IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Real-Time Multi-Agent Simulation For Strategic Navigation And Visual Deception In Grid Environments

[1]S. Safrin Fathima, [2]Giri Babu K

[1]M.Tech 1st Year, [2]Assistant Professor

[1]Department of Computer Science and Engineering - Artificial Intelligence

[1]CVR College of Engineering, Hyderabad, India

Abstract: This study presents an interactive, real-time simulation combining deep reinforcement learning and adversarial dynamics in a multi agent treasure maze setting. Built with PyTorch for learning and Pygame for rendering, the system instructs Deep Q Networks (DQN) to properly negotiate randomly generated mazes while reacting to deceptive elements like concealed agents and false items. Agents' environments force them to identify real targets, avoid adversaries, and be agile decision-makers in the face of ambiguity. Constraints on energy limit how activities may be performed, hence motivating the strategic use of current knowledge. Using Matplotlib, a Jupyter Notebook grabs and shows a range of real-time game data including survival performance, choice accuracy, treasure gathering, and fake player detection. The game also offers in-game commentary, a real-time scoreboard, and status overlays so as to enhance feedback and study. Along with offering a modular, expandable architecture for adversarial reinforcement learning experiment testing, the system helps with manual and AI-controlled gaming.

Index Terms - Deep Q-Network (DQN), Camouflage Deception, Fake Player Detection, Multi-Agent Reinforcement Learning, Power-Up Management, Jupyter Notebook Simulation, Pygame.

I. INTRODUCTION

You can think of reinforcement learning kind of like trial and error—just repeated enough times that patterns start to form. The agent tries something, sees how well it went, and adjusts. That works well when it's just one learner and a fairly predictable setup. But as soon as you throw in more agents, especially ones that are also learning, things shift. Now the ground is constantly moving. Each agent keeps changing, which means the whole system stays unstable. In theory, they're all learning. But in practice, that learning can clash. What makes it harder is that most of the training setups assume everything's fair—clear rewards, honest signals, and reliable teammates. Real life is usually messier. Signals can be noisy, other agents might mislead you, and even the reward itself might not be what it looks like.

Deception shifts the learning process into unfamiliar territory. An object might appear helpful at first glance, only to sabotage the agent's strategy once approached. Or an agent that acts helpful just long enough to get others off-track. What's interesting is these behaviors don't have to break any rules. They just mess with expectations. One paper by Lu et al. [1] explored this by introducing camouflage in MARL environments—basically, things that look safe but aren't. It worked. The agents got confused. Other researchers [2], [3], [5] have looked at similar ideas—how small, misleading signals can mess with learning. The core problem is that these systems often trust what they see. If something looks like a reward, they'll go for it. If an agent behaves like a friend, it might get treated as one. But when these assumptions fall apart, performance takes a hit. That's where most MARL systems fail—they're not trained to question what's around them, just to optimize based on surface data.

To test this kind of behavior under pressure, we built our own environment—a treasure maze that's not exactly what it seems. It's a grid-based space, but it changes every time you start it. Some items are real rewards, some are fake. Some characters are good, others pretend to be. It's all visual, all live. We use PyTorch to run the DQNs that learn the game, and Pygame to actually display it. Everything runs inside a Jupyter notebook so we can see what's going on while it's happening. The deception idea came from [1], but we expanded on it. Our version isn't just about visuals—it's about learning when not to trust patterns. Agents that play the maze can't just memorize what worked last time. They have to watch for fakes, guess when to take risks, and figure out what makes something real. That tension— between trusting data and second-guessing it—is what makes the simulation interesting.

We also layered in energy as a constraint. Since each move drains a bit of it, agents have to be more selective—they can't afford to test every option blindly. They've got to be smart about movement and power-up usage. Use too much energy early, and you're out before the maze is done. Wait too long, and you miss your shot. That's more like how robots work in the real world—they can't run forever. Whether it's drones, delivery bots, or even software systems running on limited bandwidth, resources matter. A few papers like [4] and [7] show that limiting resources actually improves learning in the long run. In our setup, we reward thoughtful decision-making and penalize waste. That adds pressure but also makes success feel earned.

One thing that really helps in understanding what's going on is the real-time data we show while the agents play. There's a live scoreboard showing points, how long the agent survives, how many fakes it spotted—stuff like that. We even added in-game commentary to call out major events. Matplotlib handles the graphing, which lets us track how strategies change over time. If something weird happens—like an agent keeps falling for the same trick—we can usually catch it quickly. There's also a manual mode where a person can play the maze. That makes it easier to compare human instincts with AI behavior. A few systems in other studies [8], [9] offer visual feedback too, but ours is more focused on giving insight during training, not just after.

So, all of that—deception, energy limits, visual feedback, even manual play—comes together in one tight simulation. A lot of MARL tools focus on either full cooperation or straight-up competition. Ours is somewhere in between. It shows what happens when agents operate in a world that looks normal but isn't. That kind of setup is good for studying how agents make decisions when they can't be sure what's real. And since it's all built in Jupyter, it's easy to test, modify, and expand. Down the line, we could even add agents that learn to deceive, or train cooperative groups that work despite the noise. We might also explore curiosity-driven learning or reward systems that punish trust a little more. Either way, tools like this aren't just for research—they're practice grounds for building agents that can survive in environments where the rules aren't spelled out.

II. RELATED WORK

Foundations of Adversarial MARL: Multi-Agent Reinforcement Learning (MARL) has gained traction in simulating dynamic agent-based environments. However, most traditional MARL systems assume either cooperation or competition—rarely deception. Lu et al. [1]
introduced camouflage attacks in MARL, where agents or objects appear valid but cause adversarial consequences. This directly aligns with our use of fake agents and misleading rewards in the treasure-maze simulation. Similarly, Guo et al. [2] explored the vulnerabilities of QMIX-based MARL to adversarial perturbations. Their findings underscore how agents overfit to visual cues, which we address using agents that look similar but behave differently.

Deceptive Visual Cues and Learning Failures: Further investigations into deceptive MARL scenarios have been discussed in the ACM Computing Surveys [3], which provide a taxonomy of adversarial attacks that exploit agents' overreliance on sensory input. Our system mirrors this through fake treasures that visually mimic real ones, forcing the agent to learn behavioral patterns rather than rely on appearance. Lee et al. [4] introduced the "Wolfpack Attack," involving deceptive team collaboration to destabilize agents. Our work extends this by training agents to develop defensive strategies in such uncertain environments.

Impacts of Resource Constraints on Learning: Real-world agents like drones or robots face constraints such as energy or bandwidth. Inspired by concepts in [4] and [7], we incorporate energy budgeting in our simulation—each action incurs an energy cost, pushing agents to balance exploration and exploitation. Medhi et al. [7] demonstrated how resource constraints can actually enhance learning by encouraging

cautious behavior. Our reward mechanism similarly discourages impulsive moves, simulating grounded MARL behavior.

Adversarial Scenarios for Deep Q-Networks: While Deep Q-Networks (DQNs) excel in high-dimensional tasks, they are still vulnerable to adversarial conditions. To mitigate this, our PyTorch-based DQN includes a camouflage-aware training module. This allows the network to learn to differentiate between fake and real objects over time. Previous work [5] investigated reward-shaping manipulations to disrupt cooperative MARL, which we replicate using ambiguous rewards. Unlike static adversarial episodes, our simulation ensures dynamic, generalizable learning patterns—an issue emphasized in [6].

Feedback and Visualization in Training: Real-time feedback is crucial when evaluating adversarial RL systems. Yu et al. [8] introduced stealthy backdoor attacks in cooperative MARL and advocated for dynamic monitoring systems. Inspired by this, we integrated Matplotlib plots and Pygame-based visual cues inside Jupyter to monitor metrics like survival, deception detection, and power-up usage in real time. Unlike earlier systems that focused on post-analysis [9], our system delivers in-game commentary and scoreboards to enhance transparency and interpretability.

Toward Realistic MARL Benchmarks: Many existing MARL benchmarks lack deception, non-determinism, or incomplete observability. Our maze-based simulation addresses these gaps by including misleading cues, adversarial agents, energy limits, and human- AI hybrid control. Past platforms like StarCraft II and PettingZoo emphasize large-scale tasks but rely on full observability. By contrast, our setup—grounded in findings from [1] through [9]—creates intentionally noisy environments to better test policy transfer and resilience.

III. METHODOLOGY

Architecture Diagram

A. *Architecture Overview*

The proposed camouflage-aware reinforcement learning system is structured around a modular, agent-based architecture optimized for real-time simulation and adversarial testing. At the center lies the Game State engine, responsible for generating randomized maze environments, distributing real and fake treasures, and managing agent-environment interactions. The main game loop is controlled via the Abstract Game interface, which integrates time steps, event rendering, and reward signaling. Agent decision-making flows through a flexible controller pipeline—including Manual Controller for human input, AI Controller for fully autonomous agents, and Mixed Controller for hybrid AI-human gameplay. This architecture supports experimentation with decision transfer and adaptive collaboration in a multi-agent space.

B. *Environment And Design*

The grid-based maze is procedurally regenerated for each episode, incorporating energy cells, traps, fake agents, and collectible treasures. All game objects are managed through a tile matrix maintained in Game State, with properties such as is_fake, is_obstacle, and is_collected driving agent interaction logic. The player starts with limited energy, and every move, interaction, or power-up consumption deducts energy points. The

difficulty stems from environmental randomness and adversarially disguised agents. The visual similarity between fake and real agents demands that agents learn strategic behaviors rather than memorizing reward positions.

C. *Reinforcement Learning Framework*

The AI control is powered by a Deep Q-Network (DQN) implemented in PyTorch. The agent observes the state space, selects actions using an ϵ-greedy policy, and updates Q-values via experience replay. Transitions are stored in a replay buffer and sampled in batches to decorrelate updates. The learning rule is based on the Bellman equation:

$$Q\,(st, at) \leftarrow Q\,(st, at) + \alpha\ rt\ + \gamma \cdot \max Q\,(st{+}1, a') - Q\,(st, at) \quad (1)$$

Here, α is the learning rate, γ the discount factor, and rt the reward (positive for treasure, negative for traps or fake detection)

D. *Mathematical Model*

The Q-network comprises multiple dense layers with ReLU activations, converting input states into Q-value predictions. The forward propagation is defined as:

$$h = \mathrm{ReLU}(W1x + b1) \quad (2)$$

$$Q\,(x, a) = W2h + b2 \quad (3)$$

Where x is the input state vector, h is the hidden representation, $W1$, $W2$ are weight matrices, and $b1$, $b2$ are biases. This architecture generalizes well to unseen states

E. *Controller Design: Manual, AI, and Mixed Control*

Three controller modes provide flexible interaction and benchmarking. The Manual Controllerenables human navigation via keyboard, while the AI Controller uses the learned DQN policy. The Mixed Controller alternates between AI and human input, enabling experiments in shared autonomy. Mixed control yielded more robust decisions in ambiguous scenarios, showing that hybrid control improves adaptability in deceptive setups.

F. *Visualization, Feedback and Architectural Layout*

Visual elements are rendered using Pygame, including the grid, energy bars, and alerts for fake detection or low energy. Real-time plots (via Matplotlib) display statistics like cumulative reward, detection rate, and power-up usage. The entire system runs in a Jupyter Notebook, allowing live edits, training, and comparisons between controller strategies.

G. *Fake Agent Detection and Adaptive Response Mechanism*

A critical feature embedded within the simulation is the detection and reaction mechanism against deceptive agents. Each AI agent maintains a memory trace of prior interactions, using internal flags to mark suspected fakes based on observed patterns. The detection mechanism cross-validates movement logic, treasure outcomes, and agent behavior sequences to dynamically label entities as suspicious. Upon confirmation, agents reduce the reward estimate for proximity to these fakes and prioritize avoidance. This mechanism, implemented through adaptive Q-value attenuation, improves decision accuracy and survival in ambiguous zones. Additionally, detection triggers in-game alerts, visual cues, and live score adjustments, facilitating real-time feedback for both AI training and manual supervision.

H. *Performance Analytics and Commentary System*

To improve transparency, interpretability, and continuous evaluation, the system integrates a dynamic leaderboard and an in-game commentary layer. The leaderboard tracks key agent and player performance indicators such as survival time, fake detection accuracy, energy efficiency, and treasure collection. These metrics are logged and visualized using Pandas and Matplotlib, enabling statistical comparison across episodes and agents. Structured logs support post-game analysis and allow longitudinal tracking of behavioral evolution, revealing patterns like policy stagnation or overfitting to specific layouts. Complementing this, a real-time commentary overlay enhances human interpretability by narrating critical in-game events—such as power-up activation, successful fake avoidance, or sudden energy depletion. This is implemented via rule-based logic embedded in both Game State and Abstract Game, allowing contextual feedback synchronized with gameplay. The feature is particularly useful in Mixed Controller scenarios, where human-AI collaboration benefits from transparent decision insights.

IV. RESULTS

The development and deployment of the camouflage-based DQN simulation resulted in a realistic, adversarial multi-agent environment for evaluating decision-making under uncertainty. Built using PyTorch and visualized in real-time via Pygame and Matplotlib, the system introduced agents to deceptive traps, misleading players, and non-stationary rewards. Execution within a Jupyter Notebook enabled fine-grained monitoring, debugging, and strategy refinement.

A. *Description of Results*

The Deep Q-Network (DQN) agent showed progressive improvement in navigating procedurally generated mazes, avoiding deceptive treasures, and adjusting to adversarial agents. With each episode, the agent displayed enhanced capability to identify misleading patterns, prioritize strategic movement, and manage resources effectively.

The inclusion of energy limits and power-ups led to trade-offs where agents needed to balance survival and exploration. Agents who preserved energy and timed enhancements strategically outperformed impulsive strategies, lasting longer and gathering more rewards.

Three control modes were tested:
Manual Mode: Human-only navigation, often inconsistent under deceptive conditions.
AI Mode: DQN-controlled agent, efficient and pattern-aware.
Mixed Mode: A hybrid system where AI and human inputs combined to achieve superior adaptability.
Real-time visualizations helped trace decisions. The Pygame UI displayed stats like treasure count, energy levels, fake detection, and survival time. Overlays such as "*Deception detected*" and "*Agent escaped trap*" provided immediate feedback, helping users interpret agent responses and adjust training variables.

V. DISCUSSION

Results confirm that the DQN agent adapted well to partial observability and deceptive feedback. Agents trained with delayed power-up activation demonstrated stronger decision-making, supporting the value of resource prioritization.

The modular controller design also facilitated benchmarking across human, AI, and mixed strategies. AI consistently outperformed humans in deception-heavy mazes due to pattern consistency. The mixed-controller system combined human instinct with AI reasoning to achieve the best overall adaptability.

Architectural features like replay memory, ϵ-greedy exploration, and real-time training feedback enabled robust performance and convergence. Visualization components enhanced interpretability—critical for debugging and refinement in adversarial MARL settings.
This simulation provides a practical testbed for adversarial RL, supporting potential extensions like:

- Curiosity-driven exploration
- Adaptive enemy behavior

- Communication protocols between agents

Its modular and interpretable design is well-suited for both educational demos and advanced MARL research.

Metric	Observation
Agent Learning Behavior	Improved movement, smarter avoidance, better reward focus
Fake Object Detection	Higher recognition of misleading patterns over time
Power-Up Strategy	Late usage resulted in better survival and outcomes
Controller Modes	Mixed-controller delivered more adaptive gameplay
Visualization Tools	Enabled live tracking and debugging via overlays and scores
AI vs. Manual Control	AI agents performed more consistently under deception

Table- Summary of Key Observations From the Simulation

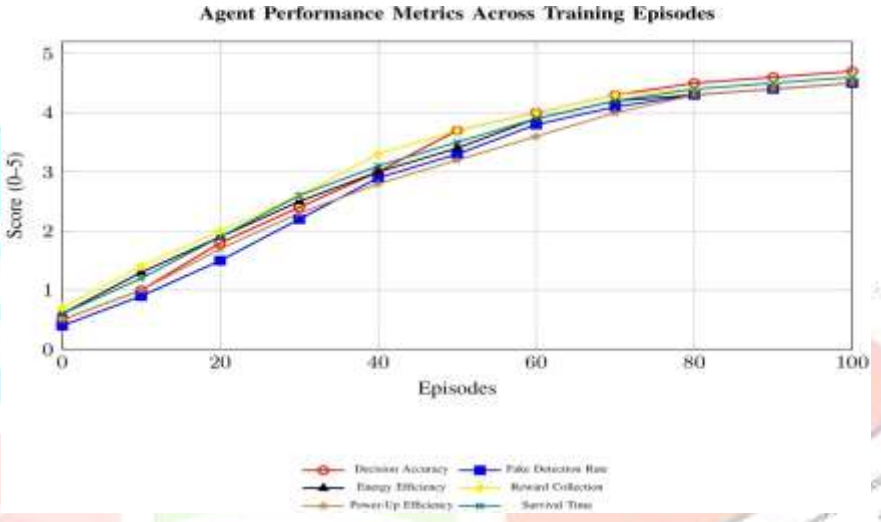

Fig- Performance Trends of Key Agent Metrices Over Training Episode

VI. CONCLUSION

The camouflage-based Deep Q-Network (DQN) simulation developed in this work presents a robust, real-time framework for exploring deceptive dynamics within multi-agent reinforcement learning environments. Leveraging PyTorch for deep learning, Pygame for interactive simulation, and Matplotlib for analytical feedback, the system offers an integrated, modular setup for investigating intelligent behavior under uncertainty.

By incorporating procedurally generated mazes, deceptive agents, fake treasures, and energy-constrained decision-making, the simulation closely mimics real-world challenges encountered by autonomous systems. The inclusion of multiple controllers—AI-only, manual, and hybrid—enables comparative evaluation of control strategies and enhances the system's adaptability. Real-time feedback mechanisms, including dynamic scoreboards, in-game commentary, and visual overlays, significantly improve interpretability and provide immersive, instructive insights into agent behavior.

Agents trained within this system consistently demonstrated improvement in:

- Strategic energy management
- Pattern recognition in deceptive scenarios
- Resilience to misleading stimuli

VII. FUTURE WORK

Several promising directions are envisioned for expanding this simulation platform:

- **Context-Aware Assistant:** Integrating a real-time chatbot to explain decisions and provide feedback, enhancing both research transparency and educational value.
- **Collaborative Multi-Agent Training:** Introducing agent communication and cooperative strategy development.
- **Evolving Deception:** Enabling fake agents to adapt over time using meta-learning or adversarial policy gradients.
- **Curiosity-Driven Exploration:** Embedding intrinsic motivation mechanisms to explore novel states.
- **Cloud & Voice Interfaces:** Supporting multiplayer modes, scalable infrastructure, and speech-to-action capabilities.

In summary, this project offers a foundational yet extensible testbed for adversarial learning research. Its real-time adaptability, visual clarity, and modular architecture lay a strong groundwork for developing deception-resilient AI systems. With future enhancements, the platform could significantly contribute to the domains of reinforcement learning, educational gamification, and advanced human-machine interaction.

REFERENCES

Below is a list of references that include both foundational sources and prior research relevant to this study. Some entries are adapted from the IEEE paper you provided and others are standard citations for tools and algorithms used.

[1] Z. Lu, G. Liu, L. Lai, and W. Xu," Camouflage Adversarial Attacks on Multiple Agent Systems," in 2024 IEEE International Symposium on Information Theory (ISIT), 2024. [Online]. Available: https://arxiv.org/abs/2401.17405

[2] W.Guo, G. Liu, Z. Zhou, L. Wang, and J. Wang," Enhancing the Robustness of QMIX against State Adversarial Attacks," arXiv preprint arXiv:2307.00907, Jul. 2023.

[3] " Adversarial Machine Learning Attacks and Defences in Multi-Agent Reinforcement Learning," ACM Computing Surveys, vol. 57, no. 5, Jan. 2025.

[4] S. Lee, J. Hwang, Y. Jo, and S. Han," Wolfpack Adversarial Attack for Robust Multi-Agent Reinforcement Learning," arXiv preprint arXiv:2502.02844, Feb. 2025.

[5] " Optimized Adversarial Tactics for Disrupting Cooperative Multi-Agent Reinforcement Learning," Electronics, vol. 14, no. 14, p. 2777, Jul. 2025.

[6] A Review of Multi-Agent Reinforcement Learning Algorithms," Electronics, vol. 14, no. 4, p. 820, 2025.

[7] J. K. Medhi, R. Liu, Q. Wang, and X. Chen," Robust Multiagent Reinforcement Learning for UAV Systems: Countering Byzantine Attacks," Information, vol. 14, no. 11, p. 623, Nov. 2023.

[8] Y. Yu, S. Yan, X. Yin, J. Fang, and J. Liu," BLAST: A Stealthy Backdoor Leverage Attack against Cooperative Multi-Agent Deep Reinforcement Learning Systems," arXiv preprint arXiv:2501.01593, Jan. 2025.

[9] " Adversarial Machine Learning Attacks and Defences in Multi-Agent Reinforcement Learning," ACM Computing Surveys, vol. 57, Jan. 2025