Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control
Journal homepage: http://kinetik.umm.ac.id
ISSN: 2503-2267
Vol. 11, No. 2, May, Pp. 217-226

An adaptive swarm clustering algorithm for game AI based on Reinforcement Learning Godot and Particle Swarm Optimization (RLGPSO)

Trisna Gelar*[1], Iwan Awaludin[1], Raditya Pasya[1], Raihan Fuad[1], Muhammad Rizqi Sholahuddin[1]
Department of Computer Engineering and Informatics, POLBAN, Bandung, Indonesia[1]

Article Info
Keywords:
Adaptive Swarm Clustering, Particle Swarm Optimization, Reinforcement Learning, Game AI, Dynamic Environments

Article history:
Received: August 30, 2025
Accepted: December 01, 2025
Published: May 01, 2026

Cite:
T. Gelar, I. Awaludin, R. Pasya, R. Fuad, and M. R. Solahudin, "An Adaptive Swarm Clustering Algorithm for Game AI Based on Reinforcement Learning Godot and Particle Swarm Optimization (RLGPSO)", *KINETIK*, vol. 11, no. 2, May. 2026.
https://doi.org/10.22219/kinetik.v11i2.2480

*Corresponding author.
Trisna Gelar
E-mail address:
trisna.gelar@polban.ac.id

Abstract
The management of extensive agent swarms presents significant challenges in dynamic, real-time environments, particularly within the context of game artificial intelligence, such as real-time strategy games. Traditional Particle Swarm Optimization (PSO) techniques demonstrate effectiveness in optimization tasks; however, they frequently exhibit suboptimal convergence and insufficient flexibility in complex and challenging scenarios. This study presents a hybrid methodology that combines Reinforcement Learning (RL) and Particle Swarm Optimization (PSO) to develop an adaptive swarm clustering system. This method utilizes a Deep Deterministic Policy Gradient (DDPG) agent operating externally through an API to dynamically adjust Particle Swarm Optimization (PSO) parameters, thereby maintaining a separation between adaptive intelligence and the simulation engine. This allows the swarm to effectively navigate and group within a procedurally generated 2D simulation environment with physical obstacles, unlike previous studies that rely on static mathematical benchmarks. A quantitative analysis employing Mixed Linear Model Regression (MLMR) indicates that this hybrid method significantly outperforms traditional, manually tuned PSO in terms of convergence time and diversity value. The RLGPSO model showed an 11.46% decrease in convergence time on highly complex maps. This result was statistically significant, with a p-value of 0.002 from the MLMR analysis. This research presents a framework for the deployment of intelligent, self-organizing agent swarms, enhancing the realism and efficacy of contemporary game artificial intelligence.

1. Introduction

Real-time strategy games like *StarCraft* serve as critical platforms for artificial intelligence research [1] due to their demands for effective resource management, strategic swarm composition, and the concurrent control of multiple units in complex and dynamic settings [2]. Swarm Intelligence (SI) is a domain of artificial intelligence that draws inspiration from the collective behaviors of social insects and animal groups, providing a systematic method to address these challenges. SI algorithms enable large groups of simple, independent agents to perform complex tasks through decentralized governance and localized interactions. This paradigm, known as collective intelligence, improves group decision-making by facilitating decentralized information transmission without centralized control. This effect is observable in natural self-organizing systems, such as bee swarms selecting a new nesting site. Swarm-based approaches demonstrate stability and scalability, making them advantageous for complex optimization problems in areas such as robotics, electrical systems [3], and game simulation [4].

PSO is a prominent swarm intelligence technique recognized for its conceptual clarity and empirical efficacy across various optimization challenges [5]. A significant drawback of traditional PSO is its dependence on fixed, experimentally adjusted parameters, usually determined through trial and error. This lack of adaptability leads to two main types of failure: early convergence, where the swarm finds a local optimum before discovering the global solution, thereby reducing diversity [6]; and fixed parameter configurations that limit the swarm's capacity to adapt to dynamic obstacles or changing optimal conditions [7]. The lag in reaction time can result in units becoming trapped or experiencing difficulties in traversing terrain, eventually leading to decreased performance.

Recent research has concentrated on adaptive and hybrid swarm methods to address these deficiencies. These methodologies integrate complementary strategies to address the constraints of singular methods, a concept that has demonstrated efficacy in optimization and machine learning domains [8]. Typical improvements encompass non-reinforcement learning processes, such as chaotic mappings that maintain population diversity, and adaptive inertia-weight strategies that dynamically adjust global and local search behaviors based on the search state [9]. While these adaptive methods are effective in improving performance in simpler scenarios, they tend to be rule-based or heuristic, depending on pre-defined, deterministic functions to modify parameters. These characteristic limits their ability to derive

the non-linear, context-specific policies necessary to optimally manage swarm behavior in the highly dynamic, complex, and unpredictable environments characteristic of modern game AI, which necessitates a more generalized, experience-driven approach.

Deep Reinforcement Learning (DRL) has emerged as method for addressing high-dimensional, temporally extended challenges characteristic of game AI among adaptive techniques. By combining deep neural networks with reinforcement learning, DRL agents can learn complex control rules directly from high-dimensional inputs, such as raw pixels, enabling them to navigate the vast decision spaces found in modern video games [10], [11]. In contrast to rule-based adaptive methods, deep reinforcement learning (DRL) can generate non-linear policies derived from experience, which makes it particularly effective for the online parameter regulation of PSO. Prior studies have employed both Deep Q-Networks (DQN) and Deep Deterministic Policy Gradient (DDPG) methods to adjust PSO parameters at runtime, supporting the use of DRL for building genuinely adaptive swarms [12]. However, the core PSO control variables (ω, c_1, c_2) are continuous in nature. Consequently, the DDPG method has employed, as it is tailored for complex control tasks in continuous action spaces, providing an effective approach for the real-time optimization of all three PSO parameters.

Researchers have combined reinforcement learning with PSO using various unique approaches. One paradigm uses RL as a strategy selector, where an agent chooses which particle's experience to follow at each step to manage exploration and exploitation trade-offs at a micro level [13], [14]. Another employs a hierarchical architecture, with a high-level RL controller providing strategic guidance to a lower-level PSO optimizer responsible for tactical refinement [15]. The approach adopted in this paper follows a third, more direct paradigm: employing RL as a dynamic parameter controller, whereby a DDPG agent observes the swarm's global state and outputs PSO parameter adjustments in real time. This paper proposes an adaptive hybrid algorithm, RLGPSO, that leverages the DDPG agent to tune PSO parameters dynamically with the objective of optimizing swarm clustering in dynamic environments.

This paper adopts a third, more direct paradigm by employing RL as a dynamic parameter controller. DDPG agent observes the swarm's global state and provides real-time adjustments to PSO parameters. This study introduces an adaptive hybrid algorithm, RLGPSO, which utilizes the DDPG agent to dynamically adjust PSO parameters to optimize swarm clustering in dynamic environments. This work makes three contributions to the fields of computational intelligence and game AI: (1) an algorithmic framework for integrating Deep Deterministic Policy Gradient (DDPG) with Particle Swarm Optimization (PSO) for online continuous parameter tuning; (2) the creation of a comprehensive, physics-based simulation environment within the Godot Engine [16] to enable real-time, dynamic experiments, surpassing traditional static mathematical benchmarks; (3) a quantitative assessment employing Mixed Linear Model Regression (MLMR), indicating a decrease in convergence time and confirming the statistical robustness of the improvements.

2. Research Method

The study technique comprises three independent phases: simulation design and implementation, model training, and performance evaluation. This systematic approach enables the development, training, and comprehensive assessment of the proposed hybrid RLGPSO algorithm in a dynamic, real-time environment. The overall process is depicted in Figure 1.

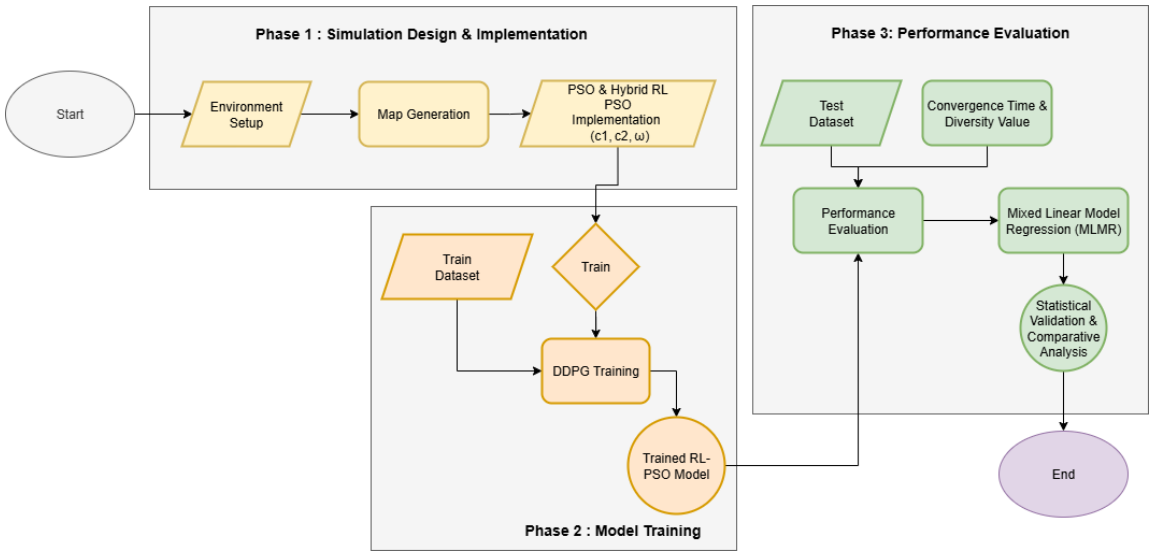


Figure 1. The Proposed Hybrid RLGPSO Algorithm within a Dynamic, Real-time Environment

2.1 Simulation Design & Implementation

This phase includes setting up the fundamental environment and executing the algorithms. All experiments are performed within a customized simulation in the Godot Engine, utilized to establish a method for swarm clustering. To create a robust and varied environment, a custom tool was developed to procedurally generate maps within a 1000×1000 pixel bounded area. *Table 1* lists the three complexity levels of maps—low, medium, and high—based on the number and type of two-dimensional obstacle shapes placed within the canvas [17], [18].

Table 1. Map Complexity Metrics

Complexity	Number of obstecles	Shape types
Low	10-30	Circle, ellipse
Medium	10-30	Regular polygon (eg. Triangle, square)
High	30-50	Combination of circle, ellips, regular polygon, irregular polygon

To avoid the inefficiency of manual map creation, a tool was developed in Godot using GDScript. The tool produces obstacle configurations utilizing adjustable trigonometric patterns (sine, cosine, and tangent) and geometric principles, facilitating rapid, consistent, and scalable map generation. The horizontal and vertical positions of barriers are determined using Parameterized Equations 1 and 2.

$$x_i = i \cdot x_{step} + x_{offset} \tag{1}$$

x_i denotes the horizontal position of object i, where i is the index of the object, beginning from 0. The term x_{step} represents the horizontal spacing between adjacent objects, while x_{offset} indicates the offset applied to the horizontal position to adjust or shift the object's location along the x-axis.

$$y_i = A \cdot \sin\left(f \cdot i \cdot x_{step}\right) + i \cdot x_{step} + y_{offset} \tag{2}$$

Where i refers to the index of the object. The parameter x_{step} defines the horizontal spacing between consecutive objects, whereas y_{offset} specifies the vertical offset applied to each object's position. The term A represents the amplitude of the wave function, and f denotes its frequency, both of which influence the object's vertical displacement in accordance with the wave's behavior.

The research design incorporates both independent and dependent variables to evaluate the algorithm's effectiveness. Independent variables: the factors manipulated in the experiments are the optimization method (conventional PSO vs. RL-PSO), the number of particles in the swarm (agents), and the complexity of the map scenario. Dependent variables: the performance is measured using convergence time (the time for the swarm to reach the target) and diversity value (a measure of swarm spread), as noted in Equation 3.

$$D(t) = \frac{1}{n_s} \sum_{i=1}^{n_s} \sqrt{\sum_{j=1}^{n_d} \left(x_{ij}(t) - \bar{x}_j(t)\right)^2} \tag{3}$$

Where $D(t)$ denotes the diversity value at iteration t, $x_{ij}(t)$ is the j^{th} position of the i^{th}particle in the iteration t, $x_{(j)}(t)$ is the average j^{th} of all the particles in the iteration t.

The application of the PSO algorithm within the Godot Engine is inspired by traditional, platform-independent formulation [19]. The method requires significant modifications to achieve compatibility with the engine's physics-based node system. Each particle in the swarm is represented as a CharacterBody2D node, serving as a physics-based object that manages its own movement and collision detection. The Godot implementation uses the physic engine through the *move_and_slide* function. This function prevents collisions and ensures smooth, physically reasonable particle gliding past obstacle surfaces. The logic for this process distinguishes between different obstacle shapes: for linear obstacles, the surface vector and its normal are used to adjust the particle's velocity via vector projection, while for circular obstacles, the surface vector is derived from the contact point and the circle's center to create a normal vector for sliding computation.

Moreover, this transition to a physics-based simulation required a recalibration of the traditional PSO parameters. Initial assessments employing typical parameter values (e.g., c_1=1.711897, c_2=1.711897, and ω=0.711897) revealed

inadequate performance in complex obstacle situations, characterized by slow movement and a restricted search radius. The parameters for the baseline traditional PSO were empirically adjusted to set a cognitive coefficient (c_1) of 1.0, a social coefficient (c_2) of 1.0, and an inertia weight (ω) of 1.1, resulting in a significant enhancement in performance within limited search spaces.

2.2 Model Training

This phase focuses on training the DDPG agent to develop a suitable policy for swarm control policy through its engagement with the environment [20]. The model is trained on a specialized dataset comprising 16 maps, categorized into 6 low-complexity, 6 medium-complexity, and 4 high-complexity maps, to improve generalization. The training procedure comprises a total of 1,024 instances. Training begins sequentially, utilizing each map for 50 consecutive iterations, progressing from low to high complexity. Subsequent episodes are carried out with randomly generated maps and varying swarm sizes to enhance the model's robustness and adaptability to novel scenarios.

This **learning** is defined by three key components. First, state and action space: the agent's state representation is a 4-dimensional normalized vector that provides a real-time snapshot of the swarm's behavior. The variables selected to form this state vector are diversity value (reflecting particle spread), iterations without improvement (indicating stagnation), mean velocity (representing average swarm speed), and finish ratio (the percentage of particles reaching the target). The action space consists of three continuous variables, which are the outputs of the DDPG's actor network. These correspond directly to the three control parameters of the PSO algorithm: ω, c_1 *and* c_2. The output values are constrained to specific ranges (ω in [1.0, 3.0] and c_1, c_2 in [0.0, 3.0]) to ensure the swarm's dynamics remain stable and effective.

Table 2. Reward Structure for RL-PSO Agent

Condition	Reward / Penalty
Global best fitness improves	+10
Global best fitness does not improve	-5
Particle improves its personal best	+3
Particle does not improve its personal best	-2
Particle is stuck (no progress over multiple iterations)	-5
Particle reaches the target (goal area)	+2

Second, **reward mechanism**: the DDPG agent learns via a hierarchical reward mechanism that provides feedback on both global and individual swarm performance. Table 2 lists the reward structure, which is designed to incentivize behaviors that lead to efficient swarm clustering and discourage stagnation. The proposed reward mechanism incorporates two hierarchical levels of assessment: global performance and individual particle behavior. Conditions related to global optimization, including improvements in the global best (*gbest*) or overall best fitness value, are assessed once per iteration, as they represent the collective dynamics of the swarm. In contrast, reward signals associated with individual particles such as enhancements in personal best positions (*pbest*), identification of stagnation, or successful target attainment are calculated independently for each particle in every iteration.

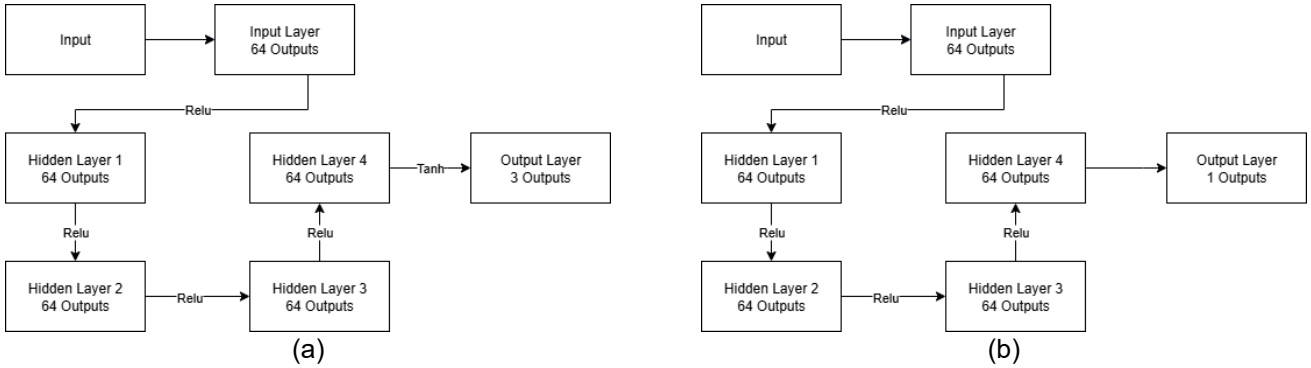


Figure 2. Adapted Actor and Critic Network Architectures[21]

Third, **network architecture**: Figure 2 illustrates the DDPG agent's fully connected feedforward neural actor and critic networks. Each network has four hidden layers with 64 neurons, using the ReLU activation function for non-linearity. From a four-dimensional state vector, the actor network outputs three continuous PSO parameter values. The final output layer utilizes a *tanh* activation function, which is subsequently scaled to ensure that the outputs remain within the defined parameter ranges. The critic network receives a concatenated vector that includes both the state and

the action proposed by the actor. The system gains the capability to estimate the action-value function, producing a single scalar output that represents the expected return for the given state-action pair. The outcome of this phase is a trained RL-PSO model that effectively manages swarm behavior by dynamically adjusting its parameters based on the current state of the swarm.

2.3 Performance Evaluation

The final stage involves a comprehensive comparative analysis to assess the efficacy of the hybrid algorithm. The trained model is evaluated against the traditional PSO baseline using a distinct testing dataset comprising 10 maps (4 low-complexity, 4 medium-complexity, and 2 high-complexity). The performance is assessed by comparing the convergence time and diversity value of the two methods. Each method is executed for 50 iterations on each map instance, and the performance metrics are averaged to mitigate stochastic fluctuation. The collected data are statistically analyzed using Mixed Linear Model Regression (MLMR) to determine the significance of the results and validate that the observed improvements are directly related to the incorporation of RL. This leads to the statistical validation and comparison analysis, thereby concluding the study.

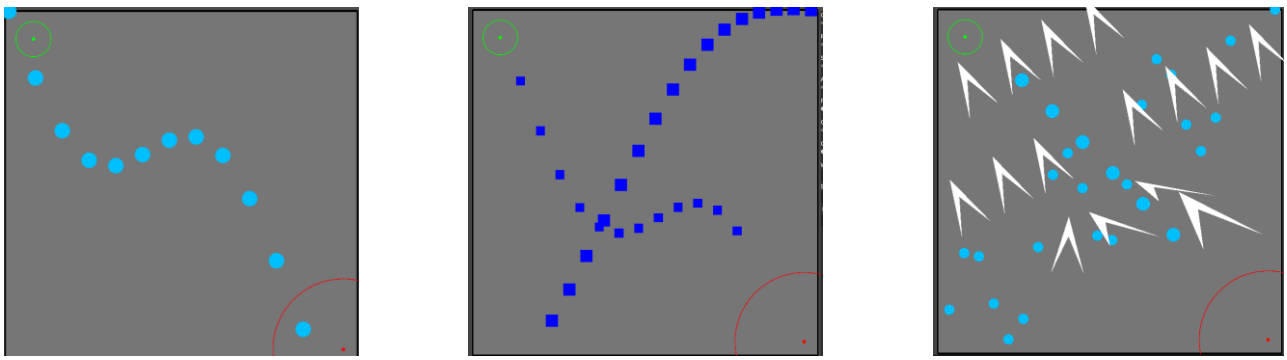
3. Results and Discussion

3.1 Simulation Design & Implementation

The initial phase involved setting up a realistic experimental environment. The main aim was to develop a simulation that would serve as a more demanding and relevant benchmark compared to traditional static mathematical functions. A comprehensive collection of maps was essential for the training and evaluation of the adaptive swarm method. Maps were categorized into three distinct levels of complexity, determined by the quantity and geometric configuration of the obstacles: low, medium, and high. This categorization was established to evaluate the algorithm's ability to adapt to various environmental constraints.

The complexity of the map was determined by the number of obstacles present, classified as follows: low to medium complexity comprises 10 to 30 obstacles, whereas high complexity includes 30 to 50 obstacles. Furthermore, the geometrical configurations of these obstacles presented a range of variations, including basic shapes such as circles, ellipses, and regular polygons, as well as more complex irregular polygons. A dedicated tool was developed in the Godot Engine to procedurally generate these maps by placing barriers using trigonometric functions, ensuring consistency and scalability.

Figure 3 illustrates procedurally generated maps that serve as the experimental environments, increasing in complexity based on obstacle quantity and geometric shape variation: (a) low complexity features fewer obstacles of simple shapes (circles, ellipses); (b) medium complexity introduces geometric polygons (e.g., squares); (c) high complexity (30–50 obstacles) presents irregular polygons, making swarm navigation and adaptability more difficult.



(a) (b) (c)

Figure 3. Example of Low, Medium, and High Complexity Maps

The traditional PSO had been successfully enhanced with the incorporation of Godot's *move_and_slide* function. This offered a physically plausible mechanism for particles to traverse by gliding along the surfaces of obstacles. This adaptation effectively addresses a notable challenge, as the conventional PSO would come to a complete stop in the event of a collision. The solution required the implementation of custom logic to address various obstacle shapes, which involved calculating a new velocity vector for the particle in accordance with the surface normal at the point of collision. A sliding motion algorithm was implemented, utilizing vector projection to adjust the particle's trajectory. *Figures 4* illustrates the mechanism used in the Godot engine, employing vector projection to modify the particle's velocity in accordance with the surface normal of the linear obstacle. This modification enables the particle to realistically slide along the barrier, effectively solving the issue of particles coming to a complete stop upon collision in standard mathematical PSO models.

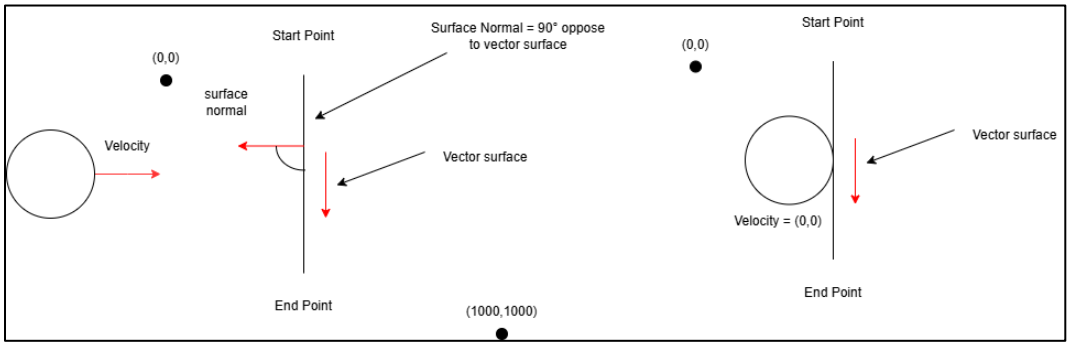


Figure 4. Visualization of Sliding Motion with Linear Obstacle

Figure 5 illustrate this behavior for both linear and circular obstacles. This diagram shows how circular obstacle collisions are handled. It demonstrates how the contact point and circle center form a normal vector from the surface vector. Real-time swarm movement in dynamic situations requires smooth obstacle avoidance without becoming trapped, which is provided by Godot's *move_and_slide* function.

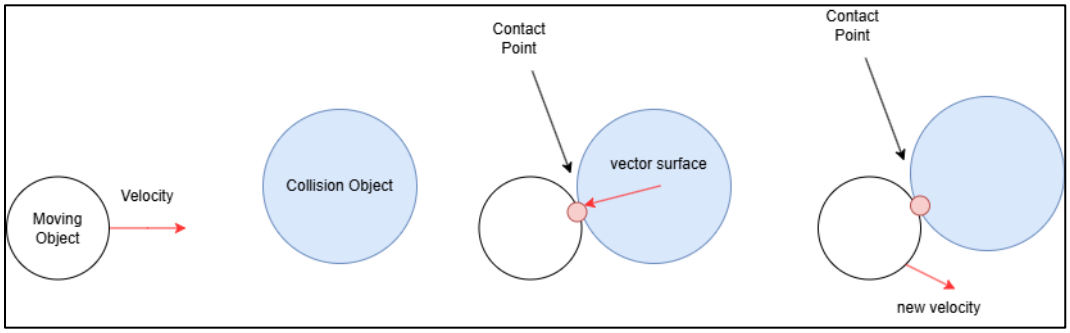


Figure 5. Visualization of Sliding Motion with Circular Obstacle

3.2 Model Training

This phase focused on training the DDPG agent to intelligently control the swarm's parameters. A core finding was the need for a robust communication framework to integrate the Godot simulation with the external RL agent (GodotRL) [22], [23]. An event-based, real-time communication system was developed for continuous learning. State information (e.g., diversity value, mean velocity) was submitted to GodotRL, which evaluated it and returned improved PSO parameters. This seamless, two-way channel was essential for the adaptive learning process and the online parameter control of the swarm. *Figure 6* illustrates the real-time, closed-loop system used for training the adaptive model. The Godot simulation environment asynchronously sends the swarm's global state (inputs) through a channel to the external godotRL DDPG agent. The agent evaluates the state and transmits the optimized PSO parameters (actions) back to Godot through a separate channel to dynamically control the swarm. Online learning and parameter management of the RLGPSO algorithm require this smooth, two-way communication.

Reinforcement Learning is cross-platform transferable due to this architectural choice. The learned policy (the neural network) is completely independent of the Godot Engine environment because the DDPG agent runs externally via a Python script and communicates through a standard network protocol. This suggests that the system's adaptive intelligence is portable. The challenge for full cross-platform validation, therefore, lies solely in replicating the customized physics behavior, specifically the recreation of the procedural map generation logic and the accurate implementation of PSO's dynamic obstacle avoidance, such as the *move_and_slide* vector projection logic, within a new physics engine (e.g., Unity or Unreal Engine). This structural separation confirms that a significant portion of the system—the adaptive policy controller—is inherently cross-platform, making the required simulation-to-simulation transfer largely an engineering task.

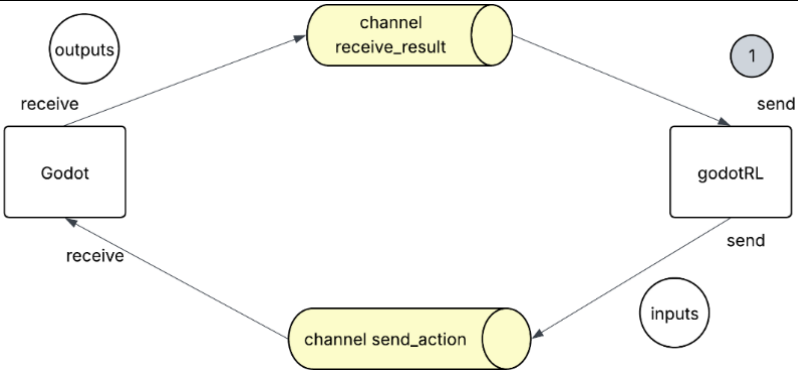


Figure 6. Communication Framework between Godot and GodotRL

The model was trained over 1,024 instances (episodes) using a sequential, then randomized, map selection process. It is critical to note that the integration of the DDPG agent introduces a computational overhead compared to static PSO, arising from the real-time forward pass through the deep neural networks and the necessary backpropagation steps for continuous online parameter learning. Future work should rigorously quantify this overhead, as it is a limiting factor for deploying the RLGPSO in resource-constrained, large-scale game AI environments.

3.3 Performance Evaluation

The final phase was a thorough comparison to demonstrate the effectiveness of the hybrid RLGPSO algorithm. The results revealed that RLGPSO outperformed traditional PSO. The trials used maps of varying complexity and swarm sizes. The performance of each algorithm was assessed through an analysis of convergence time and average diversity value. *Table 3* provides a comparison of essential performance metrics, specifically convergence time (in seconds) and diversity value, for both the traditional PSO and the RLGPSO algorithm, analyzed across different levels of map complexity and swarm sizes. The data demonstrate that RLGPSO achieves reduced convergence times and enhanced diversity values in all scenarios, with the most pronounced improvement in convergence time observed on the high-complexity map with 25 agents.

Table 3. Average Experimental Results

Independent Variables		PSO		RLPSO	
N agents	Map Complexity	Convergence (s)	Diversity	Convergence (s)	Diversity
5	Low	6.00	6.140	5.767	9.494
	Medium	7.181	6.149	6.449	10.301
	High	10.332	7.756	9.486	10.944
15	Low	6.742	8.053	5.88	9.977
	Medium	7.021	8.289	6.523	10.804
	High	11.894	11.991	11.567	12.027
25	Low	7.532	8.996	6.394	10.313
	Medium	7.927	9.386	6.972	10.939
	High	14.060	14.133	12.448	13.160

These findings were further validated using Mixed Linear Model Regression (MLMR). This method was selected to account for the hierarchical organization of the data and the random influences of individual map variations. The results indicated that the *rl_condition* variable significantly influenced convergence time, evidenced by a p-value of 0.002. The coefficient of 0.846 for the "without" condition indicates that, from a statistical perspective, the lack of RL adaptation is associated with a notable increase in convergence time. The data indicate that the incorporation of RL serves as a viable approach to enhance the efficiency of the swarm. Additionally, customization of the *swarm_size* parameter significantly affects convergence time. When the swarm size increased from 5 to 15 particles, convergence time increased by 2.081, and for 25 particles, by 2.962. These findings suggest that larger swarms require more time to converge due to increased coordination complexity. The *map_size* variable demonstrated a significant effect, as indicated by negative coefficients for the low (-3.719, p=0.001) and medium (-3.037, p=0.009) complexity maps. This suggests that convergence typically occurs faster in smaller, less complex environments.

A significant issue was identified concerning the objective function of the swarm. Using Euclidean distance to the target as the sole fitness metric can trap a swarm in a local loop. This occurs when a barrier forces the swarm to relocate to a position that is farther from the objective, as measured by Euclidean distance. The particles may revert to their previous, more advantageous position, resulting in a continuous oscillation without progress toward the objective. Using

a fundamental distance metric exclusively for navigation in complex environments is a significant limitation, as demonstrated by the behavior depicted in Figure 7. This figure illustrates the critical limitation where the swarm, when blocked by an obstacle, enters a local oscillation loop by oscillating between positions A and B. This behavior occurs because the Euclidean distance metric (used as the objective function) dictates that position A is "better" (closer to the goal) than position B, causing the swarm to revert from the necessary detour. This highlights the need for path-planning integration, possibly through the incorporation of a pathfinding or A* search method to aid navigation in complex terrain [24], [25].

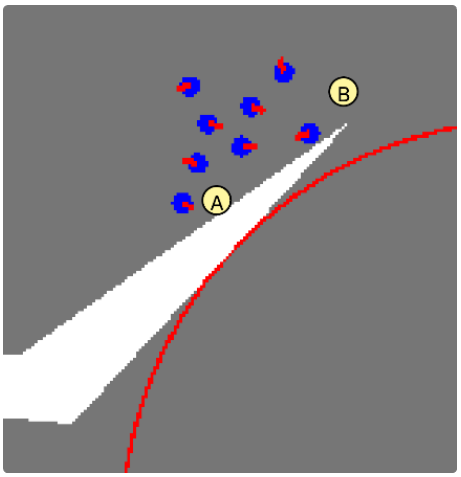


Figure 7. Visualization of Swarm Repeating Route

3.4 Dynamic Performance and Policy Stability

Understanding learning and adaptive policy stability requires longitudinal investigation. This strategy addresses model dynamics and policy volatility. The DDPG agent's performance during training was assessed using the smoothed cumulative reward, as illustrated in Figure 8. Initially, the reward data exhibited volatility, which was characteristic of the DDPG agent's necessary exploratory phase within the wide, continuous parameter space (ω, c_1, and c_2). Following this, the smoothed reward curve demonstrated a gradual increase, ultimately stabilizing at a high, positive final value. This stabilization indicated that the adaptive policy was both reliable and consistent in identifying high-quality solutions after the initial exploration period. The convergence shown in this longitudinal plot confirmed the RLGPSO model's ability to maintain a stable policy throughout training, thus mitigating policy variations and the risk of overfitting.

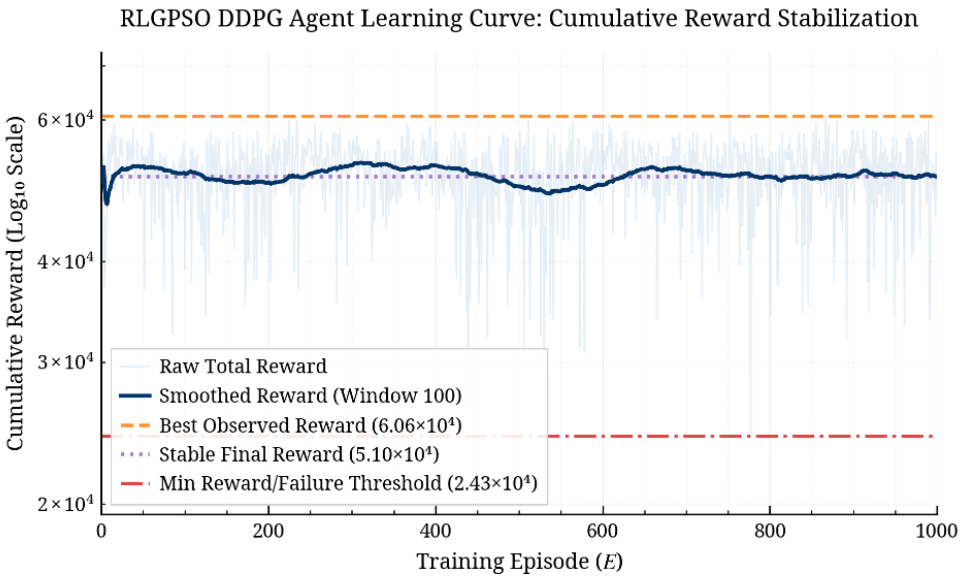


Figure 8. Cumulative Reward Learning Curve

Furthermore, Figure 9 presents the policy ensemble stabilization by plotting the average smoothed convergence speed across all low- and medium-complexity test maps. This plot shows that the maximum operational efficiency is achieved almost instantaneously at the start of the test phase, typically stabilizing within the first 12 test episodes. The rapid stabilization time indicates that the fully trained model policy generalizes effectively, thereby proving the model's resilience to both policy changes and overfitting in the operational phase.

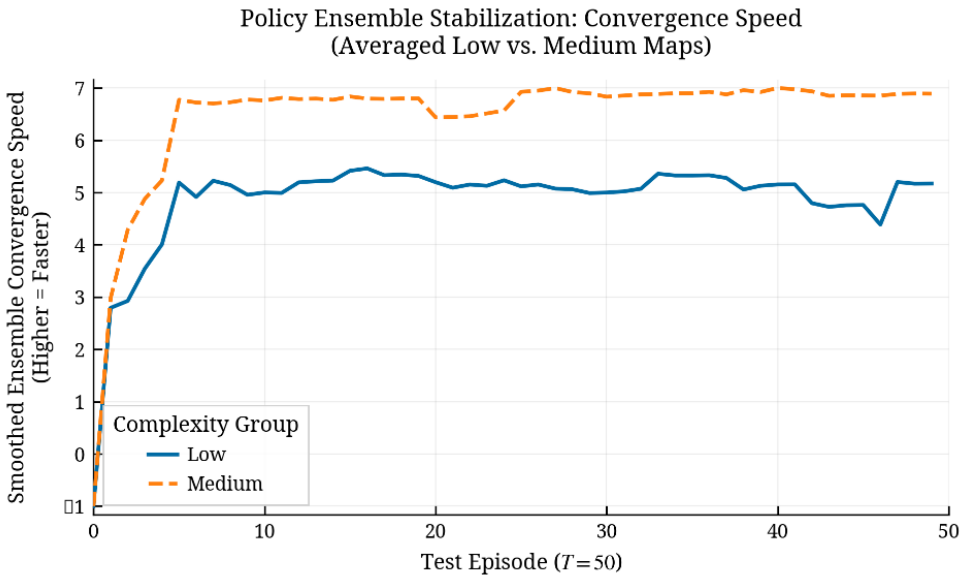


Figure 9. Policy Ensemble Stabilization (Rapid Convergence)

4. Conclusion

This study successfully introduced the Reinforcement Learning Godot Particle Swarm Optimization (RLGPSO) hybrid algorithm to address the critical limitation of static parameter configurations in traditional PSO within dynamic, obstacle-rich environments characteristic of RTS games. By employing a DDPG agent that operates in a computationally decoupled manner via an external API as a dynamic parameter controller, the RLGPSO model learned to adapt the continuous PSO parameters (ω, c_1, and c_2) in real-time, effectively mitigating the issues of premature convergence and inadequate flexibility found in fixed-parameter settings. The experimental validation carried out in a physics-based simulation on the Godot Engine, utilizing procedurally generated maps, demonstrated that this adaptive control mechanism notably enhanced swarm performance.

The RLGPSO model demonstrated consistently faster convergence and enhanced swarm diversity across various map complexities and swarm sizes. The analysis conducted using the MLMR confirmed this superior performance as statistically significant, providing strong evidence supporting the effectiveness of the hybrid approach. The longitudinal analysis of the DDPG training process confirmed its robustness, demonstrating that the cumulative reward curve stabilized quickly over the training episodes. This finding affirms the consistency and stability of the learned adaptive policy, thereby increasing confidence in its application within the simulated environment. The experimental validation carried out in a physics-based simulation on the Godot Engine, utilizing procedurally generated maps, demonstrated that this adaptive control mechanism notably enhanced swarm performance.

The RLGPSO algorithm presents a strong and effective solution for intelligent, self-organizing agent swarms. However, future research needs to tackle three key challenges identified in this study to improve its applicability and scalability in real-world scenarios. Local oscillations and poor path repetition near complex barriers may arise due to the sole use of Euclidean distance as the objective function. Path-planning methods like A* search are needed in this case. Furthermore, the deployment of the DDPG agent introduces a computational burden that needs to be carefully assessed to confirm the algorithm's practicality and adaptability for significantly larger, resource-limited real-time systems. Ultimately, while the AI policy is fundamentally cross-platform owing to its decoupled architecture (operating externally to the simulation through an API), it is crucial to conduct cross-platform validation to ensure the generalizability of the learned policy. The validation process necessitates the effective translation of custom physics behaviors, including the implementation of *move_and_slide* for obstacle avoidance, into various alternative simulation platforms.

Acknowledgement

This research was made possible through financial support provided by DIPA funds from the Bandung State Polytechnic. The authors gratefully acknowledge this support, which was granted under the Applied Research funding contract No. 158.3/R7/PE.01.03/2025.

References

[1] E. Z. Elfeky *et al.*, "A Systematic Review of Coevolution in Real-Time Strategy Games," *IEEE Access*, vol. 9, pp. 136647–136665, 2021. https://doi.org/10.1109/ACCESS.2021.3115768

[2] Y. Zhen, Z. Wanpeng, and L. Hongfu, "Artificial Intelligence Techniques on Real-time Strategy Games," in *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*, New York, NY, USA: ACM, Dec. 2018, pp. 11–21. https://doi.org/10.1145/3297156.3297188

[3] J. Tang, G. Liu, and Q. Pan, "A Review on Representative Swarm Intelligence Algorithms for Solving Optimization Problems: Applications and Trends," *IEEE/CAA J. Autom. Sin.*, vol. 8, no. 10, pp. 1627–1643, Oct. 2021. https://doi.org/10.1109/JAS.2021.1004129

[4] A. Xu, H. Li, Y. Hong, and G. Liu, "Autonomous Decision-Making for Air Gaming Based on Position Weight-Based Particle Swarm Optimization Algorithm," *Aerospace*, vol. 11, no. 12, p. 1030, Dec. 2024. https://doi.org/10.3390/aerospace11121030

[5] A. G. Gad, "Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review," *Arch. Comput. Methods Eng.*, vol. 29, no. 5, pp. 2531–2561, Aug. 2022. https://doi.org/10.1007/s11831-021-09694-4

[6] M. R. Bonyadi and Z. Michalewicz, "Particle Swarm Optimization for Single Objective Continuous Space Problems: A Review," *Evol. Comput.*, vol. 25, no. 1, pp. 1–54, Mar. 2017. https://doi.org/10.1162/EVCO_r_00180

[7] M. Kamosi, A. B. Hashemi, and M. R. Meybodi, "A New Particle Swarm Optimization Algorithm for Dynamic Environments," 2010, pp. 129–138. https://doi.org/10.1007/978-3-642-17563-3_16

[8] B. F. Azevedo, A. M. A. C. Rocha, and A. I. Pereira, "Hybrid approaches to optimization and machine learning methods: a systematic literature review," *Mach. Learn.*, vol. 113, no. 7, pp. 4055–4097, Jul. 2024. https://doi.org/10.1007/s10994-023-06467-x

[9] Z. Jiang, D. Zhu, X.-Y. Li, and L.-B. Han, "A Hybrid Adaptive Particle Swarm Optimization Algorithm for Enhanced Performance," *Appl. Sci.*, vol. 15, no. 11, p. 6030, May 2025. https://doi.org/10.3390/app15116030

[10] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A Survey of Deep Reinforcement Learning in Video Games," no. 61573353, pp. 1–13, 2019. https://doi.org/10.48550/arXiv.1912.10944

[11] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep Learning for Video Game Playing," *IEEE Trans. Games*, vol. 12, no. 1, pp. 1–20, Mar. 2020. https://doi.org/10.1109/TG.2019.2896986

[12] O. Aoun, "Deep Q-Network-Enhanced Self-Tuning Control of Particle Swarm Optimization," *Modelling*, vol. 5, no. 4, pp. 1709–1728, 2024. https://doi.org/10.3390/modelling5040089

[13] W. Li, Y. Xiong, and Q. Xiong, "Reinforcement Learning-Guided Particle Swarm Optimization for Multi-Objective Unmanned Aerial Vehicle Path Planning," *Symmetry (Basel).*, vol. 17, no. 8, p. 1292, Aug. 2025. https://doi.org/10.3390/sym17081292

[14] Z. Dong, Q. Wu, and L. Chen, "Reinforcement Learning-Based Formation Pinning and Shape Transformation for Swarms," *Drones*, vol. 7, no. 11, p. 673, Nov. 2023. https://doi.org/10.3390/drones7110673

[15] F. Zhang and Z. Chen, "A Novel Reinforcement Learning-Based Particle Swarm Optimization Algorithm for Better Symmetry between Convergence Speed and Diversity," *Symmetry (Basel).*, vol. 16, no. 10, p. 1290, Oct. 2024. https://doi.org/10.3390/sym16101290

[16] K. H. Sharif and S. Yousif Ameen, "Game Engines Evaluation for Serious Game Development in Education," in *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, IEEE, Sep. 2021, pp. 1–6. https://doi.org/10.23919/SoftCOM52868.2021.9559053

[17] Y. Chen and H. Sundaram, "Estimating Complexity of 2D Shapes," in *2005 IEEE 7th Workshop on Multimedia Signal Processing*, IEEE, Oct. 2005, pp. 1–4. https://doi.org/10.1109/MMSP.2005.248668

[18] M. Rothgänger, A. Melnik, and H. Ritter, "Shape Complexity Estimation Using VAE," 2024, pp. 35–45. https://doi.org/10.1007/978-3-031-47715-7_3

[19] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intell.*, vol. 1, no. 1, pp. 33–57, Oct. 2007. https://doi.org/10.1007/s11721-007-0002-0

[20] E. H. Sumiea *et al.*, "Deep deterministic policy gradient algorithm: A systematic review," *Heliyon*, vol. 10, no. 9, p. e30697, May 2024. https://doi.org/10.1016/j.heliyon.2024.e30697

[21] S. Yin *et al.*, "Reinforcement-learning-based parameter adaptation method for particle swarm optimization," *Complex Intell. Syst.*, vol. 9, no. 5, pp. 5585–5609, Oct. 2023. https://doi.org/10.1007/s40747-023-01012-8

[22] E. Beeching, J. Debangoye, O. Simonin, and C. Wolf, "Godot Reinforcement Learning Agents," Dec. 2021. https://doi.org/10.48550/arXiv.2112.03636

[23] M. Ranaweera and Q. H. Mahmoud, "Deep Reinforcement Learning with Godot Game Engine," *Electronics*, vol. 13, no. 5, p. 985, Mar. 2024. https://doi.org/10.3390/electronics13050985

[24] A. Rafiq, T. Asmawaty Abdul Kadir, and S. Normaziah Ihsan, "Pathfinding Algorithms in Game Development," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 769, no. 1, p. 012021, Feb. 2020. https://doi.org/10.1088/1757-899X/769/1/012021

[25] S. R. Lawande, G. Jasmine, J. Anbarasi, and L. I. Izhar, "A Systematic Review and Analysis of Intelligence-Based Pathfinding Algorithms in the Field of Video Games," *Appl. Sci.*, vol. 12, no. 11, p. 5499, May 2022. https://doi.org/10.3390/app12115499