


# AlchemyNavigation: Comparative evaluation of a dynamic navigation system for unity virtual worlds

Maciej Kempny<sup>1</sup>, Marcin Barszcz<sup>1\*</sup> , Krzysztof Dziedzic<sup>1</sup>,  
Maria Skublewska-Paszkowska<sup>1</sup>, Pawel Powroznik<sup>1</sup>

<sup>1</sup> Lublin University of Technology, Nadbystrzycka 38D 20-618 Lublin, Poland

\* Corresponding author's e-mail: m.barszcz@pollub.pl

## ABSTRACT

Navigation systems are fundamental components in modern game engines, facilitating the realistic and efficient movement of autonomous agents within virtual environments. This paper presents an evaluation of AlchemyNavigation, a custom navigation system developed for the Unity engine, designed to address limitations of standard solutions such as Unity's NavMesh. This work introduces novel contributions to the field of dynamic navigation systems by proposing a triangle-based real-time modification approach that advances beyond traditional static navigation meshes, offering significant implications for procedurally generated environments and interactive virtual worlds. A series of controlled experiments were conducted to compare both systems in terms of computational performance, scalability, and flexibility. The evaluation focused on key performance indicators, including frame time, CPU utilization, and memory consumption, across multiple hardware configurations and varying agent densities. The results demonstrate that NavMesh maintains superior stability and lower computational overhead in large-scale simulations. AlchemyNavigation, in turn, offers enhanced adaptability and runtime modification capabilities, making it well suited for procedurally generated and dynamic environments. The study confirms that custom navigation systems can complement or even outperform native solutions by providing additional features tailored to specific project requirements. These findings add to current studies on AI-driven navigation frameworks in game development and immersive virtual worlds.

**Keywords:** Unity Engine, NavMesh, AlchemyNavigation, pathfinding, agent-based simulation, dynamic navigation.

## INTRODUCTION

Over the years, navigation systems have constituted a vital element of artificial intelligence in computer games. These systems enable realistic and efficient movement of agents within virtual game environments. Standard solutions, such as the NavMesh available in the Unity engine, despite their widespread popularity and extensive application, exhibit several significant limitations. Studies presented in [1, 2] emphasize that NavMesh encounters considerable challenges in dynamically modifying the navigation space and flexibly defining non-standard agent behaviors. Developing high-quality navigation systems is crucial for enhancing immersion and realism in modern computer games, particularly in the

context of open-world games and applications of virtual (VR) and augmented (AR) reality. This also contributes to a better understanding of agent behaviors in complex environments. Therefore, this study introduces methods enabling the development of an alternative navigation system that integrates both standard methods and innovative approaches to navigation space management [3-5]. The research focuses on two navigation systems available for use in projects developed using the Unity engine: the built-in NavMesh system and a proprietary solution called AlchemyNavigation.

The significant contributions of this study can be summarized as follows:

- Development of the proprietary AlchemyNavigation system for Unity, enabling real-time creation and modification of navigation space

and improving flexibility in environment management.

- Comprehensive comparative analysis with the native NavMesh system under identical test scenarios.
- Implementation of a unified measurement methodology using Unity profiling tools to monitor key indicators such as frame time, CPU usage, and RAM consumption.
- Creation of a dedicated Unity test application with a complex multi-level base scene.

The primary objective of this article is to examine the impact of navigation systems on application performance within the Unity engine. The study emphasises on comparing the customization and adaptability capabilities of navigation systems, with special consideration given to procedurally generated and dynamically changing game worlds. The scientific novelty of this study lies in introducing *AlchemyNavigation* as a dynamic navigation system for Unity that supports real-time triangle-based modifications of the navigation space. Unlike the widely used NavMesh, which is limited to static or semi-dynamic environments, the proposed solution enables flexible runtime adaptation in procedurally generated and dynamically evolving virtual worlds. Furthermore, the study provides a reproducible and comprehensive comparative evaluation of both systems under controlled experimental conditions, using a unified performance measurement methodology. This combination of methodological rigor and system innovation fills a clear research gap in the domain of AI-driven navigation frameworks for interactive 3D environments.

## BACKGROUND STUDY

The exploration of navigation systems within the Unity engine has become increasingly important in both game development and research contexts. A central concept is the NavMesh, which represents traversable space in the form of polygons. Zikky [6] reviewed its effectiveness in 3D environments, emphasizing why it became the mainstream choice for developers: the method is straightforward to implement, efficient in standard pathfinding tasks, and well-integrated into Unity. By simplifying the representation of the environment, NavMesh allows agents to move realistically even in complex virtual spaces, making

it a foundation for both static and semi-dynamic simulations. Nevertheless, researchers have actively explored alternatives to NavMesh. Alonso et al. [3] demonstrated how deep reinforcement learning (RL) could be applied to navigation in large 3D Unity maps. Their work showed that RL-trained agents are capable of learning optimal navigation strategies and adapting to changes in the environment, often surpassing the performance of classical algorithms. This approach is particularly valuable in dynamic scenarios, where environmental conditions cannot be pre-defined at design time. Similarly, Pyke and Stark [7] examined navigation in the context of unmanned aerial vehicles (UAVs), testing particle swarm optimization (PSO) and D\* Lite algorithms within Unity. Their studies confirmed that the choice of algorithm strongly impacts pathfinding efficiency, computational cost, and memory usage, underscoring the trade-offs developers must consider when selecting navigation strategies for dynamic and complex environments.

Another important issue concerns the performance optimization within Unity itself. Lee [8] analyzed the engine's limitations, highlighting challenges when rendering large numbers of dynamic objects. These performance bottlenecks directly influence the responsiveness of navigation systems, as heavy rendering workloads reduce available CPU resources for pathfinding tasks. Strategies proposed in that study, such as optimizing object hierarchies and reducing unnecessary updates, are relevant for ensuring efficient real-time navigation. Complementary findings were presented by Messaoudi et al. [9], who examined CPU and GPU consumption in Unity-based applications. Their results emphasized the importance of resource management, showing how both processor and graphics load can affect the stability of navigation when deployed on different devices.

The use of Unity in augmented reality and intelligent navigation research has further broadened the scope of studies. Fajrianti et al. [10] developed the Indoor Navigation System Using Unity and Smartphone, which applied simultaneous localization and mapping to provide real-time user guidance indoors, where GPS is unreliable. This demonstrates Unity's adaptability beyond entertainment, extending to assistive technologies. Meanwhile, Ma et al. [11] proposed NavTuner, a system that learns scene-sensitive navigation policies. Their work demonstrated how adaptive parameters allow navigation systems to dynamically adjust performance to environmental conditions,

improving efficiency compared to static approaches. This adaptability reflects a wider trend of integrating machine learning into navigation systems.

Hybrid planning methods is another study scope. Zhou et al. [12] investigated combining global planners with local strategies in robotic navigation. Their study showed that such hybrid solutions improve overall adaptability and efficiency, especially in environments with frequent local obstacles. Juliani et al. [13] further expanded on this by presenting Unity as a platform for intelligent agents. They showed how policies learned in simulation can be successfully transferred to real-world contexts, bridging the gap between virtual testing and physical deployment.

Beyond algorithms and learning methods, Unity has been employed in diverse application domains. Laksono and Aditya [14] demonstrated how Unity can be used to visualize topographic and geospatial data in 3D. This capability supports more realistic navigation in areas such as urban planning, training simulations, and disaster management. Mas'udi et al. [15] focused on decision trees for non-player character navigation in Unity-based games. Their work highlighted how structured decision-making can enrich gameplay, offering players more realistic and challenging interactions with agents. Gunes and Dilipak [16] applied Unity to urban pedestrian navigation, analyzing how systems cope with dynamic obstacles and transfer points, thus addressing issues relevant to real-world mobility. Finally, Gazis and Katsiri [17] reviewed the role of Unity in serious games, identifying how navigation systems contribute to training, education, and broader applications beyond entertainment. Taken together, the literature highlights two complementary trends:

- Optimization of performance – ensuring that Unity-based navigation systems remain efficient across hardware and scale well to large agent populations.
- Enhancement of adaptability – integrating machine learning, hybrid planning, or AR extensions to support navigation in dynamic and unpredictable environments.

These insights provide essential context for evaluating AlchemyNavigation, which combines the deterministic stability of geometric approaches with novel runtime flexibility, aiming to address the limitations of Unity's standard NavMesh.

## STUDIED SYSTEMS

This section is dedicated to describing the studied systems and their distinctive and unique features, developed based on their technical documentation [18, 19].

### AlchemyNavigation system

AlchemyNavigation is a proprietary navigation system that can be added to a Unity project via the Package Manager tool. It is a new solution that enables real-time dynamic modification of the navigation space, making it particularly valuable in the context of procedurally generated environments. This tool offers comprehensive agent customization capabilities through inheritance mechanisms and motion modifiers, ensuring precise control over their behavior. It stands out with its flexible navigation path management using a layering system and a weight assignment mechanism, effectively influencing agents' route selection preferences. It features advanced functionalities that allow for the creation and modification of the navigation space during program execution, as well as modules adapted for overwriting.

At the core of the package is the AlchemyNavigationSystem manager, which coordinates all key functions and exposes configuration of the navigation space, including partitioning into up to 32 surfaces/layers. The system offers three complementary ways to build the navigable space: (i) script-based insertion and removal of triangles, with each operation returning a persistent handle for later updates; (ii) an editor-level FacesHolder component that allows to visually add, enable/disable, or transform triangles at any time while leveraging standard Unity GameObject operations; and (iii) baking from scene geometry, performed either offline in the editor or at runtime to initialize large areas efficiently. In practice, static baking via editor tools produces a set of GameObjects equipped with FacesHolder components that the user can further modify, toggle, or transform, whereas dynamic baking during program execution is driven by scripts and yields sets of triangles that can be added to – or removed from – the navigation space as gameplay evolves. Agents may be instantiated either as custom, non-standard implementations or by using the built-in simple agents, depending on the required behaviors and integration complexity

The simple agents provide standard movement suitable for most game characters. Additionally, their behavior can be modified by adding special components called movement modifiers, which can be used to implement collision avoidance behavior.

### NavMesh system

The NavMesh system was chosen for comparison with the developed AlchemyNavigation system because it is one of the most popular and widely used solutions for navigation and pathfinding in virtual environments, especially in game development and interactive simulations. NavMesh provides robust and well-optimized algorithms for agent navigation on static surfaces. However, it has certain limitations, particularly when dealing with highly dynamic environments or non-standard movement behaviors, which the AlchemyNavigation system aims to address. Therefore, comparing the two allows us to evaluate the advantages and improvements of the proposed method over a widely accepted industry standard.

The comparative analysis in this study focuses on Unity NavMesh as it represents the most widely adopted navigation system integrated natively into Unity. While alternative solutions exist (e.g., custom A-based systems or crowd simulation frameworks), NavMesh serves as an industry baseline, ensuring a relevant and reproducible evaluation. NavMesh is a navigation system available by default in every Unity project. It is characterized by a high level of integration with the environment and versatility. The system consists of four main elements:

1. NavMesh – a data structure describing the surface on which agents can move.
2. NavMesh Agent – a component that allows the creation of characters that can move on the navigation surface and avoid collisions with other characters and obstacles.
3. NavMesh Obstacle – a component that enables the creation of dynamic obstacles that should be avoided by agents.
4. Off-Mesh Link – a component that allows the creation of shortcuts between locations on the navigation surface.

Beyond its core elements, Unity exposes high-level components that streamline building and maintaining navigation surfaces in the editor and at runtime. NavMeshSurface builds per-agent

navigation surfaces that can be enabled or disabled as needed. NavMeshModifier and NavMeshModifierVolume label areas either by scene hierarchy or by explicit 3D regions, allowing developers to adjust traversal properties without rebaking the entire level. Finally, NavMeshLink creates dynamic connections between otherwise separate surfaces (e.g., across gaps, doors, or ladders), enabling off-mesh transitions during gameplay. In the NavMesh system, the navigation surface is created through a baking process, which collects rendering components from all objects marked as navigation-static in the scene. It then processes the contained geometry data and approximates surfaces that agents can traverse.

## RESEARCH METHODOLOGY

The research methodology was designed to ensure reproducibility, clarity, and a comprehensive evaluation of the two navigation systems. The process consisted of three stages:

- preparation of research workstations – three computers with distinct hardware specifications were selected to test scalability and hardware dependency,
- development of experimental scenes in Unity – a standardized base scene was created with specific features to simulate realistic multi-level environments,
- implementation of navigation variants and measurement – both AlchemyNavigation and NavMesh were tested under identical conditions, with systematic recording of performance metrics using dedicated profiling tools.

### Research workstation

The study was conducted in an identical manner on three different research stations, each featuring distinct hardware components and parameters. The specifications of the machines are presented in Table 1.

### Scene preparation

A Unity project was created featuring a base scene (Figure 1), which contained the elements summarized in Table 2. To ensure a comprehensive analysis, several variants of the base scene were prepared (Table 3), differing in the active navigation system and its configuration.



**Table 1.** Technical specifications of the stations used in the studies

No.	Processor	GHz	Graphics card	RAM (GB)
1.	Intel Core i7-6700K	4.00	Nvidia GTX 1660 SUPER	32
2.	Intel Core i7-8750H	2.20	Nvidia GTX 1050Ti	16
3.	Intel Core i7-7200U	2.50	Nvidia 940MX	4

## Measurement tools and procedure

The CPU and RAM load tests were conducted using a custom-built Unity application that incorporated the tested navigation systems, as well as a baseline scenario without any active navigation system. The following measurement tools and methods were applied:

- Unity.Profiling API (ProfilerRecorder) [20] – used to collect frame time (ms), CPU usage (%), and RAM usage (MB). Data were sampled every 1 second across a runtime of 20 minutes for each test variant.
- Windows Task Manager validation – RAM readings from Unity profiler were cross-validated against Task Manager to ensure accuracy.
- Measurement interval – average values of frame time and memory were calculated from 1-second intervals across the full runtime.

This combination ensured accurate and consistent monitoring of system performance across different machines and agent densities.

## Performance indicators and selection rationale

The performance evaluation focused on three key indicators:

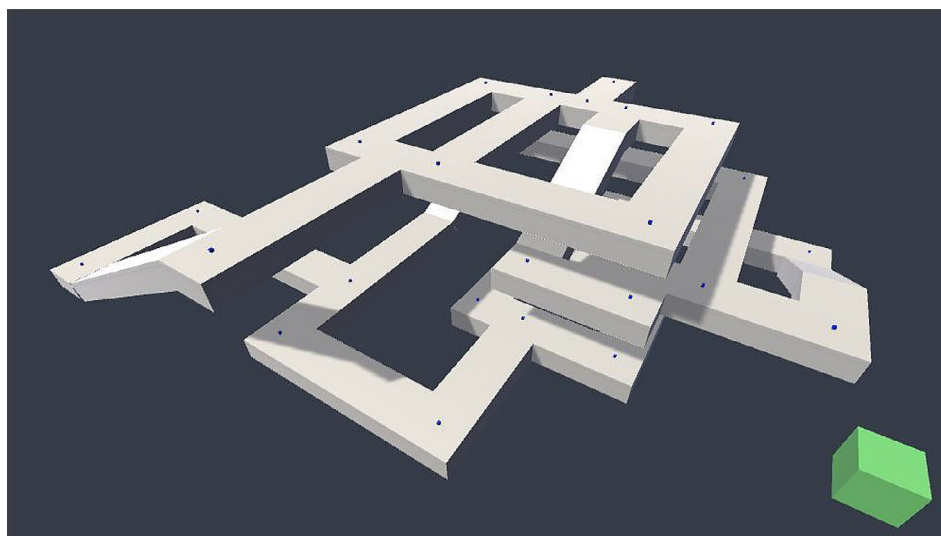
**Table 2.** Features of the base scene

No.	Feature of the scene
1.	The scene consists of one hundred cuboid objects
2.	Objects are placed at different height levels to form multiple floors
3.	The floors are interconnected, creating a larger, non-primitive structure

**Table 3.** Variants of the base scene

No.	Scene variant
1.	No active navigation system
2.	Active AlchemyNavigation system (without collision avoidance)
3.	Active AlchemyNavigation system (with collision avoidance)
4.	Active NavMesh system (without collision avoidance)
5.	Active NavMesh system (with collision avoidance)

- Frame time (ms) – determines simulation smoothness and responsiveness; essential for real-time interactive environments.
- CPU utilization (%) – critical for navigation systems since pathfinding and agent updates are primarily CPU-bound processes in Unity.
- RAM consumption (MB) – ensures scalability by confirming that navigation systems do not

**Figure 1.** Base scene of the author's application for evaluation

excessively increase memory usage with higher agent counts.

We limited the analysis to frame time, CPU utilization, and RAM because Unity's pathfinding and agent updates are CPU-bound in our setup, while GPU load remained largely unaffected by the navigation choice. Path-quality metrics (e.g., trajectory optimality, collision counts) are acknowledged as important but were out of scope here and are listed as limitations for future work.

Other potential criteria, such as GPU load, were not prioritized, as navigation computations in Unity are CPU-dominant. Similarly, agent trajectory accuracy and collision counts were not used as independent metrics because both systems employ deterministic pathfinding algorithms. Instead, collision avoidance effects were implicitly captured through CPU load and frame time variations.

### Experimental conditions

For each scene variant, measurements were taken with 10, 150, and 500 active agents on all three machines. Tests were carried out with and without collision avoidance enabled for both AlchemyNavigation and NavMesh. All trials were conducted under identical environmental

conditions, ensuring fair comparison between the two navigation systems.

## RESULTS

The average measurement results for the tested systems in terms of frame time are presented in Figures 2 and 3.

Based on the obtained results for the AlchemyNavigation system, it can be concluded that both the hardware components and the number of agents have a noticeable impact on frame time. Another noteworthy observation is that for a larger number of active agents, enabling the collision avoidance function causes a sharp increase in CPU usage. For 150 agents, the largest difference recorded was 2.24 ms, whereas for 500 agents, it reached 38.98 ms.

For the NavMesh system, the results indicate that agent count number and hardware components significantly affect frame time. An interesting observation is that the CPU load increase caused by enabling collision avoidance is noticeable but remains stable and moderate (with a worst-case increase of about 50%). Although these trends mirror those reported for AlchemyNavigation, the absolute frame-time

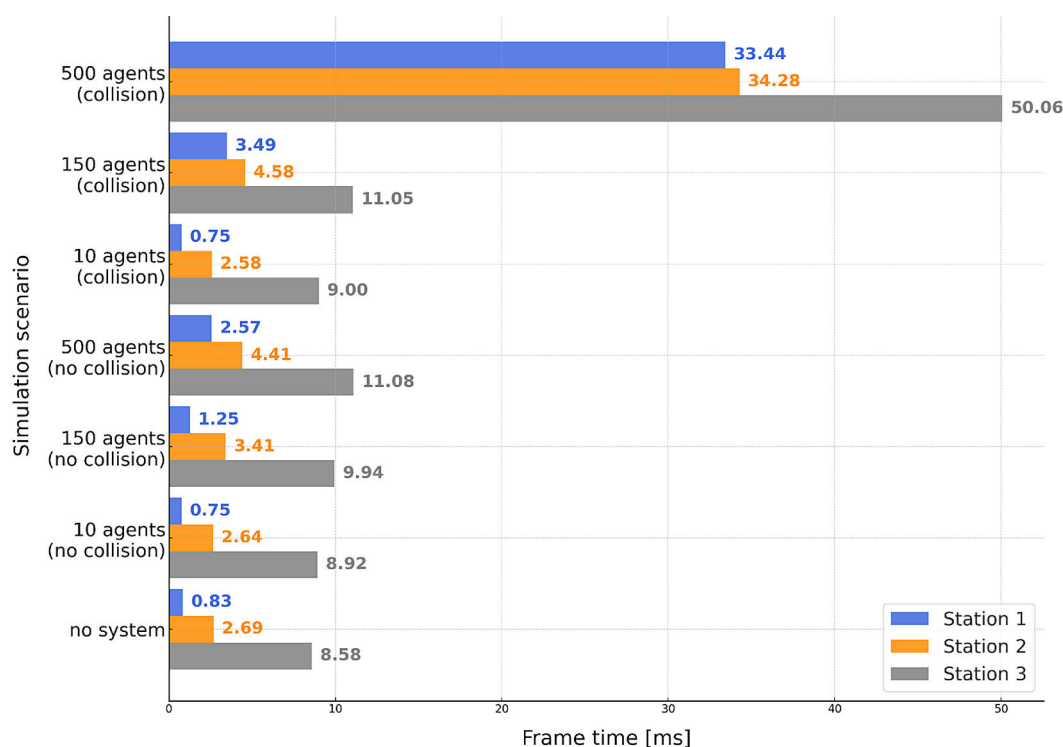


Figure 2. Average frame time for the AlchemyNavigation system

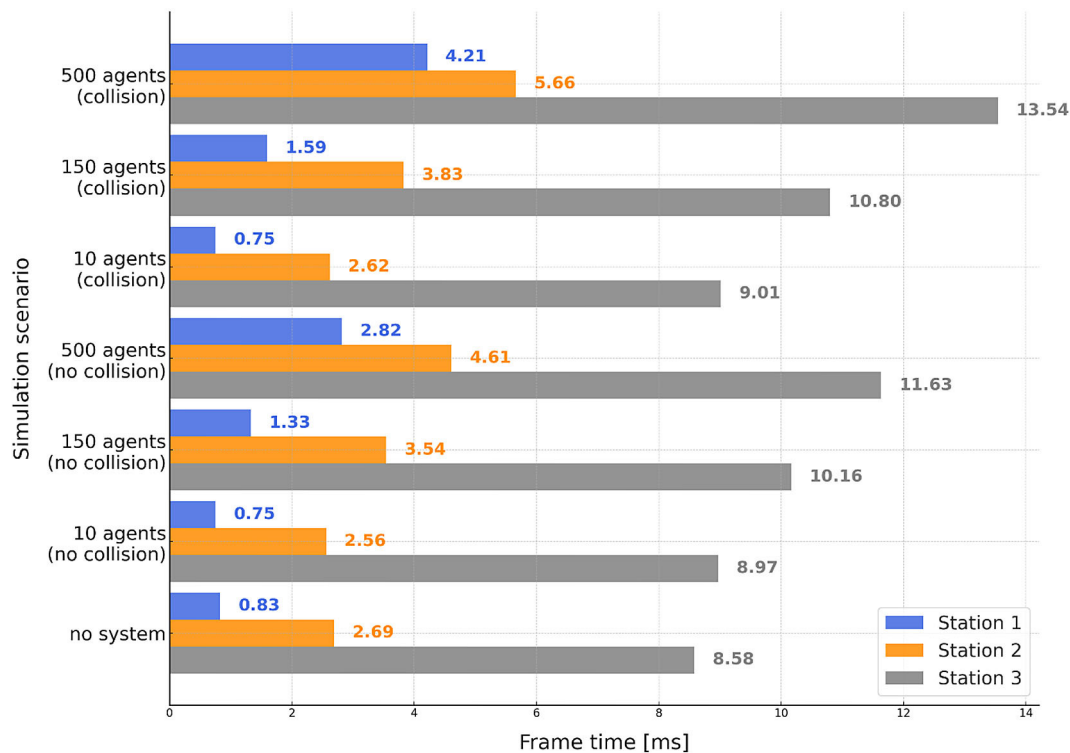


Figure 3. Average frame time for the NavMesh system

levels differ due to implementation details. In NavMesh, crowd handling and local avoidance are amortized by a built-in controller that batches neighbor searches and stabilizes updates; in our current AlchemyNavigation setup, avoidance

checks scale less favorably under congestion (e.g., narrow corridors), which produces episodic spikes at high agent counts. Consequently, both systems show the same qualitative behavior (agent count, hardware, and avoidance matter),

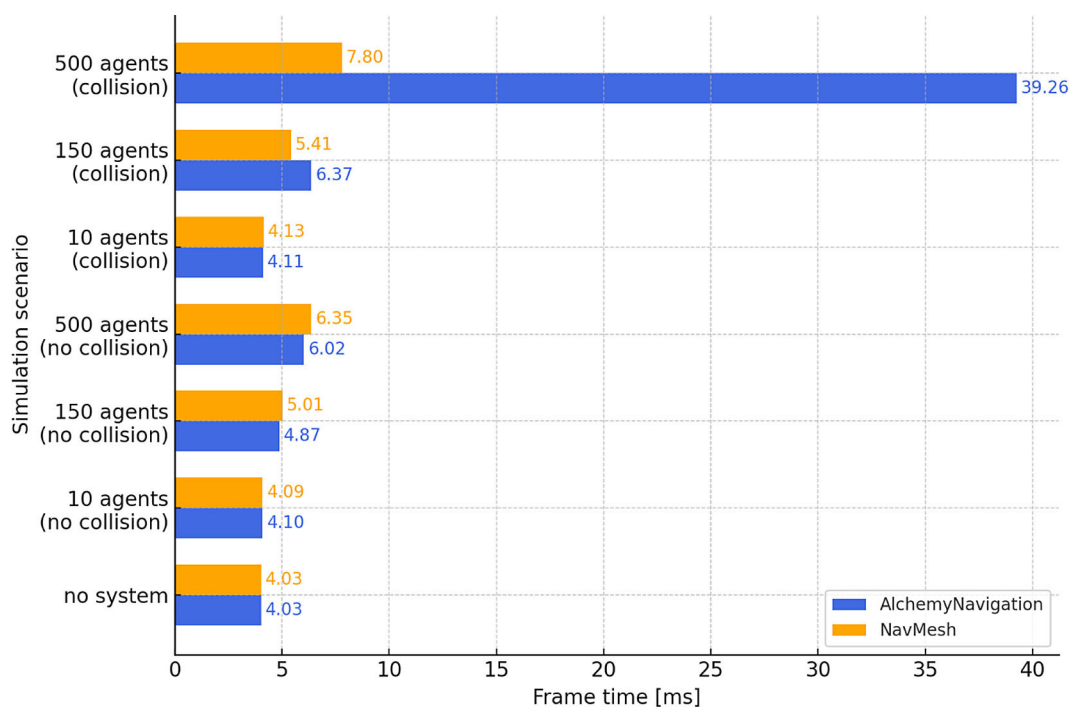


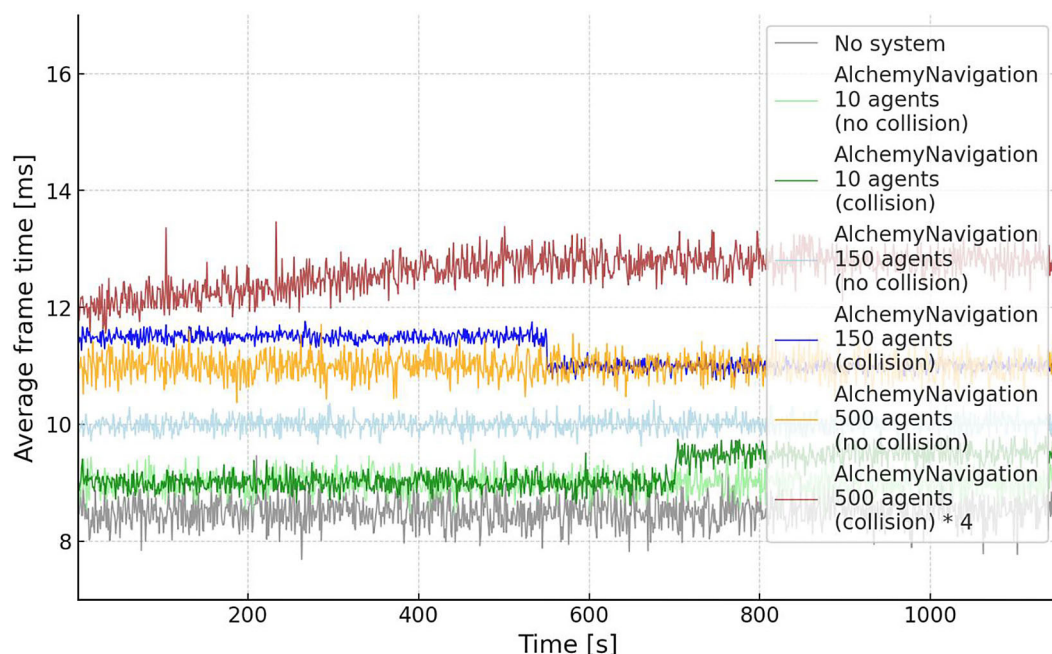
Figure 4. Comparison of average frame time collected from all stations for the studied systems

but their magnitudes diverge, especially under heavy load. In Figure 4, the analysis results of the collected data for both analyzed and compared systems are presented. The first noticeable conclusion from the comparative analysis is that for a small number of agents (10), regardless of the navigation system or whether the collision avoidance function is enabled, the increase in frame time is almost imperceptible – a maximum of about 2%. The AlchemyNavigation system is slightly faster when collision avoidance is disabled. This is because, with avoidance disabled, AlchemyNavigation uses a lighter agent pipeline – direct triangle-level traversal and minimal steering logic – whereas NavMesh retains a baseline management overhead for agent controllers and surface bookkeeping even without avoidance. However, when collision avoidance is enabled, the NavMesh system consumes fewer CPU resources and remains more stable, especially when handling a large number of active agents (a difference of 31.46 milliseconds in the test with 500 agents and collision avoidance enabled). This behavior stems from NavMesh’s crowd controller and batched local-avoidance updates, which reduce oscillations and neighbor-query costs at scale. Spatial partitioning and cached path corridors amortize per-frame work, yielding lower CPU usage and narrower frame-time bands under heavy congestion – consistent with the  $\approx 31.46$  ms gap observed at 500

agents. Figures 5 and 6 show the time-series of average frame time for AlchemyNavigation and NavMesh. With collision avoidance disabled, both systems are nearly steady – departures from the mean are infrequent and small. When avoidance is enabled and the agent count is 150 or 500, periodic oscillations emerge, consistent with congestion in narrow corridors. In AlchemyNavigation, the series settles near 11–12 ms and exhibits episodic spikes; for 150 agents it initially stabilizes around 11.5 ms and later drifts toward  $\approx 11$  ms. In the corresponding NavMesh runs, values remain within a 9.5–11.5 ms band. These patterns suggest that the fluctuations are driven by corridor bottlenecks and by differences in how the systems batch local-avoidance updates and neighbor queries.

Figure 7 compares the average RAM usage values obtained in all trials. For both tested systems, the results indicate no significant relationship between the number of agents and RAM usage. The largest recorded difference was 1.57%, which is small enough to consider the memory usage differences between the systems negligible.

Figures 8 and 9 present the average RAM usage values recorded in the trials for both systems. Based on the recorded RAM usage trends, the following conclusions can be drawn. In both tested systems, RAM usage remains very stable, with variations never exceeding one megabyte.



**Figure 5.** Average frame time trends for the AlchemyNavigation system



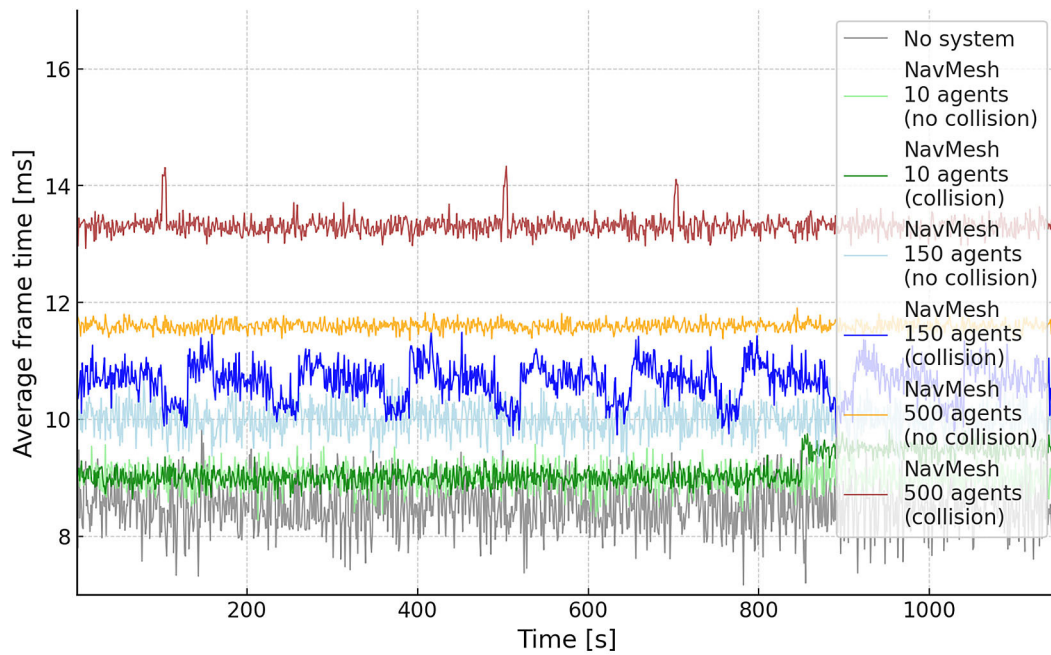


Figure 6. Average frame time trends for the NavMesh system

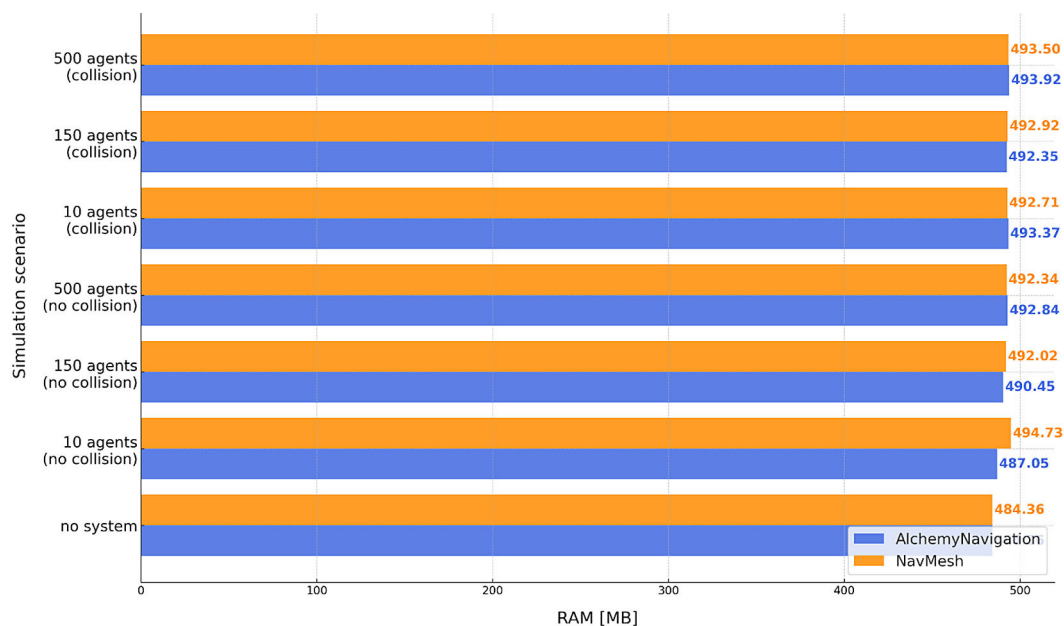


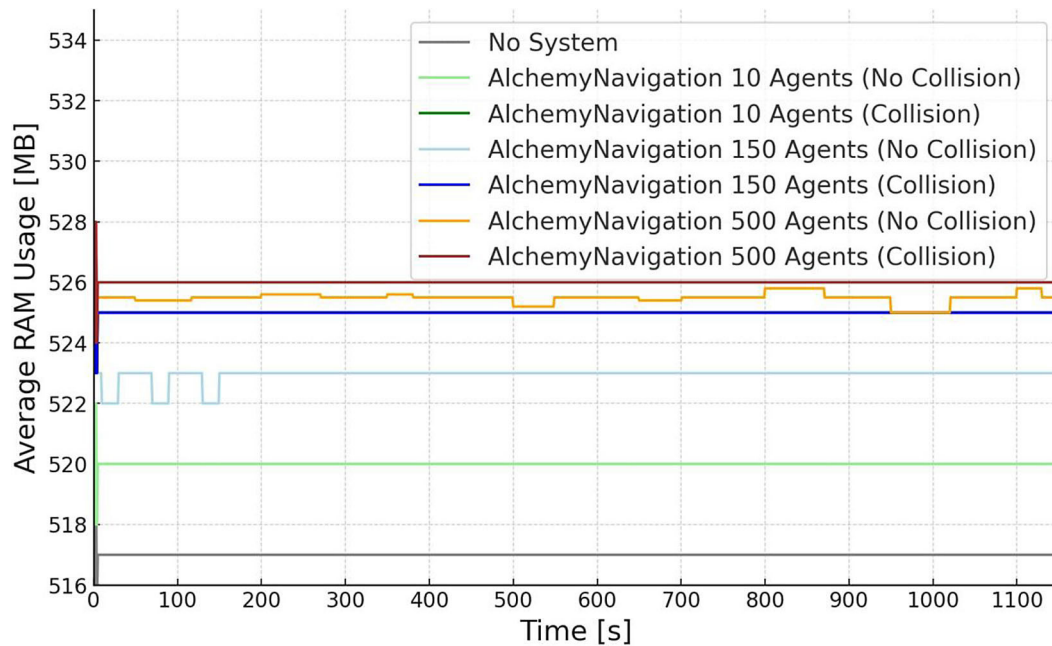
Figure 7. Comparison of average RAM usage from all test stations for the studied systems

However, during the first few seconds, there are sharp changes in RAM usage. This is likely due to initialization of the navigation spaces and the fact that all agents request paths simultaneously.

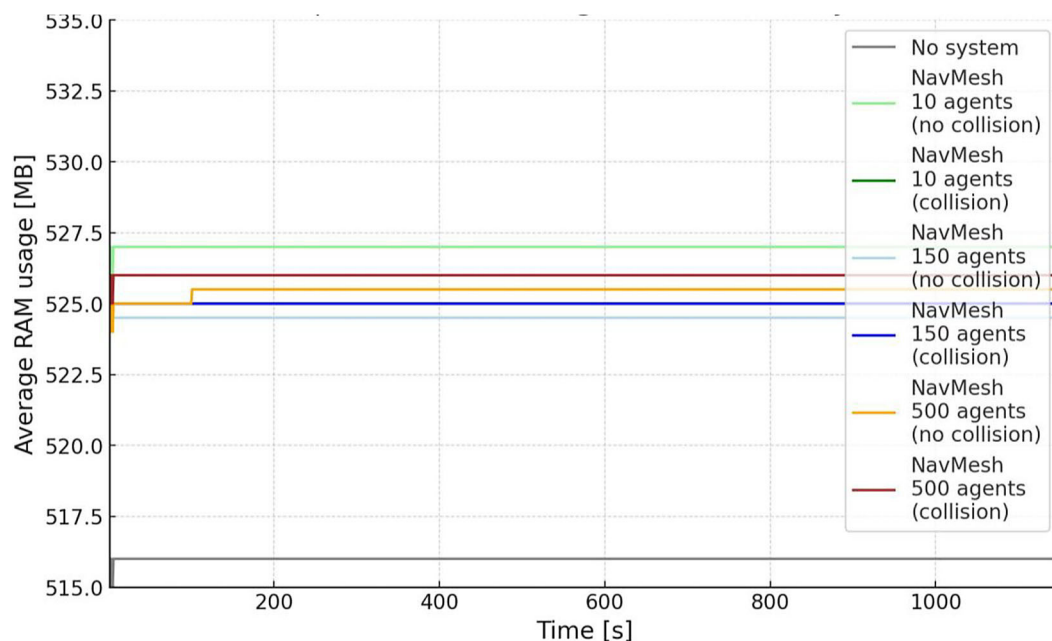
A key observation is that in AlchemyNavigation tests with fewer agents, the values stabilize at slightly lower levels (520 MB for 10 agents and 523 MB for 150 agents). In contrast, NavMesh tests show very similar RAM consumption across all tests, stabilizing at approximately 526 MB.

### Frame rate analysis

Frame rate is one of the key indicators of a navigation system's efficiency. It determines the time required to compute a single frame, which directly affects the smoothness of animation and responsiveness of the system. The results clearly indicate that an increasing number of agents result in a proportional decrease in frame rate (or increase in frame time) for both analyzed systems. However, the scale of this impact differs depending on the navigation system and its configuration.



**Figure 8.** Average RAM usage trends for the AlchemyNavigation system



**Figure 9.** Average RAM usage trends for the NavMesh system

In the case of AlchemyNavigation, enabling collision avoidance causes a significant increase in frame time, especially for a large number of agents. For 500 agents, the difference in frame time compared to the version without collision avoidance reaches 38.98 ms. This large delta arises from dense-crowd local-avoidance costs and burst re-planning. With 500 agents, the effective neighbor sets grow and trigger more frequent short-horizon steering updates; in our maps,

corridor bottlenecks intensify these interactions. In the current AlchemyNavigation setup, triangle-level updates and per-agent checks scale less favorably under congestion, producing episodic frame-time spikes and a higher average.

In the case of NavMesh, the performance degradation is more moderate. Even for 500 agents, the additional computational overhead from collision avoidance does not exceed 50% of the base frame time. This ceiling reflects NavMesh's

crowd controller, which batches neighbor queries, uses spatial partitioning, and maintains cached path corridors. These mechanisms amortize per-frame work during congestion, so the avoidance overhead grows more gently with agent count and remains below roughly half of the base frame time in our tests.

These results suggest that NavMesh is better optimized for handling a large number of moving agents, while AlchemyNavigation provides greater flexibility but at the cost of a higher computational load. Frame rate measurements were conducted on three machines with different hardware configurations. The results indicate that:

- CPU speed has a significant impact on performance. On the machine equipped with an Intel Core i7-6700K (4.00 GHz), the frame time was noticeably lower (frame rate higher) compared to the machine with an Intel Core i7-7200U (2.50 GHz) for the same number of agents.
- Graphics card performance did not significantly affect navigation system efficiency, as pathfinding computations were primarily CPU-bound.
- RAM capacity also did not play a major role, as neither system exceeded available memory limits.

These results confirm that navigation systems in Unity are primarily CPU-dependent, and increasing processor clock speed and efficiency directly translates into better navigation performance.

### RAM usage analysis

The results show that the number of agents does not significantly affect RAM usage. The largest recorded difference was 1.57%, which is negligible. This means that both AlchemyNavigation and NavMesh have efficient memory management mechanisms, allowing them to operate even with a large number of agents without excessive RAM consumption. The analysis of RAM usage trends indicates that:

- In the first few seconds of the simulation, there is a sharp increase in RAM usage, likely due to navigation space initialization and simultaneous pathfinding requests from all agents.
- After the initial phase, RAM usage stabilizes and remains constant throughout the simulation.
- AlchemyNavigation with fewer agents (10–150) required slightly less RAM, stabilizing at 520–523 MB.

- NavMesh consumed a similar amount of RAM across all tests, stabilizing at approximately 526 MB.

These results confirm that both systems efficiently allocate and manage memory, making them suitable for large-scale simulations.

### Collision avoidance impact analysis

Collision avoidance is a crucial feature of any navigation system, as it ensures realistic agent behavior and prevents overlapping movement paths.

The study confirmed that enabling collision avoidance significantly increases CPU usage. However, the scale of this impact varies between the systems:

- In AlchemyNavigation, enabling collision avoidance caused a noticeable increase in frame time as the number of agents grew. This was particularly evident in the test with 500 agents, where the additional processing cost reached 38.98 ms per frame.
- In NavMesh, the impact of collision avoidance was more moderate and stable. Even for 500 agents, the additional computational overhead was limited to an increase of up to 50% in frame time.

These results suggest that AlchemyNavigation provides a more flexible but computationally expensive implementation of collision avoidance, whereas NavMesh offers a more optimized and scalable solution. The experiments also examined how agents behaved in crowded environments, such as narrow corridors or intersections. The key observations were:

- In AlchemyNavigation, agents frequently stopped and dynamically adjusted their paths, which sometimes led to unnatural movement patterns.
- In NavMesh, agents followed more predictable trajectories and handled congestion more smoothly, which suggests a more stable pathfinding algorithm.

These findings indicate that AlchemyNavigation might require additional tuning for large-scale environments with high agent density, while NavMesh is better suited for handling complex crowd movements.

## DISCUSSION

The results of this study highlight the trade-off between flexibility and performance observed in AlchemyNavigation compared to Unity's native NavMesh. AlchemyNavigation provides dynamic triangle-based modifications of the navigation space, offering greater adaptability for procedurally generated and dynamic environments. However, this flexibility results in increased computational cost, particularly when collision avoidance is enabled with high agent densities.

Our experiments demonstrated that NavMesh maintains more stable performance in large-scale simulations, with significantly lower increases in frame time when collision avoidance was activated (up to 50% overhead) compared to AlchemyNavigation, which recorded a maximum increase of 38.98 ms per frame with 500 agents. This indicates that NavMesh is more efficient in handling large crowds and stable environments. Comparable observations have been reported in the literature, where hybrid navigation strategies combining global planning with local reinforcement learning improved efficiency by approximately 20% and reduced collision rates by 34% [21]. These results align with our findings that deterministic solutions such as NavMesh and AlchemyNavigation achieve predictable outcomes but differ in scalability.

While AlchemyNavigation relies on deterministic geometric algorithms, its performance trends are similar to challenges described in reinforcement learning frameworks, which include training costs and stability limitations in real-world applications [22]. Our study shows that deterministic methods still provide predictable and controlled outcomes, which is especially valuable in scenarios requiring reliability. At the same time, modular systems such as AlchemyNavigation could be extended with adaptive mechanisms, consistent with approaches explored by Joshi [23] and others developing procedural generation frameworks [24, 25]. In scenarios with 500 agents, NavMesh managed congestion more smoothly, while AlchemyNavigation agents often re-routed dynamically, leading to occasional unnatural movement patterns. Similar scalability issues have been addressed through hierarchical pathfinding, such as MA-HNA\* [26], which demonstrated significant improvements in handling large agent populations. Likewise, parallel reinforcement learning approaches [27] showed

increased decision accuracy in multi-agent environments. These studies support the notion that scalable solutions often combine hierarchical or parallelized strategies – features that could potentially enhance AlchemyNavigation in future development. Although our results focused on CPU and RAM metrics, they also point to the potential of combining AlchemyNavigation with adaptive tuning methods. For example, reinforcement learning enhanced with transformers [29] or image-based trajectory planning [28] has been presented to improve data efficiency and generalization. The modular design of AlchemyNavigation would allow the integration of such approaches, bridging deterministic stability with adaptive flexibility.

Moreover, the results suggest that the two systems should be chosen contextually: NavMesh remains superior for large-scale, stable simulations where efficiency is critical, whereas AlchemyNavigation is particularly suited for dynamic and procedurally generated environments. These findings are consistent with broader research indicating that hybrid approaches – whether based on swarm optimization [8], reinforcement learning [3, 30, 31], or multi-agent coordination [32, 33] – aim to balance adaptability with computational efficiency.

In summary, this study confirms that AlchemyNavigation complements rather than replaces NavMesh. Its ability to adapt environments at runtime fills a gap in Unity's native navigation tools, while its limitations under heavy load align with known challenges of dynamic navigation systems. By positioning our findings within the context of reinforcement learning, hierarchical planning, and perception-driven approaches, we demonstrate how AlchemyNavigation can evolve in line with current study directions while retaining the benefits of deterministic control.

## LIMITATIONS AND FUTURE WORK

This study has several limitations that should be acknowledged. The evaluation was conducted exclusively in the Unity engine. While Unity was chosen deliberately due to its widespread adoption in both academic and commercial development, further studies are needed to verify the generalizability of AlchemyNavigation in other environments such as Unreal or Godot, as well as in non-gaming contexts.



The experiments were limited to three desktop hardware configurations. Tests on mobile and VR/AR platforms, which operate under stricter computational constraints, are essential to assess scalability and performance in resource-limited conditions.

The analysis focused on computational performance indicators (frame time, CPU, RAM). Metrics describing path quality or subjective assessments of agent behavior realism were not systematically included. Future research should complement performance evaluation with user studies and quantitative path metrics such as path length deviation or collision rates.

The presented results are based on averaged values without applying statistical significance testing. Future studies should incorporate statistical analyses (e.g., ANOVA, t-tests) to strengthen the reliability and reproducibility of the findings.

Future research directions include:

- Testing in more complex environments with a larger number of dynamic obstacles and diverse spatial structures.
- Comparative analyses with hybrid path planning methods and tools such as Unity ML-Agents [9, 30, 31].
- Integration with reinforcement learning approaches, including multi-agent reinforcement learning [32, 33], to improve adaptability and flexibility.
- Application of perception-based methods such as transformer networks [28] and image-based trajectory planning [29] to enhance efficiency and generalization.
- Verification of system performance on mobile devices and under VR/AR conditions [12]

## CONCLUSIONS

The conducted study demonstrated that the proprietary AlchemyNavigation system is an effective alternative to the standard NavMesh solution in the Unity environment. The analysis confirmed that AlchemyNavigation stands out with its ability to dynamically modify the navigation space in real time, which is particularly important in procedurally generated and dynamically changing environments. At the same time, it was found that this flexibility is associated with higher CPU load, especially in scenarios involving a large number of agents and when collision avoidance is enabled. The comparison of both systems

also indicated that NavMesh ensures more stable performance and lower computational costs in large-scale simulations, whereas AlchemyNavigation provides broader adaptive capabilities at the expense of efficiency. The memory usage analysis showed that both solutions are characterized by comparable and stable RAM consumption, regardless of the number of agents.

In summary, AlchemyNavigation complements the standard NavMesh system by offering additional functionality for real-time navigation space modification, while maintaining acceptable computational efficiency.

## Acknowledgements

AlchemyNavigation system is available at <https://kempnynmaciej.github.io/alchemy-navigation/index.html>.

## REFERENCES

1. Medina RCS, Gittabao HSB, Agustin VA. Modifying JPS algorithm using navmesh data structure applied in 3D using Unity. *United Int J Res Technol (UIJRT)*. 2023; 4(8): 41–7.
2. Demyen D, Buro M. Efficient triangulation-based pathfinding. In: *AAAI*. 2006; 6: 942–7.
3. Alonso E, Peter M, Goumard D, Romoff J. Deep reinforcement learning for navigation in AAA video games. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21)*. 2021; p. 2133–2139. <https://doi.org/10.24963/ijcai.2021/294>
4. Cui X, Shi H. A\*-based pathfinding in modern computer games. *Int J Comput Sci Netw Secur*. 2011; 11(1): 125–130.
5. Barnouti NH, Al-Dabbagh SSM, Naser MAS. Pathfinding in strategy games and maze solving using A\* search algorithm. *J Comput Commun*. 2016; 4(11): 15–25. <https://doi.org/10.4236/jcc.2016.411002>
6. Zikky M. Review of A\* (A star) navigation mesh pathfinding as the alternative of artificial intelligent for ghosts agent on the Pacman game. *EMITTER Int J Eng Technol*. 2016; 4(1): 141–9. <https://doi.org/10.24003/emitter.v4i1.117>
7. Pyke LM, Stark CR. Dynamic pathfinding for a swarm intelligence based UAV control model using particle swarm optimisation. *Front Appl Math Stat*. 2021; 7: 744955. <https://doi.org/10.3389/fams.2021.744955>
8. Lee T. Unity engine dissection: Improvement points in comparison between Unity engine and

- open-source engines. 2024. <https://doi.org/10.47116/apjcri.2024.11.33>
9. Messaoudi F, Ksentini A, Simon G, Bertin P. Performance analysis of game engines on mobile and fixed devices. *ACM Trans Multimedia Comput Commun Appl.* 2017; 13(4): 1–28. <https://doi.org/10.1145/3115934>
10. Fajrianti ED, Funabiki N, Sukaridhoto S, Panduman YYF, Dezheng K, Shihao F, et al. INSUS: Indoor navigation system using Unity and smartphone for user ambulation assistance. *Information.* 2023; 14(7): 359. <https://doi.org/10.3390/info14070359>
11. Ma H, Smith JS, Vela PA. NavTuner: Learning a scene-sensitive family of navigation policies. In: 2021 IEEE/RSJ Int Conf Intell Robots Syst (IROS). IEEE; 2021; 492–9. <https://doi.org/10.48550/arXiv.2103.01464>
12. Zhou X, Gao Y, Guan L. Towards goal-directed navigation through combining learning based global and local planners. *Sensors.* 2019; 19(1): 176. <https://doi.org/10.3390/s19010176>
13. Juliani A, Berges V-P, Teng E, Cohen A, Harper J, Elion C, et al. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627.* 2018. <https://doi.org/10.48550/arXiv.1809.02627>
14. Laksono D, Aditya T. Utilizing a game engine for interactive 3D topographic data visualization. *ISPRS Int J Geo-Inf.* 2019; 8(8): 361. <https://doi.org/10.3390/ijgi8080361>
15. Mas'udi NA, Jonemaro EMA, Akbar MA, Afrianto T. Development of non-player character for 3D kart racing game using decision tree. *Fountain Inform J.* 2021; 6(2): 51–60. <https://doi.org/10.21111/fij.v6i2.4678>
16. Gunes M, Dilipak H. Shortest path problem in pedestrian transfers application in Unity. *Gazi J Eng Sci.* 2020; 6(2). <https://doi.org/10.30855/gmbd.2020.02.03>
17. Gazis A, Katsiri E. Serious games in digital gaming: A comprehensive review of applications, game engines and advancements. *arXiv preprint arXiv:2311.03384.* 2023. <https://doi.org/10.37394/232018.2023.11.2>
18. Kempny M. *AlchemyNavigation documentation.* 2021 [cited 2024 Jun 20].
19. Unity Technologies. *NavMesh documentation.* 2021 [cited 2024 Jun 20].
20. Unity Technologies. *Unity.Profiling documentation.* 2021 [cited 2024 Jun 20].
21. Wang X, Sun Y, Xie Y, Bin J, Xiao J. Deep reinforcement learning-aided autonomous navigation with landmark generators. *Front Neurobot.* 2023; 17: 1200214. <https://doi.org/10.3389/fnbot.2023.1200214>
22. Tang C, Abbatematteo B, Hu J, Chandra R, Martín-Martín R, Stone P. Deep reinforcement learning for robotics: A survey of real-world successes. *Annu Rev Control Robot Auton Syst.* 2025; 8: 153–188. <https://doi.org/10.1146/annurev-control-030323-022510>
23. Joshi AS. Reinforcement learning-enhanced procedural generation for dynamic narrative-driven AR experiences. In: *Proceedings of GRAPP 2025; 2025.* arXiv:2501.08552.
24. Fachada N, e Silva RC, de Andrade D, Códices N. Unity snappable meshes. *Software Impacts.* 2022; 13: 100363. <https://doi.org/10.1016/j.simpa.2022.100363>
25. Cheliotis K. ABMU: An agent-based modelling framework for Unity3D. *SoftwareX.* 2021; 15: 100771. <https://doi.org/10.1016/j.softx.2021.100771>
26. Rahmani V, Pelechano N. Multi-agent parallel hierarchical path finding in navigation meshes (MA-HNA\*). *Comput Graph.* 2020; 86: 1–14. <https://doi.org/10.1016/j.cag.2019.10.006>
27. Liu C, Liu D. Deep reinforcement learning algorithm based on multi-agent parallelism and its application in game environment. *Entertain Comput.* 2024; 50: 100670. <https://doi.org/10.1016/j.entcom.2024.100670>
28. Babiarz A, Kustra M, Wen S. Application of deep learning methods for trajectory planning based on image information. *Adv Sci Technol Res J.* 2024; 18(6): 247–258. <https://doi.org/10.12913/22998624/191924>
29. Huang W, Zhou Y, He X, Lv C. Goal-guided transformer-enabled reinforcement learning for efficient autonomous navigation. *IEEE Trans Intell Transp Syst.* 2024; 25(2): 1832–45. <https://doi.org/10.1109/TITS.2023.3312453>
30. Fadhol MD, Marcellino F, Rabani DD, Azzahra FF, Amalia S, Abdiansah. AI-driven traffic simulation using Unity: implementing finite state machines for adaptive NPC behaviour. *J Intell Comput Health Inform.* 2024; 5(2): 73–78. <https://doi.org/10.26714/jichi.v5i2.14595>
31. Zhang S, Li Y, Dong Q. Autonomous navigation of UAV in multi-obstacle environments based on a deep reinforcement learning approach. *Appl Soft Comput.* 2022; 115: 108194. <https://doi.org/10.1016/j.asoc.2021.108194>
32. Li H, Yang P, Liu W, Yan S, Zhang X, Zhu D. Multi-agent reinforcement learning in games: research and applications. *Biomimetics.* 2025; 10(6): 375. <https://doi.org/10.3390/biomimetics10060375>
33. Zeng W, Yan X, Mo F, Zhang Z, Li S, Wang P, et al. Knowledge-enhanced deep reinforcement learning for multi-agent game. *Electronics.* 2025; 14(7): 1347. <https://doi.org/10.3390/electronics14071347>