# Unity ML Agent: Introduction to Reinforcement Learning

1st Soheil Shirvani
*Department of Computer Science*
*University of California Riverside*
Riverside, CA, USA
sshir030@ucr.edu

***Abstract*—In this report, I present my experiments using Unity ML-Agents on a selection of example environments, highlighting the use of different input modalities and action spaces. I explored discrete environments such as Gridworld, which uniquely utilized image inputs, and Basic, both of which were successfully solved using Deep Q-Network (DQN) and Double DQN (DDQN). In contrast, continuous environments like 3DBall and Worm demonstrated the effectiveness of Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC), with Worm yielding notable results despite limited training resources and 3DBall being fully solved. Although additional environments like Crawler and Push Block were attempted, their inherent complexity combined with limited training did not allow for acceptable results. The report also discusses the capabilities of the Unity ML-Agent package, which provides functionalities to define environments, agent interactions, and reward functions based on actions within the environment. Utilizing these tools, I developed a custom environment where an agent must move left or right to catch falling bricks for rewards, showcasing the flexibility and potential of ML-Agents in creating tailored scenarios. This report concludes as my final project report for Introduction to Reinforcement Learning (RL) Course.**

***Index Terms*—Reinforcement Learning, Unity ML Agent**

## I. Summary of Contributions

This report presents a comprehensive study on implementing and tuning various RL algorithms such as DQN, DDQN, SAC, and DDPG across different Unity ML-Agent environments [1], each posing unique challenges and requiring tailored approaches. Here is a summary of the core contributions from the study:

1) **Adaptation of Reinforcement Learning Algorithms:** Successful adaptation and tuning of DQN and DDQN for discrete environments, and SAC and DDPG for continuous environments. This demonstrates the flexibility of these algorithms in managing various types of action spaces and state inputs.
2) **Performance Benchmarks Across Environments:**
   - **Basic:** Achieved a reward of 0.93, approaching the benchmark of 1.0, demonstrating the effectiveness of the algorithms in simple scenarios.
   - **3DBall:** Attained performance close to the perfect score of 100, showcasing excellent control and stabilization skills in a continuous task.
   - **Gridworld:** Reached an average reward of 0.6 across multiple runs with different seeds, against a benchmark of 0.8, indicating proficient handling of high-dimensional image inputs.
   - **Worm:** Managed a highest score near 100, significantly below the benchmark of 800, reflecting challenges in complex continuous environments with limited resources.
   - **Crawler:** Excluded from this report due to the lack of positive outcomes, highlighting the need for more robust training strategies in highly dynamic environments.
3) **Custom Environment Development:** Designed and implemented a novel environment where an agent can only move left, stay, or move right to catch falling bricks, with each brick caught increasing the reward by one. The agent receives image input of the environment, testing the algorithms' capabilities to handle spatial judgments and reward-based decision making.
4) **Empirical Validation:** Extensive testing in these environments not only validates the general applicability of the chosen algorithms but also underscores their potential for customization based on the specifics of the task. The findings contribute valuable insights into the strengths and limitations of each algorithm within different contexts.

## II. Introduction and Related Works

**Reinforcement Learning and Unity ML-Agent Simulation Environment.** RL is a pivotal area of machine learning where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards [2]. This approach has been widely applied across various domains including robotics, games, and autonomous driving, where the ability to learn complex behaviors is crucial [3], [4]. Unity Technologies has significantly contributed to this field with its Unity ML-Agents Toolkit, which provides a rich simulation environment for training and evaluating RL agents [5]. This platform supports a variety of environments that can be tailored to mimic real-world scenarios, thereby offering researchers a versatile tool to develop, test, and refine their RL algorithms in controlled yet complex settings [6].

**Reinforcement Learning Algorithms and Markov Decision Processes.** RL is fundamentally based on the framework of Markov Decision Processes (MDPs), where an agent's decision-making process is modeled as a sequence of states, actions, and rewards [7]. Classic RL algorithms like Q-learning and its variants, such as DQN, have set foundational principles for learning optimal policies. More recent advances include algorithms like DDQN which addresses overestimation of Q-values, and Actor-Critic methods such as SAC and DDPG, which balance the benefits of policy-based and value-based approaches [8]. These modern algorithms enhance the agent's ability to learn efficiently in environments with continuous action spaces and high-dimensional state inputs, pushing the boundaries of what can be achieved through autonomous decision-making.

**Applications and Limitations in Real-World Settings.** While many studies have demonstrated the ability of RL agents trained in simulation environments like Unity ML-Agents to perform complex tasks, transferring these models to real-world applications often presents significant challenges [9]. Although successful implementations exist, such as robots performing navigation and manipulation tasks [10], these models frequently suffer from issues related to generalization, scalability, and uncertainty in dynamic and unstructured environments. The performance of RL agents is often tightly coupled with the specifics of the training environment, leading to a degradation in effectiveness when faced with novel scenarios. [11] Therefore, there is a continuous need for developing methodologies and simulation environments like unity ml agent toolkit that enhance the robustness and adaptability of RL models to ensure their practical applicability in varied and unpredictable real-world situations.

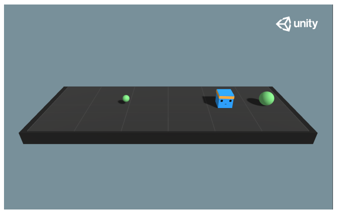
## III. Results

In this section, I detail the outcomes achieved across various Unity ML-Agent environments, including Basic, 3DBall, GridWorld, and Worm, utilizing algorithms such as DQN, DDQN, SAC, and DDPG. For each environment, I discuss the specific states, actions, and rewards, alongside the implementation details and the results obtained.

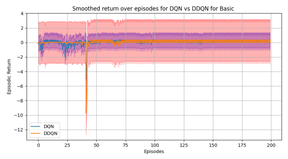
### A. *Basic.*

In the Basic environment, a relatively straightforward task is set for the agent, focusing on linear movement with the objective of maximizing rewards by navigating left or right. This environment encapsulates the essence of reinforcement learning by simplifying the decision-making process to a fundamental level. The agent, which is the sole entity in this scenario, is tasked with moving towards the most rewarding state, which is evaluated using a discrete set of possible actions: moving left, remaining stationary, or moving right. The reward function is designed to penalize unnecessary movement with a -0.01 penalty for each step taken, while providing incremental rewards of +0.1 and +1.0 for reaching suboptimal and optimal states, respectively.

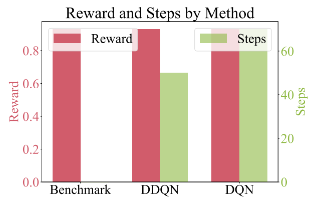
The simplicity of the Basic environment, defined by a single vector observation for the agent's current state and no visual observations, enables a straightforward evaluation of the reinforcement learning algorithms employed, namely DQN and DDQN. These algorithms were effectively used to train the agent in discerning and performing the actions that maximize rewards. The set benchmark mean reward for this environment is 0.93, acting as a performance metric for the agent. In the course of the training, both DQN and DDQN were finely tuned to strike a balance between exploring new strategies and exploiting known rewarding actions, aiming to meet or exceed this benchmark. The comprehensive results analysis reveals how these algorithms facilitated the agent's learning and strategy optimization in this minimalistic setting. The outcomes of implementing DQN and DDQN in this scenario are further illustrated in the Figure 1.



(a) Basic Environment Image



(b) Results of implimenting DDQN and DQN for Basic environment



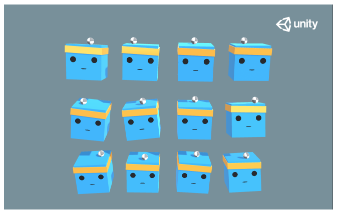
(c) Final mean Reward of last 10 episode

Fig. 1: Results for Basic Environment

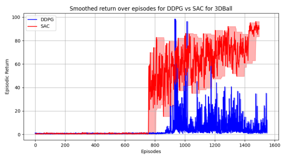
An interesting observation in the Basic environment is the impact of exploration randomness on the agent's learning behavior. In several training runs, agents initially gravitated towards the smaller rewards due to their relative ease of attainment compared to the optimal state. Since receiving a small reward is more beneficial than wandering aimlessly or remaining stationary, the agent learns to consistently achieve this lesser reward. This behavior underscores the necessity for multiple exploratory attempts to reach the larger reward, often requiring a strategic balance of exploration and exploitation influenced by the randomness and initial seed settings. This phenomenon highlights the crucial role of randomness and exploration strategies in reinforcement learning, illustrating how they can significantly affect the convergence outcomes and the overall learning trajectory of an agent.

### B. *3DBall.*

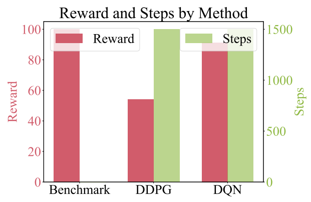
In the 3DBall environment of the Unity ML-Agents toolkit, agents face the challenge of balancing a ball on their heads, a task that not only tests their control mechanisms but also their ability to maintain stability under varying conditions. This environment, which includes 12 agents of the same kind operating under identical behavior parameters, offers a rich platform for evaluating the nuances of reinforcement learning



(a) 3D Ball Environment Image



(b) Results of implementing SAC and DDPG for 3D Ball environment



(c) Final mean Reward of last 10 episode

Fig. 2: Results for 3D Ball Environment

algorithms, particularly in scenarios that demand continuous action decisions. Each agent must continuously adjust its rotations along the X and Z axes to keep the ball from falling, with the reward structure providing +0.1 for every step the ball remains balanced and a penalty of -1.0 if the ball falls off. The complexity of the task is further augmented by variations in the physical properties of the ball such as scale, gravity, and mass, which can be adjusted to modify the difficulty and dynamics of the task.

The agents utilize two continuous actions to achieve their objective, with the observation space providing detailed feedback on their own rotation as well as the position and velocity of the ball. This setup not only mimics a real-world balancing task but also allows for the exploration of advanced control strategies facilitated by deep reinforcement learning algorithms like DDPG (Deep Deterministic Policy Gradient) and SAC (Soft Actor-Critic). These algorithms, designed to handle continuous action spaces effectively, were implemented to optimize the agents' performance in maintaining balance under various configurations of ball physics. The results of implementing DDPG and SAC, showcasing their effectiveness in managing the continuous control necessary for this environment, are detailed in the Figure 2 for the 3DBall environment.

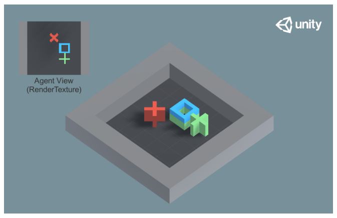
The training duration lasted approximately 3-4 hours for each algorithm. During these sessions, both SAC and DDPG demonstrated strong performance by successfully achieving the benchmark goal of 100 rewards. However, SAC consistently outperformed DDPG, showcasing its superior efficiency and effectiveness in this specific task. Despite these achievements, the training can lack of generalizability in the learned behaviors. Further training, potentially extended to around 10 hours, might allow the models to develop more robust and adaptable strategies, potentially enhancing their performance. All trainings were done on my labtop using Nvidia RTX 2060 GPU.

## C. *GridWorld.*

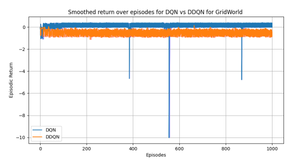
In the GridWorld environment, agents are tasked with a complex navigation challenge within a structured grid-like setting that incorporates multiple goals and obstacles. The environment is designed to test the agents' ability to discern and reach the correct goal while avoiding incorrect ones and navigating around obstacles. Each agent operates under identical behavior parameters, with a reward system that deducts a small penalty for every step taken (-0.01) to encourage efficiency, provides a significant positive reward for reaching the correct goal (+1.0), and imposes a severe penalty for mistakes (-1.0 when an incorrect goal is reached, ending the episode). The agents do not use vector observations but rely on visual observations from a top-down perspective of the grid. This setup includes action masking, which is essential for effectively managing the discrete action space that encompasses five potential movements: the four cardinal directions plus the option to remain stationary. This mechanism ensures that the agents only consider viable moves at any given point, enhancing the strategic depth of the environment.

For this project, modifications were made to the original GridWorld environment to streamline the challenge and reduce complexity. Specifically, the goal was standardized to always be green, and the incorrect or penalty-inducing goals were designated as red. This alteration simplifies the decision-making process for the agents, as they now only need to identify and move toward green goals, avoiding the red ones. Furthermore, the environment was adjusted to solely provide image inputs, eliminating vector observations to focus the learning process on visual cues. This change aims to more closely mimic real-world scenarios where agents must rely on visual information to make decisions. The results of these modifications and the impact on the agents' ability to navigate the GridWorld are illustrated in the Figure 3, showcasing the effectiveness of the training and the strategic capabilities developed by the agents in this simplified context.

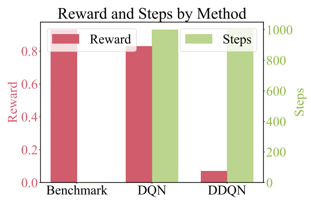
In the modified GridWorld environment, the training sessions were notably lengthy, extending to approximately 10 hours, primarily due to the complexity introduced by image-based state inputs and the necessity of convolutional layers to process these visual cues effectively. Despite these extended training efforts, the results fell short of the benchmark expectations. This shortfall is largely attributed to the requirement for a considerably larger set of experiences to refine the agents' decision-making capabilities effectively. A significant limitation in this context was the inability to run multiple agents in parallel, which restricted the rate of experience accumulation—a crucial factor for efficient training in complex environments. Additional training sessions, even when extended, consistently yielded similar outcomes, underscoring the challenges posed by the low quality of the top-down state images, which further complicates the agents' ability to discern and navigate effectively towards the goals. The simplified goal colors, intended to ease this challenge, unfortunately did not lead to results comparable to benchmark. The



(a) GridWorld Environment Image



(b) Results of implementing DQN and DDQN for GridWorld environment



(c) Final mean Reward of last 10 episode

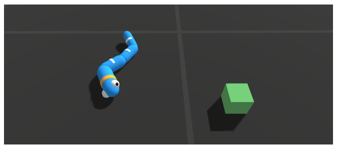
Fig. 3: Results for GridWorld Environment

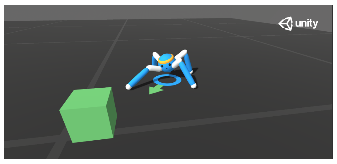
performance constraints highlighted the inherent difficulties in training robust models in visually complex and dynamically challenging environments, particularly when relying on single-agent setups.

## D. *Worm, Crawler*

In the Worm environment, agents control a segmented worm with a head and three body parts, aiming to navigate effectively towards a goal. The environment hosts 10 identical agents, each operating under a unique geometric reward function that multiplies individual rewards, thereby encouraging the agents to optimize for all possible rewards rather than focusing on the easiest ones. This setup promotes comprehensive skill development over simple task execution. The agents' performance is assessed based on the alignment of the body's velocity and direction with the goal, both normalized between 0 and 1. Agents interact through a complex set of continuous actions, involving up to 9 target rotations for the joints, and rely on a detailed vector observation space of 64 variables that track positional, rotational, and dynamic states of each limb. Despite these intricacies, the benchmark mean reward stands ambitiously at 800. An image of the environment is given in Figure 4.

The Crawler environment represents a more challenging extension of the Worm setup, featuring a creature with four arms and forearms designed to move toward a specified direction without falling. Similar to Worm, it employs a geometric reward function to drive the behavior of 10 agents, fostering a focus on maximizing comprehensive rather than partial success in actions. The agents must coordinate 20 continuous actions targeting joint rotations to achieve alignment between the head's direction and the goal velocity, again normalized for comparability. The environment's complexity is heightened by an extensive vector observation space comprising 172 variables, significantly increasing the challenge with a benchmark


(a) Worm Environment Image



(b) Crawler Environment Image

Fig. 4: Unsolved Crawler and Worm Continuous Environments

mean reward set at 3000. An image of the environment is given in Figure 4.

Despite rigorous training efforts spanning 10 hours, neither SAC nor DDPG algorithms managed to produce satisfactory results in the Worm and Crawler environments. The primary difficulty lies in the vast continuous action space, which complicates the learning process significantly. This vastness often leaves the algorithms appearing uncertain of effective strategies even after extensive training, highlighting a fundamental limitation in their current capabilities. Both environments' reliance on a single agent per training instance exacerbates this issue by severely limiting the amount of experience that can be gathered in any given time, slowing down the training efficiency.

The challenges of these environments are further compounded by their intricate dynamics and the need for precise control over multiple joints and limbs. Each agent's performance heavily relies on its ability to interpret a large array of sensory inputs and translate them into a coherent set of actions, a task that grows exponentially difficult with the increase in the number of controllable elements. The single-agent setup not only hinders rapid data collection but also restricts the diversity of interactions and scenarios an agent can learn from, delaying its ability to generalize and adapt to new or slightly altered tasks. This setup underscores a crucial gap in current training methodologies, which struggle to scale effectively with the complexity of the task environments such as Worm and Crawler, where strategic exploration and robust generalization are key to achieving high performance.

## E. *Push Block*

The Push Block environment presents a platforming challenge where an agent is tasked with manipulating a block to a designated goal area. This environment involves a single agent capable of executing a variety of actions, including turning clockwise or counterclockwise, moving in four distinct directions, or remaining stationary. The agent's decision-making process is informed by a continuous vector observation space consisting of 70 variables derived from 14 ray-casts, each detecting one of three possible objects: a wall, the goal, or the block. The reward function is designed to incentivize efficiency and success: the agent incurs a small penalty of -0.0025 for every step taken to encourage swift completion of the task, and gains a significant reward of +1.0 if the block successfully contacts the goal. Additional environmental factors

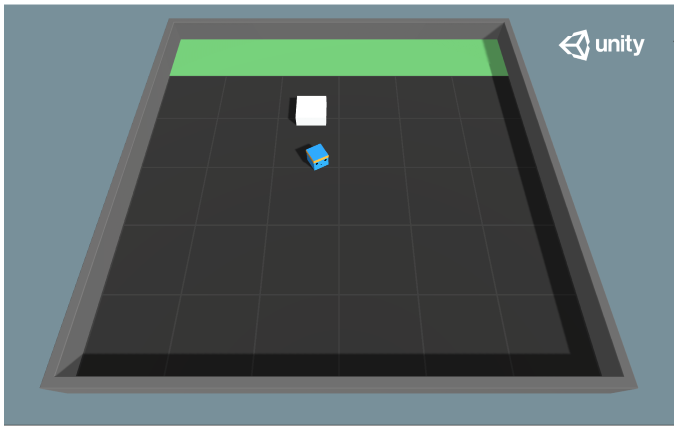


Fig. 5: Push Block Environment Image

such as block scale, dynamic and static friction coefficients, and air resistance (block drag) can be adjusted to modify the challenge's complexity, with a benchmark mean reward set at 4.5. Figure 5 illustrate the environment.

Solving the Push Block environment proves particularly challenging due to the large dimensionality of the observation space and the complexity of the task. The agent must interpret 70 continuous state variables to navigate effectively, a task complicated by the scarcity of positive reinforcement. Initially, it takes approximately 5-6 hours of training before the agent even manages to touch the block and receive a positive reward. This rarity of positive feedback significantly hampers the learning process, as the agent struggles to associate actions with beneficial outcomes. The vast majority of actions lead to negligible or negative outcomes, making it difficult for the agent to learn productive behaviors.

To enhance the learning efficiency in the Push Block environment, several adjustments could be considered. Improving the reward function to include incremental positive feedback as the agent moves towards the block could significantly aid in teaching the agent the desired behavior. This method of rewarding could provide more frequent and immediate feedback, helping to shape the agent's actions more effectively. Additionally, experimenting with more sophisticated algorithms such as Proximal Policy Optimization (PPO) might offer better performance due to its ability to handle large action spaces and sparse rewards more effectively than simpler methods like DQN or DDQN. Implementing curriculum learning, where the task difficulty is gradually increased as the agent's performance improves, could also be beneficial. Such strategies would not only speed up the initial learning phase but also encourage the development of more generalizable skills, potentially leading to better performance and faster convergence to optimal strategies.

## IV. Custom ML Agent Environment

Creating custom training environments using the Unity ML-Agents Toolkit allows for the development of intelligent agents tailored to specific tasks such as non-player character behavior

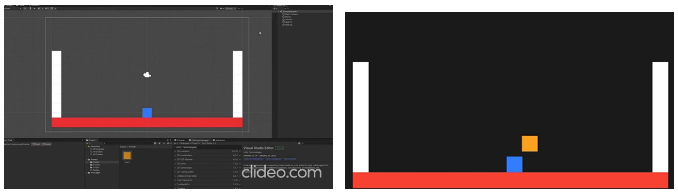


Fig. 6: Custom Environment: Bricks Game

in games, simulation-based training, or complex decision-making processes in controlled scenarios. These custom environments enable precise control over the challenges and tasks presented to agents, ensuring that the learning process is closely aligned with desired outcomes, whether for entertainment, educational, or research purposes. By designing targeted training scenarios, developers can enhance the interactions within digital environments, leading to improved system efficiency and user experience.

In the Unity ML-Agents framework, the creation and training of agents revolve around extending the `Agent` class and implementing its core methods to define agent behavior. These methods include `OnEpisodeBegin()`, for initializing or resetting the environment at the start of each episode; `CollectObservations(VectorSensor sensor)`, for gathering environmental data that the agent uses to make decisions; `OnActionReceived()`, where the agent executes actions and rewards are assigned based on the outcomes; and `Heuristic()`, for manual control or testing using heuristic-driven decisions. The ML-Agents `Academy` class orchestrates the simulation, ensuring smooth communication between the Unity environment and the external Python training scripts, managing the lifecycle of episodes, and optimizing the agent's policy based on performance.

A practical example of this setup can be seen in an environment like 'Ball3DAgent', where the agent's task is to balance a ball on a platform. Here, specific implementations of the `Agent` methods manage the environment's dynamics—resetting the platform and ball, collecting tilt and velocity data, and adjusting the platform's angle to keep the ball balanced. Such environments not only facilitate the development of specific agent behaviors but also allow for the iterative refinement of these behaviors through continuous testing and training, ultimately leading to agents that perform effectively and adaptively in their designated roles.

In the custom environment I developed, the agent operates within a 2D platform where it must navigate left, right, or remain stationary to catch falling bricks from above, enhancing its interaction and decision-making skills under dynamic conditions. The environment is visually represented as a simple 2D image that serves as the state input to the agent, coupled with three discrete action choices. I designed two distinct reward functions to train the agent effectively: the first function awards a +1 reward for each successful catch and a -1.1 penalty for missing a brick, aiming to strongly discourage misses, while the second function offers a +0.3 reward for every step the agent correctly positions itself to catch the next falling brick, addressing the issue of reward scarcity by promoting consistent good positioning. The episode ends after 100 steps, which typically includes around 10 brick falls, creating a compact and challenging scenario for the agent to learn efficient and precise movements. Figure 6 illustrate my custom design implementation.

Unfortunately, the training sessions for this custom environment were not completed to satisfaction. Similar to the challenges faced in the GridWorld scenario, the necessity of using convolutional layers to process the image inputs in both DQN and DDQN architectures significantly extended the training duration. After approximately six hours of training, the results were still suboptimal, suggesting that a more extended training period, possibly around 20 hours, might be required to achieve satisfactory results. Preliminary projections indicate that with sufficient training time, the agent could potentially reach a score of +10 in the first reward setup or similarly high scores in the second reward configuration, given the enhanced logic and continuous reward structure it employs. This extended training would allow for a more thorough optimization of the neural network parameters, potentially leading to more effective learning outcomes.

## V. Limitations and Future Works

**Limitations.** This study faced several limitations that impacted the depth and scope of the experiments conducted. Primarily, the limited availability of resources, particularly GPUs, and the constraint of training time, which did not extend beyond a week, significantly restricted the potential for extensive training sessions necessary for environments like Crawler and Push Block. Additionally, the generalization capabilities of the models remain untested, introducing uncertainties about how well these agents would perform outside the controlled settings of their training environments. Without simulations bridging the simulation-to-reality gap, the real-world applicability of these trained models cannot be confidently asserted, representing a critical limitation in predicting their effectiveness in practical scenarios.

Furthermore, the integration of Unity ML-Agents with the gym wrapper poses an additional limitation, as it only supports wrapping one agent per environment. This is a significant drawback for two reasons. Firstly, it restricts the possibility to test environments involving collaborative tasks or competitive two-player games, which could have provided valuable insights into the dynamics of multi-agent interactions. Secondly, while the default configuration in ML-Agents can run multiple instances (often up to ten) of the same agent simultaneously to expedite experience gathering and accelerate training, the gym wrapper's limitation to a single agent significantly slowed the training process. This restriction not only increased the training time but also limited the amount of diverse experiences the agent could learn from simultaneously, making the training less efficient and potentially less effective.

**Future Works.** Looking ahead, there are several promising directions for expanding this research. Implementing advanced reinforcement learning algorithms such as Asynchronous Advantage Actor-Critic (A3C) and Proximal Policy Optimization (PPO) could potentially enhance the performance and efficiency of the training processes [12], [13]. Furthermore, introducing new environments, such as Food Collector, along with more custom scenarios, will help in assessing the robustness and adaptability of the algorithms. A critical aspect of future work will involve testing the generalization of the implemented algorithms to unseen test cases, which is essential for validating their practical utility. Additionally, efforts should be made to bridge the sim-to-real gap, aiming to conduct simulations that more closely mimic real-world conditions, thus providing a more reliable basis for evaluating the feasibility of transferring simulated training into real-world applications.

## VI. Conclusion

This report has thoroughly explored the application of various reinforcement learning algorithms within the Unity ML-Agents framework, successfully solving Basic and 3DBall, partially solving GridWorld, and facing significant challenges with Worm, Crawler, and Push Block due to the complexities of these environments and the need for more extensive training resources. The experiments employed DQN and DDQN for discrete environments and SAC and DDPG for continuous environments, showcasing the versatility of these methods across a complete set of environments featuring both image and vector states. Furthermore, the creation of a custom ML agent environment demonstrated the potential of Unity ML-Agents in developing intelligent systems, though the solution to this environment remained elusive within the project's timeframe. This comprehensive study has highlighted the practicality and applicability of reinforcement learning agents and the usefulness of Unity ML-Agents in training them. However, the transition from simulation to real-world application presents future challenges, as the sim-to-real gap must be addressed to verify the uncertainty and real-world viability of the developed models, suggesting a significant avenue for future research.

## Acknowledgment

I would like to extend my deepest gratitude to Professor Yu, whose expertise and insightful lectures have profoundly enriched my learning experience this quarter. I am also immensely thankful to our teaching assistant, whose support, discussions, and well-crafted assignments were instrumental in translating complex theories into practical knowledge. This quarter has been a wonderful journey, thanks to your incredible dedication and support.

## References

[1] M. Mattar, J. Shih, V. Berges, C. Elion, and C. Goy, "Announcing ml-agents unity package v1. 0."

[2] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[3] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[4] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2021.

[5] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar *et al.*, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018.

[6] M. Lanham, *Learn Unity ML-Agents–Fundamentals of Unity Machine Learning: Incorporate new powerful ML algorithms such as Deep Reinforcement Learning for games*. Packt Publishing Ltd, 2018.

[7] M. A. Wiering and M. Van Otterlo, "Reinforcement learning," *Adaptation, learning, and optimization*, vol. 12, no. 3, p. 729, 2012.

[8] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[9] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *arXiv preprint arXiv:1904.12901*, 2019.

[10] H. Hu, K. Zhang, A. H. Tan, M. Ruan, C. Agia, and G. Nejat, "A sim-to-real pipeline for deep reinforcement learning for autonomous robot navigation in cluttered rough terrain," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6569–6576, 2021.

[11] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," in *International conference on machine learning*. PMLR, 2019, pp. 1282–1289.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[13] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative multi-agent games," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 611–24 624, 2022.