of animating in a game engine and has more straightforward requirements. The second method employed is procedurally animating the dog's head and neck. We base our choice on our available resources (i.e., 3D models, animation clips, data), but nothing prevents one from using other data-driven and machine learning approaches.

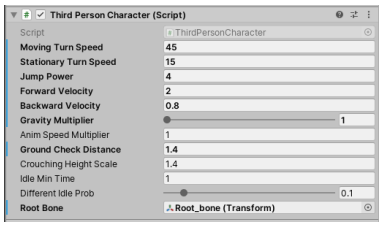

Figure 4.5: Character controller parameters. These parameters controls how the agent physically behaves, such as maximum velocity, turning speed, jumping power, etc. It can be tuned for different behaviors and goals.

Figure 4.5 shows the character controller parameters while 4.6 shows the controller internal states (*Animator* in the Unity3D) controlling how the animation clips are blended. A node in this graph contains the animations that can be played and their blend tree, defining how to mix them based on continuous parameters smoothly. In the same way, a connection describes conditions and how transitions are made, for example, the time taken or in which part of the animation clip the transitions are allowed. Note that the states and variables are not visible from the outside. On top of the animator, a script sets the internal variables according to the environment state and commands received. Following there is the definition of the interface receiving the planner commands:

- X-axis - Controls the character forward/backward speed and is a real value in $[-1, 1]$,
- Y-axis - Controls the character steering left/right and is a real value in $[-1, 1]$,
- Jump - A binary value that controls if the character should jump or not,
- Crouch - A binary value that controls if the character should crouch or not.

Indeed, while the internal updates of this module happen at 60Hz, the above controls have no guarantees of being executed immediately or at all, as it depends on the controller's internal state. Even though this is not visible to the planner, it has to learn and adapt to what the low-level controller offers. This separation between the received actions and motion synthesis ensures that the generated animations follow the controller's intended design independently of the planner.

Of course, that same design and input interface can impact the agent's learning speed and final performance [5].

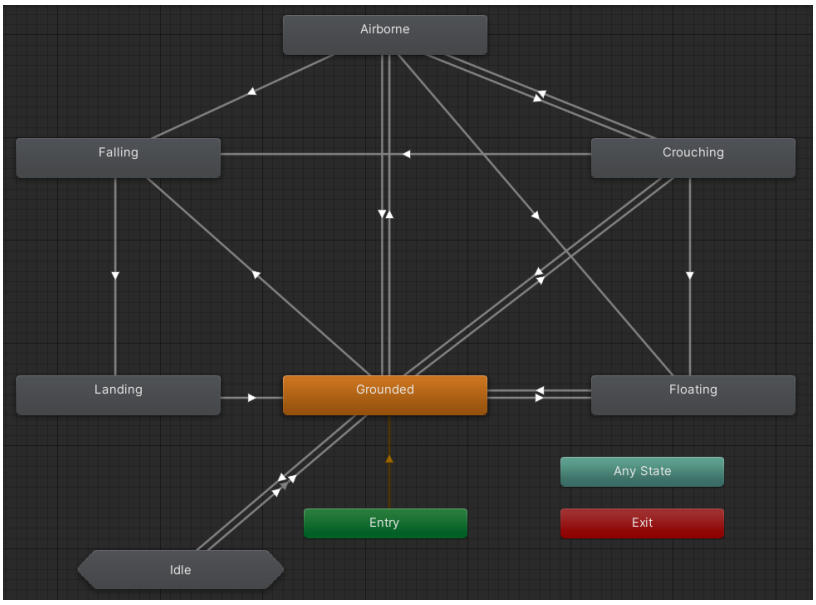

Figure 4.6: The Unity3D Animator Graph. This graph is the state machine containing the blend-tree for animation clips. Each of these states relates to the physical condition of the agent. They can describe if the dog's paws are touching the solid ground, floating or whether it is airborne.

[5] In our previous work (Souza und Velho, 2021) we explore a few variations on the modeling of the control interface and its impact on the agent's performance. One of them is relative to using discrete commands for the X and Y-axis, which slightly speeds up the learning, but degrades the agent's performance, especially on dexterous tasks.

Our second motion synthesis module consists of procedurally animating the head and neck of our dog character. This module is used for making the dog look at specific things during its animation, hence this technique was easier and more controllable than, for example, creating individual animation clips and inserting them on the previous Animator. Another point of using more than one way of generating animations is to highlight the freedom allowed by the hierarchical and modular design of the agent. Next, we continue with more details about the mid-level task planners.

Task Planner

The perceived intelligence of an agent is directly related to what they can achieve or the tasks they can complete. Our modeling treats the task planning in the mid-level with a few different modules using both machine learning and other heuristics.

This section focuses on modeling the modules mentioned earlier while leaving specific details of each behavior for the later chapters.

Reinforcement learning planners

Our first module uses deep reinforcement learning to handle locomotion and dexterous movements. While the set of inputs or observations that this module receives is fixed, it can use different learned models (artificial neural networks) for different tasks, such as playing fetch or jumping the hulahoop.

Figure 4.7 shows one of the sources for the agent sensing with raycast, which enables the agent to detect the near environment and

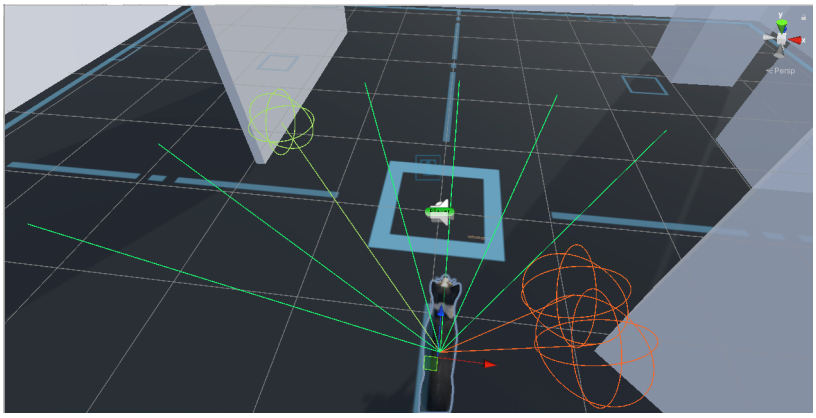

Figure 4.7: Agent raycast sensor. This sensor has a fixed angle and radius corresponding to a partial observation of the local environment.

goal such as the walls, the player, or the stick to fetch. This planar sensing works similarly to an image generated by a projective camera. It also has the benefits of already being segmented (each type of object has its tag) and is inexpensive compared to the computational power needed to process raw images. Souza und Velho investigates in more detail the usage of raw images and hand-crafted observations when learning and its implications in the agent's performance. Here, we choose a compromise that dramatically diminishes the computational power needed during the training phase, making the development iteration faster.

Other observations the agent receives are:

- the direction to the target or goal, working like a compass when the goal is far away and not detected by the raycast sensor,
- the agent's velocities and local coordinate system functioning as the agent's self-awareness [6].

Besides the "direction to the target" observation, everything the agent observes does not depend on any underlying knowledge of the environment dynamics. We call this a self-contained agent that is easier to work in different environments without major adaptations.

The outputs of this module are precisely the same inputs of the motion synthesis controller (X-axis, Y-axis, Jump, Crouch), with decisions being made at a 30Hz rate. High rates make the agent look more responsible to the environment, but its actions may appear not smooth. Low rates make the agent less responsible and negatively impacts training difficulty. We chose to use an exponential moving average to smooth the actions without significantly losing the agent responsiveness [7].

Deterministic and Stochastic Planners

Heuristic-based controllers use fixed procedures scripted by a human expert, usually using condition-action rules, state machines, or search. For simple tasks, these methods are much more straightforward than using learning-based controllers, without losing on quality.

Here, we use the condition-action type to control where our dogs look. Given a set of rules and interest points, we directly define the

[6] It is also noteworthy that the agent's self-awareness could contain information of, for example, the motion synthesis internal state, but it would require specific adjustments for different controllers.

[7] Interestingly, our choice for 30Hz with smoothing reflects the size and requirements of our application environment. Other tasks with different agents or environments' sizes could differ significantly.

steps to move and animate the dog head deterministically (we will cover more details of this behavior in Section 5.2). Such condition-action planners are easy to implement for simple tasks with a few well-known conditions and actions. Still, they can get complicated for an unknown or extensive set of conditions and actions.

Next, our second planner (used for the wandering behavior) is more refined with a stochastic behavior. We use rejection sampling to find a valid random route and use the reinforcement learning planner to move the dog through this route. This combination of planners makes the development of more complex behavior much more manageable. While one could directly randomize the agent's actions, it would not achieve the same effect, for example, because the route generation works on a different time scale and considers other variables that may not be observable by the agent. Likewise, directly computing the actions, for example, walking straight to the immediate goal, would make the agent robotic without dynamic; dynamics that are excellently handled by the reinforcement learning module, considering its local observations and state.

The following section overviews the general training procedures, reward design, and other considerations.

4.2 *Training overview*

Until this point, we have seen our mid-level controller's input and output interface, but not the agent reward system and learning process. Here we will cover these topics, including details of the framework used: the Unity3D *ML-Agents*[8].

[8] Juliani u.a. 2018 JULIANI, Arthur ; BERGES, Vincent-Pierre ; VCKAY, Esh ; GAO, Yuan ; HENRY, Hunter ; MATTAR, Marwan ; LANGE, Danny: Unity: A general platform for intelligent agents. In: *arXiv preprint arXiv:1809.02627* (2018)

The ML-Agents toolkit is specifically developed to create intelligent agents using the Unity3D game engine to simulate the environment and acquire experiences (data). It has a simple interface and a few examples for deriving agents from. Additionally it is open-source, allowing it to be modified to suit any purpose. Our first interaction with the ML-Agents was with its early experimental version v0.3.1 (2018), later updated to the first release, v1.0.2 (2020).

Figure 4.8 shows the general learning schematic. The agent lies inside the Unity3D where the simulation happens with the observations and rewards collected by the ML-Agents. Furthermore, these observations are transferred to the ML-Agents in Python's environment, responsible for learning the policy and sampling actions. Finally, the sampled action is sent back to the agent.

The Python's side uses the Tensorflow/Pytorch as backbone for implementing and training the neural networks. Also, various parameters related to the algorithm used and the network layout can be set in a python configuration file, while the environment and agent details are directly specified in the Unity3D.

Moreover, various agents and environment instances can run concurrently during the training, effectively speeding up the learning, with the minor disadvantage of being a little less sample-efficient and not equivalent to a single agent/environment.

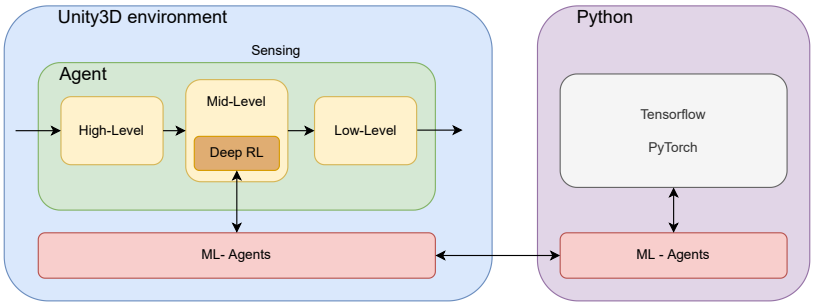

Figure 4.8: Diagram of our agent modeling and the connections with the ML-Agents toolkit. Note that the communication with the external Python's module only happens during the training, whereas inference uses Unity3D's inference engine *Barracuda*

Our training uses three important algorithms for successfully learning. They are:

- Proximal Policy Optimization [9] - is a recent policy approximation method developed with the challenges of deep learning in mind. Thus, it seeks a balance of learning performance, sample and computational efficiency. Using a surrogate (clipped) objective to approximate the policy through the stochastic gradient, it tries to emulate the benefits of more complex methods, such as trust-region [10], that ensures smooth updates of the policy, but being more computationally and sample efficient. Continuous control tasks, such as our agent, present better convergence and smooth updates of the policy during training.
- Curiosity[11] - is an intrinsic reward method to tackle the agent's exploitation-exploration dilemma and helps avoid sub-optimal solutions. Like the idea of "curiosity", it rewards the agent when it explores state-actions for which its knowledge and predictions are inaccurate. Its weakness is getting stuck on trying to learn from high-entropy or completely random sources, which cannot be well predicted, gladly it is not the case of our controlled training environment.
- Generative Adversarial Imitation Learning (GAIL)[12] - uses ideas from Generative Adversarial Networks (GANs)[13] applied to imitation learning. GANs revolutionized by introducing a method to learn how to sample from a distribution, with one of its most impressive applications being style transfer between images.

  Standard imitation learning, as illustrated by *Behavioral Cloning*, using suffers from the need for many expert examples to learn effectively. Comparatively, GAIL uses the GAN technique to imitate a behavior provided by a limited amount of expert examples, making it a compelling approach to learning complex behaviors.

These techniques can be used in conjunction to learn complex behaviors in a simple way. Here we use GAIL as a form to bootstrapping our agent policy. Using a few minutes (3 mins) of recorded expert examples, we can bootstrap a policy that is later improved using reinforcement learning. Tasks that otherwise would not be learnable by our agent [14] solely with RL became easily tractable, simplifying

[9] Schulman u. a. 2017 Schulman, John ; Wolski, Filip ; Dhariwal, Prafulla ; Radford, Alec ; Klimov, Oleg: Proximal policy optimization algorithms. In: *arXiv preprint arXiv:1707.06347* (2017)

[10] Schulman u. a. 2015a Schulman, John ; Levine, Sergey ; Abbeel, Pieter ; Jordan, Michael ; Moritz, Philipp: Trust region policy optimization. In: *International conference on machine learning* PMLR (Veranst.), 2015, S. 1889–1897

[11] Pathak u. a. 2017 Pathak, Deepak ; Agrawal, Pulkit ; Efros, Alexei A. ; Darrell, Trevor: Curiosity-driven exploration by self-supervised prediction. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, S. 16–17

[12] Ho und Ermon 2016 Ho, Jonathan ; Ermon, Stefano: Generative adversarial imitation learning. In: *Advances in neural information processing systems*, 2016, S. 4565–4573

[13] Goodfellow u. a. 2014b Goodfellow, Ian ; Pouget-Abadie, Jean ; Mirza, Mehdi ; Xu, Bing ; Warde-Farley, David ; Ozair, Sherjil ; Courville, Aaron ; Bengio, Yoshua: Generative adversarial nets. In: *Advances in neural information processing systems* 27 (2014)

[14] It is worth noting that the simplicity of our agent sensing also plays a role in the difficulty of a task. With more advanced sensing, for example, more rays or even raw images, tasks that need fine control can take advantage of better sensing. In fact, the art in developing intelligent agents is balancing the required computational power with smart approaches.

the need to develop complex reward systems or curricula.

Following we detail our reward system shared between learned brains, other unique specificities for each brain are left for later Chapter 5.

Reward System

Our background chapter defined the agent reward as a scalar given at each step related to the action-state pair. Although this general definition fits various cases well, most of the time, a single non-zero reward is given only at an ending state, where the final outcome is known, for instance, win/lose. These sparsity in rewards is one of the challenges when learning and significantly slows the process, up to the point of being computationally intractable.

Developing an environment where meaningful experiences with non-zero rewards are often found is crucial (even when starting from random policies, which are typical for most learning algorithms). Here we call these design choices as *reward system*.

Our dog agent brains share few components for its rewards and they are:

- $+1.0$ - when the goal is reached. This type of reward is the most common, and for objectives that are reachable in a few steps it may be sufficient for learning. In our case, given the agent decision rate (30Hz) and maximum velocity, it may take several steps to approach the goal and hence the distance to the objective also influences the amount of steps necessary. Therefore, a good curriculum is initializing the agent near the objective and increasing its distance according with the learning.

- $+0.001(v_a * d_{\text{tgt}})$ - every time step, where v_a is the agent velocity vector and d_{tgt} is the normalized direction to the target. Reducing the distance to the objective may be an indicator of progress. In a cluttered environment, the agent may need to turn around obstacles, and this reward would be negative, even if the agent is taking the right steps to complete the goal. For this reason, the intensity of this signal is low and, in turn, not overtaking the $+1$ of the real objective. Of course, the choice of the weight depends on how many steps the agent takes to complete the task. Our particular environment varies between 50 to 300 steps, and the best case 0.001 weight may accumulate to 0.3. Interestingly, this type of tuning is solely based on the training environment and does not prevent the agent from performing in a larger environment afterward. In general, such reward has positive effects accelerating the learning process when the true reward is scarce, as long as it is well employed.

- -0.2 - when colliding with obstacles. This is one of the tricky rewards to set as it can play against the main objective. Low values could be ignored if colliding allows to faster or the same speed

when reaching the goal. Still, high values can entirely hinder learning as it would be better to stay still than receive penalties. Finding the correct value requires searching empirically, and as it depends both on other reward sources and the environment, it can get too complex quickly.

- -0.1 - when jumping. This penalty was set to avoid jumping when not necessary. In general, it does not affect the completion of any task, but makes the behavior more similar to a real dog. One reason for such a penalty is the motion synthesis module, where jumping allows for a fast turn in the air.
- -1.0 - when leaving the training area or getting stuck. Both of these ending states are undesired and hence are the opposite of completing the task.

Another important point is defining ending states. Although most behaviors could continue indefinitely, using a good ending criterion enables focusing on generating meaningful experiences and avoiding current rewards to affect actions that did not contribute to them. One barrier we use is having a maximum number of steps, ensuring that if the agent takes too long to find a final state, we end the episode as these experiences are probably not contributing to the learning.

This covers the basics of the agent's reward that is shared among the learned behaviors. Details of the environment and goal setup for each specific behavior and their challenges are presented in their respective sections in Chapter 5. Thereupon we continue with the training procedures using the ML-Agents.

Training procedures

So far, in previous sections, we have seen various aspects of our abstraction and modeling. Here, we enter the details of training an agent with the ML-Agents framework and our procedures.

Once the agent, environment, and rewards are modeled, one can build it into an executable from which the experiences are obtained. Training begins by launching the ML-Agents from python with parameters pointing to the environment executable and a configuration file (YAML) [15] . It is also possible to follow the training evolution with the *Tensorboard* Python library. Statistics of mean reward, episode length, and many others are available in real-time, enabling one to quickly identify the progress or lack thereof, therefor, improving the development and testing process.

The configuration file contains all parameters for controlling the algorithm used for reinforcement learning and imitation learning. The table 4.1 shows the values and a brief explanation of each parameter we used.

In addition to choosing our parameters following best practices, we didn't search extensively. A few reasons for this choice are to focus on the agent and environment modeling, which alone requires a good amount of effort and fine-tuning.

[15] When launching the training, the ML-Agents accepts a couple of other parameters, such as the number of environments to run parallel, window size, headless mode, loading from a previous section, etc. Their minuteness can be found on the ML-Agents documentation page as they are situational.

Name	Value	Description
Algorithm		
Trainer	PPO	Algorithm used for reinforcement learning.
Learning rate	3.0×10^{-4}	Intensity of the gradient descent update.
Maximum steps	1.0×10^{8}	Maximum number of steps experienced for the agent in the simulation environment
Gamma	0.99	Reward discount factor
Normalize	true	Normalize policy inputs
Batch Size	1024	Number of experiences used for each gradient descent step.
Buffer Size	40960	Number of experiences collected between the policy update steps.
Number of Epochs	3	Number of policy updates done using the same set of experiences.
Time Horizon	1000	Length in time steps of the influence of a reward over the agent's past actions and states. Upper bound for the cumulative reward summation.
Network Layout		
Hidden Units	512	Number of perceptrons per hidden layer.
Number of Layers	2	Number of hidden layers.
Curiosity		
Strength	0.002	Intensity of the curiosity reward signal.
Encoding Size	256	Number of hidden units used for the curiosity encoding network.
GAIL		
Strength	0.01	Intensity of the GAIL reward signal.
Encoding Size	128	Number of hidden units used for the GAIL encoding network.
Behavioral Cloning		
Strength	0.5	Intensity of the Behavioral Cloning reward signal.
Steps	5.0×10^{6}	Number of decision steps experienced

Table 4.1: Parameters used during training.
Trainer - ML-agents also offer the Soft Actor-Critic algorithm and the possibility of implementing your own trainer.
Learning rate - Higher values can increase the learning speed but also make it unstable or unable to train.
Maximum steps - Usually, our agent learns a behavior with fewer steps, but it continues to improve during the entire training section.
Batch size - Smaller sizes can increase the initial learning, but it makes the learning less stable later on. Problems with continuous control typically use higher values than discrete control.
Buffer size - Higher values slow down the initial learning, but generally, the policy updates are more stable. When running with multiple agents, it's common to multiply its value by the number of agents to avoid instabilities.
Number of epochs - Higher values can speed up the learning at the cost of biasing it towards the current experiences. Larger buffer sizes allow for a higher number of epochs.
Time Horizon - Its size depends on the typical agent episode length and should be enough to capture the behavior trajectory and its reward.
Hidden units & number of layers - Complex behaviors may require more units and layers to learn a good policy, but on the other hand, bigger networks are harder to train.
Strength - The strength of rewards other than the extrinsic ones should be small enough not to overshadow the actual goal. When such rewards are too large, they make the learning unstable and can diverge at any time during training.
Encoding size - It depends on the policy's input size and should be adequate to represent the agent's state space well.
Steps - The number of steps to use behavioral cloning in the initial phase of training.

Our training is done entirely headless [16], without the need of a GPU, and indeed it is usually faster on CPU. This behavior reflects our design choices, using low-dimensional vector observations and a small artificial neural network. It also allows for excellent performance for real-time applications without requiring expensive hardware and enables us to train locally with our available resources. Figure 4.9 contains the layout of our artificial neural network used for our agent's brain. During the training, a few other networks are used for GAIL, Curiosity, and the PPO critic, but they are not required for inference.

[16] It does so by passing the argument *no-graphics* for the ML-agents.

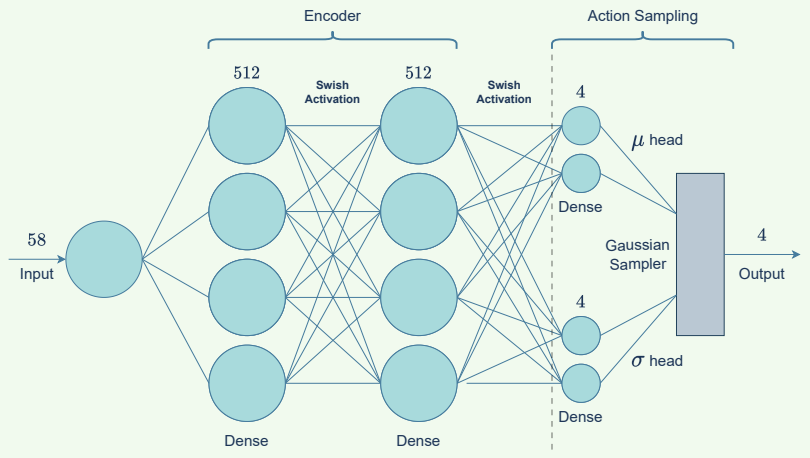

Figure 4.9: Dogbot's policy neural network. Its layout is simple and consists of two parts; the first is the *encoder* which can have n dense layers with m hidden units each. In our case, it has 2 layers and 512 hidden units, directly reflecting the ML-Agents configurable parameters. The second part is the action sampling, a Gaussian sampler based on learned values for μ and σ for continuous actions. The layout of the sampler is not configurable as it depends exclusively on the number of output actions. Interestingly, during the training is possible to follow up the entropy of the sigma values, which is a good indicator of the policy convergence.

When training, we collect experiences from eight executables with eight agents each, totaling 64 agents running concurrently. Likewise, our buffer size already accounts for this number of agents. Although the speed up is significant, it is not one-to-one, as it speeds up only the experience collection but not the gradient descent updates (Tensorflow/Pytorch libraries already parallels those).

The examples used for bootstrapping with imitation learning are directly recorded with the Unity3D Editor using the agent's heuristic interface. This interface can implement the behavior in any fashion as long as it outputs a valid action for the agent. Moreover, our implementation takes input from the keyboard/mouse to control the agent. Interestingly, it is much easier to use such an interface with a high-level control like ours than those using low-level torque-joint control, which is not straightforward to a human control.

Overall, the recorded examples have 1 to 3 minutes and are not perfect, but do complete the goal by achieving a final state. This approach serves well for many tasks that humans can complete but are hard to solve heuristically. Of course, this is just one of the tools available for creating intelligent agents; other tasks for which recording examples are impossible can benefit from other techniques.

The other two techniques we take advantage of during training are randomizing the environment goal and obstacles, and implementing a curriculum by controlling the task's difficulty through the number of obstacles and distance of the goal. Figure 4.10 shows a few examples of our training scene.

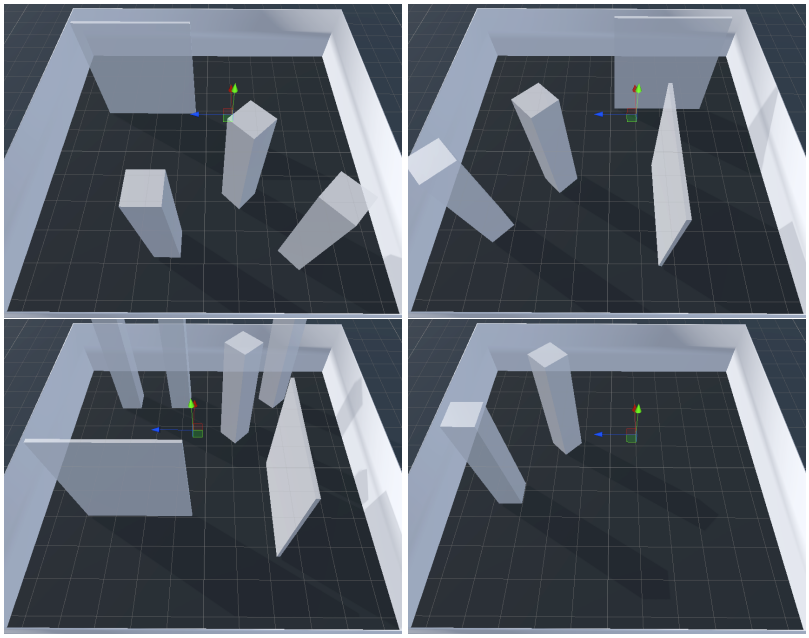

Figure 4.10: A few examples of our obstacles placement randomization in the training environment. The number of obstacles influences the scene's difficulty, while the variation in the positioning helps with the agent's generalization of the learning.

In essence, training an agent with ML-Agents does not require any particular Python knowledge and is straightforward for simple cases. In general, designing agents that can learn can be a much more arduous task. Finally, the ML-Agents does not restrict any possibility when creating agents or training; complex use cases can directly change its internal code, which is open-source and well organized.

5
Agent behaviors

We covered our general agent's functioning and training procedures in the past chapter, yet without details of specific behaviors. Here we dive deeper into these behaviors, which we split, as already said, into three categories: *Command triggered*, *Self-initiated* and *Interactive*. We propose these conceptual categories because they can be well defined and cover different aspects of an intelligent agent for interactive media.

The first class of command triggered behaviors enfolds the functionality types. A well-defined command or trigger initiates the behavior with a static outcome expected. As an illustration, we can think of calling your pet and hoping to capture its attention or that it approaches you. These kinds of behaviors are the most common, and they bring the idea of intelligence by functionality. Other more complex examples could be personal assistants answering a demand.

Admittedly, this class of behaviors covers many use cases passing the sensation of intelligence, but they look mechanical and are more similar to the idea of automation than artificial life. Now that we have touched on this point, an agent designed for interactive media seeks not only intelligence but passing the impression of being alive. This need brings the other two behavior classes into play.

The next class, the self-initiated behaviors, contributes significantly to the perception of life. It contains all behaviors triggered by the agent to satisfy its needs and desires. Examples are wandering, smelling things, looking at things, etc. Although these can be triggered by specific events (*i.e.*, looking in the direction of a loud noise), it differs from the first class as not being a functionality, but a means to emulate the *personality* of the agent or character. Therefore it is fundamental to simulating life and representing the uniqueness of an individual agent.

Finally, the last class of interactive behaviors is fuzzier. We think of it as the primary source of empathy and bond and can be regarded as the ability to initiate interactions or play along. While this class could be a mix of the previous classes, we believe that separately treating these behaviors is beneficial as it serves another conceptual purpose and requires a different approach in its implementation.

With this, we summarize our behavior classes and their intent, enabling us to enter in detail in the following sections about which

behaviors we implement in each category and their challenges.

5.1 Command triggered behaviors

As we have seen before, the class of command triggered behaviors can be compared to a functionality initiated by a specific circumstance. While they can relate to automation-like behaviors, it does not mean they are purely mechanical, but that their triggers and goals(ending states) are well defined and objective.

Our pool of such behaviors includes the *Call* (the dog approaches the player after receiving a command), the *Fetch* (the dog fetches a stick thrown by the player) and the *Hulahoop jump* (the dog dexterously jumps through a hoop). Moreover, its essential feature is locomotion.

The call and fetch use simple locomotion without dexterous movements or specific short timings. Indeed, it is possible to complete those tasks in different ways; for instance, the agent can approach from any direction and travel further distances. On the other hand, the hulahoop jump requires dexterity with precise timing for the jump action.

Together, these three behaviors cover different aspects of complexity: starting with simple and less restricted locomotion, dexterous locomotion, and finally sequential objectives for the fetch task. Additionally, combining these can lead to exciting results and more complex behaviors. In the following sections, we develop further into each of those behaviors.

Fetch & call

The fetch and call behaviors are our exemplars for basic locomotion. Although we call it basic locomotion, it has a few challenges: handling the obstacles and the initial learning.

Starting with the obstacles, balancing the objectives of avoiding the obstacles, and reaching the desired position during the learning depends not only on the assigned rewards but also on the environment setup. A fixed reward and environment difficulty often leads to an unbalanced behavior of either following only the position goal or avoiding collisions.

We solved this balance problem by implementing a curriculum where the agent first learns how to achieve the goal without obstacles and then adapts the behavior to environments with an increasing number of obstacles. This could also be understood as a form of bootstrapping from a previous behavior [1]. Another facet of our agents sensing the environment is its relatively simple ray-cast sensor. As shown in the figure 5.1, the limited length of the sensor and the fixed size of the traced spheres limits the distance that our agent can detect and also is susceptible to aliasing. Our previous work (Souza und Velho, 2021) showed that visual observations paired with convolutional networks could achieve great results at the cost of a much more significant computational burden when training. Here

[1] Indeed our first implementation used two separated scenes and bootstrapped from the brain learned on the environment without obstacles. These two ideas are, in essence, the same solution seen from different perspectives, with the curriculum being the broader abstraction.

our objective is to achieve the same level of performance with simple sensors and more advanced modeling.

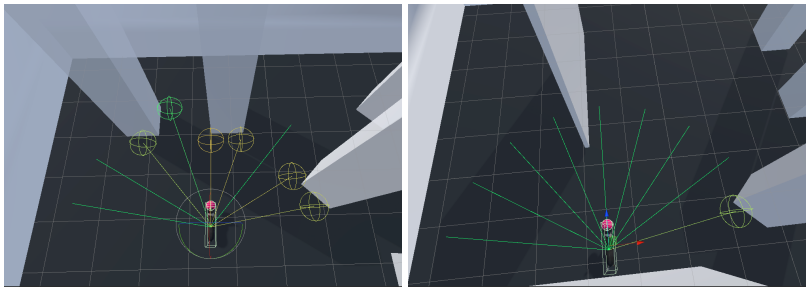

Figure 5.1: Instances of Dogbot's ray-cast sensor. Its lower computational cost has drawbacks in ray length and aliasing.

Besides the low resolution of the ray-cast sensor; in a virtual environment, it is simpler to control and adjust the sizes of the collision hitboxes allowing our sensor to be effective. We follow this approach for the target goal and the fetch stick, ensuring that the agent does not miss them. One could imagine this as running the agent and environment simulation on a coarse scale while displaying a more refined environment to the user.

The second challenge for Dogbot's agent was the initial learning being too slow. One reason for this behavior is its high decision rate (30Hz) combined with the delay of the motion synthesis module between a few animation transitions [2] . Therefore, if the agent does not commit enough time when these transitions occur, the action may not happen or cause a significant movement. However, slowing the agent's decision rate can make it less responsible in other cases.

The solution for this limitation was adding a forward bias to our agent's movement, considering its behavior was supposed to be moving in that direction most of the time. Interestingly, the solution was straightforward, but finding the source of these effects and the complex interplay of factors requires a significant amount of time and debugging. Afterward, this bias also functions as a method of reducing the burden of learning all of our agent's four branches of control (X-axis, Y-axis, jump, crouch) simultaneously [3].

Another step to increase the learning speed is lowering the goal's difficulty through its distance and positioning it in front of the agent. This measure alone could significantly speed up the initial learning, exemplifying the importance of developing suitable learning environments where rewards are often found.

[2] An excellent example of this combination is when the agent is idle and starts moving. There is a slight delay mixing between the idle animation and the walking/running animation. While this delay could be removed, it would affect the motion synthesis's visual quality, which is not desirable in our use case.

[3] The PPO algorithm has successfully handled tasks with much higher dimensions than our agent's control branches. However, the higher the dimension and interplay of them, the higher the difficulty of learning. This phenomenon is well known as the *curse of dimensionality*.

Hulahoop jump

In the previous section, we handled the basic locomotion used by fetch and call behaviors. Granted, the insights of these behaviors carry over for our agent's global locomotion, including our next step on command-triggered behavior: the hulahoop jumping.

Although the locomotion is the base for the hulahoop jump, this new behavior requires a more dexterous control of the agent move-

ment, such as approaching the hoop at a specific angle and timing its jump through depending on the hoop angle and height.

From the user experience point of view, these pet tricks behaviors are an easy way to engage the player interacting with the dog and get absorbed into the environment or narrative.

Interestingly, this behavior had many iterations until a functional brain was learned, and even after that, a few tweaks were needed to suit its artistic purpose. While these challenges had to be solved jointly, we will cover them individually with no specific order.

One of our agent's artistic requirements when jumping through a hoop was to avoid colliding with the hoop edge. This need translated to a penalty applied every time such collision happened. Unlike our obstacles collision penalty that could or not be directly in the way of the goal, this penalty was firmly against our goal. Since the agent has to approach the hoop before jumping, if it misses the timing of the jump or doesn't jump at all, the penalty received will be broadcast to all previous actions, which in reality were good actions leading to approaching the hoop.

Unfortunately, achieving balance for the interplay of these rewards was not possible; in general, the reward would be either positive or negative, enforcing the behavior or preventing it altogether. Even though we believe a curriculum could probably solve the learning, it would demand developing many more levels of difficulty and even so would generate various meaningless experiences. Any wrong action would be much more unforgiving than our previous locomotion behavior.

A more straightforward solution is using imitation learning to bootstrap our behavior and keep improving it afterward with reinforcement learning. This solution seamless overcomes our reward balance, requiring just 3 minutes of recorded experience [4] . The compelling point of bootstrapping is allowing the agent to learn how to complete the task, even if clumsily. In turn, this balances the reward of completion and the penalty of colliding with the hoop. Now the reward dynamics becomes optimizing the return of completing the task with a greater reward if no collision happens, instead of the previous unstable behavior of positive and negative rewards.

[4] Experience that can be easily recorded with the Unity3D Editor and ML-Agents toolkit. These experiences are not required to be perfect exemplars; they only need to complete the goal acquiring the final reward.

Working on improving an initial behavior should be simple. Looking solely at the training measures, it was evolving well, the episode mean reward was increasing, and the episode length was decreasing. However, visual inspection of our agent behavior was intriguing; it completely deviated from the intended behavior of jumping through the hoop.

In fact, the agent was exploiting our reward system, in Figure5.2 we show three versions of how we detected and rewarded the agent for jumping the hoop. The first version of the task's completion detection was simply ending the episode with a $+1$ reward when the agent collided with the disk in the center of the hoop. Although it is a necessary condition, it is not sufficient. Even though the bootstrapped behavior started jumping right, exploiting the reward

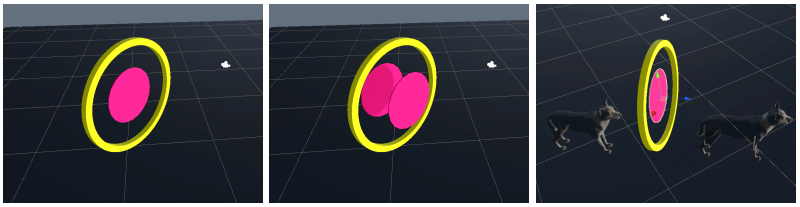

Figure 5.2: The three versions of the hoop condition check used for training.
(Left) The first naive version checks the objective completion with a single collision disk (in pink).
(Center) The second version checks the collision with two disks, one at each side of the hoop.
(Right) The last version, with one disk. Additionally, it checks the plane's side containing the hoop in which the agent was before and after the jump action.

was more advantageous in the long run. The agent could jump at an acute angle touching the disk without passing through the hoop (Figure 5.3), and hence completing the task faster than having to align itself to jump.

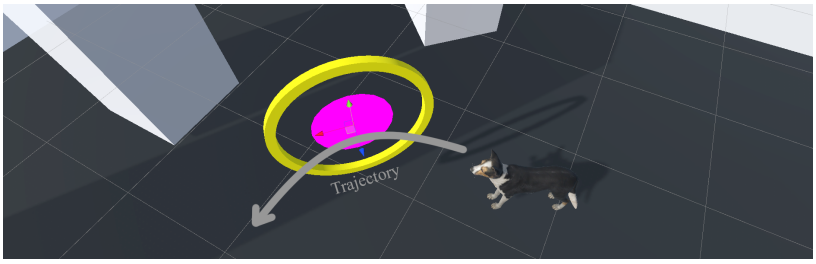

Figure 5.3: The first case of reward exploitation. The agent jumps from an acute angle touching the collision checker (Pink disk) but drifts away mid-air to avoid colliding with the hoop.

On a second try, using the two collision disks approach should ensure that the agent went through the hoop. Still, we were caught on another pitfall: the misconception that the agent would keep moving (remember that the episode ends when the agent touches the disks). Laughably, our agent's whims lead it to stop in the middle of the action hovering on the hoop (Figure 5.4).

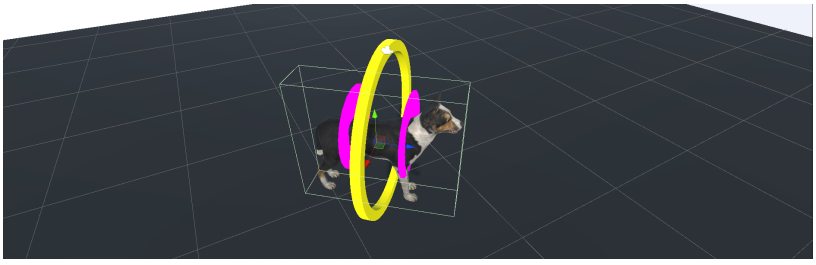

Figure 5.4: The second case of reward exploitation. The agent jumps facing the hoop and touches both collision checkers (Pink disks) as expected but fails to land on the ground. The landing issue is caused by the agent's hitbox (Light green) colliding with the hoop a short time after the jump action. This behavior does not happen while training since after the final reward is given by hitting both pink disks, the episode ends.

Here we have interesting remarks about condition checking and rewarding the agent. First, in our background section, we state that a reward is associated with a state-action pair. Still, the hoop case depends on a sequence of actions and other conditions that are not observable by the agent (*i.e.*, its past actions, the disk's collision, and the landing afterward). A more complex observation could avoid our pitfall, but correctly checking the task completion is a broad approach while keeping the agent simple. Next, it's noticeable how correctness should be a priority when assigning a reward. As it is an optimization, the agent will certainly go for the low-hanging fruit leading to unexpected behaviors, especially on virtual environments with 3D engines where many computations and procedures are approximated

given the necessity of being real-time.

After understanding these observations, it was straightforward to fix our agent behavior. Before assigning the reward, we look at the agent's position before and after jumping. The reward is given if they are on opposite sides of the plane containing the hoop and the center disks registered a collision between the jumping and landing.

To summarize, developing the reward system for behaviors that do not rely on a specific ending state but a short series of actions should strive for correctness. Like programming, defining pre-conditions, post-conditions, and the exact time the reward is assigned serves as best practices and can avoid various scenarios of the reward exploitation.

5.2 *Agent's self-initiated behaviors*

The previous section detailed the functionality of command-triggered behaviors and the ones we implemented for our agent. Even though an agent can have numerous behaviors of that class, they don't cover an essential aspect of living beings: satisfying their own will, curiosity, and needs. Here, we call this class of behaviors as *self-initiated*, as it makes sense from an agent-centered perspective.

For our specific intend, the behaviors of the class, as mentioned earlier, are used to model secondary or idle activities with lower priority than those triggered by player commands. Our Dogbot's idle behaviors are two: *wandering* and *look at*, despite their simplicity, their role of breaking the monotony of an idle agent staying still for long periods is a crucial concept for interactive environments. While they are sufficient to our narrative, it does not prevent other more complex behaviors from being used. When simulating large worlds with a persistent state, incorporating other behaviors that meet the common needs of a living being could be desirable.

On a general level, the idle behaviors share interesting properties: they can take advantage of stochastic approaches and may work on a lower decision rate (1Hz). A coarser time scale allows it to stay on a higher control level and delegate tasks to other controllers when needed. Additionally, its low priority does not delay or affect the handling of different inputs and behaviors because each controller runs concurrently. These properties derive from not having a specific goal or expected behavior, allowing for varying which behavior is triggered, for how long it runs, and its update rate resulting in more diversity for the agent actions [5].

Finally, as mentioned earlier, the behaviors' modeling is based on necessary pre-conditions, the trigger probability, and a cool-down after the action, with each behavior having its set of parameters.

[5] Coincidentally, it may even be harder to notice when one behavior is not working correctly; it may go easily unnoticed since it has no exact goal.

Look at & Wandering

It was not by chance that we chose to add the *Look at* and *Wandering* features to compose our idle behaviors. First, the ability to look

at different directions improves our agent animation and adds the notion of attention or focus, significantly contributing to the agent's perceived liveness.

As a mechanism of signaling focus, eye contact is even more critical for the specific case when the dog gazes at the player. Various studies on cognitive science theorize on how the perceived visual contact modulates the subsequent communication and interaction in humans, for example, Senju und Johnson which calls it "the eye contact effect". In fact, this effect can be noticeable, for instance, when one is chatting with a group; if no one is making eye contact, the person may stop by perceiving a lack of interest, in the other hand, if at least a single person keeps the eye contact it may motivate one to continue chatting. Although our agent is not a human, it still benefits from eye contact, especially when the user is in a VR environment. Also, in contrast to human eye contact, it is unlikely to introduce adverse effects for our dog interaction, such as making the player shy.

Figure 5.5 shows three sequential frames of our agent turning and looking at the player. This behavior follows a stochastic heuristic considering the agent's current state, the possible attention triggers, and then setting a target, how long to gaze, and a cool-down until the same action can happen again [6] .

Next, our second behavior, wandering, is designed to activate when there is no player interaction. In such situations, leaving the agent completely still would break the feeling of a living or intelligent creature [7] . While we have a few animations to play in this state (the dog shaking/stretching), they can quickly get repetitive. In contrast, the wandering being a parametric procedural approach is unlikely to generate the same trajectory of actions.

The wandering trajectory consists of random checkpoints placed in the environment with the agent moving between them using the previous reinforcement learning controller for locomotion. During its checkpoints traveling, the agent can make pauses, look around or even cancel the behavior, for example, when it is near the player. Also, the trajectory construction follows a greedy-stochastic approach based on the agent's local surroundings and a few parameters, for instance, minimum and maximum distance to travel, vision cone, minimum and maximum number of checkpoints, etc.

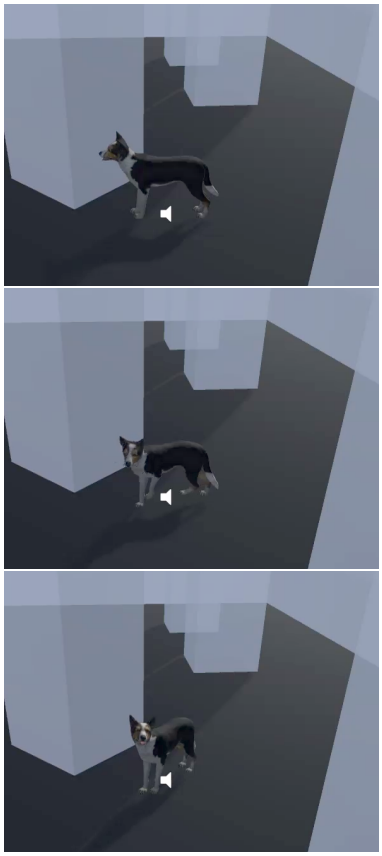

Figure 5.5: Frame sequence of Dogbot's looking at the player, slightly turning the body and head.

[6] It can get interesting when the dog either ignores a cue or follows a moving target, for example.

[7] Wandering near the player can also help it engage in the other command-triggered activities, which is a positive point for our intention.

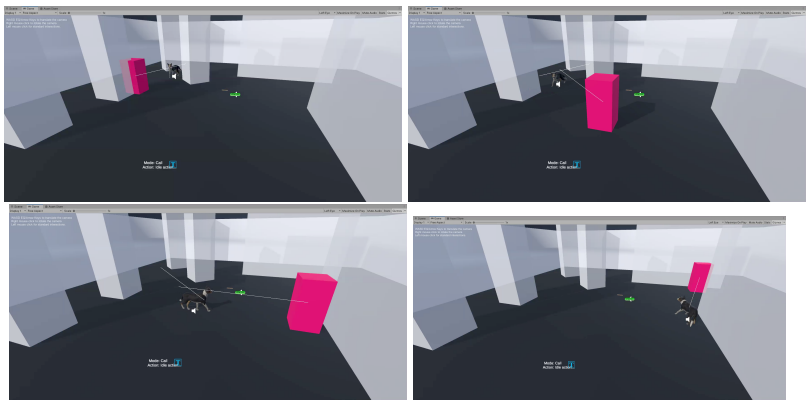

Figure 5.6: This behavior combines both the learned movement controller with randomly generated trajectories (with checkpoints in pink). We greedily construct these trajectories by choosing random directions inside the agent view cone and walking random lengths. To make it more befitting, the agent can also stop for random amounts of time and look around from time to time.

Figure 5.6 shows how a possible trajectory would be generated, step by step. One of the caveats of using the RL locomotion module is that we need to restrict the agent velocity as wandering is a slow-paced action. Here, it is done directly by clipping the motion controller input (forward velocity); it would not be as straightforward for a controller based on low-level joint torque control. Another necessity is the smoothness of the locomotion actions; for the wandering, the smoothing of the controller outputs is a must, as any flickering is much more perceptible when the agent is moving slowly [8] .

Finally, our choice for mixing both the "look at" and "wandering" works well for our intended idle behaviors. It is not a rule for all self-initiated behaviors. Here it is primarily a way of adding variability for the idle state.

[8] We have mentioned the smoothing of the locomotion controller outputs in the Chapter 4 (System Overview) and one of the primary motivators for it is the wandering behavior constraint.

5.3 *Player-Agent interactive behaviors*

In the previous sections, we have seen two opposite and complementary types of behaviors; command-triggered and self-initiated behaviors. The first has a well-defined goal and expected behavior, similar to a taught trick. The second relates to the internal needs and desires of the agent where the objective and expected behavior is unknown to the external observers.

While these two classes can encompass, if not all, most of the behaviors of an agent simulating a living being, we believe a third conceptual class contains behaviors that mix properties of the categories above and other specificities.

Here we name this new class of behavior as *player-agent interactive*, with their defining traits being the ability to be initiated from both parts, the player or the agent, and not having a defined goal or expected outcome. These characteristics suit well empathetic and affective behaviors. A good example is when playing with your pet; it may start from a player or dog action, while the outcome of the interaction is not well defined, it may depend on, for instance, the affective relation, personality, and mood of both the player and agent. As an illustration, when the pet owner comes home, the dog may run to welcome it; the owner may not be in its best mood and ignore the pet; On the other hand, if it is a stranger trying to approach the pet, it may run away or get aggressive.

In general, those interactions share similarities; while they are incredibly reactive, from the agent's perspective, their progress also depends on the agent's current internal state and memory. Modeling those can be very challenging, requiring everything from previous behavior classes and more; grasping the hidden triggers and goals for them being the most significant obstacle. Yet, they are the step-up for agents, as they model the social interaction and bonding between living beings. Indeed, these types of interactions are the ones that motivate having a pet. For this reason, we are convinced that emulating anything that resembles such behaviors can dramatically impact the perception of liveness and intelligence of a virtual agent.

Following, we develop our "play" mode behavior characteristics and requirements. Although it is far from complete modeling for empathetic and affective behavior, it is the first step in this direction and covers a few necessary conditions.

Play mode behavior

Our first choice for emulating an interactive behavior was having a play mode where the agent and the player can freely exchange by moving around. This pick is the simplest and more common for a pet and can take advantage of our past knowledge of locomotion.

From our initial analysis of how the dog should behave in play mode, the crucial requisites were to have a natural-looking interaction where the dog presents a type of synchronous movement with the player—achieving our desired behavior required working on three fronts: the locomotion properties, the movement triggers, and the goals or target movements, accounting for the class-specific necessities and the limitations of our motion synthesis module.

Starting with locomotion, the first identified properties for interactive behaviors were that they happen on a small range containing the actors and have fast responses inducing the idea of synchronism. Because of those, the agent must be able to make, for example, sharp turns, stay in range, and react to abrupt movements. Another related point was how collisions are differently handled compared to previous behaviors; here, given the nature and the possibility of sharp movements, it is not uncommon for collisions to happen a few times and play a role in the naturalness.

We needed to adapt the motion synthesis to perform abrupt movements by tuning its parameters and the training reward weights to meet these demands. On the motion synthesis side, the first step was to allow the dog to initiate any movement with higher acceleration. Afterward, the second issue dealt with was our agent's ability to make sharp turns. At full speed, our dog had a minimum radius circle when turning; that radius avoided the sliding effect usually seen on animations that are not synchronized with their spatial motion. [9]. However, for our behavior, where both the player and the dog are expected to be moving during their actions, shifting the player's attention allows us to cheat and ignore this restriction and make sharp turns without needing new animation clips. The limitation, as mentioned above, was related to our animation set, but nothing would prevent one from having more animation clips for these specific cases. Lastly, we added masking for the crouch and jump actions to synchronize them with specific target placements to avoid introducing more penalties in the reward system.

With our necessary move-set established, we can advance on the training details. As highlighted early, we set a loose collision and higher time penalty for this behavior to achieve a faster reaction of our agent. It is also noteworthy that even without the changes in the reward system, training a new model would be needed. While the

[9] The sliding effect can be very noticeable at close and medium distances when the viewer is fixed. When this effect is too strong, it negatively impacts the quality of motion and can distract the player.

previously learned model could control the dog and complete the goal, it was sloppy given the drastic changes to its motion synthesis dynamics.

The next requisite is setting the movement triggers and the target position, which are fundamental to the behavior's outcome. The first thing to notice is that we cannot directly start this behavior from the dog as we have no control over the player's actions. Still, we can take measures to make it more likely to happen, such as approaching the player and catching the player's attention. In contrast, the player triggers are more manageable; they can be based solely on the player's movement. Here we use the hand's movement relative to the body. In our case, the VR controllers provide tracking. Some cautions should be taken when handling the tracking's noise input and also taking into account the player's movement proportionally to its body height.

For the target setup, we developed a heuristic to position it in front of the player view cone according to its movement. The vital point is handling edge cases when the dog is already in the target position or too close to the player. Lastly, reducing the target collision check size and checking it only against the dog's nose (figure 5.7) solved the issue of having the dog moving in a small range and always facing forward the player.

Figure 5.7: Dog nose collider used when in play mode.

With this combination of factors, we could closely resemble the playing behavior, with the downside of having to set the triggers and targets explicitly. For our case, having control over the player triggers is desirable; nevertheless, a promising approach would be including the target positioning in the learning process as a controlling variable, either mimicking real data or with a strategic reward. Such a solution would require the additional generation of the hand movement or any other trigger used. It would be exciting to investigate generative techniques for this problem.

The last touch of this behavior was on the player's sense. Without visual and audible cues, it could be rather tricky for the player to grasp the functioning of the interaction. Adding a toy to the player's hand and audible signals (*i.e.*, dog bark) when triggering the behavior made

it straightforward to use without tutorials or explicit instructions.

6
Proof of concept scene

The previous chapter covered the behaviors with their functioning and design choices. Yet, no specific details of their combined usage on an actual application were mentioned, even though their performance in such an environment has dramatically impacted their development direction.

Here, we will address our testbed scene, which worked as a proof of concept, directing our efforts to relevant perception aspects of interacting with an intelligent agent. Indeed, we can attribute various fine-tuning to the testing done in virtual reality, which showed where our agent was under-performing and could be improved. Moreover, other aspects such as the user interface, controls, and usability are vital elements to extract all the potential intended for each behavior.

Particularly, this proof of concept scene translates into our two objectives:

- demonstrating our proposed modeling and abstractions in a functional application,
- serving as a base toolkit for our intelligent agent, which accepts further additions.

Extras such as artistic worlds and narratives would effectively give life to new applications without the need for new behaviors or drastic modifications to our game logic. Even though, nothing prevents such changes from being done, although they would require a more profound understanding of our intrinsic components.

Following, we continue with our development journal (Section 6.1) and how each new step helped shape the final result. In the Section 6.2 we detail the application features and the user interaction. Lastly, we explore how one could use our implementation on other applications and further developments biased towards artistic content.

6.1 Development journal

This section will present a kind of development journal of our agent and modeling. Up to this point, we constructed our agent in an iterative linear fashion. However, as most research is conducted, their goals and modeling evolves in a non-linear way, where one can go

back and forth in each step as the understanding of the matter gets deeper and the ideas clearer. Our process wasn't different, especially considering that the feedback from our test scene played a critical role in directing the next steps.

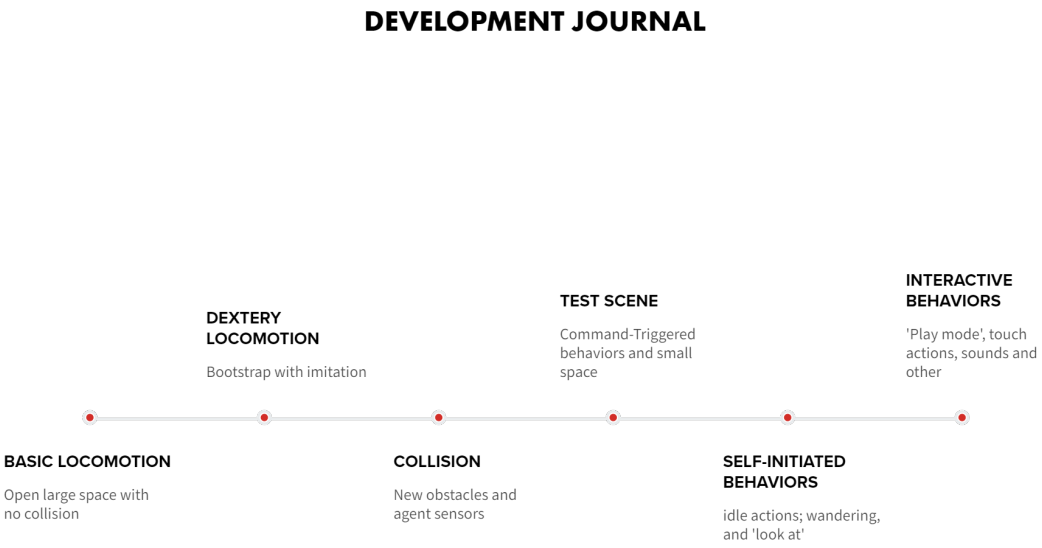

Figure 6.1 shows our simplified timeline. Starting with the basic locomotion, we defined our dog character controller, including its input interface, the animation blending, and the large base environment ($110m \times 110m$) without obstacles. In this phase, we experimented with various modeling choices to be the foundation of the upcoming development [1] . After having an operational base, it was time to try out more complex controlling with dexterous locomotion. This complexity required the addition of bootstrapping the learning process with imitation and flourished in the hulahoop jump behavior.

Our next step before being able to construct a test scene was handling collision. We needed to redesign our agent sensor and reward system for this task, leading to new training dynamics. Finding the balance demanded tuning all aspects of our agent.

Finally, with proper collision handling, we could move to our test scene composed of the command-triggered behaviors, the user interface and controls, and game logic. This scene allowed us to test and feel how our agent would perform in Virtual Reality and offered interesting insights into our environment's strengths and weaknesses. One of our first changes was adding the option for a smaller setting, and fine-tuning how the agent would position itself against the player, for better visualization and interaction. It also

[1] Much of our findings on the aforementioned phase were reported in our previous work (Souza und Velho, 2021).

brought to our attention how the other two types of behaviors (self-initiated and interactive) were crucial for the player's perception of the dog's liveness. Interestingly, while the idle actions requested more development on the game logic side, the interactive behavior was challenging on the locomotion side.

Together, those last addictions completely changed how one would perceive the interactions. It effectively pushed our agent closer to the intended AI goal of simulating life-like behavior. There is still a long way to go, for instance, working on how the agent would deal with memory and past experiences. Nevertheless, even minor details, such as the 'look at' have a significant positive impact.

6.2 *The scene: putting everything together*

In the previous section, we have seen how the development progressed, including aspects related to the player usage perspective. Here we will continue on this point of view, presenting the scene area, user controls, and behaviors.

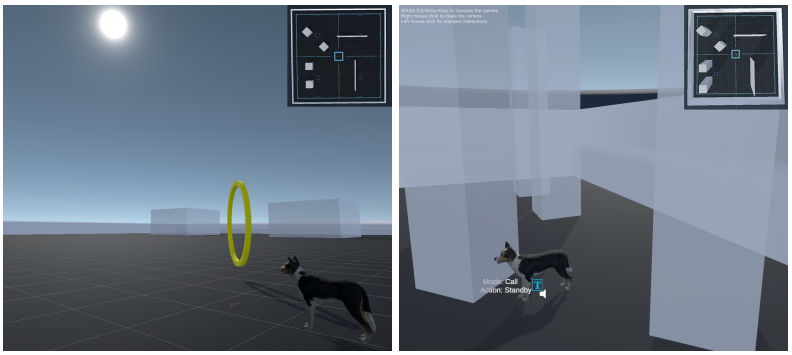

Figure 6.1: (1) Big and (2) Small environment with a similar configuration. Player and miniature top view of the

Figure 6.1, shows the large and small environment. Their visuals are simple, but they can mimic the dynamics of outdoor and indoor, respectively. In a large environment, it is interesting, for example, to play fetch and use the teleport to locomote through the map. Conversely, a smaller and more cluttered space calls for more focused interactions, such as the play mode or even looking closer and touching the dog. Interestingly, these closer interactions where the details are more evident pushed our efforts to improve the dog's behavior and movement.

Moving forward on the behaviors, figures 6.2, 6.3 and 6.4 shows a few frames of different behaviors in action. The first shows the 'fetch' where the player can grab and throw the stick around; next, the hulahoop jump with the hoop positioned by the player; and lastly, the 'look at'.

Inside the virtual reality scene, playing fetch is very similar to the real-world activity, the player needs to grab and throw the stick, and the dog reacts accordingly.

The hulahoop jump evokes a circus-like experience, where the player can position the ring anywhere. The initial idea was to let the player hold the hoop, and the dog would jump as many times

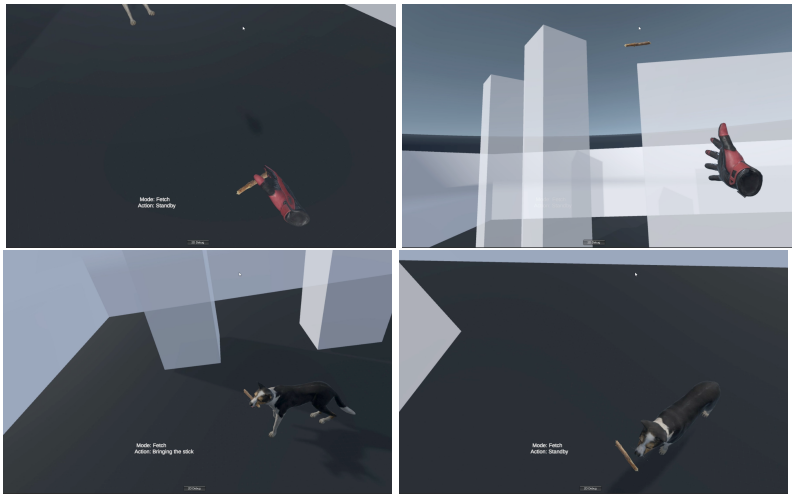

Figure 6.2: Fetch throwing stick

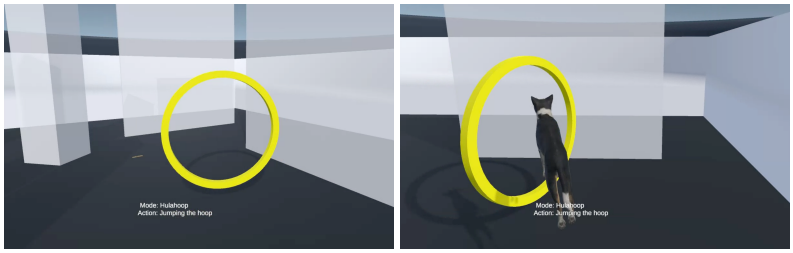

Figure 6.3: Dog jumping hoop

for the duration. However, when there is a collision, the positioning must be blocked or disabled. Given our inability to match the virtual and real space collision blocking, we opted for a safer approach to positioning them to activate the behavior to avoid weird sensations of unmatching actions and responses in VR. Figure 6.5 shows our visual indicator for bad and good positioning without blocking the player's movement.

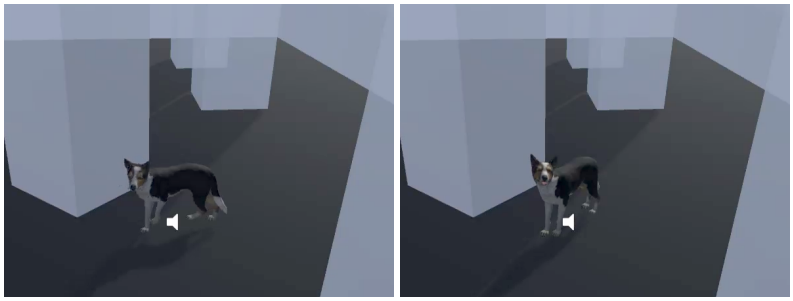

Figure 6.4: Look at

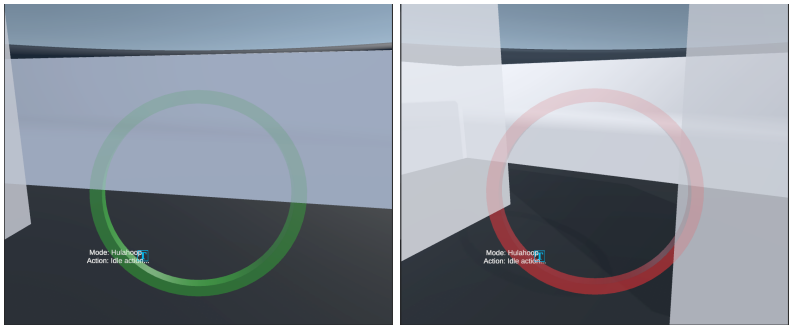

Figure 6.5: In detail, the player's visual indicator while positioning the ring.

Our default bindings for the Vive controller are shown in the figure 6.6. Figure 6.7 has the steam VR page for setting up the buttons bindings. This page will show the outline of the current controller; the "plus" on each physical button allows one to set up the action bindings.

Our feedback UI is shown in Figure 6.8, we display the current control mode and the dog's current action on it. The current available modes are: *Call*, *Fetch*, *Hulahoop* and *Play* which enables their respective behaviors through a trigger button (Note that the self-initiated behaviors can be triggered in any mode without restrictions). This choice was made specifically for the VR controller. Since a few buttons are already assigned to the player's movement, such as teleport and sharp turn, it would be easier to have a single "action button" that can trigger any action depending on the selected mode.

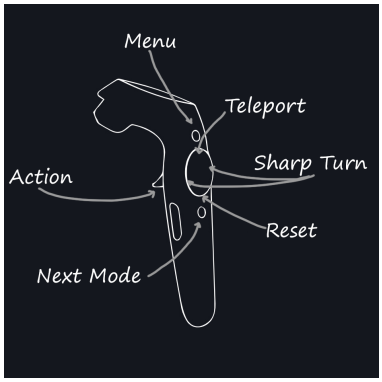

Figure 6.6: Default action bindings for the Vive controller.

Figure 6.7: VR Controller input mapping configurator. The left and the right controller can be configurated separately or mirrored, and every physical button can accept any action.

On the controller side, we have the following action defined:

- **Teleport** - uses the controller pointer to set the location to teleport,
- **Sharp Turn** - turns the camera (Left/Right) without moving the headset,
- **Next mode** - loops through the modes,
- **Reset** - resets the current dog action,
- **Action** - triggers the current mode action, which also serves as 'Grab' in the fetch mode.

Figure 6.8: In detail, the debug UI showing the current mode and dog's ongoing action.

Our scene as a toolkit

Our test scene's last perspective is seen as a toolkit and demo for other applications. Those new applications could use our behaviors together with other artistic elements. A good example would be its usage with a narrative context, including other components, such as actors. In those cases, the dog could follow a fixed script or be completely free, performing as the actor's commands (for instance, when entertaining toddlers).

Another feature that could be improved is the environment and graphics complexity of the scene. As an illustration, an outdoor set of a park could be a great place to play with the dog or help tell a narrative story. Figure 6.9 shows a possible usage of our dog in another shared VR environment as a pet.

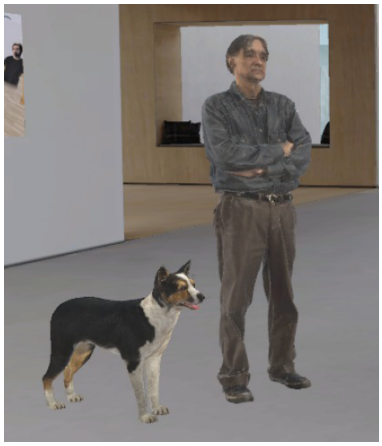

Figure 6.9: Example of usage in other application

Lastly, from an implementation viewpoint, we offer an easy way to train new behaviors. For objectives that can be modeled with collision checking, we offer a tag system, where different objects can be tagged, and each tag has its specific reward. This system allows to set new objectives or penalties and adjust their weight without coding. Additionally, if other types of observations, and pre or post-actions, are needed, they can be added through a *event delegate* allowing for extending the agent functionalities with little effort. Figure 6.10 shows the UI to setup the tags and delegates respectively.

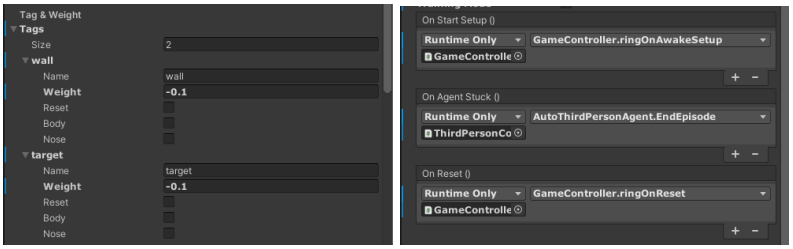

Figure 6.10: (1) Tag System with the hulahoop behavior's configuration. Each tag entry has a name, reward weight, if the agent should be reset and which agent's colliders react to it, in our case, the entire body or the nose.
(2) Event delegate system. One can set multiple different callbacks for each type of event. In the picture, we have the setup for the hulahoop behavior.

Our demo implementation is available on GitHub as a unity project and as a package; anyone can either derive their new application from our project or start a new one using the components of our package.

With these possibilities and the test scene as a demo, we believe one can effectively use it as a toolkit for developing new applications without a cold start.

7
Conclusion

In the previous chapters, we have motivated and constructed a virtual intelligent agent following our proposed hierarchical conceptualization using contemporary deep reinforcement learning techniques. Here, we will discuss the outlook of these agents in a top-down fashion, starting with our vision of their roles and future, going through our approach, and ending with down-to-earth next steps and applications.

Our vision

On the real-world automation branch, agents (virtual or not) have clear objectives, such as making a process cheaper, reliable, precise, etc. Their social impact, for instance, on human jobs, has been discussed for many years, and while there are still controversies, it is not a taboo anymore. On the other hand, virtual agents resembling living beings both in visual and cognitive aspects are flourishing just now. Moreover, there is still a taboo to be broken on human interaction with synthetic life, which we believe will be dismissed as their long-term impact and the shaping of the future unrolls.

In this regard, we believe that nowadays these agents' central role is related to communication in its simpler sense of transmitting a message, especially for media, narratives, and socio-education applications. More than the message itself is how it is transmitted. Although computational power, graphics, and machine learning technology evolved significantly with breakthroughs such as deep learning, the way we interface with computers and use them as means did not encounter the same spectrum of qualitative changes.

In general, our current means of interaction are still similar to the past century, such as using a keyboard and screen; video conferences (roughly identical to a telephone/television experience). Only recently, we could grasp a more sophisticated interaction with personal assistants (*i.e.*, Alexa) that does not appear to be a simple automated robot.

In our vision, the true breakthrough will be a qualitative change in interactivity, where the machine is not a means of communication but also take part of the communication itself. The ability to immerse in virtual and augmented reality and interact with other humans and

with artificial life agents seamlessly on a massive scale will certainly shape new ways of communication, affecting our interpersonal relations and social constructions. Its applicability in education and healthcare is among the most impactful areas with excellent potential for innovation, for instance, in education, how content is presented, or how psychological trauma is treated in healthcare.

One could argue that every quantitative development in graphics fidelity and intelligent agents until now was a necessary step for achieving the future immersion that can bring the qualitative changes mentioned early. It is also noteworthy that the same immersion and agents can only positively impact our society when well designed from both technological and ethical perspectives. Precisely because of that, the extent of its interdisciplinary is not exclusive to technical areas such as artificial intelligence or computer graphics. The types of application and research direction we choose will directly affect the shape of these virtual spaces with artificial life, be it the likes of the now trending *metaverse* or any other form it will take in the future.

With this premise, in the next section, we discuss our conceptual approach and its realization in the form of our DogBot application (using the current technology), pushing the future in the direction of our vision.

Considerations about our work

In our work, we dealt with the hierarchical construction of the agent in three levels, motion synthesis, task planning, and task selection. Although these levels cover a great variety of agents and tasks, their usage in interactive media requires attention to other details that are more subjective or hard to quantify. The explanation for this is intrinsically related to our previous discussion of communication. A simple analogy, as giving a speech is not only choosing the right words but the pace and voice tone; creating an agent able to transmit the right message (in our case, simulating a living dog) depends on choices based on perception.

Our hierarchy covers the mechanical and interactive aspects of the agent while perceptual choices guide its development process and options. Interestingly, while the challenges of learning a specific task using reinforcement learning are well defined and covered in our background chapter, achieving a behavior that performs similar to living beings still requires a tremendous human effort in design development. We believe these peculiar deep traits we try to emulate through design choices are currently in an "unlearnable" category because of two conjugated factors. First, they are not easy to quantify or qualify and hence can't be inserted into the learning and rewarding processes other than with real-world collected data. And second, they are a sub-product of the cognitive level we have not achieved yet, depending on a broad set of previous experiences and environmental factors unrelated to a specific task per se.

Overall, from another perspective, this dependency on the "human

touch" is, in our opinion, essential for art to transmit a message that may be well known but in a unique way that the artist understands and feels it. Indeed, the advancements in methods and theory approaching the cognitive level will lower the burden of the process but not substitute artistic creativity.

Regarding the understanding, we can associate our types of behaviors with the transmitter-receiver analogy in communication, with the caveat that one of the actors is an intelligent agent. The first and more straightforward case is the command-triggered actions representing a one-to-one message from the user to the dog without a sequel. Next, the self-initiated type works as a broadcast from the agent to the world. Specifically, in our case, it affects the agent's liveness perception, capturing the user's attention and opening the door for other activities. Lastly, the interactive behavior is the double direction communication where the following steps depend on the user and agent's response (actions).

Conclusively, our proof of concept scene implements examples of these behaviors and their functioning in a ludic application under our limitations and possibilities. Nevertheless, it has all the base elements to serve as a framework for applications with narratives communicating a message through the combination of behaviors and acts or even more sophisticated behaviors that can have deeper meaning individually. Another point is why an intelligent dog agent is attractive. Besides the benefits of implementation and simplicity, a virtual pet is much more approachable than a virtual human. Hence, it may work as a middle step between adopting other agents incorporating additional features and human traits.

Technical next-steps

Now going to a down-to-earth analysis of the subsequent possibilities for extending our work, there are two sides to explore. The first concerns the role of tools and applications, and the second concerns specific extensions that could be applied to our work in the near future.

On the first front, the success of these intelligent agents depends on their social acceptance and incorporation into real-world applications to become commonplace (similar to how photo filters have dominated the smartphone applications). In this direction, the Unity *ML-Agents* and the Unreal *MetaHumans* are reasonable steps on accessible tools (among others) for creating these agents. An excellent example of how good applications can popularize new technologies is the Augmented Reality Pokemon Go game that became trending a few years ago. The missing piece is still the VR headsets to be more accessible to bring immersive experiences to the masses. Nevertheless, assuming its popularization, it is not far-fetched to imagine users bringing their pets to a virtual space or creating digital twins of their real pets in this space.

Lastly, we believe our work could benefit from a few apparent

expansions from a technological and short-term perspective. For instance, having more behaviors, using a more refined motion synthesis controller (such as (Zhang u. a., 2018)), and being available in a multi-user and multi-agent space. These would certainly positively impact the user immersion and interaction. Additionally, adding the memory of previous experiences and customization on a per-user basis could improve the feeling of uniqueness for the general experience. This memory could work on affection and how likely the dog would obey a command or initiate an interaction. One possible way to explore this uniqueness could be similar to what *recommender systems* do through, for instance, collaborative filtering.

Acknowledgements

The author is partially supported by CNPq doctoral scholarships. This research was done in the *Visgraf* Computer Graphics laboratory at IMPA. *Visgraf* is supported by the funding agencies FINEP, CNPq, and FAPERJ.

Bibliography

[Agrawal und van de Panne 2016] Agrawal, Shailen ; Panne, Michiel van de: Task-based locomotion. In: *ACM Transactions on Graphics (TOG)* 35 (2016), Nr. 4, S. 1–11

[Baker u. a. 2019] Baker, Bowen ; Kanitscheider, Ingmar ; Markov, Todor ; Wu, Yi ; Powell, Glenn ; McGrew, Bob ; Mordatch, IGOR: Emergent tool use from multi-agent interaction. In: *Machine Learning, Cornell University* (2019)

[Bengio u. a. 2007] Bengio, Yoshua ; LeCun, Yann u. a.: Scaling learning algorithms towards AI. In: *Large-scale kernel machines* 34 (2007), Nr. 5, S. 1–41

[Bengio u. a. 2009] Bengio, Yoshua ; Louradour, Jérôme ; Collobert, Ronan ; Weston, Jason: Curriculum learning. In: *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, S. 41–48

[Cavazza u. a. 2002] Cavazza, Marc ; Charles, Fred ; Mead, Steven J.: Interacting with virtual characters in interactive storytelling. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, 2002, S. 318–325

[Dai u. a. 2021] Dai, Zihang ; Liu, Hanxiao ; Le, Quoc V. ; Tan, Mingxing: CoAtNet: Marrying Convolution and Attention for All Data Sizes. In: *arXiv preprint arXiv:2106.04803* (2021)

[Deng u. a. 2009] Deng, J. ; Dong, W. ; Socher, R. ; Li, L.-J. ; Li, K. ; Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09*, 2009

[Emmelkamp und Meyerbröker 2021] Emmelkamp, Paul M. ; Meyerbröker, Katharina: Virtual reality therapy in mental health. In: *Annual Review of Clinical Psychology* 17 (2021), S. 495–519

[Goodfellow u. a. 2014a] Goodfellow, Ian ; Pouget-Abadie, Jean ; Mirza, Mehdi ; Xu, Bing ; Warde-Farley, David ; Ozair, Sherjil ; Courville, Aaron ; Bengio, Yoshua: Generative Adversarial Nets. In: Ghahramani, Z. (Hrsg.) ; Welling, M. (Hrsg.) ; Cortes, C. (Hrsg.) ; Lawrence, N. (Hrsg.) ; Weinberger, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 27, Curran Associates, Inc.,

2014. – URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[Goodfellow u. a. 2014b] Goodfellow, Ian ; Pouget-Abadie, Jean ; Mirza, Mehdi ; Xu, Bing ; Warde-Farley, David ; Ozair, Sherjil ; Courville, Aaron ; Bengio, Yoshua: Generative adversarial nets. In: *Advances in neural information processing systems* 27 (2014)

[Hedberg und Alexander 1994] Hedberg, John ; Alexander, Shirley: Virtual reality in education: Defining researchable issues. In: *Educational Media International* 31 (1994), Nr. 4, S. 214–220

[Hessel u. a. 2018] Hessel, Matteo ; Modayil, Joseph ; Van Hasselt, Hado ; Schaul, Tom ; Ostrovski, Georg ; Dabney, Will ; Horgan, Dan ; Piot, Bilal ; Azar, Mohammad ; Silver, David: Rainbow: Combining improvements in deep reinforcement learning. In: *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018

[Ho und Ermon 2016] Ho, Jonathan ; Ermon, Stefano: Generative adversarial imitation learning. In: *Advances in neural information processing systems*, 2016, S. 4565–4573

[Holden u. a. 2017] Holden, Daniel ; Komura, Taku ; Saito, Jun: Phase-functioned neural networks for character control. In: *ACM Transactions on Graphics (TOG)* 36 (2017), Nr. 4, S. 1–13

[Holden u. a. 2016] Holden, Daniel ; Saito, Jun ; Komura, Taku: A deep learning framework for character motion synthesis and editing. In: *ACM Transactions on Graphics (TOG)* 35 (2016), Nr. 4, S. 1–11

[Hsu 1999] Hsu, Feng-hsiung: IBM's deep blue chess grandmaster chips. In: *IEEE micro* 19 (1999), Nr. 2, S. 70–81

[Juliani u. a. 2018] Juliani, Arthur ; Berges, Vincent-Pierre ; Vckay, Esh ; Gao, Yuan ; Henry, Hunter ; Mattar, Marwan ; Lange, Danny: Unity: A general platform for intelligent agents. In: *arXiv preprint arXiv:1809.02627* (2018)

[Kavanagh u. a. 2017] Kavanagh, Sam ; Luxton-Reilly, Andrew ; Wuensche, Burkhard ; Plimmer, Beryl: A systematic review of virtual reality in education. In: *Themes in Science and Technology Education* 10 (2017), Nr. 2, S. 85–119

[Krizhevsky u. a. 2012] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems* 25 (2012), S. 1097–1105

[Kuffner Jr 2000] Kuffner Jr, James J.: *Autonomous agents for real-time animation*, stanford university, Dissertation, 2000

[LeCun u. a. 2015] LeCun, Yann ; Bengio, Yoshua ; Hinton, Geoffrey: Deep learning. In: *Nature* 521 (2015), Nr. 7553, S. 436–444

[Lee u. a. 2018] LEE, Seunghwan ; YU, Ri ; PARK, Jungnam ; AANJANEYA, Mridul ; SIFAKIS, Eftychios ; LEE, Jehee: Dexterous manipulation and control with volumetric muscles. In: *ACM Transactions on Graphics (TOG)* 37 (2018), Nr. 4, S. 1–13

[Levine u. a. 2016] LEVINE, Sergey ; FINN, Chelsea ; DARRELL, Trevor ; ABBEEL, Pieter: End-to-end training of deep visuomotor policies. In: *The Journal of Machine Learning Research* 17 (2016), Nr. 1, S. 1334–1373

[Levine u. a. 2011] LEVINE, Sergey ; LEE, Yongjoon ; KOLTUN, Vladlen ; POPOVIĆ, Zoran: Space-time planning with parameterized locomotion controllers. In: *ACM Transactions on Graphics (TOG)* 30 (2011), Nr. 3, S. 1–11

[Lin 1992] LIN, Long-Ji: Self-improving reactive agents based on reinforcement learning, planning and teaching. In: *Machine learning* 8 (1992), Nr. 3-4, S. 293–321

[Ling u. a. 2020] LING, Hung Y. ; ZINNO, Fabio ; CHENG, George ; VAN DE PANNE, Michiel: Character controllers using motion VAEs. In: *ACM Transactions on Graphics (TOG)* 39 (2020), Nr. 4, S. 40–1

[Liu u. a. 2016] LIU, Libin ; PANNE, Michiel Van D. ; YIN, KangKang: Guided learning of control graphs for physics-based characters. In: *ACM Transactions on Graphics (TOG)* 35 (2016), Nr. 3, S. 1–14

[Lowe 1999] LOWE, David G.: Object recognition from local scale-invariant features. In: *Proceedings of the seventh IEEE international conference on computer vision* Bd. 2 Ieee (Veranst.), 1999, S. 1150–1157

[Lowe 2004] LOWE, David G.: Distinctive image features from scale-invariant keypoints. In: *International journal of computer vision* 60 (2004), Nr. 2, S. 91–110

[Luo u. a. 2021] LUO, H. ; CHEN, A. ; ZHANG, Q. ; PANG, B. ; WU, M. ; XU, L. ; YU, J.: Convolutional Neural Opacity Radiance Fields. In: *2021 IEEE International Conference on Computational Photography (ICCP)*. Los Alamitos, CA, USA : IEEE Computer Society, may 2021, S. 1–12. – URL https://doi.ieeecomputersociety.org/10.1109/ICCP51581.2021.9466273

[Luo u. a. 2022] LUO, Haimin ; XU, Teng ; JIANG, Yuheng ; ZHOU, Chenglin ; QIU, QIwei ; ZHANG, Yingliang ; YANG, Wei ; XU, Lan ; YU, Jingyi: Artemis: Articulated Neural Pets with Appearance and Motion Synthesis. In: *arXiv preprint arXiv:2202.05628* (2022)

[McCarthy 2007] MCCARTHY, John: What is artificial intelligence? (2007)

[Mnih u. a. 2013] MNIH, Volodymyr ; KAVUKCUOGLU, Koray ; SILVER, David ; GRAVES, Alex ; ANTONOGLOU, Ioannis ; WIERSTRA, Daan ; RIEDMILLER, Martin: Playing atari with deep reinforcement learning. In: *arXiv preprint arXiv:1312.5602* (2013)

[Mnih u. a. 2015] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Rusu, Andrei A. ; Veness, Joel ; Bellemare, Marc G. ; Graves, Alex ; Riedmiller, Martin ; Fidjeland, Andreas K. ; Ostrovski, Georg u. a.: Human-level control through deep reinforcement learning. In: *nature* 518 (2015), Nr. 7540, S. 529–533

[Naderi u. a. 2017] Naderi, Kourosh ; Rajamäki, Joose ; Hämäläinen, Perttu: Discovering and synthesizing humanoid climbing movements. In: *ACM Transactions on Graphics (TOG)* 36 (2017), Nr. 4, S. 1–11

[Nair u. a. 2018] Nair, Ashvin ; McGrew, Bob ; Andrychowicz, Marcin ; Zaremba, Wojciech ; Abbeel, Pieter: Overcoming exploration in reinforcement learning with demonstrations. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)* IEEE (Veranst.), 2018, S. 6292–6299

[Nakada u. a. 2018] Nakada, Masaki ; Zhou, Tao ; Chen, Honglin ; Weiss, Tomer ; Terzopoulos, Demetri: Deep learning of biomimetic sensorimotor control for biomechanical human animation. In: *ACM Transactions on Graphics (TOG)* 37 (2018), Nr. 4, S. 1–15

[North und North 2016] North, Max M. ; North, Sarah M.: Virtual reality therapy. In: *Computer-assisted and web-based innovations in psychology, special education, and health*. Elsevier, 2016, S. 141–156

[Pathak u. a. 2017] Pathak, Deepak ; Agrawal, Pulkit ; Efros, Alexei A. ; Darrell, Trevor: Curiosity-driven exploration by self-supervised prediction. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, S. 16–17

[Peng u. a. 2017] Peng, Xue B. ; Berseth, Glen ; Yin, KangKang ; Van De Panne, Michiel: Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. In: *ACM Transactions on Graphics (TOG)* 36 (2017), Nr. 4, S. 1–13

[Pierson und Gashler 2017] Pierson, Harry A. ; Gashler, Michael S.: Deep learning in robotics: a review of recent research. In: *Advanced Robotics* 31 (2017), Nr. 16, S. 821–835

[Rothbaum u. a. 1997] Rothbaum, Barbara O. ; Hodges, Larry ; Kooper, Rob: Virtual reality exposure therapy. In: *Journal of Psychotherapy Practice & Research* (1997)

[Russell und Norvig 2002] Russell, Stuart ; Norvig, Peter: Artificial intelligence: a modern approach. (2002)

[Schaal 1999] Schaal, Stefan: Is imitation learning the route to humanoid robots? In: *Trends in cognitive sciences* 3 (1999), Nr. 6, S. 233–242

[Schaul u. a. 2015] Schaul, Tom ; Quan, John ; Antonoglou, Ioannis ; Silver, David: Prioritized experience replay. In: *arXiv preprint arXiv:1511.05952* (2015)

[Schulman u. a. 2015a] SCHULMAN, John ; LEVINE, Sergey ; ABBEEL, Pieter ; JORDAN, Michael ; MORITZ, Philipp: Trust region policy optimization. In: *International conference on machine learning* PMLR (Veranst.), 2015, S. 1889–1897

[Schulman u. a. 2015b] SCHULMAN, John ; MORITZ, Philipp ; LEVINE, Sergey ; JORDAN, Michael ; ABBEEL, Pieter: High-dimensional continuous control using generalized advantage estimation. In: *arXiv preprint arXiv:1506.02438* (2015)

[Schulman u. a. 2017] SCHULMAN, John ; WOLSKI, Filip ; DHARIWAL, Prafulla ; RADFORD, Alec ; KLIMOV, Oleg: Proximal policy optimization algorithms. In: *arXiv preprint arXiv:1707.06347* (2017)

[Senju und Johnson 2009] SENJU, Atsushi ; JOHNSON, Mark H.: The eye contact effect: mechanisms and development. In: *Trends in cognitive sciences* 13 (2009), Nr. 3, S. 127–134

[Simon 1991] SIMON, Herbert A.: The architecture of complexity. In: *Facets of systems science*. Springer, 1991, S. 457–476

[Souza und Velho 2021] SOUZA, Caio ; VELHO, Luiz: Deep Reinforcement Learning for Task Planning of Virtual Characters. In: *Intelligent Computing*. Springer, 2021, S. 694–711

[Sutton 1988] SUTTON, Richard S.: Learning to predict by the methods of temporal differences. In: *Machine learning* 3 (1988), Nr. 1, S. 9–44

[Sutton und Barto 2018] SUTTON, Richard S. ; BARTO, Andrew G.: *Reinforcement learning: An introduction*. MIT press, 2018

[Terzopoulos 1999] TERZOPOULOS, Demetri: Artificial life for computer graphics. In: *Communications of the ACM* 42 (1999), Nr. 8, S. 32–42

[Tobin u. a. 2017] TOBIN, Josh ; FONG, Rachel ; RAY, Alex ; SCHNEIDER, Jonas ; ZAREMBA, Wojciech ; ABBEEL, Pieter: Domain randomization for transferring deep neural networks from simulation to the real world. In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)* IEEE (Veranst.), 2017, S. 23–30

[Torabi u. a. 2018] TORABI, Faraz ; WARNELL, Garrett ; STONE, Peter: Behavioral cloning from observation. In: *arXiv preprint arXiv:1805.01954* (2018)

[Velho und Alevato 2022] VELHO, Luiz ; ALEVATO, Bernardo: Humanos Digitais e Avatares / VISGRAF Lab - IMPA. 2022 (TR-02-2022). – Technical Report

[Vinayagamoorthy u. a. 2006] VINAYAGAMOORTHY, Vinoba ; GILLIES, Marco ; STEED, Anthony ; TANGUY, Emmanuel ; PAN, Xueni ; LOSCOS, Céline ; SLATER, Mel: Building expression into virtual characters. (2006)

[Watkins 1989] Watkins, Christopher John Cornish H.: Learning from delayed rewards. (1989)

[Wooldridge 1999] Wooldridge, Michael: Intelligent agents. In: *Multiagent systems* 6 (1999)

[Wooldridge und Jennings 1995] Wooldridge, Michael ; Jennings, Nicholas R.: Intelligent agents: Theory and practice. In: *The knowledge engineering review* 10 (1995), Nr. 2, S. 115–152

[Yu u. a. 2018] Yu, Wenhao ; Turk, Greg ; Liu, C K.: Learning symmetric and low-energy locomotion. In: *ACM Transactions on Graphics (TOG)* 37 (2018), Nr. 4, S. 1–12

[Zell u. a. 2019] Zell, Eduard ; Zibrek, Katja ; McDonnell, Rachel: Perception of virtual characters. In: *ACM Siggraph 2019 Courses*. 2019, S. 1–17

[Zhang u. a. 2018] Zhang, He ; Starke, Sebastian ; Komura, Taku ; Saito, Jun: Mode-adaptive neural networks for quadruped motion control. In: *ACM Transactions on Graphics (TOG)* 37 (2018), Nr. 4, S. 1–11