INTERNATIONAL JOURNAL OF INNOVATIVE COMPUTING
ISSN 2180-4370
Journal Homepage : https://ijic.utm.my/

Exploring Agent Behavior and Performance in Shortened Simulations via Multiagent Reinforcement Learning

Stanislav Safranek*
Department of Information Technologies
University of Hradec Kralove
Hradec Kralove, Czech Republic
Email: stanislav.safranek@uhk.cz

Brian Kirk
New Mexico Institute of Mining and Technology
New Mexico, United States of America

Submitted: 15/9/2025. Revised edition: 28/11/2025. Accepted: 5/1/2026. Published online: 28/7/2026
DOI: https://doi.org/10.11113/ijic.v16n1-2.683

***Abstract*—This study explores the potential of Multiagent Reinforcement Learning (MARL) for autonomous navigation in a discrete two-dimensional environment. We design and implement an agent that learns an optimal path through a 5×5 grid via repeated interactions and a reward-based mechanism. Over 500 training episodes, we examine the convergence speed of the learned policy, the stability of agent behavior, and the success rate in reaching the goal. Our approach combines artificial neural networks with a multiagent framework, enabling decentralized decision making and scalable adaptation. We discuss critical factors affecting learning stability, including reward function design and network architecture, and outline avenues for extending the methodology to more complex, real-time tasks. The findings demonstrate MARL's promise in solving navigation problems efficiently and provide concrete recommendations for tuning training parameters and network structures to enhance performance and robustness.**

***Keywords*—Agent-based Simulation, Artificial Neural Networks (ANN), Multiagent Reinforcement Learning, Pathfinding Optimization, Reinforcement Learning Stability, Statistical Performance Analysis**

I. INTRODUCTION

The integration of Artificial Neural Networks (ANN) and Multi-Agent Systems (MAS) represents a significant advancement in the development of intelligent adaptive systems. This article explores the theoretical foundations and practical applications of combining ANN and MAS, highlighting their synergy and benefits. Multi-Agent Systems (MAS) are a branch of distributed artificial intelligence where multiple autonomous agents operate in a shared environment to achieve goals. Each agent uses sensors and actuators to interact based on local perception and decision-making capabilities, including cooperation and coordination [17]. Effective communication uses protocols like the Agent Communication Language (ACL) [3], while coordination and cooperation strategies improve system performance [13]. Achieving consensus is crucial in dynamic environments [8]. Artificial Neural Networks (ANN) consist of interconnected nodes (neurons) that process information using a connectionist approach. Typical architectures include feedforward, recurrent, and convolutional neural networks [5]. Activation functions introduce non-linearity, enabling the learning of complex patterns, while training algorithms like backpropagation adjust the weights [11]. Techniques like regularization and dropout ensure robust performance [15]. Reinforcement learning (RL) allows agents to learn to make decisions by interacting with the environment to maximize cumulative rewards. The Bellman equation facilitates updating value estimates, while algorithms like Q-learning optimize the learning process [16]. Integrating ANN with MAS enhances agents' capabilities by enabling learning and adaptation, leading to more scalable and efficient systems. Neural networks help approximate optimal policies in high-dimensional spaces, improving coordination and real-time decision-making [9]. The combination of these technologies creates intelligent systems capable of addressing complex challenges. This article aims to highlight the benefits and challenges of integrating ANN and MAS, supported by recent advancements and practical applications, and to show how this integration can lead to innovative solutions across various industries.

II. LITERATURE REVIEW

Multi-agent systems (MAS) are a type of distributed artificial intelligence where multiple autonomous agents interact within a shared environment to achieve individual or collective goals. Each agent operates based on its local perception and decision-making capabilities, which may include cooperation, competition, coordination, and negotiation with other agents. The key theoretical aspects of MAS include:

- *Agent Definition*: An agent is an entity that perceives its environment through sensors and acts upon that environment using actuators. Agents are characterized by autonomy, social ability, reactivity, and pro-activeness [17].
- *Agent Communication*: Effective communication is essential for MAS. Agents use various protocols and languages, such as the Agent Communication Language (ACL), to exchange information, negotiate, and coordinate their actions [3]
- *Coordination and Cooperation*: Coordination refers to the process of managing interdependencies between activities performed by agents. Cooperative strategies include forming coalitions, joint plans, and sharing [13].
- *Consensus and Agreement*: In MAS, reaching consensus is critical, especially in scenarios like sensor networks, robotics, and resource allocation. Consensus algorithms ensure that agents agree on certain variables or states despite disturbances and delays [8].
- *Learning and Adaptation*: Agents can adapt their behaviors based on experiences. Reinforcement learning (RL) is often employed in MAS to allow agents to learn optimal policies through interactions with their environment [2].

Artificial neural networks (ANN) are computational models inspired by the human brain, consisting of interconnected nodes (neurons) that process information using a connectionist approach. The theoretical aspects of ANN include:

- *Neural Network Architecture:* The architecture of an ANN includes input, hidden, and output layers. Each neuron in a layer is connected to neurons in the subsequent layer through weighted edges. Common architectures include feedforward neural networks (FNN), recurrent neural networks (RNN), and convolutional neural networks (CNN) [5]
- *Activation Functions*: Neurons use activation functions to introduce non-linearity into the network, allowing it to learn complex patterns. Common activation functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU) [7].
- *Training Algorithms*: Training an ANN involves adjusting the weights of the connections based on the error of the output. Backpropagation, coupled with gradient descent, is a widely used training algorithm that minimizes the error by propagating it backward through the network [11].
- *Loss Functions*: The loss function measures the difference between the predicted output and the actual output. Common loss functions include mean squared error (MSE) for regression tasks and cross-entropy loss for classification tasks [6].
- *Generalization and Overfitting*: Generalization refers to the ability of an ANN to perform well on unseen data. Overfitting occurs when the network learns the training data too well, including noise and outliers, leading to poor performance on new data. Techniques like regularization, dropout, and early stopping are used to prevent overfitting [15].

Reinforcement learning (RL) is a type of machine learning where agents learn to make decisions by taking actions in an environment to maximize cumulative rewards. The theoretical aspects of RL include:

- *Agent and Environment*: In RL, an agent interacts with an environment, making decisions and receiving feedback in the form of rewards. The goal is to learn a policy that maximizes the cumulative reward over time [16].
- *Bellman Equation*: The Bellman equation provides a recursive decomposition of the value function, essential for dynamic programming and RL algorithms. It helps in updating value estimates based on rewards and future value estimates [1]
- *Learning Algorithms*: Common RL algorithms include Q-learning and policy gradient methods. Q-learning is an off-policy algorithm that learns the value of actions without following the current policy, while policy gradient methods optimize the policy directly using gradient ascent [14].

Integrating MAS with ANN combines the strengths of both fields to address complex problems:

- *Adaptive and Intelligent Agents*: ANN can endow agents with learning capabilities, allowing them to adapt and make decisions based on patterns in data. This is particularly useful in dynamic environments where agents need to learn and react to changes [16].
- *Scalability and Efficiency*: ANN helps in approximating optimal policies and value functions in high-dimensional spaces, making MAS more scalable and efficient. For example, neural network-based approaches can solve high-dimensional pathfinding problems more efficiently than traditional methods [9].
- *Improved Coordination*: ANN can be used to develop sophisticated models for predicting other agents' behaviors, leading to better coordination and cooperation strategies within MAS [19].
- *Real-Time Decision Making*: By leveraging the fast computation capabilities of ANN, MAS can achieve real-time decision making, which is crucial in applications like autonomous driving, robotics, and financial trading [10].

The theoretical aspects of MAS, ANN, and RL provide a strong foundation for understanding their integration. The synergy between these technologies can lead to robust,

scalable, and intelligent systems capable of addressing a wide range of complex real-world problems.

1) Recent Advances in Integrating Neural Networks with Multi-Agent Systems

In recent years, several innovative approaches utilizing neural networks for solving high-dimensional optimization problems, particularly in multi-agent systems (MAS), have emerged. These studies represent significant advancements in the efficiency, stability, and scalability of these systems. The first study introduces a method that combines Hamilton-Jacobi-Bellman (HJB) and Pontryagin Maximum Principle (PMP) approaches to parameterize the value function using a neural network (NN). This approach allows for obtaining approximately optimal controls in real-time without solving an optimization problem during deployment. Offline training of the neural network ensures optimal control over a large portion of the state space, effectively scaling to high dimensions and mitigating the curse of dimensionality. The method's effectiveness is demonstrated through multi-agent collision-avoidance problems in dimensions up to 150, with empirical results showing linear scaling of parameters with the control problem's dimension [9].

The second paper by Zhang and Wu addresses the consensus problem for uncertain nonlinear MAS under disturbances. It employs neural networks and backstepping to create adaptive controllers, ensuring that all system signals are globally bounded and consensus errors converge to a predefined accuracy despite disturbances. The method's innovation lies in its global precision consensus control scheme, validated through simulations in scenarios such as mobile robots and sensor networks [19].

The authors Pan, Ji, Lam and Cao introduce a method for predefined-time adaptive neural tracking control in nonlinear MAS. Using a new lemma within the backstepping framework, this approach achieves predefined-time stability, allowing users to specify the exact convergence time. Neural networks and finite-time differentiators ensure stable system signals, with followers tracking the desired trajectory within the predefined time. The method simplifies parameter tuning and demonstrates its effectiveness through extensive simulations [10].

In the fourth paper, Wu, Liu, Li, Zhang, and Hu address achieving consensus in multi-agent systems with unknown, time-varying input dead-zones. The authors propose an adaptive neural network control method combined with an event-triggered mechanism, ensuring followers can track a virtual leader within a predefined time while avoiding Zeno behavior. This approach mitigates communication congestion and improves convergence rates and system stability [18].

In the last paper authors Zhang, Niu, Zhang and Wang present a method for achieving consensus tracking in high-order nonlinear MAS. This approach models followers in a non-strict-feedback structure and uses unknown functions for control gains. RBF neural networks compensate for unknown nonlinearities, and finite-time differentiators ensure stability and zero tracking error within a predefined time. The control scheme's effectiveness is validated through theoretical analysis and simulations, maintaining system stability and achieving precise tracking. Together, these studies highlight significant advancements in using neural networks for real-time control and consensus tracking in multi-agent systems, demonstrating robustness, efficiency, and scalability in various complex scenarios [20].

III. METHODOLOGY

This study focuses on examining artificial neural network (ANN) and multi-agent system (MAS) unification in a virtual environment. The proposed framework is based on reinforcement learning, where the agent learns to optimize the decision process using the feedback provided by the environment. To validate this approach a simple environment was developed for testing as shown in Fig. 1.

A 5x5 grid was created that contained obstacles and a designated target. The goal of the agent is to navigate and find the target point via the shortest available pathway. To do this the agent selects an action at each step based on the state and receives a reward or penalty depending on the proximity to the goal. The objective is to maximize the cumulative reward, as described in the reinforcement learning framework by Sutton and Barto [16]. The environment was inspired by the authors Farhan, Gohre & Junprung, but it was simplified to meet our needs [4].

The experimental configuration and training involved one agent per simulation step with independent goal pursuit behavior and autonomous learning capability. Artificial neural networks were employed to approximate the decision policy based on experience of all agents, and the multi-agent system concept was employed as a structural framework for future scalability.

Each training session consisted of three stages: environment initialization, running the training loop, and performance testing. No epsilon-greedy strategy was employed in the training loop. Table I presents the entire algorithmic process with pseudocode for training the agent on the 5×5 grid. Training was executed in Python on the Google Colab environment, cloud computational facilities that allow for collaborative programming.

TABLE I. PSEUDOCODE FOR TRAINING AND ANALYSIS
(*Source:* Own Research)

```
1. SETUP
   • ENV_SIZE ← 5; MAX_EPISODES ← 500; BLOCK_SIZE ← 
50
2. ENVIRONMENT
   CLASS Env
     METHOD reset():
       grid ← zero_matrix(ENV_SIZE)
       place start, goal, obstacles
       visits ← zero_matrix(ENV_SIZE)
       RETURN start_pos
     METHOD step(action):
       move agent if no obstacle
       visits[current_pos]++
       reward, done ← check_goal(current_pos)
       RETURN (current_pos, reward, done)
3. MODEL
   FUNCTION create_model():
     build NN with 2 hidden layers (ReLU) and 4-unit 
output (linear)
```

```
     compile with Adam & MSE
     RETURN model
4. TRAINING
   FUNCTION train(model, env):
     FOR episode IN 1…MAX_EPISODES:
       state ← env.reset(); steps ← 0; done ← FALSE
       WHILE steps < max_steps AND NOT done:
         action ← model.predict(state)
         (state, r, done) ← env.step(action)
         record transition
         steps++
       record steps, done, env.visits
     RETURN step_list, success_list, total_visits
5. ANALYSIS
   FUNCTION analyze(steps, success_list):
     Efficiency: fit linear regression on (episode,
steps) → slope β
     Convergence: compute std dev of steps in blocks
of BLOCK_SIZE
     Success: compute cumulative success rate
     RETURN β, std_list, cumulative_success
6. PLOTTING (for slides)
   plot_scatter_with_line(episodes, steps, β)
   plot_line(std_list)
   plot_line(cumulative_success)
7. MAIN
   env ← Env()
   model ← create_model()
   (steps, success, visits) ← train(model, env)
   (β, std_list, cum_succ) ← analyze(steps, success)
   generate_plots()
      print_summary(β, last(cum_succ))
```

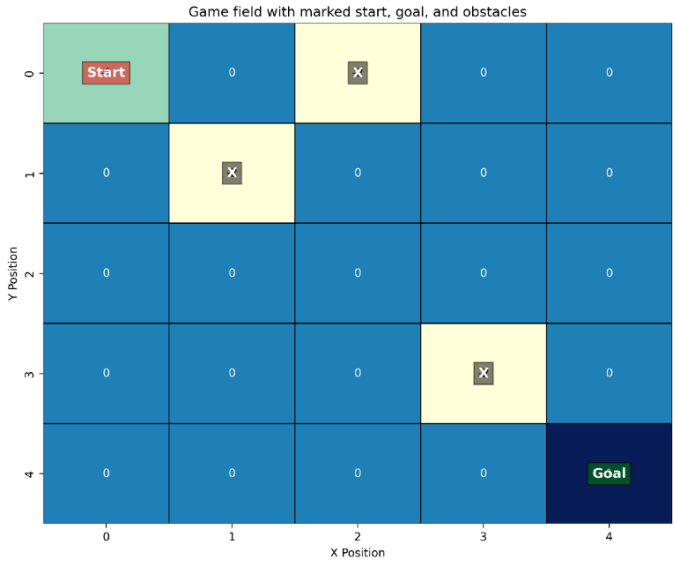

Fig. 1. Game field with marked start, goal, and obstacles. (Source: own research)

IV. RESULTS

Table II shows the simulation results in absolute numbers. The parameter epsilon was not explicitly implemented in the simulation, which means that the agent chose actions deterministically based on the prediction of the neural network (ε = 0). The decay of randomness in the agent's behavior therefore does not come from the epsilon-greedy policy, but from the initial random initialization of weights and the stochastic nature of training. The resulting performance variability thus reflects internal random influences of learning, not controlled exploration of the environment.

TABLE II. MAIN RESULTS FROM SIMULATIONS
(*Source*: Own Research)

Monitoring/Episodes	Number of laps in an episode	Number of successful viewings
500	400	493
Number of unsuccessful follow-ups	Numbers of unsuccessful Rounds	
7	16; 107; 108; 156; 171; 413; 492	

The simulation produced some basic descriptive statistical outputs as shown in Table III. The result of the statistical analysis shows several metrics that provide insight into the agent's performance in individual training rounds. The basic simulation setup was 500 rounds of repetition, with each round containing a maximum of 400 steps. These values were also obtained using the Python language with standard outputs. The average number of steps to reach the goal is 107.460 compared to the median, which is lower at 77 steps, which is much less than the found average. This suggests that the distribution of steps is likely skewed and that during testing, the agent often needed more steps to find the goal. The standard deviation indicates high variability in the number of steps between individual rounds, which is due to the randomness of the training process. The agent's success rate is high, specifically over 98%, with 493 completed rounds in absolute numbers. The interval range is 105 according to the analysis, indicating a high data dispersion between the first and third quartiles. The low kurtosis value suggests that the distribution of the number of steps is flatter than a normal distribution. A skewness value of 1.37 indicates a "rightward" tilt, meaning there are more episodes with a higher number of steps than those with fewer steps. This representation of values indicates that while the agent has a high success rate, it also shows a significant degree of variability in performance, as suggested by the variance and IQR.

TABLE III: BASIC STATISTICAL OUTPUTS
(*Source*: Own Research)

Average steps	Median steps	Standard deviation	Success Rate (%)
107.460	77.000	86.107	98.600
Variance	Kurtosis	Skewness	IQR
7414.437	1.531	1.366	105.000
95% Confidence Interval Lower		95% Confidence Interval Upper	
99.894		105.026	

This raises three issues for verification:

- *Question 1*: Does the average number of steps needed to reach the goal decrease with each additional training episode?
- *Question 2*: Does the variability in the number of steps between individual episodes decrease after a certain number of episodes?
- *Question 3*: Does the agent's success rate (i.e., the number of completed episodes) increase with each additional training episode?

A. Question 1

The principle of agent learning is based on interaction with the environment and feedback through reinforcement learning. This process involves exploration and exploitation, which lead to efficient training and improving the agent's abilities. Exploration is the process where the agent explores new state possibilities in the field to gather information about the environment, aiming to discover new paths and possible strategies, which may lead to short-term lower rewards for achieving the goal. However, in the long term, it is an essential part of learning, especially for optimizing the agent's policy. A simple example is when an agent finds a short route (according to the agent's judgment) from the start to the goal and keeps using it, not discovering potentially more efficient routes. Learning the fastest strategy to achieve the goal with a high reward is called exploitation. In such a case, the agent decides based on the currently best-known strategy, which is suitable when trying to maximize profit in the short term [12]. The first hypothetical question explores whether there is a trend of improving the agent's performance during the learning process. Specifically, it examines whether the agent requires fewer steps with each subsequent episode. Effective training should lead to the agent gradually reducing the number of steps in subsequent episodes. A statistical method that can be used for this is linear regression, which describes the statistical relationship between the independent variable and the dependent variable:

$$Y=\beta_0+\beta_1 X \quad (1)$$

, where Y is the dependent variable, X is the independent variable, β_1 represents the slope of the regression line and β_0 presents the y-intercept. In our case, the independent variable is each episode, and the dependent variable is the number of steps.

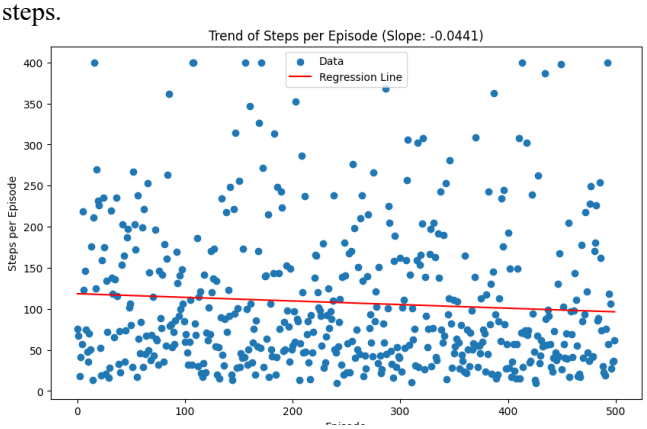

Fig. 2. Trend of Steps per Episode. (*Source*: own research)

In Fig. 2, the graph shows the trend of the number of steps over individual episodes. The blue dots in the graph represent the number of steps for each episode, and the red line represents the regression line, which illustrates the trend in the graph. The equation of the regression line is:

$$Y=-0.0441X+118.46 \quad (2)$$

The slope of the regression line is -0.0441, indicating that as the number of episodes increases, the average number of steps per episode decreases, albeit very slightly in this simulation. Specifically, in this case, the number of steps decreases by 0.0441 steps per episode. The negative slope of the line supports the hypothesis of training efficiency, as the agent gradually learns and improves its performance (number of steps) to reach the goal. It should be noted that the correlation coefficient is very weak (R^2 = 0.0055), but this trend is important and significant for confirming the hypothesis that the agent can improve its strategy to achieve the goal with increasing episodes. Despite the low dependence, indicating slow learning, this hypothesis can be accepted.

B. Question 2

Another question investigates whether the agent stabilizes its performance over the course of episodes, specifically examining the degree of variability. High variability indicates rather unpredictable results, which may suggest that strategies are inconsistent and the agent faces challenging decision-making environments. Conversely, a reduction in variability could indicate that the agent is finding more efficient paths. If variability decreases, it can be inferred that the agent is effectively utilizing its experiences and more frequently choosing optimal routes, which can enhance the predictive capabilities of the training. Using statistical methods, it is possible to identify phases of training that may need optimization.

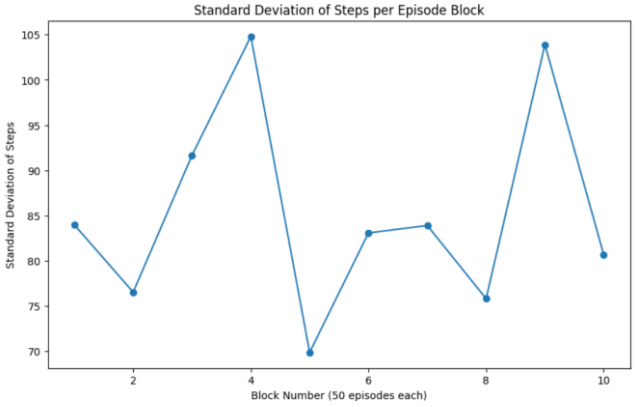

Fig. 3. Standard Deviation of Steps per Episode (*Source*: Own research)

A simple visual inspection reveals that the variability in the agent's number of steps fluctuates significantly. The question was examined by dividing the data into 10 blocks of 50 episodes each, with the standard deviation calculated for each block. Fig. 3 shows that the variability in the number of steps between episodes fluctuates throughout the training. In some blocks (e.g., block 4 and block 9), we see an increase in variability, while in other blocks (e.g., block 2 and block 6), the variability decreases. Fig. 3 suggests that the variability in the number of steps between episodes fluctuates during training. Although there are periods when variability decreases, the overall trend is not clearly declining. This fluctuation may be caused by different phases of learning and exploring new

strategies. Further training and statistical analysis will be conducted to confirm or refute the hypothesis of convergence.

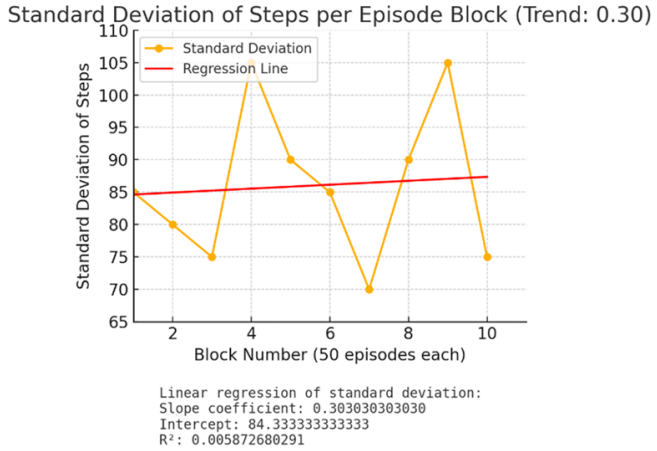

Fig. 4. Standard Deviation of Steps per Episode with regression line (*Source*: Own research)

Fig. 4 displays the fitting of the previous figure using a regression line. The slope coefficient is 0.303, indicating a slight trend in variability in the number of steps between episodes. The coefficient β0 indicates the expected standard deviation for the first block. The correlation coefficient is very low, at R^2=0.00587, explaining very little variability in the data. These data are likely influenced by additional factors. The previous statistical test will be supported by a second method, specifically the ANOVA method. The reason for this is to verify that there is no statistically significant difference in the degree of variability between the individual groups.

TABLE IV. ANOVA TABLE (Source: Own Research)

ANOVA table				
	Sum_sq	df	F	PR(>F)
C (Block)	899.0	9.0	0.809181	0.608941
Residual	11110.0	90.0	NaN	NaN

The results of the ANOVA test showed that the p-value is 0.608941, which is significantly higher than the commonly used threshold of 0.05 to determine statistical significance. This means that there is no statistically significant difference between the standard deviations in different blocks of episodes. The F-value of 0.809181 further confirms that the differences between groups are minimal compared to the variability within groups. Based on these results, we cannot reject the null hypothesis, which states that the variability between blocks is the same. This conclusion suggests that the agent does not demonstrate a systematic reduction in performance variability over time, thus refuting the convergence hypothesis. Therefore, the agent does not exhibit more consistent performance in later phases of training compared to its initial performance.

C. Question 3

The final hypothetical question aims to demonstrate the agent's success with each additional training process, where it learns based on experiences, i.e., feedback. This is to prove whether, within reinforcement learning, the agent genuinely improves in strategy selection and decision-making in each environment. If the learning is successful, it increases the agent's performance, making the training algorithm effective. The cumulative success rate graph provides a detailed view of the agent's performance during training episodes. Data analysed, obtained, and visualized from the simulation includes information on the number of steps and success rates in individual episodes, with unsuccessful episodes being 16, 107, 107, 156, 171, 413, and 492. In the first 200 episodes, it is apparent that failures, or unfinished episodes, are "relatively" frequent. Looking at Fig. 5, it is evident that choosing 400 steps per episode was relatively high, and further exploration would benefit from selecting lower step parameters. Fig. 6 further shows that the success rate is initially lower, reflecting the fact that the agent is learning the path and, as episodes progress, the success rate slows and converges to approximately 98%. This development can be supported by reinforcement learning (RL) theory, specifically the concepts of exploration and exploitation. At the beginning of training, the agent explores its environment (exploration) and learns which actions lead to the best outcomes. This often results in a lower success rate, as the agent tries new strategies and learns from mistakes. As training continues, the agent begins to utilize learned strategies more (exploitation), leading to a higher success rate. Dips in the graph can be interpreted as moments when the agent again tries new strategies or faces more complex tasks. The overall upward trend, however, confirms that the agent is learning and improving its ability to complete tasks, consistent with RL theory, where the agent gradually becomes more efficient through repeated interactions with the environment and feedback on its actions

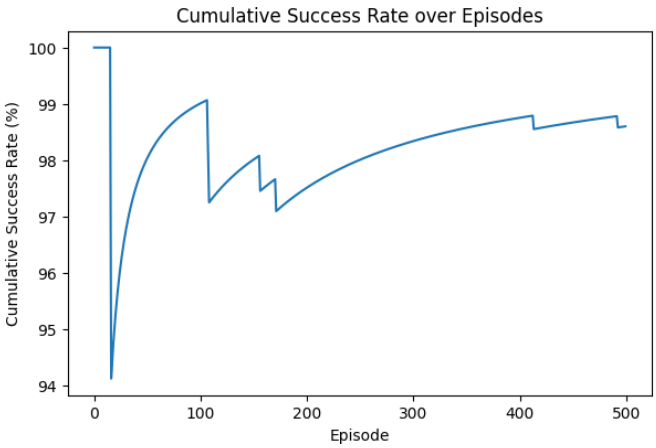

Fig. 5. Cumulative success rate over Episodes (Source: Own research)

V. DISCUSSION

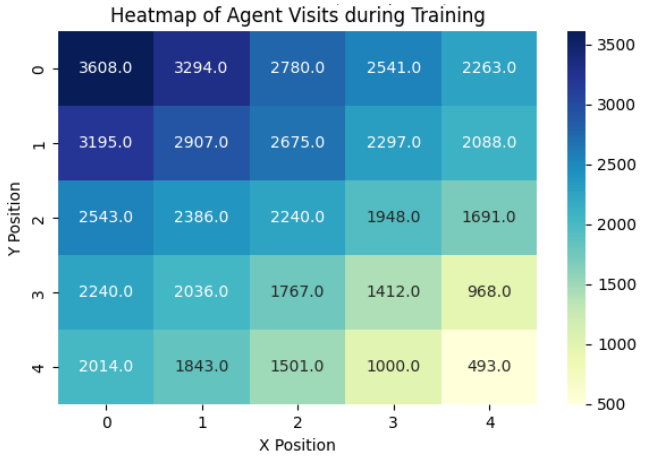

Fig. 6. Heatmap of agent visits during Training. (*Source*: Own research)

This heatmap shows the frequency of agent visits during training at positions in a 2D space. The training included 500 episodes with a maximum of 400 steps per episode. The X-axis represents the agent's positions in the horizontal direction (X position), while the Y-axis represents the positions in the vertical direction (Y position). The intensity of the colors corresponds to the number of visits at a given position, with darker colors indicating a higher number of visits. At description of the heatmap we can see:

- *Most visited positions*: The highest number of visits were recorded at positions (0,0) and (0,1), where the agent was 3608 and 3195 times, respectively. The position (0,0) is the starting position of the agent in the environment.
- *Moderate visit frequency*: Positions from (1,1) to (2,2) have visit frequencies ranging from 1500 to 3000 visits.
- *Less visited positions*: Positions towards the bottom right corner (4,4) have a lower number of visits, specifically 493 visits. The position (4,4) was the target cell and terminated an episode once visited by the agent. This frequency value corresponds with the number of total successful episodes.
- *Overall trend*: The agent spent more time in the upper left part of the space and less time towards the bottom right part.

This heatmap provides a visual representation of how the agent moved during training and where it spent the most time. This can be useful for analyzing its behavior and optimizing the training process.

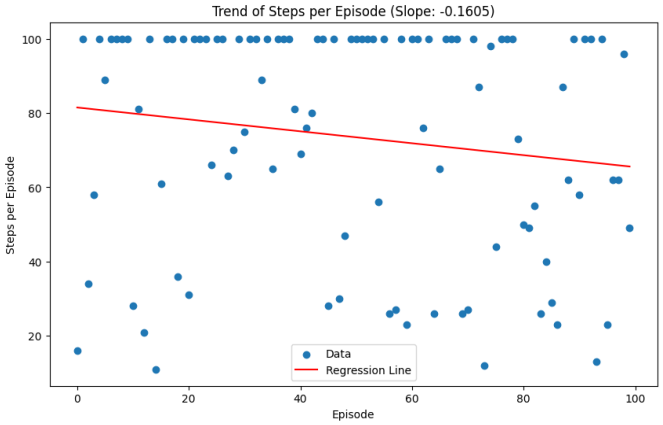

Fig. 7. Heatmap of agent visits during Training. (*Source:* Own research)

To understand the outputs, a shorter version of the simulation was conducted, limited to 100 rounds with 100 steps each as shown in Fig. 7. Following the hypotheses and their interpretation in the previous chapter, the differences in the results can be discussed:

- *Simulation Outcomes*: Out of 100 rounds, 44 rounds were not completed, representing 44% of the total rounds.
- *Regression Line*: The graph shows a regression line with a slope of -0.1605, indicating a slight downward trend in the number of steps per episode.

Discussion on Differences in Results:

- *Completion Rate*: With 44 rounds not completed, the completion rate is 56%. This suggests that the agent successfully completed just over half of the rounds in the shorter simulation.
- *Slope of Regression Line*: The slope of the regression line is -0.1605, which indicates a slight decrease in the number of steps needed to complete an episode over time. This trend might suggest that the agent is gradually improving and is able to reach the goal faster, although the decrease is very small.
- *Data Dispersion*: The data shows a large dispersion in the number of steps per episode, which may indicate inconsistent performance of the agent. Some episodes were completed with a very low number of steps, while others required close to the maximum of 100 steps.
- *Comparison to Longer Simulation*: Comparing this shorter simulation to the original longer one (500 episodes with a maximum of 400 steps) might reveal differences in the agent's learning and adaptation over time. The shorter simulation provides insight into the agent's initial performance and potential early learning difficulties.

Overall, this shorter simulation provides valuable insights into the agent's early-stage behavior and performance, which can inform further modifications and improvements in the training process.

Fig. 8 illustrates the cumulative success rate over episodes. The curve shows strong fluctuations at the beginning of training, which can be attributed to the exploratory behaviour of the agent. As the simulation progresses, the success rate becomes more stable and slightly increases toward the final episodes. This indicates that the agent gradually adapts its behaviour and improves its decision-making strategy over time.

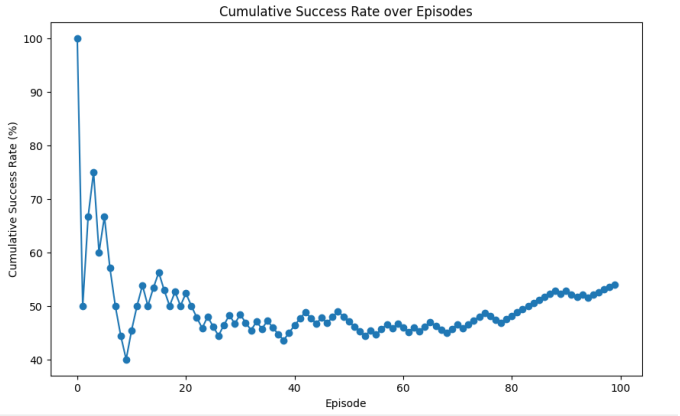

Fig. 8. Cumulative success rate over episodes. (*Source*: Own research)

VI. CONCLUSION

The integration of Artificial Neural Networks (ANN) and Multi-Agent Systems (MAS) holds significant potential for developing advanced intelligent systems capable of solving complex problems in various real-world scenarios. This combination leverages the strengths of both technologies, offering enhanced learning, adaptation, scalability, and efficiency.

The theoretical foundations of MAS provide a robust framework for understanding how autonomous agents can effectively communicate, coordinate, and cooperate within a shared environment. ANN contributes powerful tools for pattern recognition, decision-making, and learning from data, enabling agents to improve their performance over time. Reinforcement Learning (RL) further enhances these capabilities by allowing agents to learn optimal strategies through interactions with their environment.

Recent advancements in the field, as demonstrated by various studies, highlight the practical applications and benefits of integrating ANN with MAS. These include improved real-time decision-making, better coordination among agents, and the ability to solve high-dimensional optimization problems more efficiently. The empirical results from simulations and experiments show that such integrated systems can achieve high success rates, adaptability, and robustness. Despite these advancements, there are challenges that need to be addressed, such as optimizing training processes and ensuring consistent performance across different scenarios. Future research should focus on refining these integrations, exploring new algorithms, or expanding the range of applications.

In conclusion, the synergy between ANN and MAS represents a promising direction for the development of intelligent systems. By continuing to explore and enhance this integration, researchers and practitioners can unlock new possibilities and drive innovation across various industries.

ACKNOWLEDGMENT

The research has been partially supported by the Faculty of Informatics and Management UHK specific research project, addressing modern research topics with increased student involvement.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest regarding the publication of this paper.

REFERENCES

[1] Bellman, R., & Dreyfus, S. (2010). *Dynamic programming* (Princeton Landmarks in Mathematics ed., with a new introduction). Princeton University Press.

[2] Buşoniu, L., Babuška, R., & De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *38*(2), 156–172. https://doi.org/10.1109/TSMCC.2007.913919.

[3] Chopra, A. K., Artikis, A., Bentahar, J., Colombetti, M., Dignum, F., Fornara, N., Jones, A. J. I., Singh, M. P., & Yolum, P. (2013). Research directions in agent communication. *ACM Transactions on Intelligent Systems and Technology*, *4*(2), 1–23. https://doi.org/10.1145/2438653.2438655.

[4] Farhan, M., Gohre, B., & Junprung, E. (2020). Reinforcement learning in analogic simulation models: A guiding example using Pathmind. In *Proceedings of the 2020 Winter Simulation Conference (WSC)*. https://doi.org/10.1109/WSC48552.2020.9383916.

[5] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

[6] Murphy, K. P. (2013). *Machine learning: A probabilistic perspective*. MIT Press.

[7] Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)* (pp. 807–814).

[8] Olfati-Saber, R., Fax, J. A., & Murray, R. M. (2007). Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, *95*(1), 215–233. https://doi.org/10.1109/JPROC.2006.887293.

[9] Onken, D., Nurbekyan, L., Li, X., Fung, S. W., Osher, S., & Ruthotto, L. (2023). A neural network approach for high-dimensional optimal control applied to multiagent path finding. *IEEE Transactions on Control Systems Technology*, *31*(1), 235–251. https://doi.org/10.1109/TCST.2022.3172872.

[10] Pan, Y., Ji, W., Lam, H.-K., & Cao, L. (2024). An improved predefined-time adaptive neural control approach for nonlinear multiagent systems. *IEEE Transactions on Automation Science and Engineering*, 1–10. https://doi.org/10.1109/TASE.2023.3324397.

[11] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0.

[12] Russell, S. J., Norvig, P., & Chang, M.-W. (2022). *Artificial intelligence: A modern approach* (4th ed.). Pearson.

[13] Sandholm, T., & Lesser, V. R. (1995). Issues in automated negotiation and electronic commerce: Extending the contract

net framework. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS)* (pp. 12–14).

[14] Schulman, J., Chen, X., & Abbeel, P. (2017). Equivalence between policy gradients and soft Q-learning (Version 4). *arXiv*. https://doi.org/10.48550/arXiv.1704.06440

[15] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

[16] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

[17] Wooldridge, M. J. (2009). *An introduction to multiagent systems* (2nd ed.). John Wiley & Sons.

[18] Wu, W., Liu, J., Li, F., Zhang, Y., & Hu, Z. (2023). Prescribed settling time adaptive neural network consensus control of multiagent systems with unknown time-varying input dead-zone. *Mathematics*, *11*(4), 988. https://doi.org/10.3390/math11040988,

[19] Zhang, C., & Wu, J. (2021). Global adaptive consensus control for multiagent systems with predefined accuracy. *Complexity*, 2021, 1–19. https://doi.org/10.1155/2021/3396482

[20] Zhang, Y., Niu, B., Zhang, J.-M., & Wang, X.-M. (2021). Predefined-time adaptive neural-network-based consensus tracking control for nonlinear multiagent systems with zero tracking error. *IEEE Access*, *9*, 132205–132214. https://doi.org/10.1109/ACCESS.2021.3115118.