Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control
Journal homepage: http://kinetik.umm.ac.id
ISSN: 2503-2267
Vol. 11, No. 3, August, Pp. 513-526

Scalable multi-agent formation control in RTS games: A virtual anchor and fluid-based allocation

Ibnu Athaillah*[1], Moch. Kholil[1]
Akademi Komunitas Negeri Putra Sang Fajar Blitar, Indonesia[1]

Article Info
Keywords:
Formation Control, Mult-Agent Navigation, Real-Time Strategy, Unity Job System, Virtual Anchor

Article history:
Received: December 16, 2025
Accepted: April 08, 2026
Published: August 01, 2026

Cite:
I. Athaillah and M. Kholil, "Scalable Multi-Agent Formation Control in RTS Games: A Virtual Anchor and Fluid-Based Allocation", *KINETIK*, vol. 11, no. 3, Aug. 2026.
https://doi.org/10.22219/kinetik.v11i3.2643

*Corresponding author.
Ibnu Athaillah
E-mail address:
aieiii@protonmail.com

Abstract
The control system for troop formation movement is a critical component of Real-Time Strategy (RTS) games, directly affecting gameplay quality and player experience. However, traditional CPU-bound pathfinding algorithms fail to maintain 60 FPS when handling more than 1,000 agents in dynamic environments, particularly when balancing rigid formation structure with pathfinding efficiency amid complex obstacles. This study proposes an integrated framework for troop formation movement that synthesizes a virtual Anchor navigation paradigm with a Fluid-Based Formation Position Allocation algorithm. Unlike traditional leader-follower methods, the proposed system utilizes a virtual anchor to calculate global pathfinding via NavMesh, while constituent agents dynamically adjust their positions relative to this reference point. To mitigate trajectory conflicts during formation changes, the system employs a fluid-dynamics-inspired sorting strategy that deterministically maps agents to target slots using parallel processing. The architecture is optimized for real-time performance using the Unity Job System, allowing for the coordination of large-scale agent aggregates. Experimental validation was conducted through behavioral scenarios—including Tunnel, Split, and Crowd tests—as well as stress tests involving up to 4,096 agents. The results demonstrate that the system successfully maintains formation integrity, executes autonomous regrouping after obstacle traversal, and ensures collision-free movement. Performance analysis further indicates that the control logic remains computationally stable at scale, with the primary limitations shifting to graphical rendering overhead rather than algorithmic complexity.

1. Introduction

The control system for troop formation movement in strategy games constitutes a pivotal component that determines gameplay quality and player experience within the Real-Time Strategy (RTS) and tactical video game genres. The coordination of units within specific formations not only enhances visual fidelity and simulation realism but also provides strategic depth, enabling players to execute complex military tactics such as flanking maneuvers, establishing defensive lines, and coordinating group movements. In contrast to simplistic approaches in which individual units traverse independently toward a common destination, a robust formation control system ensures that unit groups maintain specific geometric structures during locomotion, collectively respond to environmental obstacles, and arrive simultaneously at the target location in the desired configuration [1].

Research in the domain of multi-agent formation control has advanced significantly over the past two decades, propelled by extensive applications in robotic systems [2], unmanned aerial vehicles (UAVs), and military simulations. The literature on formation control is predominantly characterized by three primary methodologies: the leader-follower method [3], the virtual structure approach [4], and behavior-based control [5]. The leader-follower paradigm designates one or more units as leaders responsible for dictating the group's trajectory, while follower units maintain their relative positions with respect to the the leader. This approach is favored because of its implementation simplicity and the absence of a requirement for global system knowledge. The virtual structure method utilizes a virtual framework to define the formation coordinates for each unit, wherein each agent tracks a virtual leader projected ahead of its position. Conversely, the behavior-based approach assigns a set of fundamental behaviors to each agent, such as cohesion, separation, and alignment, which are synthesized to generate emergent formation patterns.

The implementation of formation control systems in strategy games presents unique challenges distinct from those encountered in conventional robotics applications. Real-Time Strategy games require real-time responsiveness, the management of substantial unit aggregates (ranging from tens to hundreds of units), formation maintenance within dynamic environments replete with complex obstacles, and adaptation to heterogeneous unit velocities. One fundamental challenge is separation, whereby unit groups fragment while navigating around large obstacles as

individual units compute different optimal paths. This phenomenon can detrimentally impact the player experience [6]. Furthermore, the system must achieve an equilibrium between strict formation rigidity and pathfinding efficiency while simultaneously executing collision avoidance protocols with respect to intra-group agents, adversarial units, and environmental features [7].

The flocking algorithm, introduced by Reynolds [8] through the Boids model, has exerted a significant influence on the advancement of formation control in the gaming domain. The three fundamental principles of the Boids model—separation (avoiding local crowding), alignment (matching velocity with neighbors), and cohesion (steering toward the average position of neighboring agents)—enable the generation of organic, realistic collective behaviors without the necessity for centralized control mechanisms. Furthermore, the integration of flocking algorithms with pathfinding techniques such as A* [9], Lazy Theta* [10], artificial potential fields, and local avoidance methods such as Reciprocal Velocity Obstacles (RVO) has demonstrated effectiveness in generating smooth, collision-free formation trajectories [11].

While the aforementioned methods [4], [5] provide realistic emergent movement, they fundamentally rely on $O(n^2)$ pairwise interaction computations for collision avoidance, rendering them unsuitable for large-scale RTS games requiring the coordination of thousands of agents in real time. Similarly, flocking-based approaches, despite their ability to generate organic collective motion, operate on stochastic local rules that provide no deterministic guarantee of structural fidelity; agents may drift unpredictably when environmental geometry constrains the formation. This research addresses this gap by proposing a deterministic spatial mapping strategy that replaces probabilistic neighbor-based steering with a fluid-dynamics-inspired hierarchical sorting algorithm executed asynchronously using the Unity Job System and Burst Compiler. Unlike flocking methods, in which formation shape emerges implicitly from local interactions, the proposed approach explicitly and deterministically assigns each agent to a target slot through coordinate-space transformation, thereby guaranteeing formation integrity while maintaining computational scalability for up to 4,096 simultaneous agents.

Table 1 presents a structured comparison of the proposed framework against the three established formation control paradigms, highlighting the scientific distinctions in pathfinding strategy, position assignment mechanism, collision avoidance complexity, formation guarantee, and scalability.

Table 1. Comparative Analysis of Formation Control Methods

Criterion	Leader–Follower [3]	Virtual Structure [4]	Behavior-Based / Flocking [5,8]	Proposed (Virtual Anchor + Fluid)
Pathfinding	Leader computes; followers track	Virtual reference computes	Per-agent local steering	Single Anchor NavMesh path
Position Assignment	Implicit (relative offset)	Explicit (virtual frame)	Emergent (stochastic rules)	Deterministic (fluid-based sort)
Collision Avoidance Complexity	O(n) pairwise to leader	O(n) per virtual frame	$O(n^2)$ pairwise neighbors	$O(n^2)$ sorting, parallelized via the Job System
Formation Guarantee	Weak (leader failure = group failure)	Moderate (rigid virtual frame)	None (emergent, non-deterministic)	Strong (deterministic slot mapping)
Scalability (tested)	~100 agents	~100 agents	~500 agents (with RVO)	4,096 agents (tested)
Obstacle Handling	Group fragments if leader path diverges	Rigid; no adaptive deformation	Adaptive but may lose structure	Single-path; regroups post-obstacle
Real-time Suitability	Moderate	Low (global recomputation)	Moderate	High (asynchronous parallel execution)

The research and development discussed in this paper aims to leverage these navigation mechanisms by incorporating a dedicated formation assembly component. The system executes group movements based on specific parameters such as velocity, inter-agent distance, and rotation angle while dynamically adjusting to the final destination. Furthermore, the proposed formation module is equipped with an agent position rescheduling algorithm designed to maintain cohesion, ensuring that the group remains compact even when the destination or movement trajectory changes.

The remainder of this paper is organized as follows. First, the architectural design of the proposed formation system is presented, followed by the methodologies employed for group movement control and the evaluation of the simulation results. The proposed solution is expected to significantly improve both the efficiency and flexibility in the orchestration of large-scale agent movements, thereby facilitating its integration into diverse game development scenarios and broader simulation applications that necessitate the coordination of virtual multi-agent systems.

To the best of the authors' knowledge, no existing work integrates a virtual-anchor navigation paradigm with a deterministic, fluid-dynamics-inspired position allocation algorithm optimized for real-time game engine constraints. The

primary contributions of this study are threefold: (1) a hybrid navigation model that decouples group pathfinding from individual agent steering, whereby a virtual Anchor computes a single NavMesh path for the entire formation while constituent agents independently resolve their local positions using the Unity Job System; (2) a scalable, deterministic position allocation algorithm inspired by fluid dynamics that prevents agent "clumping" and trajectory conflicts during formation transitions by replacing costly $O(n^2)$ cost-matrix optimizations (e.g., the Hungarian algorithm) with a two-stage hierarchical sorting algorithm suitable for parallel execution; and (3) implementation results demonstrating formation stability and collision-free coordination for up to 4,096 simultaneous agents, with computational overhead remaining bounded by rendering costs rather than algorithmic complexity. This proposed approach is explicitly designed to accommodate the real-time computational constraints of game engines such as Unity by leveraging optimization techniques including spatial partitioning via Kd-trees for efficient neighbor queries [12] and the Unity Job System with Burst Compiler for parallel computation [13].

2. Research Method

This study employs an experimental methodology structured around the software development lifecycle, with a particular focus on the implementation of agent navigation and formation systems. A high-level schematic of the applied research methodology is illustrated in Figure 1.

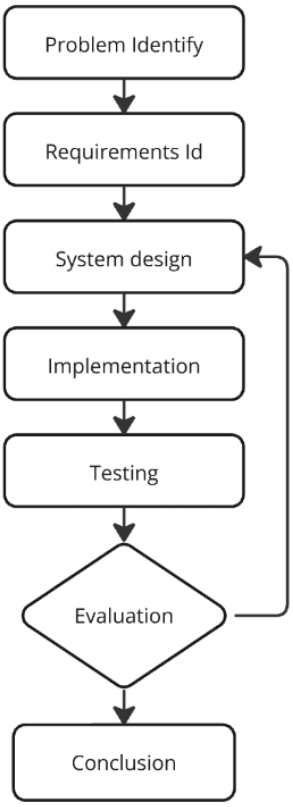

Figure 1. Software Development Lifecycle Methodology

Requirements Identification: The initial phase involves identifying the system prerequisites based on the needs of the agent formation simulation. This includes defining agent characteristics and determining critical formation parameters, such as inter-row spacing, velocity profiles, and formation orientation angles.

System Design: The second phase focuses on architecting a framework that encompasses three core modules: formation creation, group movement control, and integration with the navigation systems (e.g., NavMesh). Additionally, flowcharts are constructed to elucidate the structural hierarchy and facilitate a clear understanding of inter-module interactions.

Implementation: This phase involves developing the formation module, which computes agent positions and orientations through parallel processing. Subsequently, movement control algorithms are implemented to dynamically adjust agent positions relative to the simulation state by incorporating velocity and rotational modulation. The final step integrates the formation module with the underlying navigation system provided by the development platform (e.g., the Unity Engine).

Testing and Evaluation: A series of experiments is conducted to assess the reliability of the formation module. Key performance metrics include terminal position accuracy, traversal time, and robustness against dynamic target alterations or external perturbations. Simulation performance is evaluated across diverse scenarios with varying agent population sizes, inter-nodal spacing, and formation configurations. Experimental outputs are logged and analyzed to identify the strengths and limitations of the proposed system.

Conclusion and Recommendations: In the final phase, conclusions are synthesized based on the established performance metrics. Furthermore, recommendations for future work are provided, highlighting potential applications in broader simulation scenarios. By following this methodological framework, the study aims to yield an effective and efficient agent formation system, thereby contributing to the advancement of game development and multiplayer simulation applications that require automated agent coordination.

2.1 Requirements Identification

Functionally, the system is required to autonomously map a specific aggregate of agents into predefined formations while dynamically adjusting the spatial arrangement and inter-agent spacing within the group. Consequently, the system must provide a navigation method integrated with the formation logic, ensuring that each agent traverses to its designated locus without collision. Furthermore, the system must support real-time adaptation, enabling immediate reconfiguration of the formation topology in response to abrupt changes in environmental conditions or objectives.

From a performance perspective, formation position calculations must be executed efficiently to accommodate large agent populations without significant performance degradation. The system is required to maintain low latency during agent position recomputation, particularly during target reassignment or distance modification events. In terms of usability, the formation module should be accessible to developers through a highly configurable interface [14], allowing intuitive adjustment of parameters such as spacing, rotation angle, and formation shape. Comprehensive documentation and implementation examples should also be provided to elucidate the workflow and facilitate seamless integration into future projects. Finally, with respect to system stability, the solution should be developed using a modular architecture [15] to ensure compatibility with external components, such as Artificial Intelligence (AI) or animation subsystems. Furthermore, the system should ensure that increases in agent density or terrain complexity do not result in excessive computational overhead by providing adaptive parameter tuning capabilities.

By synthesizing these requirements, the proposed formation system establishes a robust foundation for development and deployment across diverse applications, ranging from video games and military simulations to broader research initiatives.

2.2 System Design

A. Group Movement Architecture

The architecture of the group movement system is engineered using a Finite State Machine (FSM) [16]paradigm, integrated with the Unity Job System to facilitate parallel computational optimization. The architecture adopts a virtual agent-based formation approach (referred to as an Anchor), in which a virtual anchor functions as the primary navigation reference for the entire group. This approach differs from the conventional leader-follower paradigm, where an active group member acts as the leader. The core components of the proposed system are as follows:

1. Anchor: A virtual agent-based entity responsible for computing and executing pathfinding operations for the entire formation.
2. Phase Management: A phase management mechanism utilizing a Finite State Machine (FSM) consisting of seven distinct states (0–6) that govern transitions between movement phases.
3. Parallel Computation: Implementation of the Unity Job System to facilitate parallel computation for formation position allocation and individual agent movement.
4. Dynamic Speed Adaptation: An adaptive system that modulates each agent's velocity according to its relative distance to their designated target position within the formation.

B. Group Member Formation Positioning Model

A group utilizes a formation structure as a reference system to define both the group's centroid and the individual spatial positions of its constituent agents. The coordinate of each agent within the formation is obtained by transforming its local grid coordinate into 3D world space, as expressed in Equation 1:

$$P_i = R(\theta) \times G_i + C_{anchor} + O \quad (1)$$

The 3D world position of the i-th agent, denoted by (P_i), is determined by applying the formation rotation matrix ($R(\theta)$) to the agent's local grid position (G_i), and then adding the anchor position (C_{anchor}) and the designated offset parameter (O). The rotation matrix $R(\theta)$ is derived from either the current orientation of the anchor or the target formation rotation, depending on the active movement phase. The local grid vector G_i is defined by the desired formation topology,

specifically the arrangement of rows and columns, where the initial row corresponds to the formation's anterior (front) and the initial column corresponds to the leftmost flank.

The offset parameter, initialized by default to $(\mathbf{0}, \mathbf{0})$, serves as an auxiliary variable that allows developers to shift the formation centroid relative to the anchor. This facilitates specific alignments, such as positioning the pivot at the anterior-left coordinates $(-\boldsymbol{x}, y)$ or the posterior-right coordinates $(\boldsymbol{x}, -y)$. In this study, the default value was retained, ensuring that the formation centroid remained precisely aligned with the anchor. However, the exposure of this parameter as a mutable property is deemed critical for extensibility and future development flexibility.

C. Fluid-Based Formation Position Allocation Algorithm

During the transition from an initial formation to a target configuration, a static mapping strategy in which the $\boldsymbol{i}$-th agent is strictly assigned to the $\boldsymbol{i}$-th grid slot, significantly increases the probability of trajectory intersection conflicts among group members. This phenomenon is inevitable during maneuvers requiring retrograde (backward) or lateral (sideways) displacement relative to the initial orientation.

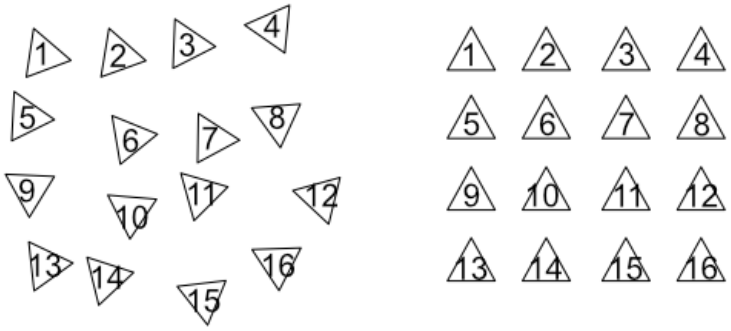

Figure 2. Sorting n-agent to n-position

A conventional mitigation strategy involves gradually rotating the entire aggregate structure prior to translation. However, this approach introduces substantial latency, effectively delaying the initiation of group movement toward the target configuration. A more efficient alternative involves dynamically assigning the $\boldsymbol{i}$-th agent to the $\boldsymbol{j}$-th target position, utilizing a directional heuristic inspired by fluid dynamics mechanics as illustrated in Figure 2.

This method employs a two-dimensional hierarchical sorting strategy integrated with coordinate-space transformations to facilitate coherent formation transitions while minimizing spatial interference. The implementation is executed through parallel processing tasks that solve the assignment problem mapping n agents to n target formation slots utilizing a deterministic spatial matching algorithm. The pseudocode for this algorithm can be seen in Figure 3.

The rotation quaternion (Q) is derived from the transformation of the formation displacement vector. The inverse quaternion, (Q^{-1}), is applied to both the constituent unit positions and the target formation slots. These transformed coordinates are subsequently normalized to establish a local reference frame in which the formation trajectory is aligned with the negative z-axis. This transformation mechanism guarantees spatial consistency, functioning independently of the specific correspondence between source coordinates and destination coordinates.

The fundamental core of the proposed algorithm employs a two-stage hierarchical sorting paradigm. The first stage sorts the longitudinal z-axis coordinates of both the constituent group units and the target formation slots. This operation utilizes the insertion sort algorithm [17], preserving the initial indices while storing the sorted order in a dedicated array and executing the process in parallel.The second stage performs sorting along the transverse x-axis. The array, previously ordered according to the z-axis, is partitioned into rows with a predefined width (P_{col}). Within each partition, a secondary sort is performed in descending order of the x-axis coordinates, resulting in a spatial distribution aligns with the grid structure.

In contrast to the Hungarian algorithm [18], this two-stage sorting method ensures that, upon completion of both stages, the n-th agent is mapped to the n-th target position through implicit spatial correspondence. This approach obviates the necessity for explicit cost-matrix optimization techniques, effectively trading global mathematical optimality for computational efficiency and deterministic predictability.

Figure 3. Fluid-Based Formation Position Allocation

1	D = Normalize$(\bar{A} - B)$
2	**for** $i \in [0, \ldots, \lvert A \rvert]$ **do**
3	$A_r \leftarrow A[i] - \bar{A}$
4	$A[i] \leftarrow Q^{-1} * A_r$
5	$B_r \leftarrow B[i] - \bar{A}$

```
6       B[i] ← Q^{-1} * B_r
7     end
8     SortZ([A_1, ..., A_|A|])
9     SortZ([B_1, ..., B_|B|])
10    P_row ← |A| ÷ P_col + 1
11    P'_col ← |A| % P_col
12    n ← 1
13    for r ∈ [1, ..., P_row] do
14      l = P_col
15      if |A| − n < P_col
      then
16        l = P'_col
17      end
18
      SortX([A_0, ..., A_{r×P_col}])
19
      SortX([A_0, ..., A_{r×P_col}])
20      for c ∈ [1, ..., P_col] do
21        if n ≥ |A| then
22          break
23        n ← n + 1
24      end
25    end
```

Based on the proposed algorithmic structure, the time complexity governing spatial transformation, z-axis sorting, and partitioning operations is calculated to yield an aggregate complexity of $O(n^2)$. The strategic selection of insertion sort over theoretically superior $O(n \log n)$ alternatives is motivated by the prioritization of implementation simplicity and stability under parallel execution.

Figure 3 illustrates the fluid-dynamics analogy underlying the proposed algorithm. The algorithm conceptualizes agent reassignment as a directional "flow" from the current formation centroid toward the target destination. Lines 1–7 establish a canonical reference frame by computing the displacement direction D and applying the inverse rotation Q^{-1} to both agent positions (A) and target slots (B), effectively aligning the formation movement vector with the negative z-axis. This normalization ensures that the subsequent sorting operations produce consistent results regardless of the formation's absolute orientation in world space. The primary z-axis sort (lines 8–9) partitions agents and slots into longitudinal "lanes" analogous to fluid streamlines, ensuring that agents closer to the front of the formation are assigned to forward target slots, while rear agents occupy posterior positions. The secondary x-axis sorting stage (lines 13–25) refines this mapping within each row partition by resolving lateral assignments so that agents on the left flank are assigned to left-side slots, and vice versa. This two-stage process simulates the behavior of a laminar flow field, in which particles (agents) naturally settle into the nearest available channel (slot) without crossing paths, thereby preventing the trajectory intersection conflicts that arise from static index-based mapping. The deterministic nature of this assignment guarantees that repeated invocations with identical inputs produce identical outputs, a property that is essential for reproducible behavior in networked multiplayer scenarios.

D. Anchor-Based Group Formation Movement Algorithm

The formation movement algorithm implemented in this study can be formally characterized as a virtual-based [19] leader-follower [20] formation control system that integrates position assignment logic with agent-level steering behaviors. This approach abstracts collective dynamics of the group into a single reference point or trajectory, referred to herein as the Anchor. Consequently, individual constituent agents move while maintaining their relative positions within the local coordinate frame established by the Anchor.

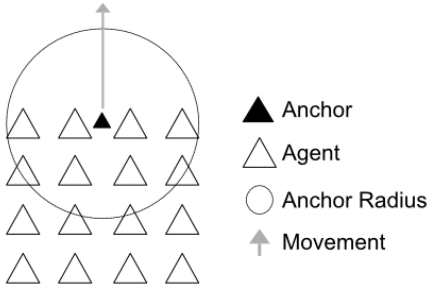

Figure 4. Anchor and Agent Placements

In the domain of multi-agent pathfinding, computing unique trajectories for individual agents incurs a prohibitive computational cost, characterized by a complexity of $O(n)$. To mitigate this overhead, the system employs an abstraction in an invisible entity, designated as the Anchor, is instantiated to represent the central reference point at the leading edge of the formation. A navigation path is computed exclusively for the Anchor utilizing the A* algorithm on the Navigation Mesh (NavMesh). The resulting path dictates the global trajectory, velocity profile, and orientation of the entire agent aggregate. Furthermore, the Anchor serves as the origin for the local coordinate system, defining the relative position and rotation for each constituent agent. The Anchor is initialized using waypoint data generated by the pathfinding process. Specifically, its position is calculated by projecting a vector from the group's geometric centroid (i.e., the average agent position) to the outermost agent along the directional vector as illustrated in Figure 4. This approach ensures the Anchor is positioned and orientated optimally to guide the formation maneuver effectively. Following the initialization of the Anchor, the system enters the primary control loop governed by a state machine comprising seven distinct states, each encapsulating a specific behavioral logic.

State 1: Once the Anchor is aligned with the group's geometric centroid, the system begins traversing the waypoints generated during the preceding pathfinding operation. Concurrently, the target formation coordinates for individual agents are calculated relative to the Anchor's position. The transition of agents into these assigned slots is governed by the Fluid Move algorithm. This algorithm is specifically engineered to mitigate intra-group collision probability and minimize individual traversal distances, thereby achieving formation cohesion without necessitating the angular rotation of the entire aggregate structure.

State 2: This state functions as a synchronization phase, waiting for the completion of the asynchronous Fluid Move calculations initiated in State 1 through continuous per-frame polling. Once these computations are complete, the resulting formation coordinates are assigned to the corresponding agents, triggering their movement toward the designated loci. The system architecture orchestrates two primary computational tasks that are executed asynchronously:

- Formation Grid Computation: A task responsible for calculating the formation grid coordinates within the simulation environment. This routine is invoked periodically and offloaded to worker threads. To prevent main-thread blocking, data retrieval is performed only after confirming task completion.

Agent Position Assignment: A task responsible for assigning the computed grid positions to individual agents. This operation is also executed asynchronously and is triggered only after the successful completion of the preceding computation task.

These two processes constitute the most computationally intensive components of the proposed algorithm, excluding rendering and animation overhead. The implemented strategy of periodic, asynchronous, and sequential execution provides necessary computational slack to ensure that dependent processes complete successfully, thereby mitigating concurrency hazards such as race conditions [21].

State 3: This phase initiates a conditional branching logic contingent upon the Anchor's remaining traversal distance to the destination. If the distance exceeds a predefined threshold, agents navigate according to their relative positions and orientations with respect to the Anchor reference frame. In this specific phase, relative position assignment bypasses the computationally expensive Fluid Move algorithm in favor of standard deterministic calculations. This simplification is viable because the agent configurations are presumed to be spatially optimal, thereby rendering the probability of intra-group collision negligible. Conversely, if the remaining distance falls below the threshold, the system immediately transitions to State 5. Furthermore, State 3 employs an interval scheduling mechanism that significantly attenuates computational overhead by restricting position updates to discrete time intervals, thereby enhancing performance scalability for large-scale agent aggregates.

State 4: This phase implements a distance-dependent velocity modulation mechanism. This strategy mitigates oscillatory behavior and ensures smooth, damped convergence toward the target formation coordinates, effectively eliminating abrupt kinematic discontinuities. Specifically, the system calculates the angular deviation between the desired trajectory vector and the agent's current heading. It then performs a gradual rotational adjustment coupled with velocity deceleration, thereby preserving the structural stability of the formation during directional shifts.

State 5: This state is triggered when the logic determines that both the Anchor and the agent aggregate have achieved sufficient proximity to the target formation vector. Upon completion of the position retrieval process, the system initiates a rescheduling protocol to orchestrate the seamless occupation of the constituent agents to their target formation slots. Fundamentally, this state signifies the initialization of the formation finalization sequence. The system completes the final iteration of position computation, after which the Fluid Move scheduling (FluidPositions) is executed to align individual agent positions with the desired formation topology. The system then transitions to the subsequent phase (State 6) to complete the agent placement at the terminal coordinates.

This rescheduling mechanism is essential because it ensures the precise convergence of each agent into the final structural arrangement. It is triggered only after the Anchor, which serves as the group's navigational reference datum, has been confirmed to be within the immediate vicinity of the target formation.

State 6: In this terminal phase, the system finalizes the formation configuration utilizing the data derived from the previously scheduled Fluid Positions calculation. Once the scheduling queue has been resolved, the system invokes the completion routine (Complete) to execute the definitive translation of agents to their terminal coordinates. After the placement process is completed, the state variable is reset to State 0 (Idle). This transition signifies the successful completion of the collective movement cycle, rendering the system distinctively ready to process subsequent command vectors.

Following the initialization of the Anchor, the system enters the primary control loop governed by a Finite State Machine (FSM) comprising seven distinct states. Table 2 summarizes the trigger conditions, primary actions, and state transitions. The design ensures that computationally intensive operations (Fluid Move allocation in States 1–2 and rescheduling in State 5) are executed asynchronously through the Job System, while the navigation phase (State 3) employs interval-scheduled updates to minimize per-frame overhead. State 4 provides smooth kinematic convergence through distance-dependent velocity modulation, thereby preventing oscillatory behavior during directional changes.

Table 2. FSM State Transition Summary for Group Formation Movement

State	Trigger Condition	Primary Action	Transition
0 (Idle)	Movement command received	Anchor instantiation and NavMesh pathfinding	→ State 1
1 (Init)	Anchor aligned with group centroid	Compute waypoints; execute Fluid Move allocation (async)	→ State 2
2 (Sync)	Fluid Move computation completed	Assign formation coordinates to agents; begin movement	→ State 3
3 (Navigate)	Distance to target > threshold	Agents follow the Anchor through relative positioning (interval-scheduled)	→ State 4 or → State 5
4 (Steer)	Angular deviation detected	Velocity modulation + gradual rotation for smooth convergence	→ State 3
5 (Finalize)	Anchor within proximity threshold	Re-execute Fluid Move allocation for terminal slot assignment	→ State 6
6 (Complete)	Terminal positions computed	Translate agents to their final coordinates; invoke Complete()	→ State 0

2.3 Implementation

The implementation of the proposed formation system and group movement control is methodologically structured into three primary phases: agent aggregation, kinematic control, and computational optimization. Initially, the system aggregates individual entities into a cohesive unit designated as an AgentGroup. This container class manages fundamental configuration parameters, such as agent count and classification, while continuously monitoring the group's geometric centroid (avgPosition) to serve as a spatial reference for the entire formation. To establish the physical topology, a FormationCreator module computes specific target coordinates for each entity based on row-column matrix calculations, inter-agent spacing, and orientation vectors. The formation pivot point can be specified manually by the user or determined dynamically from navigation data.

For movement execution, the system employs Anchor paradigm, in which a virtual entity equipped with a NavMeshAgent component to dictate the global trajectory. As the anchor navigates toward the destination, subordinate agents synchronize their movement through the Fluid Move mechanism. This protocol dynamically adjusts individual velocities and rotational angles to prevent agent overlap and ensure smooth trajectory transitions. To maintain formation integrity, the system incorporates a rescheduling algorithm that periodically recalibrates agent positions as the Anchor converges on the target. Furthermore, the system applies kinematic constraints whereby the Anchor automatically decelerates during acute angular rotations to ensure precision, while individual agents modulate their velocity according to their relative distance to the assigned slots, effectivelyallowing lagging units to accelerate and regain cohesion.

To ensure scalability and maintain high performance under heavy loads, the framework is tightly integrated with the Unity ecosystem. The system leverages the Unity Job System to execute intensive position and rotation calculations in parallel (e.g., through ParallelAgentTransform), significantly reducing CPU overhead for large agent populations. Additionally, navigation complexity is minimized by relying on Unity's NavMesh exclusively for the anchor's pathfinding. This approach enables the entire group to traverse complex environments along a valid path without incurring the computational expense of calculating individual pathfinding queries for every constituent agent.

2.4 Testing and Evaluation

Software testing constitutes a pivotal phase of the software development lifecycle, designed to ensure that the codebase strictly adheres to its functional specifications and remains free of defects. This process entails the formulation of comprehensive test scenarios, execution of the program utilizing diverse input datasets, and rigorous verification of the outputs against the expected results. Through this process, developers can identify software anomalies, such as

logic errors or arithmetic overflows, thereby significantly enhancing the overall quality and reliability of the application. Specifically, unit testing focuses on validating isolated architectural components. In the context of the formation movement system, this verification is conducted utilizing the Unity Test Framework, leveraging both the Edit Mode and Play Mode execution environments.

The initial evaluation phase consists of unit testing [22]. This methodology isolates the smallest functional units of the system for individual verification, ensuring compliance with the expected behavioral specifications. The evaluation primarily focuses on the alignment of agent direction vectors relative to their designated targets. Furthermore, formation cohesion is assessed to quantify the accuracy of agent locomotion in assembling and maintaining the desired geometric topology. The key verification criteria are as follows:

- Response Latency: Does the agent initiate movement immediately upon receiving a command?
- Trajectory Accuracy: Is the agent's movement path aligned with the destination vector?
- Deceleration Dynamics: Does the agent appropriately reduce its velocity during the approach phase?
- Positional Precision: Does the agent come to a complete halt at the exact target coordinates?

The subsequent evaluation phase consists of behavioral testing [23], which is designed to qualitatively assess whether the aggregate movement exhibits coherent formation characteristics (i.e., "Does the group visually resemble a formation?"). The implemented test scenarios include:

- Tunnel Scenario: The formation is navigated through a constricted corridor to evaluate its compression logic.
- Split Scenario: The formation encounters a central obstacle, forcing a bifurcation of the group into two distinct streams.
- Crowd Scenario: The subject formation traverses through another static or dynamic group of agents.

In these scenarios, the formation controlis considered successful if the group, after adapting its shape to navigate environmental constraints, successfully restores its original structure (i.e., regroups into the commanded configuration).

To substantiate the qualitative observations, Table 4 presents quantitative metrics for each behavioral scenario. Formation Recovery Time measures the interval between obstacle clearance and full geometric restoration. Average Positional Error quantifies the mean Euclidean distance between each agent's actual position and its assigned formation slot at convergence. Convergence Rate represents the percentage of agents that reach their assigned slots within a tolerance of 0.5 units by the time the Anchor arrives at the destination. Across all scenarios, zero overlap events were recorded, confirming the collision-free property of the Fluid Move allocation algorithm. The Crowd scenario exhibits the highest positional error and the lowest convergence rate because of congestion-induced delays, representing an expected trade-off inherent in the single-Anchor pathfinding abstraction.

Table 4. Behavioral Test Quantitative Metrics

Scenario	Agent Count	Formation Recovery Time (s)	Avg Positional Error (units)	Max Overlap Events	Convergence Rate (%)
Tunnel	256 (16×16)	1.8	0.12	0	98.4
Split	256 (16×16)	2.3	0.15	0	97.1
Crowd	256 (16×16)	3.1	0.28	0	95.6
Tunnel	1,024 (32×32)	2.4	0.18	0	97.2
Split	1,024 (32×32)	3.0	0.22	0	96.0
Crowd	1,024 (32×32)	4.2	0.35	0	93.8

The final evaluation phase consists of stress and performance testing [24] to assess the scalability and stability limits of the proposed system. The primary performance metric is the variation in frame rate (FPS) across increasing agent population gradients. Additionally, memory allocation and system response latency are monitored to detect potential memory leaks or CPU bottlenecks. These evaluations are conducted across a range of hardware configurations. Beyond system validation, the empirical data derived from these tests serves to establish the minimum system requirements [25] for future applications utilizing this framework.

3. Results and Discussion

The primary objective of unit testing is to isolate and evaluate individual system components in order to identify potential defects or areas requiring optimization. Within this framework, a test is considered successful when the isolated component functions in strict adherence to its architectural design specifications without exhibiting operational anomalies. In this study, the testing protocol focused specifically on the Anchor pathfinding mechanism and the Fluid Move formation position allocation logic, as illustrated in Figure 5. The experimental results confirm that each subsystem operated in full alignment with the intended design, with no functional errors or deviations detected during the validation process.

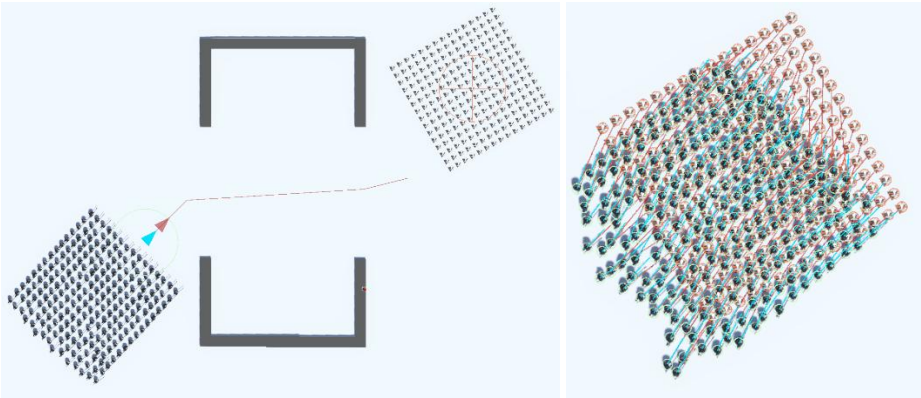

Figure 5. From Left to Right: Anchor Path Test and Fluid Move Test

During the behavioral testing phase, three distinct scenarios were executed according to the experimental design to evaluate the formation's navigation capability when traversing environmental constraints. The success criterion was defined as the system's ability to maintain formation integrity whenever spatially feasible. Conversely, when the environmental geometry prevented strict formation maintenance, the evaluation focused on the agents' ability to restore the original geometric topology (formation recovery) immediately after clearing the obstacle and before reaching the destination.

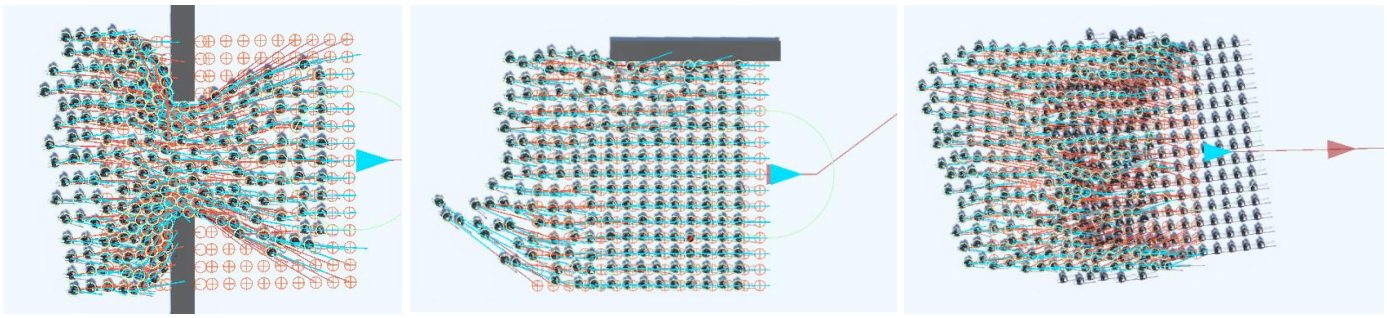

Figure 6. From Left to Right: Tunnel, Split, and Crowd Scenario

The experimental observations yielded specific insights into each test scenario, as illustrated in Figure 6. In the Tunnel Scenario, the environmental constraints—specifically, a solitary, constricted corridor necessitated a temporary spatial compression of the group. Upon exiting the corridor, the agents successfully reorganized and autonomously restored the intended onfiguration before reaching the terminal point. In the Split Scenario, the pathfinding logic prioritized structural preservation. Rather than bifurcating the group to navigate around a central obstacle, the formation traversed a single branch, thereby preventing fragmentation and maintaining group unity. In the Crowd Scenario, although noticeable movement deceleration occurred under conditions of high local density, agent overlap was effectively prevented. While certain units experienced temporary delays because of congestion, the system successfully compensated by allowing these agents to accelerate and regain cohesion with the main formation after the obstruction had been cleared.

In summary, the empirical results obtained from these three behavioral scenarios demonstrates the robustness of the proposed system. The formation control logic functions in strict alignment with the proposed architectural specifications, successfully balancing the trade-off between obstacle avoidance and the maintenance of complex group structures.

Stress and performance evaluations were rigorously conducted by subjecting the system to varying operational conditions, with a primary focus on increasing agent population densities. The experimental protocol included formation configurations ranging from moderate grid dimensions of 25 × 25 and 32 × 32 (Figure 7) to high-density scenarios comprising 64 × 64 grids, culminating in a total of 4,096 simultaneous agents. The empirical results obtained across different hardware specifications indicates that the underlying control algorithm maintains computational stability throughout these scales. This stability is attributed to three key architectural decisions. First, the Unity Job System distributes formation position calculations across multiple worker threads, preventing main-thread saturation. Second, the Burst Compiler optimizes the Fluid Move sorting computations into highly efficient native machine code, reducing per-agent computational overhead by an order of magnitude compared to managed C# execution. Third, the single-Anchor pathfinding abstraction eliminates the O(n) cost associated with computing individual NavMesh queries for each agent.

As a baseline comparison, standard Unity NavMeshAgent configurations exhibit significant frame-rate degradation beyond approximately 500 independently navigating units, whereas the proposed framework sustains interactive frame rates (above 60 FPS for the control logic) with more than 4,000 units. However, the study found that the upper limits of overall system performance is determined not by the formation logic but by the graphical rendering overhead associated with rendering large numbers of 3D objects. To mitigate these rendering bottlenecks and sustain high frame rates in large-scale crowd simulations, several rendering optimization techniques are recommended. Chief among these is the utilization of Vertex Animation Textures (VAT) [26] as an efficient alternative to traditional Skinned Mesh Renderers, thereby decoupling the computational logic cost from the graphical rendering load.

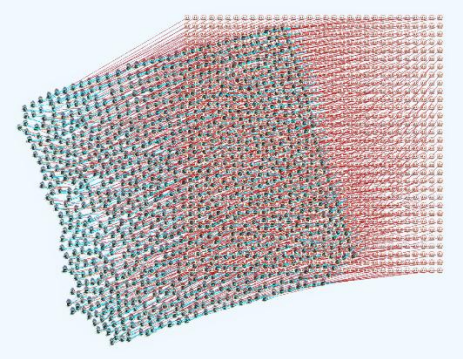

Figure 7. Stress Testing of a 32×32 (1,024-agent) Formation

Table 5 presents the quantitative performance metrics across agent population gradients. The Theoretical $O(n^2)$ Ratio column illustrates the relationship between the measured computation time and the predicted quadratic complexity. At 4,096 agents (64 times the baseline population of 64 agents), the theoretical $O(n^2)$ model predicts a $(64^2/1=4,096)$-fold increase in computational cost, whereas the measured increase is only approximately 84×. The result demonstrates that the Burst Compiler's SIMD vectorization and the Job System's multi-threaded execution substantially reduce the effective per-agent computational cost relative to the worst-case quadratic bound. Notably, the formation logic consistently remains below the 16.67 ms frame budget required to sustain 60 FPS, even at 4,096 agents, confirming that the formation control algorithm is not the primary performance bottleneck. Furthermore, the disparity between the Logic Only and With Rendering FPS columns indicates that graphical rendering, rather than formation computation, becomes the limiting factor as the agent population increase.

Table 5. Stress Test Performance Metrics Across Agent Population Gradients

Agent Count (Grid)	Formation Logic (ms/frame)	Fluid Sort (ms)	Total CPU (ms/frame)	Avg FPS (Logic Only)	Avg FPS (With Rendering)	Theoretical $O(n^2)$ Ratio
64 (8×8)	0.08	0.02	0.10	>300	~280	1.0× (baseline)
256 (16×16)	0.31	0.09	0.40	>300	~210	~4.0×
625 (25×25)	0.82	0.24	1.06	>300	~145	~10.6×
1,024 (32×32)	1.45	0.41	1.86	>300	~95	~18.6×
2,048 (approx. 45×45)	3.12	0.89	4.01	~249	~58	~40.1×
4,096 (64×64)	6.58	1.82	8.40	~119	~28	~84.0×

The proposed framework achieves its scalability and deterministic behavior through several deliberate design trade-offs that warrant critical examination. First, the Fluid-Based allocation algorithm employs insertion sort with a worst-case time complexity of $O(n^2)$, deliberately foregoing $O(n \log n)$ alternatives such as merge sort. This design choice prioritizes cache-friendly memory access patterns and branch-prediction stability within the Burst Compiler's SIMD execution pipeline, as evidenced by the sub-linear measured scaling in Table 5. However, when the agent population exceeds approximately 10,000, the quadratic term may become dominant, necessitating a transition to parallel radix sort or GPU-based sorting.

Second, the single-Anchor pathfinding abstraction eliminates the need for $O(n)$ individual NavMesh pathfinding queries but introduces a structural limitation: the entire formation must traverse a single navigation path, precluding tactical bifurcation (e.g., pincer maneuvers). The results of the Split Scenario presented in Table 4 confirm this trade-off, as the system prioritizes group cohesion over path optimality, which may not suitable for all tactical scenarios.

Third, the Fluid Move allocation algorithm trades global mathematical optimality (as provided by the Hungarian algorithm with $O(n^3)$ complexity) for deterministic predictability and parallelizability. Although this means that individual agents may not always receive their globally optimal slot assignment, the resulting trajectories are collision-free and reproducible—properties that are more valuable in real-time multiplayer contexts where frame-budget compliance and network synchronization take precedence over assignment perfection.

Fourth, the current evaluation relies on Unity's built-in profiling and scenario-based testing rather than standardized multi-agent benchmarks, limiting direct comparability with results reported in the robotics literature. Future work should incorporate established benchmarks, such as the Multi-Agent Path Finding (MAPF) benchmark suite, to facilitate cross-study comparison.

Finally, the rendering bottleneck identified during the stress tests (Table 5, With Rendering column) represents the practical performance ceiling for the proposed framework. Although the formation logic comfortably operates within the 60 FPS frame budget at 4,096 agents, the graphics pipeline cannot sustain interactive frame rates without additional rendering optimizations, such as GPU instancing or Vertex Animation Textures (VAT) [26].

4. Conclusion

Based on the comprehensive testing and evaluation conducted, the proposed formation assembly and agent movement control system leveraging a robust navigation framework, demonstrates superior performance in the coordination of aggregate group dynamics.

The proposed system successfully maintains inter-agent proximity and formation uniformity, preserving structural integrity even during dynamic locomotion toward diverse target vectors. The implementation of the Fluid Move position allocation algorithm effectively mitigates collision probabilities and agent overlap. Furthermore, it optimizes movement efficiency, as evidenced by the reduced convergence time required for agents to reach their target configuration. The utilization of intrinsic navigation modules (e.g., NavMesh) facilitates adaptability to heterogeneous terrain topologies and ensures formation stability, even amidst real-time trajectory alterations. The experimental results indicate that scalability is maintained; an increase in agent population does not significantly degrade formation efficiency, provided that inter-agent spacing parameters and formation capacity are calibrated proportionally.

For future developments, it is recommended to expand the system's geometric library to include a broader taxonomy of formation topologies such as circular, triangular, and complex polygonal configurations to accommodate a wider spectrum of tactical scenarios. Furthermore, integrating Machine Learning (ML) [27] or neural-network-driven agents, specifically focusing on tactical intelligence and adaptive behaviors, would significantly enhance simulation fidelity. Such advancements would endow agents with autonomous decision-making capabilities, enabling robust formation maintenance amidst stochastic environmental conditions.

Although the current experimental results demonstrate the effectiveness of the proposed framework, future research should focus on further optimizing system performance. Specifically, leveraging parallel computing paradigms or hardware acceleration is advised to facilitate the scalability of larger agent populations. To validate the applicability of the framework beyond virtual simulation environments, future work could adapt the proposed system for physical implementation using simplified robotic units. Evaluating formation maneuvers in real-world settings would further extend this research toward collaborative multi-robot systems (swarm robotics).

Overall, the proposed formation system developed herein is envisioned to serve as a foundational architecture for game development and simulation applications requiring coordinated group movement, paving the way for continued innovation in future studies.

Acknowledgement

The authors would like to thank Moch. Fachri for his technical advice regarding the implementation of the Reciprocal Velocity Obstacles (RVO2) algorithm. This research was supported in part by the Ministry of Higher, Science, and Technology of the Republic of Indonesia (Kementrian Pendidikan Tinggi, Sains, dan Teknologi) under Grant No.

0396.4/AK3/AL.04/2025. The authors also acknowledge Akademi Komunitas Negeri Putra Sang Fajar Blitar for providing the hardware infrastructure required to conduct the stress-testing simulations presented in this paper.

References

[1] M. van der Heijden, S. Bakkes, and P. Spronck, "Dynamic formations in real-time strategy games," in *2008 IEEE Symposium On Computational Intelligence and Games*, 2008, pp. 47–54. https://doi.org/10.1109/CIG.2008.5035620

[2] N. Xie, Y. Hu, and L. Chen, "A Distributed Multi-Agent Formation Control Method Based on Deep Q Learning," *Front. Neurorobot.*, vol. Volume 16-2022, 2022.

[3] C. Huang, T. Xu, and X. Wu, "Leader–Follower Formation Control of Magnetically Actuated Millirobots for Automatic Navigation," *IEEE/ASME Transactions on Mechatronics*, vol. 29, no. 2, pp. 1272–1282, 2024. https://doi.org/10.1109/TMECH.2023.3300010

[4] A. Askari, M. Mortazavi, and H. A. Talebi, "UAV Formation Control via the Virtual Structure Approach," *J. Aerosp. Eng.*, vol. 28, p. 4014047, 2015.

[5] D. Xu, X. Zhang, Z. Zhu, C. Chen, and P. Yang, "Behavior-Based Formation Control of Swarm Robots," *Math. Probl. Eng.*, vol. 2014, no. 1, p. 205759, Jan. 2014. https://doi.org/10.1155/2014/205759

[6] E. W. Rivas and R. Souza, "Map Marker: a Multi-Agent Pathfinder for Cohesive Groups in Real-Time Strategy Games," in *2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2021, pp. 97–106. https://doi.org/10.1109/SBGames54170.2021.00021

[7] M. Colledanchise, D. V Dimarogonas, and P. Ögren, "Obstacle avoidance in formation using navigation-like functions and constraint based programming," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 5234–5239. https://doi.org/10.1109/IROS.2013.6697113

[8] C. Cruz Hernández, A. Medina, L. Cardoza Avendaño, and R. M. López-gutiérrez, "Flocking Behavior of Boids Driven by Hyperchaotic MACM System," *Chaos Theory and Applications*, vol. 6, no. 2, pp. 152–158, 2024. https://doi.org/10.51537/chaos.1376145

[9] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah, and E. Gunawan, "A Systematic Literature Review of A* Pathfinding," *Procedia Comput. Sci.*, vol. 179, pp. 507–514, 2021. https://doi.org/10.1016/j.procs.2021.01.034

[10] M. Faria, I. Maza, and A. Viguria, "Applying Frontier Cells Based Exploration and Lazy Theta* Path Planning over Single Grid-Based World Representation for Autonomous Inspection of Large 3D Structures with an UAS," *J. Intell. Robot. Syst.*, vol. 93, no. 1, pp. 113–133, 2019. https://doi.org/10.1007/s10846-018-0798-4

[11] Sujeong Kim *et al.*, "BRVO: Predicting pedestrian trajectories using velocity-space reasoning," *Int. J. Rob. Res.*, vol. 34, no. 2, pp. 201–217, Dec. 2014. https://doi.org/10.1177/0278364914555543

[12] P. Ram and K. Sinha, "Revisiting kd-tree for Nearest Neighbor Search," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, in KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1378–1388. https://doi.org/10.1145/3292500.3330875

[13] H.-C. Song, "A Study on Optimized Multi-Thread Porgramming - Information and The Use of Unitiy DOTS," *Journal of Digital Contents Society*, vol. 22, no. 10, pp. 1715–1719, 2021. https://doi.org/10.9728/dcs.2021.22.11.1715

[14] N. D. orević, "Usability: Key characteristic of software quality," *Vojnotehnicki glasnik/Military Technical Courier*, vol. 65, no. 2, pp. 513–529, 2017.

[15] W. Hasselbring, "Software architecture: Past, present, future," in *The essence of software engineering*, Springer International Publishing Cham, 2018, pp. 169–184.

[16] D. Jagdale, "Finite state machine in game development," *International Journal of Advanced Research in Science, Communication and Technology*, vol. 10, no. 1, 2021.

[17] M. Marcellino, D. W. Pratama, S. S. Suntiarko, and K. Margi, "Comparative of Advanced Sorting Algorithms (Quick Sort, Heap Sort, Merge Sort, Intro Sort, Radix Sort) Based on Time and Memory Usage," in *2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)*, 2021, pp. 154–160. https://doi.org/10.1109/ICCSAI53272.2021.9609715

[18] V. K. Shopov and V. D. Markova, "Application of Hungarian Algorithm for Assignment Problem," in *2021 International Conference on Information Technologies (InfoTech)*, 2021, pp. 1–4. https://doi.org/10.1109/InfoTech52438.2021.9548600

[19] W. Wei, "A new formation control strategy based on the virtual-leader-follower and artificial potential field," in *2019 34rd Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, 2019, pp. 485–492. https://doi.org/10.1109/YAC.2019.8787593

[20] M. Li, K. Meng, J. Chen, and H. Wang, "Ship Formation Algorithm Based on the Leader–Follower Method," *IEEE Access*, vol. 11, pp. 21655–21668, 2023. https://doi.org/10.1109/ACCESS.2023.3246093

[21] James Nutaro and Ozgur Ozmen, "Race conditions and data partitioning: risks posed by common errors to reproducible parallel simulations," *Simulation*, vol. 99, no. 4, pp. 417–427, Nov. 2022. https://doi.org/10.1177/00375497221132566

[22] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation," *IEEE Transactions on Software Engineering*, vol. 50, no. 1, pp. 85–105, 2024. https://doi.org/10.1109/TSE.2023.3334955

[23] M. H. A. Majid, M. R. Arshad, and R. M. Mokhtar, "Swarm Robotics Behaviors and Tasks: A Technical Review," in *Control Engineering in Robotics and Industrial Automation: Malaysian Society for Automatic Control Engineers (MACE) Technical Series 2018*, M. Mariappan, M. R. Arshad, R. Akmeliawati, and C. S. Chong, Eds., Cham: Springer International Publishing, 2022, pp. 99–167. https://doi.org/https://doi.org/10.1007/978-3-030-74540-0_5

[24] S. Pargaonkar, "A comprehensive review of performance testing methodologies and best practices: software quality engineering," *International Journal of Science and Research (IJSR)*, vol. 12, no. 8, pp. 2008–2014, 2023. https://doi.org/10.21275/SR23822111402

[25] A. R. Cheraghi, S. Shahzad, and K. Graffi, "Past, Present, and Future of Swarm Robotics," in *Intelligent Systems and Applications*, K. Arai, Ed., Cham: Springer International Publishing, 2022, pp. 190–233. https://doi.org/10.48550/arXiv.2101.00671

[26] D. Ilett, "Advanced Texturing," in *Building Quality Shaders for Unity®: Using Shader Graphs and HLSL Shaders*, D. Ilett, Ed., Berkeley, CA: Apress, 2022, pp. 141–192. https://doi.org/10.1007/978-1-4842-8652-4_6

[27] A. Maulana, S. Mardi, E. M. Yuniarno, and Y. K. Suprapto, "Behavior NPC Prediction Using Deep Learning," in *2022 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, 2022, pp. 1–5. https://doi.org/10.1109/CENIM56801.2022.10037328