Deep Reinforcement Learning for Maze Navigation on a Ball-Balancing Platform

Adriano Machado, Félix Martins, Francisco da Ana, Martim Iglesias
Faculty of Engineering, University of Porto (FEUP)
Porto, Portugal

***Abstract*—This study presents a novel application of deep reinforcement learning to maze navigation using a ball-balancing platform, extending the canonical ball and plate control problem. We implement and compare two primary approaches using Unity ML-Agents: a hierarchical strategy combining A* pathfinding with PPO-based low-level control, and an end-to-end approach using a monolithic RL policy with CNN and LSTM components. Our experimental evaluation demonstrates the effectiveness of Proximal Policy Optimization in learning complex navigation strategies across diverse maze configurations, providing valuable insights into sample efficiency and learned behaviors for nonlinear, under-actuated robotic control systems.**

***Index Terms*—Reinforcement Learning, Deep Reinforcement Learning, Proximal Policy Optimization (PPO), Maze Navigation, Unity, ml-agents, Hierarchical Reinforcement Learning**

I. Introduction

A. The Ball and Plate System: A Canonical Control Problem

The ball and plate system represents a canonical benchmark problem within the field of control engineering. Its relevance stems from its embodiment of several fundamental control challenges in a single setup. The system is characterized as being inherently open-loop unstable, meaning that without active control, the ball will inevitably fall off the plate. Furthermore, its dynamics are fundamentally nonlinear and under-actuated: it typically has four degrees of freedom (the ball's x and y position and velocity) but only two control inputs (the plate's tilt angles). These characteristics make it an exceptional testbed for validating a wide spectrum of control methodologies, from classical approaches to modern control schemes [1], [2].

B. A New Dimension of Complexity: The Maze-Solving Task

This project introduces a significant and novel extension to the classic ball and plate problem: the task of navigating a maze constructed upon the plate's surface. This addition fundamentally transforms the control objective. The classic problem is one of regulation or stabilization, where the goal is to maintain the ball at a fixed setpoint, typically the center of the plate, and reject disturbances [3], [4]. The maze-solving task, however, elevates the challenge to one of goal-directed navigation and path planning . The agent must now learn not only to control the ball's position but also to guide it along a constrained, non-trivial path from a designated start point to an exit. This requires a form of sequential decision-making, where the agent must implicitly or explicitly plan a sequence of movements to progress towards the final goal [5], [6].

C. A Reinforcement Learning Approach for a Complex, Dynamic Task

The central idea of this work is that model-free Reinforcement Learning (RL) provides a uniquely powerful and adaptable framework for solving this multifaceted control problem. The strength of RL lies in its capacity to derive effective control policies through direct, trial-and-error interaction with the environment, thereby circumventing the need for an explicit and often intractable mathematical model of the system's dynamics. The combination of the ball's nonlinear motion, complex contact forces with the plate and maze walls, and potential sensor and actuator imperfections makes analytical modeling exceedingly difficult [1], [7]. RL offers an adaptive solution that can implicitly learn the complex dynamics of the system and devise effective maze-solving strategies.

A key methodological decision in this project was the use of the Unity and ML-agents simulation environment [8], [9]. This choice was made deliberately to abstract away the challenge of inverse kinematics [10]. By simulating the plate's tilt angles as direct control outputs, it enabled a focused and uncluttered investigation into the learning dynamics and capabilities of the RL agent itself when faced with this complex task.

D. Report Structure and Contributions

This report is structured to provide a comprehensive exploration of the problem and the proposed solutions. Section II presents a state-of-the-art review of both classical and modern control methods applied to the ball and plate system. Section III then details our system design, simulation environment, and problem formulation as a Markov Decision Process. We outline our experiments in detail in Section IV and present an analysis of the corresponding results in Section V. Section VI discusses the broader implications of the findings and the study's conclusions. Finally, we discuss the study's conclusions in Section VI, outline potential future work in Section VII, and provide a guide for running the project in the Appendix.

The primary contribution of this work is a comparative analysis of several Reinforcement Learning strategies, evaluating their effectiveness in solving the novel and challenging task of maze navigation on a ball and plate system.

II. State of the Art

The control of the ball and plate system has been a subject of extensive research, serving as a platform for demonstrating

and comparing a wide array of control strategies. The literature reveals a clear evolutionary path from classical, model-based techniques to modern, data-driven learning approaches, each seeking to address the system's inherent challenges.

A. Classical Control Paradigms (Model-based Approaches)

Model-based control methods form the foundation of the ball and plate system control. These techniques rely on a mathematical model of the system's dynamics, derived from physics principles, to design the controller [1], [3].

1) Proportional-Integral-Derivative (PID) Control: The PID controller is the most fundamental and widely implemented solution for the ball and plate stabilization task [3], [11]. Its popularity stems from its conceptual simplicity and effectiveness. The control output is a weighted sum of three terms [12]:

- **Proportional (P):** This term provides an output proportional to the current error between the ball's actual position and the desired setpoint. It is responsible for the primary corrective action.
- **Integral (I):** This term accumulates past errors. Its function is to eliminate steady-state error, ensuring the ball eventually settles precisely at the target position.
- **Derivative (D):** This term is proportional to the rate of change of the error, effectively predicting future error. It provides a damping effect, reducing overshoot and oscillations as the ball approaches the setpoint.

The performance of a PID controller is critically dependent on the tuning of its three gains (K_p, K_i, K_d). Methods for tuning range from heuristic approaches like the Ziegler-Nichols method [13], which involves systematically increasing the proportional gain until the system oscillates, to computational optimization techniques like Particle Swarm Optimization (PSO), which can search the parameter space for gains that minimize a performance index like the Integral Absolute Error (IAE) [4], [11].

2) Linear-Quadratic Regulator (LQR) Control: The Linear-Quadratic Regulator (LQR) is an optimal control technique that provides a more systematic design approach than PID. It works by using a simplified mathematical model of the system, typically based on its behavior when the plate is leveled (i.e., horizontal).

The controller automatically calculates a control strategy that balances two competing goals: performance (how quickly the ball gets to its target) and the amount of control effort (how much the motors have to work). A designer can adjust settings to prioritize either a fast response or more conservative movements. LQR is particularly effective for trajectory tracking, but its performance depends on the accuracy of the underlying model [14].

3) Sliding Mode Control (SMC): Sliding Mode Control (SMC) is a robust control technique particularly well-suited for systems with significant uncertainties and disturbances, such as the ball and plate system [2]. The core idea is to force the system's behavior to follow a predefined ideal path. The controller works aggressively to get the system onto this path and then keep it there [15].

The main advantage of SMC is that once the system is following this path, it becomes very resistant to external disturbances and model inaccuracies. However, a significant drawback is a phenomenon known as "chattering." This refers to high-frequency, undesirable vibrations in the control signal caused by the controller making constant, aggressive corrections. Researchers have proposed various advanced methods to reduce this effect [15].

B. The Emergence of Reinforcement Learning (Model-Free Approaches)

Model-based control strategies are fundamentally limited by their reliance on a precise mathematical model. For a system like the ball and plate, factors such as varying friction, sensor noise, and processing delays create a significant "reality gap" between the model and the actual hardware, which can degrade performance [1].

Reinforcement learning (RL) offers a powerful, data-driven alternative. By learning a control policy directly through interaction with the environment, an RL agent can bypass the need for an explicit model [16], [17]. This approach allows the agent to implicitly master complex, unmodeled dynamics, a capability that has shown great promise in modern robotics and continuous control tasks [5], [18]–[20]. Several deep RL algorithms are particularly suited for this problem.

- **Deep Deterministic Policy Gradient (DDPG):** DDPG is an off-policy, actor-critic algorithm designed for environments with continuous action spaces. It learns a deterministic policy (the actor) that maps states to actions, and a Q-function (the critic) that evaluates the value of those actions. However, DDPG can be sensitive to hyperparameters and can struggle with effective exploration, particularly in environments with sparse or deceptive rewards [19].
- **Soft Actor-Critic (SAC):** SAC is a state-of-the-art off-policy actor-critic algorithm that modifies the standard RL objective to include an entropy maximization term. This means the agent aims to maximize both the expected cumulative reward and the entropy of its policy. This entropy term encourages broader exploration, leading to more robust and sample-efficient learning. SAC's stability and high performance have made it a popular choice for continuous control problems in robotics [21], [22].
- **Proximal Policy Optimization (PPO):** PPO is an on-policy, actor-critic algorithm that has become a benchmark due to its stability, reliability, and relative simplicity of implementation [18]. Its key innovation is a clipped surrogate objective function, which constrains the size of policy updates at each training step. This prevents the new policy from deviating too drastically from the old one, avoiding catastrophic performance collapses that can occur in other policy gradient methods and making the training process more stable and reliable [18], [23].

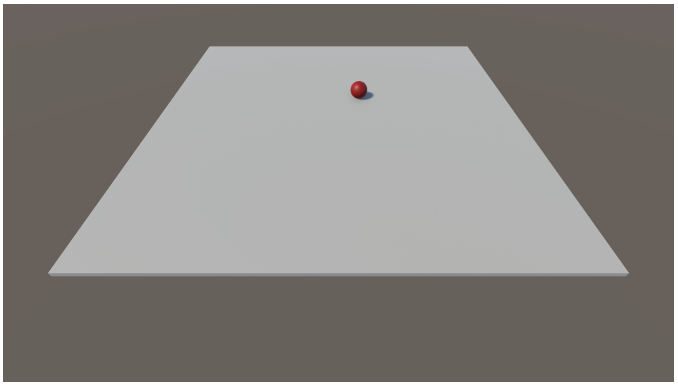


Fig. 1: Basic ball-and-plate system setup

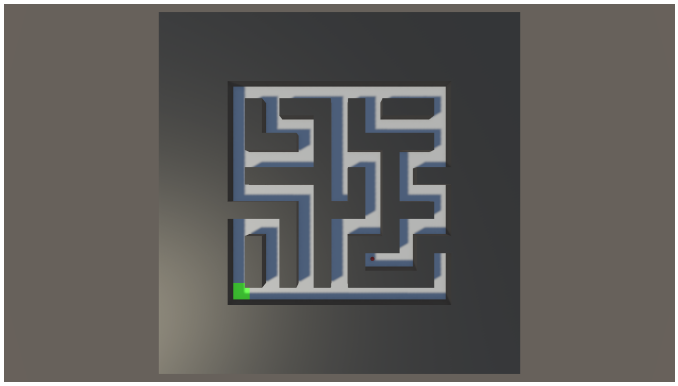


Fig. 2: Example of a generated maze. The goal is marked in green, while the ball is red

The evolution from classical controllers to deep RL represents a fundamental shift in philosophy: from designing controllers based on an imperfect model of the world to learning policies that are inherently adapted to its complexities. This project's investigation lies within this modern, data-driven paradigm. Therefore, after reviewing the available methods, Proximal Policy Optimization (PPO) was selected as the primary control strategy. Its model-free approach directly addresses the challenges of system modeling, while its noted stability offers a reliable advantage over other contemporary RL algorithms.

III. System Design and Simulation Environment

A. Basic Ball-Balancing Platform

Our first baselines were based on a simple ball-and-plate system. The system consists of a ball rolling freely on a plane, whose tilt can be actively controlled (Fig. 1).

B. The Ball-Balancing Maze Platform: Conceptual model

The system under investigation is a virtual simulation of a physical ball-balancing table. The core components are:

- **The Plate:** A flat plate, in this case rectangular, which serves as the playing surface for the ball.
- **Actuation:** The plate is actuated with two rotational degrees of freedom: pitch and roll. The simulation allows direct control over these rotations, where the agent's actions directly modify the tilt angles for the plate. This is the agent's means of influencing the system.
- **The Ball:** A sphere that rolls freely on the surface of the plate, subject to gravity and contact forces.
- **The Maze:** A static structure of walls and pathways is placed on the plate. This maze constrains the ball's movement and defines the navigation task. Every cell in the maze has a defined constant size.

The agent is responsible for controlling the plate, and has access to information like the maze structure, and the position of the ball.

1) Maze Generation and Setup: Although navigating a ball using this kind of system is a very challenging task, especially through a maze, the more interesting goal is to achieve greater generalization in solving mazes. For Reinforcement Learning, solving a static maze could involve memorizing intersections and movements, even if the ball starts in different positions across episodes. The main objective was to achieve improved generalization and measure performance in various mazes. Therefore, the maze is generated procedurally to ensure a standardized but varied set of challenges for the agent. This considerably increased the required complexity of the agent, while simultaneously increasing the robustness to variations in the mazes.

The logical structure of the maze is created using a depth-first search algorithm, specifically the recursive backtracking method. Although not explored in depth in our project, it also contains additional methods to increase or decrease the general difficulty of the maze. An example is show in Fig. 2.

C. Simulation Framework: A Critical Evaluation of Unity and ML-Agents

The choice of simulation environment is a critical methodological decision that shapes the scope and focus of the research.

The primary motivation for selecting the **Unity engine** with the **ML-Agents toolkit** was its high level of abstraction, which facilitates a focused study on reinforcement learning. In many real-world robotic systems, a task like this would be accomplished by a multi-jointed robotic arm holding the plate. To command such a robot to achieve a specific plate tilt, one would need to solve the inverse kinematics problem, which involves calculating the required joint angles for each of the arm's links [10]. This is a non-trivial mathematical and computational challenge in itself. By using Unity, it is possible to bypass this complexity.

The simulation allows for direct control over the plate's orientation. The RL agent's action space can thus be defined as the desired target angles, allowing the research to concentrate entirely on the agent's decision-making and learning process.

This approach simplifies the setup, making the complex problem of maze navigation more tractable for experimentation.

However, this simplification comes with a significant trade-off: the "sim-to-real gap". Unity's default physics engine, PhysX, is highly optimized for performance and visual realism in video games, which does not always equate to high-fidelity physical accuracy for engineering applications. Phenomena like friction, contact dynamics, and material properties may not be simulated with the precision required for a seamless transfer of a learned policy to a physical robot. Nevertheless, we argue that the learned policy, although not directly transferrable, serves as a proof-of-concept for generalizable implementations across different robotic platforms and task environments. The learned strategy to navigate the maze validates the control approach itself, which can then be tailored to the specific dynamics and controllers of various real-world systems [8], [9].

To contextualize the simulator choice, it is useful to compare Unity with other popular physics engines used in robotics research [8], [24]–[29].

The comparison in Table I clarifies that while other simulators offer higher physical fidelity, often chosen by research aiming for direct sim-to-real transfer, Unity provides a more accessible and user-friendly platform ideal for rapid prototyping and focusing on high-level agent logic, which aligns with the stated goals of this project.

D. Problem Formulation: The Markov Decision Process (MDP)

To apply Reinforcement Learning, the maze-solving task, or any other task, needs to be formally framed as a **Markov Decision Process (MDP)** or a Partially Observable Markov Decision Process (POMDP). Therefore, we need to define the action space A, the state space (observations) S, and the reward function $R(s, a, s')$, where s represents the current state, a the action taken at that state and s' the next state [16], [30]. Since we explore different tasks with different MDP formulations, we will define here only the action space, since it is common for all of them.

The action space A defines the set of actions the agent can take. For this continuous problem, the action is a 2-dimensional vector, $a = (\sigma_{roll}, \sigma_{pitch})$, representing the target roll and pitch angles for the plate [1].

The state space, or observation space in this case, defines the set of observations the agent receives from the environment at each timestep. A comprehensive state representation is crucial for the agent to make informed decisions [16].

Finally, the reward function is a very critical element of the design, providing the learning signal that guides the agent's behavior. A well-designed reward function is essential for successful and efficient learning. For instance, a naive, sparse reward function that provides a reward upon reaching the goal can make training extremely slow and effectively not tractable [16].

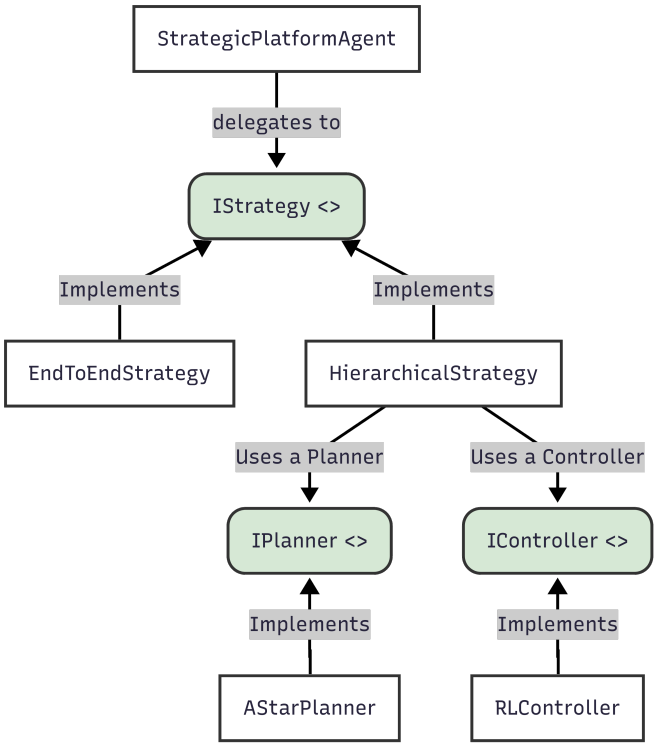


Fig. 3: Diagram showing the system architecture with the Strategy pattern implementation

E. System Architecture

Beyond the core simulation components, a fundamental design consideration of this project was to create an extensible and educational framework that could serve as a foundation for future research and development. The system architecture follows a modular approach that decouples the agent from its decision-making strategy, facilitating the exploration of different algorithmic paradigms and approaches to maze navigation.

The architecture is built around the **Strategy design pattern**, which allows for the abstraction of the agent's behavior from its implementation. This design choice enables seamless switching between different solution approaches without modifying the core agent logic. The system defines clear interfaces that serve as contracts for implementing various strategies, promoting code reusability and maintainability.

As illustrated in Fig. 3, the `StrategicPlatformAgent` serves as the main controller that delegates decision-making to interchangeable strategy implementations through the `IStrategy` interface.

The current implementation includes two primary strategy types:

- **HierarchicalStrategy:** An approach that decomposes the problem into high-level planning and low-level control components. This strategy utilizes two additional interfaces:
  - `IPlanner`: Responsible for high-level decision making, such as determining the complete path and

TABLE I: Comparison of Physics Simulation Frameworks

Framework	Primary Use Case	Physics Fidelity	Ease of Use	Sim-to-Real	Open Source
Unity/ML-Agents	Game Dev, AI/ML	Medium	High	Moderate	Open-Access
PyBullet	Robotics Simulation, ML	Medium-High	Moderate	Moderate	Yes
MuJoCo	Robotics Research	High	Low-Moderate	High	Yes
Webots	Robotics Research & Education	Medium-High	High	High	Yes

the next waypoint or sub-goal within the maze.
  - `IController`: Handles low-level control actions, such as the precise plate movements required to navigate the ball toward a specified target.

  This hierarchical decomposition is particularly valuable as it allows for hybrid approaches where different algorithmic paradigms can be combined. For instance, a classical A* pathfinding algorithm could serve as the high-level planner (`AStarPlanner`), while a reinforcement learning agent (`RLController`) could handle the low-level control tasks.
- **EndToEndStrategy:** A direct approach where the strategy handles the complete maze navigation task, suitable for end-to-end reinforcement learning implementations.

The modular design also facilitates comparative studies between different methodologies. Contributors can easily implement new strategies by adhering to the defined interfaces, enabling systematic evaluation of various approaches under identical experimental conditions. This standardization is crucial for reproducible research and fair algorithmic comparisons.

Furthermore, the architecture's educational value lies in its clarity and extensibility. New students can quickly understand the system's structure and contribute by implementing new strategies or extending existing ones. The clear separation of concerns between planning and control also provides an intuitive framework for understanding complex robotic systems, where similar hierarchical decompositions are commonly employed.

IV. Experiments

As a method of comparison, before directly tackling the more complex task of navigating a maze, we started by smaller and simpler tasks. These easier tasks provide a great baseline for comparison with the final task.

For every experiment, we used the PPO algorithm [23], with the agent being trained for a varying number of timesteps, depending on the task. While PPO is less sample-efficient compared to other methods, its robust convergence properties, which include less sensitivity to hyperparameters and stable learning, make it a reliable and widely applicable choice [18], [23]

A. Foundational Control Tasks: Baselines

1) ***Plate Balancing****:* This is the simplest task, where the objective is simply to maintain the ball on top of the plate, without having to target a specific position. To achieve better generalization, we placed the ball in a random position at the start of every episode.

The observations we provide to the agent are:

- **The platform's current tilt angles:** $\sigma_{roll}, \sigma_{pitch}$: the current roll and pitch of the plane
- **The current (local) ball position:** x, y, z
- **Ball linear velocity:** $\vec{v}_{\text{ball}}$

The reward function is very simple (1):

$$R(s,a,s') = \begin{cases} -1, & \text{if the ball falls out of the plane} \\ 0.1, & \text{otherwise} \end{cases} \tag{1}$$

Alongside cumulative reward, we measured success rate: the percentage of episodes where the ball did not fall. The agent trained for 10^6 timesteps, with episodes running for a maximum of 5000 physics steps, with one decision made every 5 steps (1000 decisions total).

2) ***Balance to target****:* This task adds an additional layer of complexity. The objective is not only for the ball to remain on top of the plane, but also to make sure it moves towards a specific target position. For better generalization, we placed the ball in a random position and also set the target position randomly, at the beginning of each episode.

The observation space was expanded to include the target's projected 2-dimensional position on the horizontal plane, disregarding its vertical component.

The reward function is similar to the previous task, but based on the distance to the target (2):

$$R(s,a,s') = \begin{cases} -1000, & \text{if the ball falls out} \\ 1 - \frac{(d_x + d_z)}{MaxDistance}, & \text{otherwise} \end{cases} \tag{2}$$

Where d_x and d_z represent the absolute difference of the positions of the ball and the target in the x and z axes (both horizontal in Unity). $MaxDistance$ represents the maximum valid distance in the plane, which depends on its size. We used a square plane with each side of size 20, so $MaxDistance = 2 \cdot Size = 20$. The formula effectively normalizes the distance value to values in $[0, 1]$, ensuring that the reward result is also within $[0, 1]$.

By providing positive rewards at every step, we encourage the agent to "survive". On the contrary, negative rewards could encourage the agent to quickly fall in order to avoid subsequent negative rewards.

B. Advanced Maze-Solving Approaches

This section details approaches to the project's main goal: navigating a ball to a target cell in a maze using a ball-and-plate system.

*1) **Hierarchical Maze Navigation**:* The hierarchical approach represents a decomposition of the maze navigation problem into two distinct but interconnected components: high-level path planning and low-level point-to-point navigation.

- **High-level planning component:** Handles strategic decision-making and global path optimization using either classical search algorithms (A*, Dijkstra's, BFS) for optimal paths through discrete maze structures, or learning-based planners with reinforcement learning.
- **Low-level control component:** Handles continuous control for navigating the ball between waypoints using either classical controllers (PID, LQR) for precise trajectory following, or reinforcement learning methods (PPO, SAC, DDPG) for adaptive control of the nonlinear ball-plate dynamics.

For this implementation, we selected the combination of A* pathfinding algorithm and a PPO-based agent for plate control.

Regarding the reinforcement learning task for the low-level controller, we use two separate feedforward multi-layer perceptrons serving as the actor and critic networks.

The agent receives the following observations:

- **Direction to target waypoint**: A normalized vector, $\vec{d}_w$, indicating the direction from the ball's current position, p_{ball}, to the target waypoint, using positions projected onto the horizontal plane.
- **Ball linear velocity**: The ball's current velocity vector, $\vec{v}_{\text{ball}}$.
- **Platform's current tilt**: $\sigma_{roll}, \sigma_{pitch}$

The reward function, $R(s, a, s')$, has multiple components designed to encourage efficient and goal-directed behavior. It is defined as a sum of three main components:

$$R(s, a, s') = R_{\text{dir}} + R_{\text{time}} + R_{\text{waypoint}} \tag{3}$$

The components are defined as follows:

- **Directional reward (R_{dir}):** A continuous reward shaping mechanism,

  $$R_{\text{dir}} = \sigma \cdot (\vec{v}_{\text{ball}} \cdot \vec{d}_w),$$

  that encourages movement towards the current target waypoint. It is computed from the dot product of the ball's velocity $\vec{v}_{\text{ball}}$ and the normalized direction to the waypoint $\vec{d}_w$. We apply a scaling coefficient σ = 0.01 to this reward component.
- **Dynamic time penalty (R_{time}):** A small negative reward given at each timestep to encourage speed. This value is dynamically calculated at the start of each episode as

  $$R_{\text{time}} = -\frac{B}{L \cdot S_{\max}},$$

  where B is a total time penalty budget (1.0), L is the path length, and $S_{\max}$ is the allowed steps per waypoint. This adaptive mechanism ensures agents are not disproportionately penalized when solving more complex mazes.

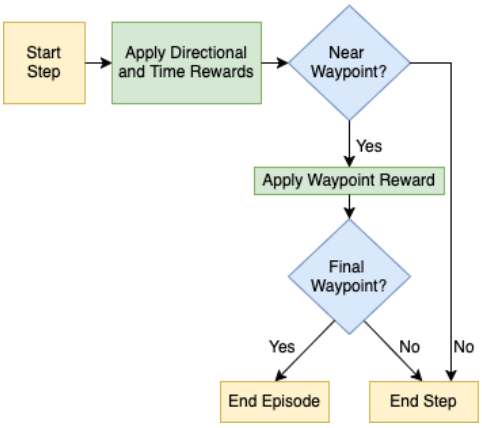


Fig. 4: Per-step decision logic for the agent in the Hierarchical Maze Navigation strategy, detailing the reward structure and waypoint progression.

- **Waypoint achievement reward (R_{waypoint}):** A sparse reward given for reaching waypoints.

  $$R_{\text{waypoint}} = \begin{cases} 1.0, & \text{if a waypoint is reached in state } s' \\ 0.0, & \text{otherwise.} \end{cases}$$

  This provides a clear signal for sub-goal completion.

Fig. 4 illustrates the algorithmic workflow of this approach, demonstrating how the multi-component reward structure from Equation 3 is applied during agent execution.

*2) **End-to-End Maze Navigation**:* In this experiment, the objective is to navigate the complete maze from start to finish using a single, monolithic RL policy. Obviously, this task requires more advanced observations, providing information about the maze, as well as a more refined reward function. It is based on a partially observable Markov Decision Process [16], [30]. Here, we provide the complete list of the observations we used.

- **Plate tilt angles (continuous):** $\sigma_{roll}, \sigma_{pitch}$: the current roll and pitch of the plane. This informs the agent about the current control inputs being applied.
- **Goal cell difference to the current ball position (discrete):** $(r_{diff}, c_{diff}) = (r_{Goal}, c_{Goal}) - (r_{Ball}, c_{Ball})$. As the formula indicates, it is defined as the difference of the cell rows and columns between the maze goal position and the current ball position.
- **Ball position shift from the center of its cell (continuous):** $CenterPosition(r_{Ball}, c_{Ball}) - RealPosition_{Ball}$, where $CenterPosition(r_{Ball}, c_{Ball})$ represents the center position of the current cell the ball is in, while $RealPosition_{Ball}$ represents the actual observed ball position. This improves the agent's fine-grained control of the current ball position, as opposed to simply providing the discrete difference to the goal.
- **Ball linear velocity (continuous):** The 3-dimensional vector $\vec{v}_{\text{ball}}$.
- **The distance of the ball to the maze goal, based on the number of cells between them (discrete):** $d[r][c]$.

It can be pre-calculated for every possible ball position using a breadth-first search.

- **A grid of wall presence around the ball:** $grid[20][20]$. This provides information about which sections of the maze the ball can move through and which are blocked. Another option would be to provide the complete encoded maze, but it doesn't scale well for larger mazes.

This strategy also requires a more complex network. We used a simple CNN to process the grid observation, producing a semantic representation that is concatenated to the rest of the observations. These observations are then processed using an LSTM unit to produce the action results, allowing the agent to have some short-term memory. Even though LSTM provides memory, we still used three stacked observations in order to further improve the performance of the agent.

The reward function we defined for this RL policy was based on the discrete distance from the ball's cell to the target's cell. We reward positively success and negatively failure. To increase robustness and penalize cheating, we penalize the agent if it positions the ball in an invalid position, although it should not be capable of doing that. This would happen if the agent could, for instance, bounce the ball on top of the maze's walls, avoiding the actual maze. Furthermore, to encourage faster navigation, the agent receives negative rewards at each step. In summary, the reward function is defined by (4):

$$R(s, a, s') = \begin{cases} 10, & \text{if } s' \text{ is the goal cell} \\ -1, & \text{if } s' \text{ is a failure state} \\ -d_{[0,1]} \cdot DR, & \text{otherwise} \end{cases} \quad (4)$$

Where $d_{[0,1]}$ represents the normalized discrete distance, with values in $[0, 1]$, between the ball's cell and the goal's cell (in the state s'), and DR represents the maximum absolute distance reward given. DR depends on the total number of training steps per training episode, to ensure the cumulative reward does not surpass the failure reward. We used 1000 decision steps, so we used a value of 0.001.

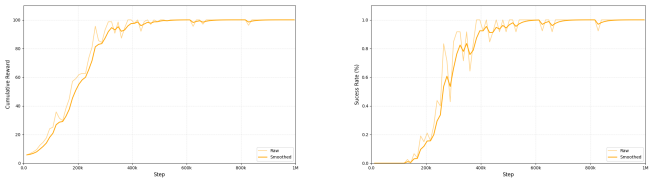
Since this task is more complex, we trained the agent for $50 \cdot 10^6$ steps.

V. Results and Discussion

A. Performance Metrics for Evaluation

To ensure a rigorous and objective comparison, we defined a set of clear performance metrics. These metrics were chosen to capture different aspects of the agents' success in the maze navigation task:

- **Success Rate (%):** The primary metric of performance, defined as the percentage of training episodes in which the agent successfully navigated the ball to the maze exit without failing or timing out.
- **Average Time to Solve (steps):** For successful episodes only, this metric measures the average number of simulation timesteps required to complete the maze. It serves as a proxy for the agent's efficiency.



(a) Reward evolution (b) Success Rate

Fig. 5: Training metrics for the agent. (a) Cumulative reward increases as training progresses. (b) Success rate increases over time.

In addition, to intrinsically compare a specific approach and its evolution during training, we used the average cumulative reward, which is a standard metric in reinforcement learning. It measures how well the agent optimized its specific reward function.

B. Baseline Task Performance

Before addressing the complete maze-solving challenge, the agent's capabilities were validated on two foundational tasks: Plate Balancing and Go-to-Target Navigation. These simpler experiments confirmed the viability of the reinforcement learning approach for the core control problem and established a performance baseline.

The first task, **Plate Balancing**, required the agent to simply keep the ball on the plate. The agent demonstrated exceptional performance, quickly learning an effective stabilization policy. As shown in the success rate plot, Fig. 5(b), the agent consistently achieved a near-perfect success rate after a brief training period, indicating it successfully learned to prevent the ball from falling off the plate for the entire duration of the episodes. This is corroborated by the cumulative reward curve, Fig. 5(a), which shows a rapid convergence to a maximum value, reflecting the agent's consistent success.

The **Balance to Target** navigation task increased the complexity by requiring the agent to not only balance the ball but also guide it to a specific target coordinate. The agent's performance, illustrated by the cumulative reward plot (Fig. 6), was also highly successful. Since the reward function for this task is proportional to the ball's proximity to the target, the high and stable final reward values indicate that the agent consistently learned to maneuver the ball to the target and maintain its position there. A further analysis of the reward formula shows that the final smoothed cumulative reward of approximately 985 corresponds to an average displacement of 0.15 from the target position along each horizontal axis. Given the ball has a radius of 0.5 and the plate has a size of 10×10, we can conclude the agent excelled at the given task.

In summary, the strong results from these baseline tasks validated that the RL agent could master the fundamental dynamics of the ball-and-plate system. This provided a solid foundation for tackling the more complex hierarchical and end-to-end maze navigation approaches.

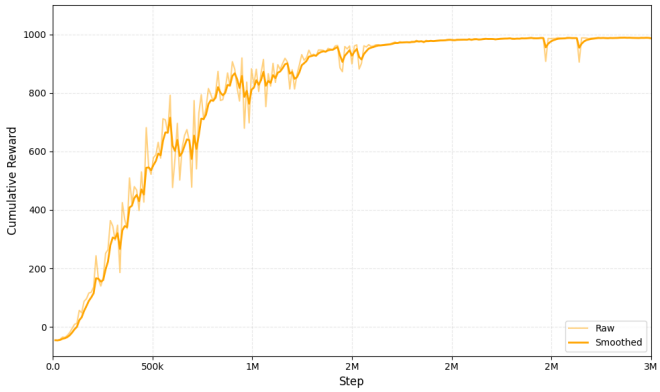


Fig. 6: Training curve showing cumulative reward as the agent learns a navigation policy toward target positions

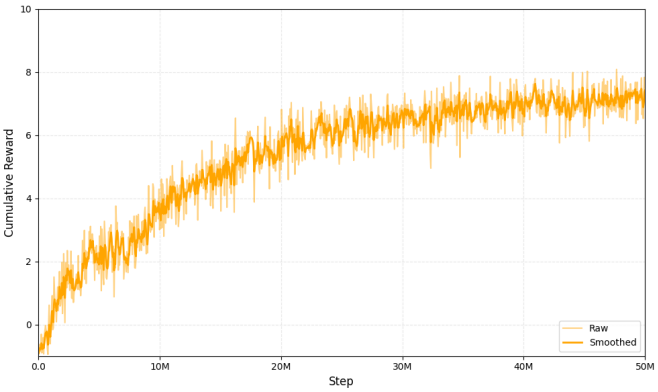
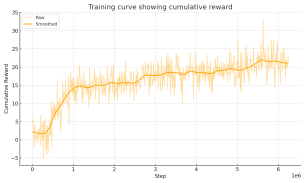
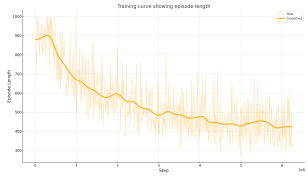


Fig. 8: Training curve showing cumulative reward as the agent learns a navigation policy through the maze to find the goal

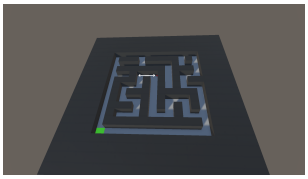


(a) Cumulative Reward

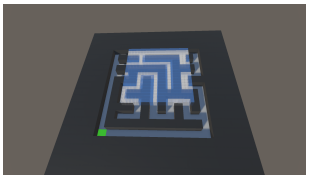


(b) Episode Length

Fig. 7: Training metrics for the hierarchical agent. (a) Cumulative reward increases as training progresses. (b) Average episode length decreases over time.



(a) Repetitive Motion



(b) Limited Grid Observation

Fig. 9: A specific failure case for the end-to-end agent. (a) The agent becomes stuck in a repetitive back-and-forth motion, indicated by the white arrows. (b) The reason for the failure: the agent's limited field of view (white and blue grid) does not allow it to see the path required to exit the area.

C. Hierarchical Approach Performance

The hierarchical approach combines classical path planning for global navigation with a reinforcement learning agent for local control. This structure proved to be highly effective.

After 6 million training steps, the agent achieved a success rate of 85.8%, with successful episodes being completed in an average of 992 steps. The training curves in Fig. 7 show consistent and stable learning progression. The cumulative reward (Fig. 7(a)) increases rapidly in the initial phase before stabilizing. Concurrently, the average episode length (Fig. 7(b)) decreases as the agent learns more efficient trajectories between the waypoints provided by the planner.

Analysis of failure cases revealed that timeouts were the primary failure mode. These typically occurred when the ball's velocity approached zero near a corner. This behavior is indicative of the agent settling into a sub-optimal policy where it prioritizes stability over progress. In these situations, the perceived risk of attempting a difficult maneuver (like turning a sharp corner) outweighs the reward for making forward progress, causing the agent to stagnate.

To counteract this, a potential solution would be to augment the reward function with a **stagnation penalty**. This would involve giving the agent a small negative reward in each step where the ball's velocity falls below a certain threshold, thus discouraging "lazy" or overly cautious behavior.

D. End-to-end Approach Performance

Achieving strong end-to-end performance required extensive experimentation, including multiple iterations of adjustments to the observation space and reward function tuning, many of which did not succeed. However, the final setup delivered great results, as detailed below.

The reward function used in this experiment has a theoretical maximum of 10, which would correspond to the ball reaching the goal cell immediately in every episode. By the end of training, our agent achieved a smoothed cumulative reward of approximately 7.51, as illustrated in Fig. 8. While this falls short of the optimal value, it demonstrates strong performance. To further assess this outcome, we monitored additional metrics: the smoothed success rate reached 80.4%, and the agent required an average of around 1040 physics steps to reach the goal.

The main advantage of the end-to-end approach is its seamless, automated learning process, which avoids the manual integration of planning and control modules required by a hierarchical system. However, this monolithic design struggles with the complexities of specific harder mazes, as shown by its lower success rate. This difficulty is due to the agent's limited visual and memory capabilities, which can cause it to get stuck, as shown in Fig. 9(a) and Fig. 9(b).

TABLE II: Comparison between Maze Navigation Strategies

Metric	Hierarchical	End-to-End
Avg. Success Rate	0.858	0.804
Avg. Steps to Completion	992	1040
No. of Training Steps	6M	50M

This happens because the agent's grid-based observation is not large enough to see the correct path forward, and its short-term memory is insufficient to support deeper exploration. Essentially, the agent cannot see the path it needs to take and isn't equipped to find it through exploration.

E. Comparative Analysis

A direct comparison highlights the distinct trade-offs between the hierarchical and end-to-end strategies, with quantitative results summarized in Table II.

The hierarchical approach proved to be the most effective strategy, achieving higher success rates (85.8% vs. 80.4%) and solving mazes slightly faster. Most notably, it demonstrated vastly superior sample efficiency, requiring only 6 million training steps compared to 50 million for the end-to-end agent.

While the end-to-end approach offers generality through automated learning, its limited local observations made it susceptible to getting stuck in complex maze sections. The hierarchical strategy's strength lies in problem decomposition: by using classical planning for pathfinding, the RL agent can focus on local navigation, leading to faster learning and higher performance.

VI. Conclusions

This study successfully demonstrated that deep reinforcement learning, specifically Proximal Policy Optimization (PPO), is a highly effective method for solving the complex, non-linear control problem of maze navigation on a ball-balancing platform. The project's primary contribution is a rigorous comparative analysis of different deep reinforcement learning strategies for this novel task.

The research highlights a key trade-off between different RL architectures. A hierarchical approach, combining A* for high-level pathfinding and a PPO agent for low-level control, proved very effective. Nevertheless, an end-to-end PPO agent, equipped with a CNN and an LSTM, also achieved promising success rates. This end-to-end approach, although more complex to tune, offers more seamless integration by internally learning the connections between high-level planning and low-level control, removing the need for a hand-coded interface between the modules.

While acknowledging the sim-to-real gap inherent in the chosen physics engine, this project establishes a strong proof-of-concept for the control strategies themselves, which can be adapted to physical systems.

By successfully training agents on procedurally generated mazes, this study underscores the generalization capabilities of RL in dynamic and uncertain robotic control environments. The developed modular framework also serves as a valuable educational and research tool for future explorations into robotic learning.

VII. Future work

While this project successfully demonstrates the viability of deep reinforcement learning for maze navigation in a simulated environment, a significant avenue for future research is bridging the "sim-to-real" gap. The current implementation, which uses the Unity engine, intentionally abstracts away certain hardware complexities to focus on agent logic. Transferring the learned policies to a physical robotic platform would require addressing several key challenges:

- **Physical System Dynamics:** The project currently bypasses the inverse kinematics problem by allowing the agent to directly control the plate's tilt angles. A physical implementation would likely involve a multi-axis robotic arm, requiring the control policy to output commands that can be translated into specific robot joint angles. Future work should involve integrating and solving this inverse kinematics challenge.
- **Sensor Integration:** The simulation provides direct information about the ball's position, velocity, and the maze structure. A real-world system would rely on physical sensors, which introduce noise and latency. Future iterations could explore:
  - Ball Tracking: The ball's position and velocity would need to be tracked in real-time. This could be achieved with a physical sensor, such as a touch-sensitive plate that functions like a touchscreen to monitor the ball's movement [7], [31]. The velocity could be estimated by the sensor system or alternatively inferred by the agent from a stack of consecutive position observations [17].
  - Maze Information: Likewise, the maze structure would no longer be automatically known. A physical system would need to acquire this layout, either by using a pre-loaded digital map or by dynamically registering the wall positions after placing the maze on a touch-sensitive surface.
- **Physics Fidelity:** The Unity PhysX engine is optimized for game performance, not engineering-grade physical accuracy. Factors like friction, material properties, and contact dynamics may differ significantly from a real-world setup. A future research direction would be to either transfer the training to a high-fidelity simulator like MuJoCo or to employ domain randomization techniques during training [24], [32]. This involves training the agent across a wide range of physics parameters (e.g., varying friction, ball mass) to create a more robust policy that can adapt to real-world conditions.

By tackling these challenges, the strategies developed in this study could be validated in a physical system, marking a critical step towards real-world application.

REFERENCES

[1] A. Knuplez, A. Chowdhury, and R. Svecko, "Modeling and control design for the ball and plate system," in *IEEE International Conference on Industrial Technology, 2003*, vol. 2, 2003, pp. 1064–1067 Vol.2.

[2] H. Bang and Y. S. Lee, "Implementation of a ball and plate control system using sliding mode control," *IEEE Access*, vol. 6, pp. 32 401–32 408, 2018.

[3] W. Wei and P. Xue, "A research on control methods of ball and beam system based on adaptive neural network," in *2010 International Conference on Computational and Information Sciences*, 2010, pp. 1072–1075.

[4] K. Latha, V. Rajinikanth, and P. Surekha, "Pso-based pid controller design for a class of stable and unstable systems," *International Scholarly Research Notices*, vol. 2013, no. 1, p. 543607, 2013.

[5] T. Zhang and H. Mo, "Reinforcement learning for robot research: A comprehensive review and open issues," *International Journal of Advanced Robotic Systems*, vol. 18, no. 3, p. 17298814211007305, 2021.

[6] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[7] J. Kumar, N. Showme, M. Aravind, and R. Akshay, "Design and control of ball on plate system," *Int J Control Theory Appl*, vol. 9, no. 34, pp. 765–778, 2016.

[8] A. Juliani, V. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *CoRR*, vol. abs/1809.02627, 2018. [Online]. Available: http://arxiv.org/abs/1809.02627

[9] C. Bartneck, M. Soucy, K. Fleuret, and E. B. Sandoval, "The robot engine — making the unity 3d game engine work for hri," in *2015 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2015, pp. 431–437.

[10] P. JhA, "Inverse kinematic analysis of robot manipulators," Ph.D. dissertation, NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA, INDIA, 2015.

[11] P. Roy, B. Kar, and I. Hussain, "Trajectory control of a ball in a ball and plate system using cascaded pd controllers tuned by pso," in *Proceedings of Fourth International Conference on Soft Computing for Problem Solving: SocProS 2014, Volume 2*. Springer, 2015, pp. 53–65.

[12] K. J. Astrom, "Pid controllers: theory, design, and tuning," *The international society of measurement and control*, 1995.

[13] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *Transactions of the American society of mechanical engineers*, vol. 64, no. 8, pp. 759–765, 1942.

[14] R. E. Kalman *et al.*, "Contributions to the theory of optimal control," *Bol. soc. mat. mexicana*, vol. 5, no. 2, pp. 102–119, 1960.

[15] V. I. Utkin, *Sliding modes in control and optimization*. Springer Science & Business Media, 2013.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:205242740

[18] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," in *Conference on robot learning*. PMLR, 2018, pp. 561–591.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019. [Online]. Available: https://arxiv.org/abs/1509.02971

[20] T. Bi and R. D'Andrea, "Sample-efficient learning to solve a real-world labyrinth game using data-augmented model-based reinforcement learning," 2023. [Online]. Available: https://arxiv.org/abs/2312.09906

[21] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018. [Online]. Available: https://arxiv.org/abs/1801.01290

[22] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," 2019. [Online]. Available: https://arxiv.org/abs/1812.05905

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

[24] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[25] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[26] O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004. [Online]. Available: http://www.ars-journal.com/International-Journal-of-Advanced-Robotic-Systems/Volume-1/39-42.pdf

[27] A. Harris and J. M. Conrad, "Survey of popular robotics simulators, frameworks, and toolkits," in *2011 Proceedings of IEEE Southeastcon*, 2011, pp. 243–249.

[28] S. Tselegkaridis and T. Sapounidis, "Simulators in educational robotics: A review," *Education Sciences*, vol. 11, p. 11, 01 2021.

[29] J. Collins, S. Chand, A. Vanderkop, and G. Howard, "A review of physics simulators for robotic applications," *IEEE Access*, vol. PP, pp. 1–1, 03 2021.

[30] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[31] A. Zeeshan, N. Nauman, and M. Jawad Khan, "Design, control and implementation of a ball on plate balancing system," in *Proceedings of 2012 9th International Bhurban Conference on Applied Sciences & Technology (IBCAST)*, 2012, pp. 22–26.

[32] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 23–30.

APPENDIX

In this section, you will find the instructions to set up the environment and run the experiments. You can find more information in the GitHub repository[1].

A. Prerequisites

Please ensure you have the following installed:

- Python 3.10.12
- Unity Hub
- Unity Editor 6.1

B. Running the Code

1) Clone the repository and navigate into the directory:
```
git clone https://github.com/
↪ Minipoloalex/maze-solver-rl.git
cd maze-solver-rl
```
2) (Optional) Install the required Python packages. This step is only required for training new models.
```
pip install -r requirements.txt
```
3) Open the project in the Unity Hub using the specified Unity Editor version.
4) Navigate to the `Assets/Scenes` folder and select the desired scene.
5) The project is pre-configured to run with a trained model. Press the Play button in the Unity Editor to start the simulation.

To switch between the Hierarchical and End-to-End strategies, select the corresponding agent prefab `GameObject` in the scene.

[1] https://github.com/Minipoloalex/maze-solver-rl