Maze Solver with Reinforcement Learning using Reward Shaping in Unity

Bruno Costa, Pedro Rodrigues
Master in Data Science and Engineering
Faculdade de Engenharia da Universidade do Porto (FEUP)
Email: {up202004966@up.pt, up201906003@up.pt}
Code Repository: https://github.com/brumocas/Maze_Solver_RL

***Abstract*—Reinforcement Learning (RL) has shown significant promise in addressing complex decision-making problems. This research investigates the application of RL to dynamic maze navigation using Unity's ML-Agents toolkit, focusing on Proximal Policy Optimization (PPO) and reward shaping for continuous tasks. The study evaluates agent performance across three scenarios: (1) a basic square environment where the agent uses raycast-based sensing to learn target-seeking behavior, (2) an intricate maze requiring sequential navigation through multiple intermediate targets, and (3) a dynamic single-target maze demanding spatial awareness and adaptive strategies. Experimental results demonstrate that RL agents can effectively adapt to varying environmental complexities but reveal challenges with raycast-based sensing in highly dynamic settings. These findings offer valuable insights into developing robust RL agents for navigation in both virtual simulations and practical real-world applications.**

***Index Terms*—Reinforcement learning (RL), Proximal Policy Optimization (PPO), Reward Shaping, Unity, Continuous Control.**

I. Introduction

DYNAMIC navigation in maze-like environments presents a challenging problem for autonomous agents, requiring the ability to process sensory inputs and adapt to changes in real-time. Traditional methods often struggle to provide robust solutions under varying conditions. Reinforcement learning (RL), by enabling agents to learn optimal strategies through interaction with their environment, offers a promising framework for addressing such challenges. Among RL algorithms, Proximal Policy Optimization (PPO) [1] has gained significant attention for its stability and effectiveness in continuous control tasks.

This paper explores the application of PPO to dynamic maze navigation tasks using Unity's ML-Agents Toolkit [2], a versatile platform for simulating and training RL agents in virtual environments. We investigate three training scenarios: (1) a simple square environment where the agent learns basic target-seeking behavior using raycasts, (2) a progressively complex maze requiring the agent to locate multiple intermediary targets before reaching the final goal, and (3) a dynamic maze where the agent directly navigates to a single target.

A key feature of our approach is the use of reward shaping, a technique designed to enhance the agent's learning process by providing continuous feedback, improving convergence speed, and enabling more effective exploration of the environment. Experimental results highlight the strengths and limitations of the proposed approach, providing insights into the design of RL agents for complex navigation tasks in dynamic settings.

II. Contributions

The main contributions of this work are summarized as follows:

- **Application of PPO for Dynamic Maze Navigation:** Proximal Policy Optimization (PPO) is employed to train an autonomous agent for navigating dynamic mazes demonstrating the algorithm's effectiveness in complex and dynamic navigation tasks in a continous control.
- **Integration of Reward Shaping:** Reward shaping is incorporated into the training process to enable faster convergence and improve learning efficiency by providing additional guidance during the agent's learning process.
- **Development of Diverse Training Scenarios:** Three distinct training scenarios of increasing complexity are designed and evaluated, ranging from a simple square environment to a dynamic complex maze, showcasing the adaptability of the agent to different challenges.
- **Comprehensive Performance Evaluation:** A systematic analysis of agent performance across all scenarios is provided, offering insights into the challenges and potential solutions for dynamic maze navigation using RL.
- **Implementation Using Unity ML-Agents Toolkit:** The study leverages Unity and its ML-Agents Toolkit [2] to create a flexible simulation environment for the design and testing of dynamic maze navigation tasks.

A. Article Structure

This paper is organized as follows: Section II reviews related work, focusing on RL and reward shaping techniques applied to dynamic maze navigation tasks. Section III details the research methodology, beginning with an introduction to the PPO algorithm in Subsection *A*. Subsections *B*, *C*, and *D* describe each developed scenario, covering the environment setup, reward shaping, implementation details and results for each case. Finally, Section V concludes the paper with a summary of the findings and suggestions for future research directions.

III. Related Work

In recent years, Proximal Policy Optimization (PPO) [1] has garnered significant attention in continuous control tasks due to its reliable convergence properties, particularly. Its application has led to remarkable results across various autonomous agent applications. For instance, in 2022, *Chao Yu et al.* [3] demonstrated the effectiveness of PPO in cooperative multi-agent games. Their research systematically analyzed PPO's performance in cooperative multi-agent settings, showing that PPO-based algorithms achieve strong performance across popular multi-agent scenarios, further underscoring PPO's strength in reinforcement learning tasks. However, in the context of autonomous navigation and path planning, the application of PPO and its performance under diverse training and testing conditions remain insufficiently explored.

Traditional maze-solving problems have often been addressed using discrete algorithms [4] [5], such as Q-learning [6], which excel in environments with well-defined state-action pairs and deterministic dynamics. However, when tasks involve more complex requirements, such as simultaneous control of rotation and velocity, these methods struggle, necessitating continuous control approaches like PPO. In 2022, *Hung et al.* [7] introduced a tuned PPO-based algorithm to solve mazes within the ML-Agents framework. Their work focused on the role of hyperparameters, such as Beta, Epsilon, Lambda, and the number of epochs, in achieving optimal performance. The study highlighted how maze complexity and agent dynamics directly influence hyperparameter selection, offering valuable guidance for future reinforcement learning applications in challenging environments. Although the application of PPO to continuous control tasks in maze-solving remains relatively sparse, recent advancements have made significant strides. In 2024, *Hamid Taheri et al.* proposed a Deep Reinforcement Learning framework utilizing an Enhanced Proximal Policy Optimization (PPO) algorithm for safe mobile robot navigation [8]. Their approach enhanced PPO's stable convergence properties by incorporating a refined neural network structure and a carefully designed reward function to optimize performance. Using LiDAR-equipped mobile robots guided by a deep neural network, the framework demonstrated effective navigation in complex environments, avoiding obstacles with precision. Experimental results from both obstacle-rich and obstacle-free scenarios underscored the Enhanced PPO algorithm's capability for collision-free autonomous navigation, highlighting its promise for sophisticated control tasks in robotics.

Aligned with Proximal Policy Optimization (PPO), reward shaping is a powerful tool to improve an agent's learning efficiency by carefully tailoring how rewards are assigned within the training environment. In their 2020 study, *Hu et al.* [9] tackled the challenge of adaptively applying shaping rewards. They proposed framing this challenge as a bi-level optimization problem: the lower level focuses on optimizing the agent's policy based on the shaping rewards, while the upper level seeks to optimize a parameterized function that adjusts the weight of these shaping rewards to ensure maximum performance relative to the true rewards. In simpler terms, their method dynamically balances the influence of shaping rewards to guide the agent effectively while still prioritizing the ultimate goal of maximizing true rewards.

The reviewed works highlight the versatility and effectiveness of PPO in addressing complex reinforcement learning tasks, particularly in continuous control scenarios. Techniques such as reward shaping and adaptive optimization further enhance PPO's performance, making it a promising approach for future applications in autonomous navigation and other challenging domains.

IV. Methodology

The proposed approach employs the PPO algorithm within a continuous action space, combined with a tailored reward shaping technique, to train an agent for scenario-specific tasks. The implementation is carried out in Unity using the ML-Agents Toolkit.

A. PPO Algorithm

PPO is an on-policy RL algorithm that optimizes the policy by minimizing the difference between the old and new policies, thereby preventing large, destabilizing updates. The key idea behind PPO is to ensure that the new policy does not deviate excessively from the old one, while still making progress toward better performance. This is achieved through the use of a clipped objective function, which constrains the magnitude of policy updates.

The PPO objective function is given by:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

In this expression, $r_t(\theta)$ is the probability ratio between the new and old policies, defined as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Here, $\pi_\theta(a_t|s_t)$ represents the probability of taking action a_t at state s_t under the new policy π_θ, while $\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the probability under the old policy. This ratio measures how much the probability of taking a particular action has changed between the old and new policies.

$\hat{A}_t$ is the estimated advantage function at time step t. The advantage function indicates how much better or worse the chosen action was compared to the expected value of the state, helping the agent determine whether the action was advantageous or not. This guides the policy to focus on actions that improve the expected reward.

The term "clip" in the objective function refers to the clipping of $r_t(\theta)$ to the range $[1-\epsilon, 1+\epsilon]$, where ϵ is a small positive value. If $r_t(\theta)$ falls outside this range, it is clipped. This clipping ensures that updates to the policy are not too large, thereby maintaining stability during training. Without clipping, the policy might change too drastically, potentially causing divergence or instability in the learning process.

The objective function in PPO serves two main purposes: it stabilizes updates by limiting the change between the old and new policies, and it encourages the policy to explore effective

actions that lead to higher rewards while maintaining a balance between exploration and exploitation. This design allows PPO to perform reliably across a range of reinforcement learning tasks, particularly in continuous control problems, making it one of the most widely used algorithms in the field.

B. Scenario 1: Target in a Simple Square Environment

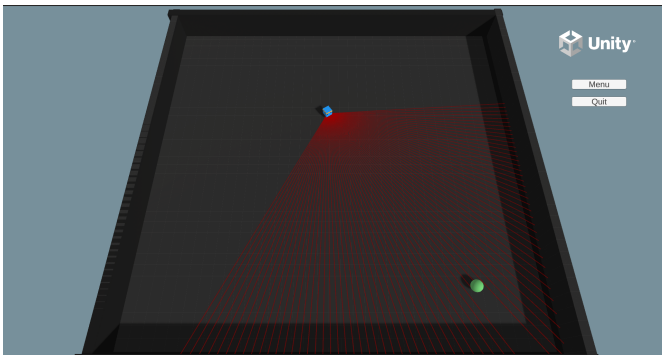

Fig. 1. Simple Square Environment with single target and respective AI agent

The first scenario, shown in figure 1, introduces the basic navigation challenge in a simplified environment. The environment consists of a square arena (46x46 units) bounded by walls and containing two elements: a blue agent and a green static target. This basic setup provides an ideal testbed for evaluating the core navigation capabilities of our reinforcement learning approach.

1) Environment Design: The world state space S consists of:

- Agent pose (position $[x, y, z]$ and orientation θ)
- Target position $[x_t, y_t, z_t]$

The agent receives a partial observation of the state space, consisting of:

- Agent's current position coordinates $[x, y, z]$
- Readings from raycast sensors that detect walls and obstacles

The RaycastSensor system in Unity ML-Agents is a powerful perception tool that enables the agent to sense its surroundings. Each ray emanating from the agent acts as a sensory input, providing normalized distance measurements to detected objects, information about whether a collision occurred, and the type of object detected (wall or target in our case). The sensor's configuration can be customized by adjusting the number of rays, their maximum detection range and their angular arrangement around the agent. These rays effectively create a local perception field around the agent, allowing it to detect and react to nearby objects and boundaries in its environment.

The action space A is continuous and two-dimensional:

- Forward/backward movement (normalized to $[-1, 1]$)
- Rotational movement (normalized to $[-1, 1]$)

The agent moves with a configured speed of 5 units per second, allowing for smooth navigation while maintaining controllability. Episodes begin with both the agent and target randomly positioned within the arena, with a minimum separation distance of 0.2 units to ensure meaningful training scenarios.

2) Reward Shapping: Through iterative development, we established an effective reward function combining several components:

$$R = R_{\text{terminal}} + R_{\text{step}} + R_{\text{rotation}} \tag{1}$$

where:

- R_{terminal} = +10.0 for reaching the target, -5.0 for wall collisions
- R_{step} = -2.0 * Δt/minDeltaTime (time-based penalty scaled by minimum possible completion time)
- R_{rotation} = -0.01 * $\|$rotation_action$\|$ (penalty for excessive rotation)

A key innovation in our reward structure is the adaptive time penalty (Rstep). For each episode, we calculate minDeltaTime as the theoretical minimum time required to reach the target, computed as the initial straight-line distance to the target divided by the agent's maximum speed (5 units/second). This creates an adaptive penalty that scales with the difficulty of each specific episode - longer distances result in smaller per-step penalties, while shorter distances enforce more strict time constraints.

The rotation penalty (Rrotation) was introduced to address a specific behavioral issue observed during training. Initially, the agent developed an inefficient strategy of frequent rotational movements instead of direct path navigation. By penalizing the magnitude of rotational actions, we encouraged the agent to prefer straighter, more efficient paths to the target. This modification led to more natural and efficient navigation behaviors, where the agent would typically orient itself toward the target and maintain a relatively straight trajectory.

This reward structure successfully balances the competing goals of reaching the target quickly while maintaining smooth and efficient movement patterns. The combination of adaptive time penalties and rotation constraints proved essential in developing an agent capable of navigating naturally and effectively to an initially unknown target in the shortest possible time.

3) Implementation Details: At the start of each episode, both the agent and the target are randomly positioned in the environment, ensuring a variety of training scenarios while maintaining a minimum separation distance. The agent's movement is handled by Unity's rigid body system, which provides realistic physics-based motion in response to the agent's actions. The implementation includes comprehensive collision detection to manage episode termination conditions, where contact with the target signals a successful completion while wall collisions result in immediate episode failure. The implementation for this scene can be found in this repository **Scene1_Code**.

4) Results: Figure 2 shows the training progress for the first scenario, presenting both the cumulative reward and the episode length across training steps. The training process revealed several key insights that led to important adjustments to the reward function. At approximately 250,000 steps, we introduced the rotation penalty (R_{rotation}) to address the agent's

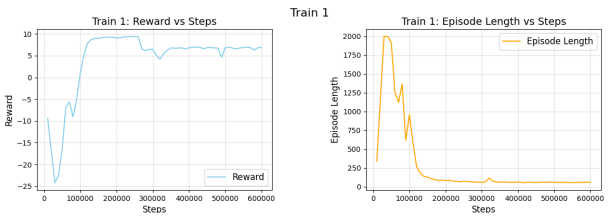

Fig. 2. Training results from Scenario 1, illustrating the cumulative reward achieved and the corresponding episode length over time

tendency to over-rotate. While this initially reduced the cumulative reward, it resulted in significantly more natural and efficient navigation patterns, with the agent taking more direct paths to the goal. By 300,000 steps, we increased the time step penalty (R_{step}), which further reduced the cumulative reward, but achieved our goal of shorter episode completion times, as evidenced by the decrease in episode length shown in the right plot in figure 2. The drops in reward near 500,000 steps showed a limitation in our minDeltaTime calculation: it did not take into account the initial rotation time required for the agent to orient towards the target. Despite this theoretical limitation in the reward formulation, the agent achieved robust performance, consistently reaching the target in a very short time.

The final trained agent demonstrated reliable targeting behaviour with impressive fast completion times, validating our reward shaping approach for this basic navigation scenario. This success is particularly evident in the plot of the length of the episodes, in Figure 2, which shows a dramatic reduction from initial episodes that required more than 1750 steps to consistent completions in less than 250 steps, indicating highly efficient navigation by the trained agent.

The entire training process for this scenario required approximately 600,000 steps. However, by using Unity ML-Agents' 100x time scale feature, the training process was significantly accelerated. We also used parallel training by creating multiple copies of the environment, which Unity ML-Agents easily handled. This parallel training approach, which required only simple scene duplication, significantly reduced overall training time while maintaining learning effectiveness.

C. Scenario 2: Navigation in a Dynamic Maze with Multiple Targets

The second scenario, shown in figure 3, introduces a more complex navigation challenge. The environment has the same external dimensions (46x46 units) as the first scenario, but incorporates static wall obstacles that create a maze-like structure. The agent must now navigate through this constrained space to reach 18 targets in a given order, significantly increasing the complexity of the navigation task.

1) Environment Design: The world state space S consists of:

- Agent pose (position $[x, y, z]$ and orientation θ)
- Position of all targets $[x_{t_i}, y_{t_i}, z_{t_i}]$

The agent receives a partial observation of the state space, consisting of:

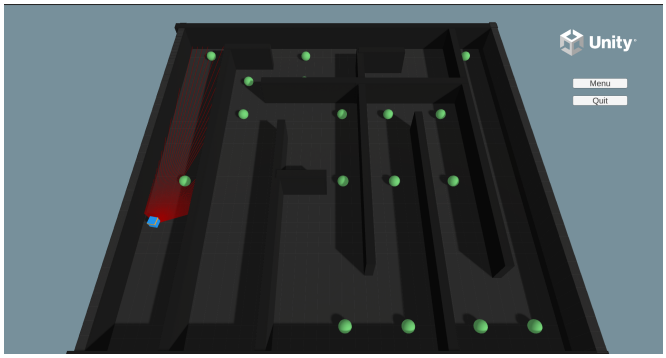

Fig. 3. Dynamic Maze with Multiple targets and respective AI agent

- Agent's current position coordinates $[x, y, z]$
- Agent's current velocity $[v_x, v_y, v_z]$
- Relative position to all remaining targets
- Readings from raycast sensors that detect walls and obstacles

The addition of velocity information to the observation space was beneficial in improving the navigation performance of the agent. As a result, the agent has learnt to make smoother turns, as it was initially prone to colliding with walls during turn manoeuvres.

For target positions, the agent receives the relative position vectors (target position minus agent position) for all targets, including those already collected. While collected targets are moved to a fixed position outside the maze, they remain in the observation space to maintain a constant input dimension of the neural network throughout training. We observed that the network learns to effectively filter out collected targets and focus on the remaining targets in the sequence. This complete target awareness also allows the agent to better plan its path through the maze sequence.

The RaycastSensor system maintains the same configuration as in the first scenario, providing normalised distance measurements and object detection information. This sensory information was important because it enabled the agent to see the target and also helped it to avoid collisions with the walls as it navigated the narrow corridors.

The action space A remains continuous and two-dimensional:

- Forward/backward movement (normalized to $[-1, 1]$)
- Rotational movement (normalized to $[-1, 1]$)

The agent moves at the same configured speed of 5 units per second. Unlike the first scenario, the episodes start with the agent at a fixed starting position in the lower left corner of the maze, while the targets maintain predefined positions that create a specific path through the maze.

2) Reward Shaping: The reward function maintains a similar structure to the first scenario but adapts to the sequential nature of the task:

$$R = R_{\text{terminal}} + R_{\text{step}} + R_{\text{rotation}} + R_{\text{progress}} \tag{2}$$

where:

- R_{terminal} = +100.0 for completing all targets, -0.5 for wall collisions

- R_{step} = -0.001 (time-based penalty)
- R_{rotation} = -0.01 * ||rotation_action|| (penalty for excessive rotation)
- R_{progress} = n + 1 for reaching the nth target in sequence

A key adaptation in this scenario is the progressive reward (R_{progress}), which scales with the target sequence. Each successfully collected target provides a reward equal to its position in the sequence plus one, creating an increasing incentive to progress through the maze. The final reward increases significantly to +100.0 for completing the entire sequence, reflecting the increased complexity of the task.

The rotation penalty remains unchanged from the first scenario and continues to encourage efficient path navigation through the corridors of the maze. The step penalty is simplified to a constant value, as the varying distances between successive goals made the adaptive penalty less relevant in this scenario.

This reward structure balances the immediate feedback needed to learn basic navigation with the long-term guidance that is required to complete the entire target sequence.

3) Implementation Details: The agent starts each episode from a fixed position at the maze entrance, establishing a consistent starting point for the sequential navigation task. The implementation retains the physics-based motion system from the first scenario, with collision detection now playing a critical role in episode termination during training. Wall collisions during training result in immediate episode termination with a penalty, encouraging the agent to learn careful navigation through the confined spaces of the maze. The implementation for this scene can be found in this repository **Scene2_Code**.

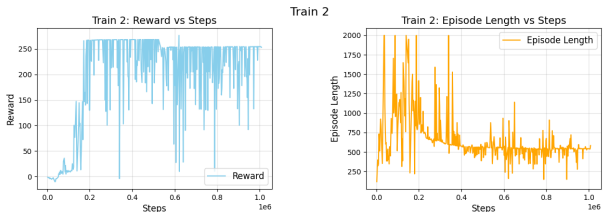

Fig. 4. Training results from Scenario 2, illustrating the cumulative reward achieved and the corresponding episode length over time

4) Experiments: Figure 4 shows the training progress for the second scenario, showing both cumulative reward and episode length across training steps. In this scenario, the task is to navigate through a maze while collecting multiple sequential targets, which has increased complexity. This complexity can be seen in the more complex reward function we chose, the number of training steps it required, and the longer time it took to complete an episode compared to the previous scenario.

The training process revealed important insights into the agent's behaviour and led to important reward adjustments. The most significant change occurred at about 500,000 steps, where we introduced the rotation penalty (R_{rotation}). While this addition caused a noticeable decrease in the maximum reward achievable in the steady state, it resulted in a corresponding reduction in episode completion time, as shown in the right plot of Figure 4. This trade-off proved beneficial as the agent developed more efficient navigation patterns through the corridors of the maze.

The progressive reward structure (R_{progress}), which increased the reward for each subsequent target collected, proved effective in guiding the agent through the target sequence. This is visible in the reward plot, where after initial training the agent consistently achieved high cumulative rewards, indicating successful completion of the entire target sequence. The episode length plot shows a significant improvement in navigation efficiency, with episodes initially requiring up to 2,000 steps being reduced to consistent completion in approximately 500 steps. This significant reduction indicates that the agent has learned not only to find all targets, but to do so via efficient paths through the maze.

Due to the increased complexity of this scenario compared to the first, the training process required approximately 1 million steps to achieve satisfactory performance. As with the first scenario, we used Unity ML-Agents' 100x timescale feature and parallel training capabilities to speed up the training process. The ability to simply duplicate the training environment proved particularly valuable in this more complex scenario, allowing efficient exploration of the larger state space while maintaining reasonable training times.

D. Scenario 3: Single Target in the Dynamic Maze

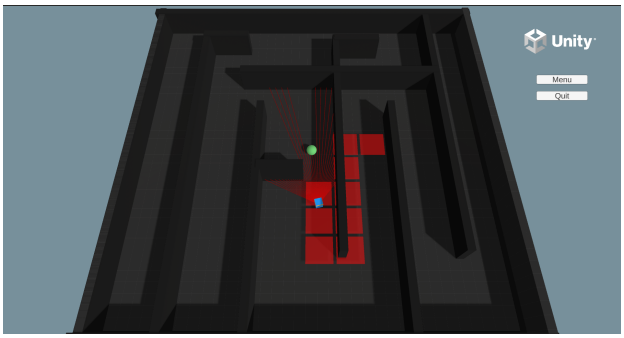

Fig. 5. Dynamic Maze with single target and respective AI agent

The third scenario, shown in figure 5, preserves the same maze structure, but introduces a novel approach to navigation. The agent is now spawned at random positions within the maze and must reach a single target fixed at the centre of the maze. To prevent cyclical behaviour, we introduced a spatial memory system that tracks the agent's exploration history.

1) Environment Design: The world state space S consists of:

- Agent pose (position $[x, y, z]$ and orientation θ)
- Target position $[x_t, y_t, z_t]$
- Grid state (visited/unvisited cells)

The agent receives a partial observation of the state space, consisting of:

- Agent's current position coordinates $[x, y, z]$
- Agent's current velocity
- Relative position to target
- Binary vector representing visited/unvisited cells

- Readings from raycast sensors that detect walls and obstacles

The environment is discretised into a grid of 1-unit cell size, chosen to keep the dimensionality of the observation space small while maintaining effective spatial resolution for navigation. The state of each cell (visited/unvisited) is tracked and visualised by semi-transparent red overlays, providing an intuitive debugging tool for developers and users to understand the agent's exploration patterns. The RaycastSensor configuration remains consistent with previous scenarios.

The action space also remains unchanged, with continuous two-dimensional control:

- Forward/backward movement (normalized to $[-1, 1]$)
- Rotational movement (normalized to $[-1, 1]$)

2) Reward Shaping: The reward function has been adjusted to encourage efficient exploration:

$$R = R_{\text{terminal}} + R_{\text{step}} + R_{\text{rotation}} + R_{\text{exploration}} \tag{3}$$

where:

- R_{terminal} = +100.0 for reaching target, -10.0 for wall collisions
- R_{step} = -0.001 (time-based penalty)
- R_{rotation} = -0.001 * ||rotation_action|| (penalty for excessive rotation)
- $R_{\text{exploration}}$ = +0.5 for new cells, -0.2 for revisited cells

The exploration reward component ($R_{\text{exploration}}$) is the key innovation in this scenario. It explicitly rewards the discovery of new areas, while penalising revisiting previously explored cells. This mechanism, combined with the spatial memory system, effectively prevents the agent from becoming trapped in cyclical patterns.

Initially, during the training phase, we incorporated only three reward components: exploration, step, and wall collision. This design aimed to encourage the agent to learn the fundamental skills of navigating and exploring the maze efficiently. Once the agent demonstrated a solid understanding of maze navigation, we introduced the additional reward component specifically tied to locating the target. This incremental approach allowed the agent to first master general exploration before focusing on the goal-oriented task of finding the target.

3) Implementation Details: The implementation introduces a grid-based tracking system that maintains the visited/unvisited state of each cell. The agent's position is continuously mapped to the corresponding grid cell, updating the spatial memory as exploration progresses. Random initialisation of the agent's position and orientation ensures robust learning under different initial conditions, while the fixed central position of the target provides a consistent goal. Wall collisions during training result in immediate episode termination, reinforcing careful navigation through the confined spaces. The implementation for this scene can be found in this repository **Scene3_Code**.

4) Experiments: Figure 6 shows the training progress for the third scenario, showing both the cumulative reward and the episode length across training steps. This scenario represents the most challenging navigation task, as evidenced by the need for approximately 6 million training steps to achieve satisfactory performance - a significant increase over the previous scenarios.

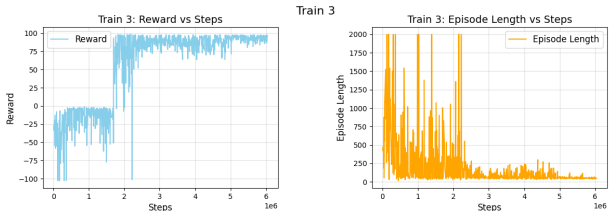

Fig. 6. Training results from Scenario 3, illustrating the cumulative reward achieved and the corresponding episode length over time

The training process provided important insights into the effectiveness of our spatial memory approach and reward structure. The addition of the grid-based exploration system showed great results, as without this spatial memory component the agent would often be trapped in circular patterns and unable to effectively explore the maze.

A significant improvement occurred at around 1.8 million steps when we added the target reward structure; this modification resulted in a dramatic increase in performance, increasing the average reward, indicating more consistent successful navigation to the target. Further refinement at 4.5 million steps, by introducing rotation penalties and adding step penalties, resulted in a further improvement with a higher and more stable reward, as shown in the left plot of figure 6.

The episode length plot in figure 6 shows a significant improvement in navigation efficiency over the training period. Initially requiring up to 2,000 steps per episode, the agent eventually achieved consistent completion in less than 200 steps, indicating that it was choosing efficient paths. This increase suggests that spatial memory combined with a refined reward structure were critical for the path planning of the agent. Despite the significant increase in training steps required compared to previous scenarios, the time-scale feature and parallel training capabilities of Unity ML agents made this extended training process manageable.

V. Conclusion

This study explored the application of PPO in maze navigation tasks, leveraging continuous control and tailored reward shaping to enhance the agent's learning process. By implementing the proposed approach within the Unity framework using the ML-Agents Toolkit, we demonstrated PPO's ability to adapt to diverse scenarios, highlighting its reliability and efficiency in optimizing complex control tasks.

Through detailed analysis, we observed how reward shaping plays a critical role in accelerating learning and improving task performance, emphasizing the importance of crafting effective reward functions for real-world applications. The results showcased PPO's potential especially in environments requiring adaptive and autonomous decision-making.

Future work will focus on extending this research to more complex and real-world navigation tasks, refining reward functions for increased learning efficiency, and exploring the integration of PPO with other reinforcement learning techniques.

In essence, this project underscores the transformative potential of RL in addressing industrial challenges, paving the way for more efficient and intelligent autonomous systems.

VI. Acknowledgments

This work was supported by the Faculdade de Engenharia da Universidade do Porto (FEUP). The authors also thank Professor António Pedro Aguiar for providing the necessary resources and support.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[2] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," 2020. [Online]. Available: https://arxiv.org/abs/1809.02627

[3] 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," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 24 611–24 624. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/9c1535a02f0ce079433344e14d910597-Paper-Datasets_and_Benchmarks.pdf

[4] S.-W. Lin, Y.-L. Huang, and W.-K. Hsieh, "Solving maze problem with reinforcement learning by a mobile robot," in *2019 IEEE International Conference on Computation, Communication and Engineering (ICCCE)*, 2019, pp. 215–217.

[5] M. Zucker and J. A. Bagnell, "Reinforcement planning: Rl for optimal planners," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 1850–1855.

[6] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[7] P. T. Hung, M. D. D. Truong, and P. D. Hung, "Tuning proximal policy optimization algorithm in maze solving with ml-agents," in *International Conference on Advances in Computing and Data Sciences*. Springer, 2022, pp. 248–262.

[8] H. Taheri, S. R. Hosseini, and M. A. Nekoui, "Deep reinforcement learning with enhanced ppo for safe mobile robot navigation," 2024. [Online]. Available: https://arxiv.org/abs/2405.16266

[9] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan, "Learning to utilize shaping rewards: A new approach of reward shaping," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 15 931–15 941. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/b710915795b9e9c02cf10d6d2bdb688c-Paper.pdf

Bruno Costa earned his Bachelor's degree in Electrical and Computer Engineering from the University of Porto, Portugal, in 2023. He is currently pursuing a Master's degree in Electrical and Computer Engineering at the Faculty of Engineering, University of Porto. His research interests lie at the intersection of autonomous driving, computer vision, perception systems, machine learning, and robotics. In 2024, he joined the INESC TEC research center, where he contributes as a researcher in machine learning and computer vision.

Pedro Rodrigues earned his Bachelor's degree in Engineering Physics from the University of Porto, Portugal, in 2023. He is currently pursuing a Master's degree in Electrical and Computer Engineering at the Faculty of Engineering, University of Porto. His research interests focus on control, perception systems, robotics, and systems engineering. In 2022, he joined the INESC TEC research center, where he conducts research in robotics and embedded systems.