Little Learning Machines: Real-Time Deep Reinforcement Learning as a Casual Creativity Game

Dante Camarena[*,†], Nick Counter[†], Daniil Markelov, Pietro Gagliano, Don Nguyen, Rhys Becker, Fiona Firby, Zina Rahman, Richard Rosenbaum, Liam A. Clarke and Maria Skibinski

Transitional Forms Inc., Toronto ON, Canada

Abstract

In this paper, we present Little Learning Machines, a groundbreaking game that enables players to take on the role of a reinforcement learning (RL) trainer. Utilizing reward and environment modeling, players train miniature robots to perform tasks, creating an open-ended space for exploring and crafting behavior. Notably, the game introduces innovative methods for executing RL in near real-time, a significant stride in the field. We delve into the technical challenges and solutions encountered in implementing a robust and dynamic simulation for this RL platform. This paper focuses on a system description, while pointing to potential avenues for enhancements and expansions to further enrich the player experience, as well as opportunities for additional research from player feedback. This pioneering game not only demystifies RL but also serves as a versatile tool for learning, research, and creativity in the realm of artificial intelligence.

Keywords

Deep Reinforcement Learning, Educational Games, Video Game, Interactive Learning, Artificial Intelligence, Neural Networks, Game Development, Unity Game Engine, Casual Creativity

1. Introduction

Is it possible for a video game to make the intricate field of Deep Reinforcement Learning (Deep RL) accessible to all? From outperforming humans in intensive tasks [1] to managing complex systems [2, 3] and refining extensive language models[4], RL has definitely demonstrated its adaptability and potential.

Despite these advancements, RL continues to be a complex topic that is often taught with a strong emphasis on theory.[5]. On the other end of the spectrum, popular science approaches show an overly simplified description of the RL procedure that provides a powerful intuition, but struggles in communicating the iterative process and pitfalls of experiments in the field. [6]. This is further complicated by the technical complexity of setting up an environment, and may be overwhelming for someone new to the field, creating an entry barrier that is hard to overcome. Interestingly, when a typical person is asked to describe RL, they often liken it to the process of training a dog, a concept most people are familiar with as it is centered around a the strong central metaphor of Reward.[7]

In this paper, we introduce **Little Learning Machines**. A game that streamlines the process of training Deep RL agents. In this game, players are introduced to a voxelized world in which they produce and train **Animo**: robots that behave autonomously. The player's primary mode of interaction with this world is through the behaviour of their Animo.

Figure 1: An Animo petting a dog.

The game is primarily intended to provide players with a casual creativity experience. [8] This is to say, players of Animo are given a rich environment with a variety of interactions to allow them to train agents and gain autotelic enjoyment from creating and fostering new behaviours. The game is further developed to allow measured exploration of the full capabilities of the space, providing a gradual unlock and reveal system similar to that found in games like *Minecraft* [9] and *Wobbledogs*[10]. Finally, the game tells a story about the difficulty of raising little

AIIDE Workshop on Experimental Artificial Intelligence in Games, October 08, 2023, University of Utah, Utah, USA

[*] Corresponding author.

[†] These authors contributed equally.

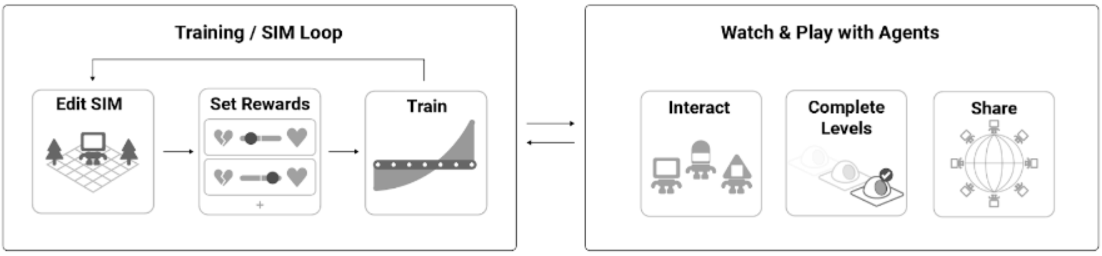

Figure 2: The Two main parts of gameplay

creatures, an interesting parable that allows lighthearted analysis of stochastic (random and unpredictable) agents.

Through this experience, we intend on making RL accessible and engaging for everyone, regardless of their technical background. Initial Playtesting has shown mastery and engagement in users ranging from Middle School to Post-Secondary. By providing players with a practical set of tools for training their agents, they can gain first-hand knowledge about the impact of their decisions on the agent's behavior. This leads to an intuitive understanding of RL concepts such as reward functions, state and action spaces, and exploration vs exploitation trade-offs. However, these results were collected in the middle of iterative and intensive game development and are not the focus of this paper. We expect to conduct additional research on player testing and feedback in future work. What follows is is a system description of the project as well as analysis of its construction.

2. Background

Our studio first ran into Reinforcement Learning while developing *Agence*[11], a co-production with the Canadian National Film Board, in which we trained reinforcement learning agents that were the main characters of our film. Instead of directly programming them, we defined a set of objectives and used RL to allow them to develop behaviours on their own. It took us a lot of experimenting to understand the effects of rewards and environment on our agents. However, as our team became familiar with the nuances of training agents, our agents' behaviour began to feel more engaging, and we began to enjoy the experience.

Throughout production of that film, we became attached to each iteration of our little agents. We were delighted to watch them overcome small obstacles, and develop curious behaviours. They would often find creative and unique ways to approach the challenges we gave them. Everyone in our team, including illustrators, producers and marketing were beginning to develop hypotheses and curiosities about training agents. Their ideas, including specific reward structures or environment curricula, often led to breakthroughs in behaviour that outpaced the work of our engineers alone. As a result, we built tooling to help them become more involved in the training process.

After years of work, this tooling evolved into a game which is the subject of this paper, *Little Learning Machines.*

2.1. Prior work

It is important to acknowledge the previous applications of training reinforcement learning models to evolve characters in games. Notable work released includes classic life-sim game Creatures (1996)[12] as well as god-game Black and White(2001) [13]. Creatures provides the player with a variation of characters to play with, but often the mechanisms underlying their behaviour are not immediately obvious to novice players. Conversely, Black & White provides a single creature to train, but the simplicity of the feedback mechanism may limit player expresivity. More recent work includes projects such as ArtBot[14] that focuses on teaching fundamental AI concept, and familiarizing the player with the process of training.

Little Learning Machines attempts to bridge the gaps across these games. It does so by focusing its gameplay on an explicit training/testing loop, providing the player a rich environment where they may attempt to train many different behaviours and a variety of agents to compare and contrast, all the time guiding the player through each step.

3. Gameplay

The gameplay of Little Learning Machines is designed to replicate the process of solving a Reinforcement Learning problem. As such it's divided into two steps, **Training in a Simulation** ("The Cloud") and **Performing in the Real World** ("The Islands"). The player assumes the role of an RL trainer, and their primary task is to train their

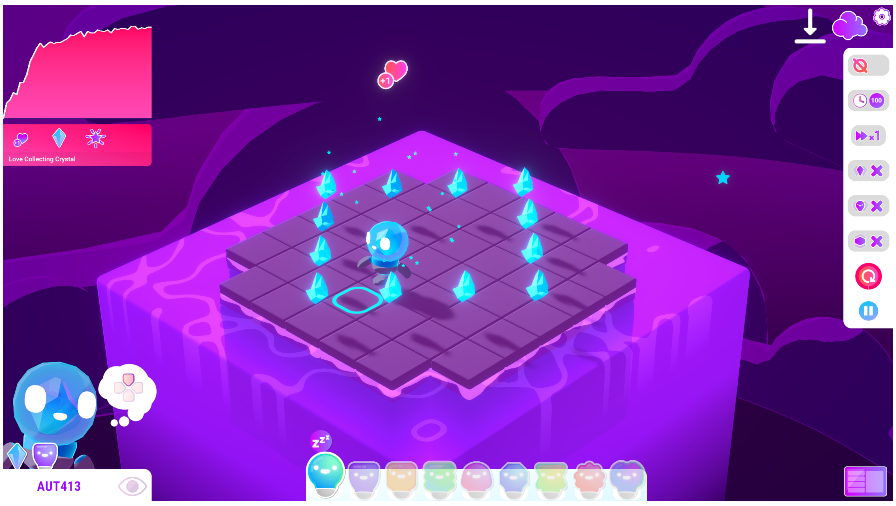

Figure 3: A screenshot of Little Learning machines during training

Animo. Players may set up environments with multiple Animo. While this slows down training and places more strain on the user's computer, it can create fun experiments where agents develop competitive behaviour.

As the Animo perform a variety of tasks in the Real World, the player gains access to new items and more Islands. This loop was instrumental in helping motivate players to explore new potential for their Animo, gradually introduce new mechanics, techniques and items, as well as provide them with a helpful balance between novelty and focus.

3.1. Training Process

3.1.1. Building an Environment

To train an agent, the player must visit a special place called "The Cloud" in which agents train. In here, the player may construct small environments to train specific behaviours. The player may modify the environment terrain by adding or removing blocks. They can also place items on a 2.5d grid, (items may have a vertical position on the map, but tunnels, bridges and overhangs are not allowed). The environment influences how the Animo learns, adapts, and behaves, adding an additional layer of strategy and personalization to the gameplay.

3.1.2. Setting Rewards

After constructing an environment and placing their Animo, the player is then required to set a reward. The reward-setting interface is described in detail below. In short, the player can set up rules to automatically provide positive or negative rewards for any events that can occur in the simulation. This allows the player to incentivize or dis-incentivize any action the Animo may take. Examples include: Collecting a Crystal, Standing still, Lighting a tree on fire, Hugging another Animo, etc.

Positive rewards that the player provides are referred to as *Love* and Negative rewards that the player provides are referred to as *Fear*. Once the player starts training, each step the Animo takes results in particles floating out of the Animo's head to indicate receiving a specific reward. In the case of receiving a negative reward, a slight shock animation makes a subtle nod to Skinner's Operant Conditioning Mechanism [15].

3.1.3. Configuring Resets

As the last step before starting training, the user can configure the environment reset. Here the player can indicate how often the training resets. The player can indicate a number of steps or a condition such as: all crystals collected or all flowers watered. The player can also apply a slight amount of randomization to their training environment, preventing the network from over-fitting [16, 17] to specific configurations of items in a level.

3.1.4. Observing training

Once the rewards are set, the player can begin training. While the neural network is trained in a background process, the player can observe an example of their "training cloud" simulated in front of them. As the network updates, the behaviour of the character in front of the player improves. The player is able to stop and continue training at any time. This allows players to reflect on their Animo's performance in real time and adjust the rewards

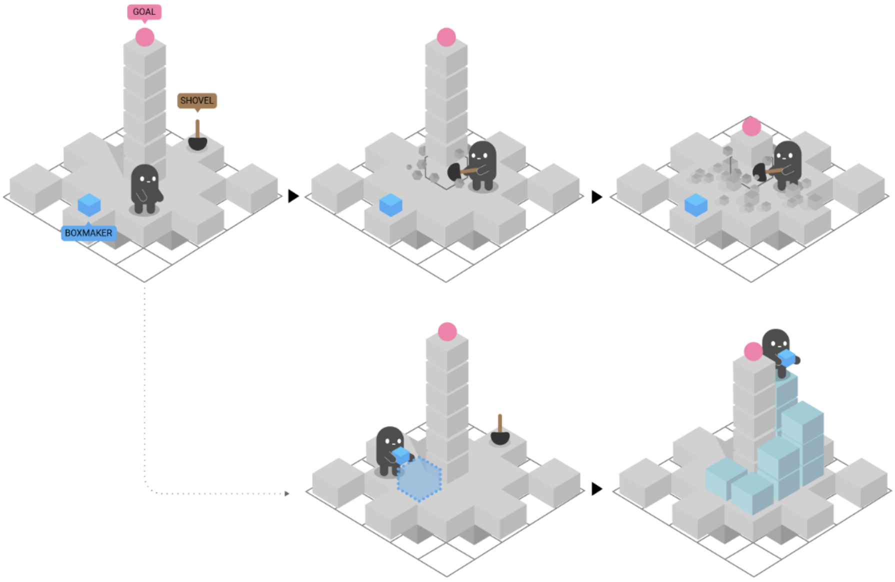

Figure 4: A quest may have more than one solution. In this diagram, the Animo may decide to use a shovel to dig blocks and lower the goal, while another approach may involve placing blocks to climb onto the goal.

accordingly. If the players aren't satisfied with their Animo's learning, they can tweak the rewards and observe how these changes affect the Animo's behavior.

This iterative process imitates the real-world application of RL, where iterative experimentation and incremental adjustments are key to successful learning. As a result, the player performs the role of a training curriculum, requiring them to set a goal that is just advanced enough for the robot to be able to learn. Players are free to continue training their existing model with different rewards and parameters. Continual training [18] is a way of exploring Animo's skill transferability and adaptability.

3.1.5. Resetting Animo

Due to a number of reasons such as: Catastrophic forgetting[19], Reaching Local Minumum[16, 17], Network Collapse[20] or Neuron Deactivation[21], the network may be unable to recover and continue learning. A player is faced with a difficult choice of having to reset the network of their Animo. This results in a freshly initialized network. This feature ensures that players always have a way to re-calibrate their strategies and try different approaches to training their Animo.

3.2. Exploration

3.2.1. Tutorial

To begin, players start out with a single Animo, on an empty island with just sand and crystals. The player is greeted by an enthusiastic teacher named *Imogen* who guides the player through the process of training agents. The first agent the player trains has to navigate a simple grid world and collect crystals. Due to the unorthodoxy of the training process, having Imogen explain some of the concepts was essential to providing players the right mindset to approach training.

3.2.2. Quest Islands

After the tutorial, The player encounters a set of islands, each thematically different. In each island, the player may find a new set of items, new Animo and Quests to complete. Quests help provide objectives for players who may need a bit more direction. In order to complete a quest, An Animo is required to perform a specific set of actions without User interaction.

Examples of Quests include: Chop 5 trees, throw the ball at the dog 3 times, collect all these crystals without stepping on any flowers, etc.

The player may not make direct modifications to Quest Islands (although they could train behaviours to do so), resulting in some items being out of reach until certain conditions are met on the island. Once items are within reach of an Animo, they may be clicked to be added to the user's inventory. Any item they have encountered may now be used in their training, allowing them to create infinite copies of the item on the cloud.

3.2.3. Home Island

As the player explores each island, they may bring home Items, NPCs and Costumes from other islands. This allows players to create spaces for their Animo to play and interact with one-another. The player is also able to train their Animo to complete large projects and permanently change the look of their main island.

4. Environment

Animo is set in a voxel grid environment reminiscent of popular games like *Into the Breach*[22] and *Crypt of Necrodancer*[23]. This choice serves to simplify the environment dynamics, enabling real-time training of agents. Players are able to construct their training environments by changing grid size, cell heights, and placing unlocked objects on the grid before starting the training in the "Cloud", however they are currently unable to interact with the environment during training.

The Environment described here was primarily designed to be extremely easy to discretize and perceive, while providing the most richness and extensibility. Our objective was to create a baseline of generic interactions that could support variety of dynamics for agent interactions.

4.1. Terrain

The game world is a heightmap-based grid, with no tunnels or bridges that would complicate the navigation. Each grid cell has a height, with cells beneath the water line being classified as either shallow or deep water, based on depth below the water line. This straightforward layout allows players to focus on the core gameplay mechanics, particularly training the RL agent to interact with objects that exist on grid cells. Each grid cell can contain objects of different types that can be broadly categorized into **Actors**, **Mediums**, and **Items** (holdable and non-holdable).

4.2. Objects

4.2.1. Items

Holdable items can be grabbed, dropped and often used by actors. Examples of such items are shovels, which decrease the height of the block in front of Animo upon use, or an axe that can be used to destroy various objects.

Non-holdable items are entities that can be interacted with in different ways. For example Tree Fruits can be planted and turned into a tree sprout, which then can be watered to produce a Tree or Fruit Tree, that in turn produces Tree fruits, completing the loop. Only a single holdable or non-holdable object can occupy a cell at a time.

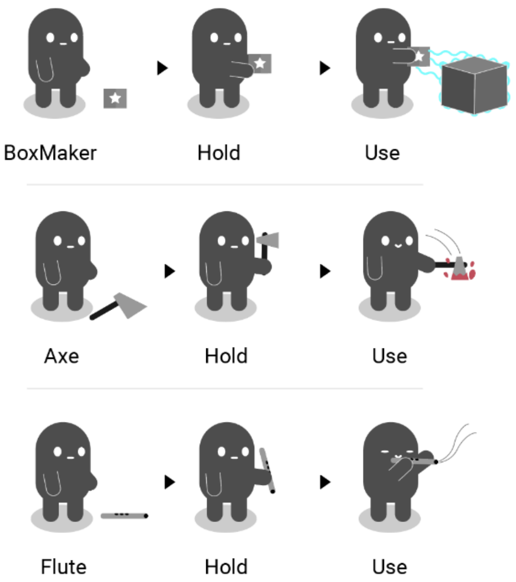

Figure 5: Despite items having different effects, their interface is the same: Grab and use.

4.2.2. Actors

Actors include: Animo, Autonimo, Dogs, Snowpals, etc. All actors outside of Animo use traditional Hierarchical State Machine based AI to perform their behaviours. This provides a sharp contrast to neural network behaviour as during the first few parts of the game, they appear smarter than Animo. However, as the game progresses, it becomes obvious that their programmed behaviours are deterministic, and unable to adapt. A single actor can exist on a given cell and can hold a single item at the time. Actors can use objects to manipulate the world or to create new objects. Animo and Autonimo have 7 actions that they can attempt to perform: wait, move forward, turn left, turn right, turn back, grab, and use. Animo can not move to a block higher than 1 above height, but can move to any height below their current cell's. Upon stepping on the cell with deep water they become helpless and do not take actions until they get out of it. The grab action allows Animo to pick up or exchange a held object with an object on the same cell. Use action can either use the held item or an object on the cell in front of Animo.

4.2.3. Mediums

Mediums are objects that do not prevent other objects from being placed on the cell that they occupy. Fire or paint are examples of such objects. Mediums are primarily used as modes of interaction between items.

4.3. Intents

Each simulation step, objects generate **Intents**. Intents are chains of actions that are evaluated simultaneously and deterministically. As intents are executed, each intent modifies a local range of grid cells. This allows the simulation to be executed in parallel using spacial partitioning. Intents that get executed in order of their priority, potentially producing new intents. Intents with similar priorities that affect overlapping regions produce conflicts. Possible intent conflicts get resolved according to predefined rules (e.g. none of the move intents attempting to move onto the same cell will get executed). Simulation step completes when there are no intents that need to be executed. This results in a turn-based simulation, where the chain of causation of intents remains preserved, allowing players to set up rewards for events that were caused only by specific Animo's actions. This is important for reward attribution in environments that feature multiple Animo training simultaneously.

5. Reward UI

Finding a way to give players control over rewards was particularly difficult. Original drafts of the game design had Animo requiring resources such as food or batteries, and had training rewards be derived from these systems. However, we found that such a system would limit the kinds of behaviours that the players could create.

We initially provided players with a menu with every interaction in the game, but navigating that menu quickly became overwhelming. We found an elegant solution that shows specific interactions by limiting the options on screen to the set of items available. Furthermore, the introduction of Iconography allowed easier access for younger players, non-English speakers and reading-impaired play testers.

6. Animo observations

Animo's ability to interact with its environment and make decisions is largely dependent on what it can observe and how it observes it. One of the main challenges of crafting sensors is to find a balance between providing enough information for the agent to make meaningful decisions, and not overwhelming it with too much information. Taking advantage of representing the environment in a way that is complementary to Animo's network architecture is important for efficient training. Animo's perception is designed to be pluggable, meaning that they can have any number of sensors, describing different parts of the environments independently of each other. The game comes with 9 unlockable Animo, each with their own unique way of perceiving the world around them.

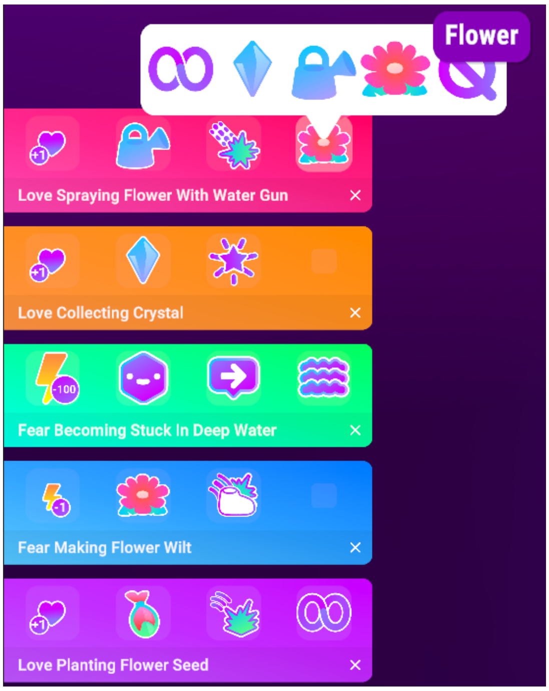
Flower
+1
Love Spraying Flower With Water Gun
×
+1
Love Collecting Crystal
×
-100
Fear Becoming Stuck In Deep Water
×
-1
Fear Making Flower Wilt
×
+1
Love Planting Flower Seed
×

Figure 6: Players use iconography to set their Animo rewards

6.1. Vector Sensors

Vector sensors are one-dimensional and do not preserve any data structure. They perceive every observed value independently, which makes them quick learners. However, they suffer from the curse of dimensionality and quickly lose performance as the number of observations grows. Compass sensors are an example of Vector sensors that are designed to inform the agent about the direction to the nearest item of player's choice. This sensor allows the agent to be aware of objects that are out of range of other sensors.

6.2. Convolutional Sensors

Convolutional sensors [24] are designed to perceive image-like data and use convolutions to exploit patterns in the observations. They might be slightly slower to run, but they learn kernels that extract specific information from the observation. They perceive a patch of the grid around the agent, rotated towards the direction that the agent is facing. Each perceived cell is parsed into relevant information that usually consists of terrain (height, ground type, occupancy), actor at cell, item held by an actor at cell, medium, and object on ground.

6.3. Attention Sensors

Attention sensors [25] perceive groups of values, called entities. This enables them to perceive a variable number of observations and, while they are significantly slower to run, their ability to learn is impressive, albeit fragile. Attention sensors perceive a number of objects that are closest to Animo. This perception system has the advantage of only focusing on objects instead of cells, which allows for a more compact representation compared to the Animo Grid Sensor.

6.4. Object Encodings

The inputs to each sensor are required to encode the type of objects in their perception as a part of state observation. One-hot encoding is a common method of approaching this task, that scales poorly as the number of object types increases. Our sensor system allows to easily switch between one-hot encoding, hand-crafted object properties, binary, and ternary object type Id encodings, allowing players to use the perception system suitable for the task at hand.

6.5. Sensor Conclusions

After conducting numerous experiments, we found that our initial hypotheses for each sensor were somewhat misguided. Just including more information was insufficient for better Animo performance. While we would often see improvement in performance relative to total training steps, this would come at a cost of increased inference time or slower model updates. This resulted in similar wall clock time for training of all three sensor types. That said, there are nuanced differences in the performance of each sensor type. Animo behaviour is qualitatively different from one another. The ease of swapping between these configurations has allowed for better analysis. Enthusiast players of the game may themselves make tweaks to sensor configurations and perhaps be able to create further conclusions.

7. Technical Architecture

Animo is one of the first games to allow players to train Deep RL agents within a game. However, the process of training DRL Agents can takes hours, Resulting in it being slow and un-engaging. In order for training itself to be engaging, it was essential to show improvement in the agents as fast as possible. However, fast techniques such as Q-Learning or Tabular learning were unable to perform at the same level as Proximal Policy Optimization.

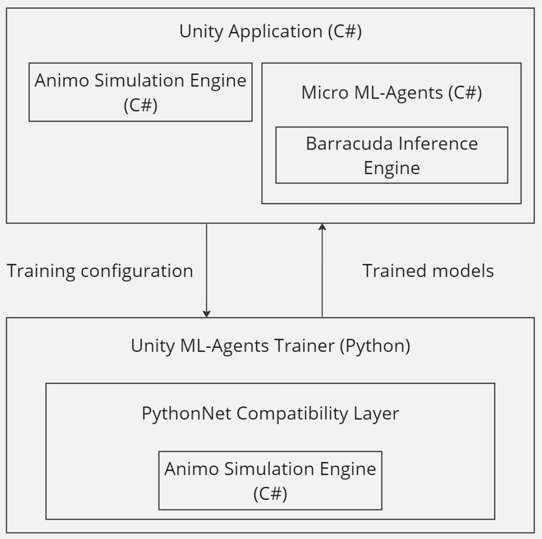
Unity Application (C#)
Animo Simulation Engine (C#)
Micro ML-Agents (C#)
Barracuda Inference Engine
Training configuration
Trained models
Unity ML-Agents Trainer (Python)
PythonNet Compatibility Layer
Animo Simulation Engine (C#)

Figure 7: A diagram of the architecture of Little Learning Machines

That said, implementation of such algorithms within engine would be a significant undertaking, and one that we would have a hard time adapting into a game. As a result, we had to perform significant engineering to leverage existing RL infrastructure. Our application makes use of an embedded python engine to access PyTorch to train agents during gameplay. All training is local to the player's computer and has shown good performance even on hardware as dated as a Surface Pro 3.

The training architecture is designed with wall clock training time being the priority. The training process is built around the modified version Unity ML-Agents, a powerful tool for training intelligent agents to be used with Unity Game Engine.[26] ML-Agents provides a flexible platform that supports multiple training algorithms, including Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO). Unity also provides a multi-platform inference engine called Barracuda, which allows us to inference models without causing disruption to the user framerate.

While early prototypes used ML-Agents directly, and modified its internals to support runtime training, recent versions of Little Learning Machines have been modified to use only a small fraction of ML-Agents, and train without the Unity Executable. The main vehicle for this approach is the development of the **Animo Simulation**, which allows us to run simulations without running the game. This eliminates the overheads of running the Unity Engine and gRPC communication protocol to generate sample trajectories during training, that is usually the case with the standard ML-Agents implementation.

As a result, The Little Learning Machines Project consists of 4 layers:

- A Unity Project, (the game) layer that allows us to use Unity Editor for development.
- Micro ML-Agents unity package, a lightweight fork of ML Agents' C# side package.
- Animo Simulation, a standalone C# dll.
- Animo Trainer A custom Python trainer, that uses PythonNet to interface with Animo Simulation directly.

The Animo Simulation is designed to be self-sufficient, allowing for potential use with other training algorithms. We have conducted several experiments with DreamerV3 [27], a new model-based training algorithm, to show that Animo Simulation can be used as a configurable benchmark environment for testing various training algorithms. During training, our modified ML-Agents Python package periodically exports current models and training statistics that are conveniently displayed within the game. This architecture, along with python, PyTorch and the rest of the dependencies are installed automatically during game setup.

Finally, the simulation uses a pluggable architecture, allowing for the easy design and implementation of new Objects, Sensors, NPCs, Reward functions and Training Algorithms. We intend on presenting the Animo Simulation as a viable benchmark for new RL algorithms in a separate conference, as it provides novel means of evaluation for said algorithms.

Micro ML-Agents can be reviewed here: https://github.com/transformsai/micro-ml-agents. Animo Trainer Simulation can be installed here: https://pypi.org/project/animo-trainer/. Both packages require additional documentation.

8. Discussion and Conclusion

Ultimately, Little Learning Machines is a bridge between culture and research. It's an experiential platform, allowing users to not just understand RL in theory, but gain an intuitive grasp of its peculiarities through exposure and experimentation. It's a shared platform for people to experiment and share different perspectives on RL. As a small sample of the kinds of communities that could interact with it, one can imagine:

- RL Researchers can test and benchmark new RL Algorithms
- Enthusiast Players can fiddle with sensors and hyperparmeters
- Modders can add new Items, objects and other content in Mods
- Creative players can come up with new environments, challenges and tests for their Animo
- Competitive players can push the performance of these algorithms to their best.
- Educators can use the platform to experientially demonstrate peculiarities of reinforcement learning.

Above all, Little Learning Machines is the easiest introduction to the addictive process of training your own artificially intelligent agents. It exposes players to a loop of training and observing ever smarter models, the delightfully frustrating process of iterating on experiments and the joy of seeing the models break through plateaus and traps. The game requires no prior experience, requires no math (except a bit of graph literacy) and no coding experience. It even installs python and PyTorch for you. It is the most straightforward way to experience reinforcement learning. And it does so not just by having you set the experiments and look at graphs, but by letting you see the Animo learn in front of your eyes. We hope that it's an inspiration for generations to come. While the technical potential of the project can be quite exciting, it is at the end of a day, a game made with care and attention, ideally enjoyed ludically by a small group of individuals.

Acknowledgments

In memoriam of Anuj Patel, this project would not be possible without his hard work.

Thanks to the following people for their support during production: Alexander Bakogeorge, Casey Bluestein, Chloe West, David Oppenheim, Erin Ray, Eve Cuthberson, Kory Mathewson, Manal Siddiqui Pablo Samuel Castro. Thanks to Level Curve Inc for their help with Audio and Music: Eliza Daly, Matt Miller, Robby Duguay. Thanks to Durham College for their advice and support: Khris Finley, Richa Thomas, Ryan Miller, Yuqi ”Stanley” Zhou, Dina Samaha, Dr. Vibha Tyagi, Tejas Vyas, Saba Siddiqi, Sharath Kumar

Thanks to the following people for their support to the project: Adam Myhill, Darren Throop, Euro Beinat, Kevin West, Paul Van Der Boor, Peter Vuong, Priya Ratti, Victor Nguyen, Vivian Gagliano. This project was possible thanks to generous funding support from the Canada Media Fund and from Ontario Creates.

References

[1] OpenAI, :, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, S. Zhang, Dota 2 with large scale deep reinforcement learning, 2019. arXiv:1912.06680.

[2] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, Z. Wang, Autonomous navigation of stratospheric balloons using reinforcement learning, Nature 588 (2020) 77–82.

[3] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas, et al., Magnetic control of tokamak plasmas through deep reinforcement learning, Nature 602 (2022) 414–419.

[4] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, R. Lowe, Training language models to follow instructions with human feedback, 2022. arXiv:2203.02155.

[5] M. Morales, Grokking deep reinforcement learning, Manning Publications, 2020.

[6] C. Course, Y. Bisk, J. Ashe, Reinforcement learning: Crash course ai 9, 2019. URL: https://www.youtube.com/watch?v=nIgIv4IfJ6s.

[7] D. Silver, S. Singh, D. Precup, R. S. Sutton, Reward is enough, Artificial Intelligence 299 (2021) 103535.

[8] K. Compton, M. Mateas, Casual creators., in: ICCC, 2015, pp. 228–235.

[9] Mojang, Minecraft, Videogame, 2011. URL: https://minecraft.net.

[10] T. Astle, Animal Uprising, Wobbledogs, Videogame, 2022. URL: https://wobbledogs.com/.

[11] P. Gagliano, C. Blustein, D. Oppenheim, Agence, a dynamic film about (and with) artificial intelligence, in: ACM SIGGRAPH 2021 Immersive Pavilion, 2021, pp. 1–2.

[12] S. Grand, D. Cliff, A. Malhotra, Creatures: Artificial life autonomous software agents for home entertainment, in: Proceedings of the first international conference on Autonomous agents, 1997, pp. 22–29.

[13] Lionhead Studios, Black & White, Videogame, 2001.

[14] M. Zammit, I. Voulgari, A. Liapis, G. N. Yannakakis, The road to ai literacy education: from pedagogical needs to tangible game design, Academic Conferences International, 2021.

[15] B. F. Skinner, Reinforcement today., American Psychologist 13 (1958) 94.

[16] A. Zhang, N. Ballas, J. Pineau, A dissection of overfitting and generalization in continuous reinforcement learning, arXiv preprint arXiv:1806.07937 (2018).

[17] C. Zhang, O. Vinyals, R. Munos, S. Bengio, A study on overfitting in deep reinforcement learning, arXiv preprint arXiv:1804.06893 (2018).

[18] K. Khetarpal, M. Riemer, I. Rish, D. Precup, Towards continual reinforcement learning: A review and perspectives. arxiv, arXiv preprint arXiv:2012.13490 (2020).

[19] P. Kaushik, A. Gain, A. Kortylewski, A. Yuille, Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping, arXiv preprint arXiv:2102.11343 (2021).

[20] V. Kothapalli, Neural collapse: A review on modelling principles and generalization, 2023. arXiv:2206.04041.

[21] G. Sokar, R. Agarwal, P. S. Castro, U. Evci, The dormant neuron phenomenon in deep reinforcement learning, 2023. arXiv:2302.12902.

[22] Subset Games, Into The Breach, Videogame, 2018. URL: https://subsetgames.com/itb.html.

[23] Brace Yourself Games, Crypt of the Necrodancer, Videogame, 2015. URL: https://braceyourselfgames.com/crypt-of-the-necrodancer/.

[24] S. Albawi, T. A. Mohammed, S. Al-Zawi, Understanding of a convolutional neural network, in: 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1–6. doi:10.1109/ICEngTechnol.2017.8308186.

[25] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, I. Mordatch, Emergent tool use from multi-agent autocurricula, 2020. arXiv:1909.07528.

[26] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, D. Lange, Unity: A general platform for intelligent agents, 2020. arXiv:1809.02627.

[27] D. Hafner, J. Pasukonis, J. Ba, T. Lillicrap, Mastering diverse domains through world models, 2023. arXiv:2301.04104.