BACKSTAGE PASS
INSTITUTE OF GAMING AND TECHNOLOGY

A.Santhosh Kumar Reddy,Bsc Honors

CROWD SIMULATION

to achieve the second year degree of

BSC.Honors

submitted to

BACKSTAGE PASS OF GAMING

Lecturer in game programming

Lecturer in Incharge: Sandeep Salimeda

September 2025

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

Abstract

This thesis investigates the creation and implementation of a real-time crowd simulation system within the Unity game engine, with an emphasis on scalability, local avoidance, and realistic agent behavior. Crowd simulation is a critical area of computer graphics and artificial intelligence that focuses on modeling the movement and interaction of multiple agents in shared environments. Beyond its traditional role in entertainment applications such as video games, crowd simulation serves broader purposes in virtual reality, urban planning, and evacuation modeling, where the need for believable large-scale agent interaction is essential.

The primary objective of this work is to design and implement a prototype capable of simulating large groups of autonomous agents navigating in real time. The system leverages Unity's NavMesh framework for pathfinding and navigation, combined with custom scripting to manage behaviors such as dynamic obstacle avoidance and adaptive crowd flow. Emphasis is placed on performance optimization, ensuring that hundreds or even thousands of agents can be simulated simultaneously without significant degradation of frame rate or responsiveness.

The agent system is designed around modular prefabs, each incorporating Unity's NavMeshAgent component for navigation and additional logic for collision handling and behavioral variation. To enhance realism, local avoidance techniques are employed, allowing agents to adapt dynamically to both static and moving obstacles within the environment. Stress testing and profiling are conducted using Unity's performance analysis tools to evaluate computational efficiency under varying agent densities. A simple user interface is also integrated, enabling runtime adjustment of parameters such as crowd size, speed, and obstacle placement.

All implementations are validated through iterative testing in Unity, across scenes of varying complexity, to assess both behavioral fidelity and system performance. The results demonstrate that scalable, real-time crowd simulations can be achieved with a balance of Unity's built-in components and custom optimizations, providing both visual realism and computational efficiency suitable for interactive applications.

By combining technical problem-solving with design considerations for realism and performance, this thesis highlights the role of crowd simulation as both a research challenge and a practical tool. The findings contribute to the broader field of interactive simulation by presenting a framework that can serve as a foundation for future applications in games, immersive environments, and real-world modeling scenarios.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, **[Sandeep Salimeda]**, for their invaluable guidance, encouragement, and support throughout the course of this thesis. Their insights into both the technical and creative aspects of computer graphics have been instrumental in shaping the direction and quality of this work.

I am also grateful to the faculty and staff of the **Department of [Computer science and Game development], [Backstage Pass of Gaming]**, for providing the academic environment and resources that made this research possible. Their dedication to fostering innovation in game development and computer graphics has been a constant source of inspiration.

Special thanks are due to my peers and colleagues who offered feedback, shared knowledge, and encouraged me during the development and testing phases of this project. The discussions and collaborative spirit were invaluable in overcoming challenges and refining the crowd simulation implementations.

Finally, I would like to extend my heartfelt appreciation to my family and friends for their unwavering support, patience, and encouragement. Their belief in me has been a source of strength throughout my academic journey.

Table of Contents (updated)

Introduction

Computer graphics and artificial intelligence play a central role in shaping interactive media such as video games, virtual simulations, and immersive environments. Among the many techniques available to developers, **crowd simulation** has become a crucial area of study, enabling the realistic modeling of large groups of autonomous agents that can navigate, interact, and adapt within complex environments. Crowd simulation not only enhances the visual and interactive quality of digital experiences but also has practical applications in fields such as urban planning, evacuation training, and virtual reality.

This thesis explores the design and implementation of a real-time **crowd simulation system in Unity**, with a focus on navigation, local avoidance, and large-scale performance. The primary aim is to investigate how agent-based simulation techniques can be used to create scalable, believable, and responsive crowds that react dynamically to both static and moving obstacles. Through the integration of Unity's NavMesh framework, custom behavior scripts, and performance optimization strategies, the project demonstrates how interactive environments can handle hundreds or even thousands of agents simultaneously.

By developing this system, the work contributes not only a practical tool for Unity-based projects but also insights into the challenges and possibilities of large-scale crowd simulation. The findings emphasize how technical approaches to navigation and avoidance can support immersive digital experiences, bridging the gap between artificial intelligence research and real-world interactive applications.

1.1 Goals and Motivation

The primary goals of this thesis are as follows:

- **Design and Implementation:** To design and implement a real-time crowd simulation prototype in Unity capable of handling large groups of agents with realistic navigation and local avoidance.

- **Scalability and Performance:** To explore methods that enable hundreds or thousands of agents to move simultaneously while maintaining computational efficiency.

- **Practical Usability:** To evaluate the system in terms of accuracy, realism, and performance for potential use in games, simulations, and virtual reality applications.

- **Contribution to Simulation Research:** To contribute to the broader field of crowd simulation by presenting an accessible, Unity-based framework that balances realism, scalability, and performance.

The motivation for this research arises from the growing importance of **believable group dynamics** in modern interactive applications. While single-character AI has been extensively developed, many real-world and entertainment scenarios require simulations of entire populations — whether it is large-scale battles in games, evacuation planning in smart cities, or immersive VR environments populated with virtual characters. By focusing on navigation, obstacle avoidance, and performance optimization, this work aims to address these demands, offering insights into how scalable crowd systems can support both creative and practical applications.

1.2 Methodology and Structure

The methodology adopted in this thesis involves the following steps:

- **Literature Review:** Studying existing techniques in crowd simulation, including navigation meshes, pathfinding algorithms, and local avoidance strategies, as well as reviewing related work in large-scale agent-based simulations.

- **Design Phase:** Defining the functional and non-functional requirements of the system, including agent behaviors, environment setup, performance targets, and scalability expectations.

- **Implementation Phase:** Developing the crowd simulation in Unity using NavMesh components and custom C# scripts for agent control, obstacle avoidance, and performance optimization. Iterative testing and refinement ensure the robustness of the system.

- **Evaluation Phase:** Conducting performance analysis by stress testing with different agent counts, measuring frame rate and resource usage, and assessing behavioral realism through observed crowd dynamics.

- **Analysis and Discussion:** Comparing the outcomes with existing crowd simulation methods, reflecting on lessons learned, and identifying limitations and areas for future work.

The structure of the thesis follows a logical progression: it begins with an introduction and background study, proceeds through the design and implementation of the system, evaluates its performance, and concludes with lessons learned and opportunities for further research

Background and Related Work

2.1 Fundamentals of Crowd Simulation

Crowd simulation is a core area of computer graphics and artificial intelligence that focuses on modeling the movement, interactions, and collective behavior of multiple autonomous agents in shared environments. It is widely used in video games, film, virtual reality, and urban simulations, where the presence of realistic crowds significantly enhances immersion and believability. Unlike single-agent systems, crowd simulation must account for large-scale navigation, real-time decision-making, and performance scalability.

At its foundation, crowd simulation combines two essential components: **global navigation** and **local interaction**. Global navigation determines how agents move from one point to another within the environment, typically using pathfinding techniques such as navigation meshes (NavMesh) or graph-based algorithms. Local interaction, on the other hand, governs how agents avoid collisions, maintain personal space, and adapt their paths in the presence of other agents or obstacles. The balance between these two layers is critical for producing realistic crowd dynamics.

A typical simulation pipeline consists of several stages. First, the environment is analyzed and processed into a navigation graph or mesh, defining the areas agents can traverse. Next, individual agents are assigned goals or destinations, and a global path is computed for each one. Finally, local avoidance and steering behaviors are applied to adjust movement in real time, ensuring that agents can adapt to dynamic conditions without compromising overall flow.

Unity provides a practical framework for implementing these principles through its **NavMesh system**, which automatically generates a walkable surface from the environment geometry. Agents, represented by the `NavMeshAgent` component, can then navigate this surface while handling pathfinding, movement, and basic obstacle avoidance. For more advanced behaviors, developers can extend the system with custom scripts, allowing for richer interactions, dynamic obstacle handling, and optimized large-scale performance.

By combining pathfinding, local avoidance, and scalable agent management, crowd simulation establishes the foundation for interactive experiences where large groups of virtual characters behave in a believable and computationally efficient manner.

code. This provides greater artistic control and is well-suited for stylized effects such as holograms, VHS distortions, and watercolor-like rendering.

Previous works in shader programming have shown how even simple manipulations of UV coordinates, fragment color blending, or mathematical functions (e.g., sine waves, noise functions) can produce highly recognizable and impactful visual effects. For example, scanline shaders simulate CRT aesthetics by offsetting pixel rows, while Fresnel-based effects enhance holographic appearances by brightening object edges. These techniques highlight the creative intersection between mathematics, art, and real-time rendering that forms the foundation for the shaders developed in this work.

2.2 Navigation Meshes and Pathfinding Algorithms

Navigation is one of the most fundamental aspects of crowd simulation, as it defines how agents move from their starting positions to their destinations while respecting the geometry of the environment. Effective navigation requires a balance between **global pathfinding**, which determines the overall route an agent should take, and **local adaptation**, which allows agents to respond to dynamic obstacles and other agents in real time.

A widely adopted technique for navigation in interactive applications is the **Navigation Mesh (NavMesh)**. A NavMesh is a simplified geometric representation of the traversable areas in a virtual environment. Unlike grid-based approaches, which divide the world into uniform cells, a NavMesh represents free space using polygons, allowing agents to move more smoothly and efficiently. Unity provides built-in tools for baking NavMeshes, automatically generating walkable surfaces based on level geometry while excluding obstacles and inaccessible areas. This approach enables agents to follow paths that appear natural and computationally efficient.

For **pathfinding**, algorithms such as **A*** (A-star) are commonly used. A* is a graph search algorithm that computes the optimal path between two points by minimizing the cost of traversal while estimating the remaining distance using a heuristic. In NavMesh-based systems, the walkable surface is represented as a graph of connected polygons, and A* (or a variation of it) is applied to find a valid path. Other algorithms, such as Dijkstra's algorithm or hierarchical pathfinding (HPA*), may be used depending on performance requirements and the complexity of the environment.

In Unity, the `NavMeshAgent` component encapsulates these techniques, providing automatic pathfinding across baked NavMeshes. Once a destination is set, the agent computes an optimal route and continuously updates its position along the path, accounting for changes in the environment. Developers can customize parameters such as speed, acceleration, and angular rotation to control movement style and realism.

However, while NavMeshes and pathfinding algorithms excel at determining global routes, they are not sufficient on their own for handling **dense crowds or dynamic obstacles**. To achieve realistic crowd flow, pathfinding must be combined with local avoidance techniques, ensuring that agents can dynamically adjust their movement to avoid collisions while still pursuing their global goals.

Thus, navigation meshes and pathfinding algorithms form the **foundation of global navigation**, upon which more advanced behaviors such as local avoidance and crowd dynamics are built.

2.3 Ignore Behavior and Local Avoidance

While navigation meshes and pathfinding algorithms provide global routes for agents, they are not sufficient for handling the complexities of real-time interactions in dense crowds. When multiple agents occupy the same environment, their paths often intersect, leading to potential collisions or unrealistic overlapping. To address this, local avoidance techniques are introduced, allowing agents to adapt their movement dynamically while still pursuing their global goals.

One approach in crowd simulation is ignore behavior, where agents selectively disregard certain influences or other agents under specific conditions. Instead of reacting to every possible collision or environmental factor, agents prioritize relevant obstacles while ignoring less critical interactions. This reduces computational overhead and prevents agents from making erratic or overly complex adjustments that may disrupt crowd flow. For example, in a sparse crowd, agents may ignore distant individuals and only compute avoidance for nearby neighbors. Similarly, an agent navigating toward a fixed destination might ignore small, temporary disturbances if adjusting would significantly delay progress.

In Unity, the `NavMeshAgent` component includes a built-in avoidance system that calculates velocity adjustments to reduce collisions between agents. However, in large-scale simulations, the default system can become computationally expensive or produce unnatural results. By incorporating ignore behavior strategies, developers can simplify calculations, ensuring that only the most relevant avoidance decisions are made. This leads to smoother motion and improved performance when simulating hundreds or thousands of agents simultaneously.

Local avoidance techniques often draw from established models such as Reciprocal Velocity Obstacles (RVO) and Social Force Models, which mathematically describe how agents adjust their velocities to avoid collisions while maintaining natural movement. In practice, these models can be simplified with ignore behavior rules, enabling agents to focus only on immediate threats and avoid excessive micro-adjustments.

In summary, ignore behavior and local avoidance complement global navigation by enabling real-time adaptability in crowded environments. While pathfinding ensures agents follow optimal routes, avoidance strategies ensure that movement remains

2.4 Applications of Crowd Simulation (Games, VR, Urban Planning)

Crowd simulation is a versatile technology with applications spanning **entertainment, research, safety, and urban development**. By enabling the realistic movement and interaction of large numbers of agents, it supports both immersive digital experiences and practical real-world problem-solving.

1. Games and Interactive Media:
In video games, crowd simulation is used to populate virtual worlds with non-player characters (NPCs) that move believably, creating the illusion of living, dynamic environments. Examples include large battle sequences in strategy or role-playing games, bustling city streets in open-world adventures, and background characters in sports or racing games. A well-implemented crowd system enhances immersion without requiring manual scripting for each character. Moreover, real-time optimization ensures that these simulations remain performant on consumer hardware.

2. Virtual Reality (VR) and Immersive Experiences:
In VR applications, the sense of presence relies heavily on how convincingly virtual environments are populated. Crowd simulation allows VR experiences to replicate realistic scenarios such as concerts, traffic flows, or training exercises. For example, evacuation drills in VR use simulated crowds to help trainees understand panic dynamics and learn appropriate responses. The interactivity of VR demands smooth real-time simulation, where agents not only navigate realistically but also respond adaptively to the user's presence and actions.

3. Urban Planning and Architecture:
Beyond entertainment, crowd simulation plays a significant role in **urban development and infrastructure design**. Architects and planners use simulations to model how people move through public spaces such as train stations, stadiums, or shopping malls. By testing scenarios like rush-hour traffic or emergency evacuations, planners can identify potential bottlenecks and optimize layouts for safety and efficiency. Similarly, crowd models assist in evaluating pedestrian flow in smart cities, helping to design spaces that balance capacity, comfort, and accessibility.

4. Emergency and Safety Training:
Another critical application is **evacuation modeling**, where simulations replicate how people might behave in emergencies such as fires, natural disasters, or large public gatherings. These scenarios allow authorities to design safer evacuation routes, evaluate crowd management strategies, and train responders without the risks associated with real-world drills.

In summary, crowd simulation is not limited to entertainment but extends to domains where human movement and group dynamics are central. Its ability to model complex, large-scale interactions in real time makes it a valuable tool for both **immersive experiences and practical decision-making** in modern society.

2.5 Related Research and Tools

The field of crowd simulation has been widely studied across computer graphics, artificial intelligence, and human behavior modeling. A number of research efforts and tools have contributed to advancing techniques for navigation, local avoidance, and scalability in both academic and industrial applications.

Research Contributions:
Early crowd simulation approaches were grounded in **social force models** (Helbing & Molnár, 1995), which describe pedestrian movement as the result of attractive and repulsive forces between agents and their environment. These models laid the foundation for simulating realistic pedestrian dynamics, especially in evacuation scenarios. Later, **Reynolds' steering behaviors** (1987) introduced agent-based techniques such as seeking, fleeing, and wandering, which became the basis for many interactive simulations in games.

More recent contributions focus on **Reciprocal Velocity Obstacles (RVO)** and **Optimal Reciprocal Collision Avoidance (ORCA)**, which provide mathematically efficient methods for real-time local avoidance between multiple agents. These approaches allow large numbers of agents to move smoothly without collisions, making them suitable for real-time applications such as games and VR.

In the context of large-scale simulations, **hierarchical pathfinding** and **flow-based methods** have been proposed to reduce computational cost, enabling thousands of agents to be simulated simultaneously. Researchers also study **behavioral realism**, incorporating psychological and sociological factors to model panic, group cohesion, or leader–follower dynamics.

Tools and Frameworks:
Several software frameworks and tools exist for implementing crowd simulation:

- **Unity NavMesh System:** Provides built-in pathfinding and basic local avoidance through the `NavMeshAgent` component. It is widely used for game development due to its accessibility and integration with Unity's workflow.

- **Reciprocal Velocity Obstacles (RVO2 Library):** An open-source library offering efficient multi-agent collision avoidance, often integrated into simulations requiring large-scale crowds.

- **Massive Software:** A commercial tool originally developed for film production (e.g., *The Lord of the Rings*), which enables highly detailed and customizable crowd behaviors for cinematic purposes.

- **SUMO (Simulation of Urban MObility):** An open-source traffic simulation package used in research for modeling pedestrian and vehicle flow in urban environments.

- **AnyLogic and Pathfinder:** Professional tools used in evacuation modeling and urban planning, capable of simulating pedestrian flows and emergency scenarios.

Unity's Role:
For this thesis, Unity was chosen due to its balance of accessibility, flexibility, and real-time capabilities. Its NavMesh system provides a strong foundation for pathfinding, while its extensibility through C# scripting allows the integration of custom behaviors and optimizations. Combined with profiling tools, Unity enables iterative testing and refinement, making it an ideal platform for developing crowd simulation prototypes aimed at both entertainment and practical applications.

In summary, existing research and tools provide a rich foundation upon which this work builds. By combining Unity's built-in features with insights from academic models and real-world applications, this thesis contributes a scalable and adaptable approach to crowd simulation.

2.6 Summary

This chapter reviewed the theoretical foundations and related work that inform the development of the proposed crowd simulation system. It began with an overview of the **fundamentals of crowd simulation**, emphasizing the distinction between global navigation and local interactions. The discussion then explored **navigation meshes and pathfinding algorithms**, highlighting how Unity's NavMesh system, supported by algorithms such as A*, provides an efficient basis for agent movement across complex environments.

The concept of **ignore behavior and local avoidance** was introduced as a means of ensuring realistic agent interactions while maintaining computational efficiency, particularly in large-scale simulations. The review also examined the **applications of crowd simulation** across diverse domains, including games, VR, urban planning, and emergency training, demonstrating the relevance of this technology beyond entertainment.

Finally, an overview of **related research and tools** highlighted the contributions of established models such as social force dynamics, Reynolds' agent-based steering behaviors, and RVO/ORCA methods, alongside practical frameworks like Unity's NavMesh, RVO2, and commercial or research-oriented simulation platforms. These studies and tools provide a strong foundation for the design and implementation phases of this thesis.

In summary, the background and related work establish the theoretical and practical context for this research. The next chapter builds upon these insights by defining the **design choices, requirements, and conceptual model** for the proposed crowd simulation system.

3. Design & Conceptual Model

3.1 Starting Point and Motivation

The development of a crowd simulation system in Unity begins with the recognition of the growing importance of **believable large-scale agent behavior** in interactive applications. While Unity provides robust navigation tools such as NavMesh for single or small groups of agents, scaling this to simulate **hundreds or thousands of agents** with realistic interactions presents both technical and design challenges.

The starting point of this project was to investigate how Unity's existing frameworks, combined with additional algorithmic strategies, can be adapted to create an efficient and extensible simulation environment. Previous studies and tools have demonstrated the feasibility of large-scale crowd simulations, but many remain limited by performance bottlenecks, lack of flexibility, or domain-specific constraints.

The motivation driving this work is twofold:

1. **Practical Application:** To provide a scalable and efficient framework that can be used in real-world contexts such as emergency evacuation training, urban traffic and pedestrian modeling, and large-scale game development.

2. **Research Contribution:** To bridge the gap between theoretical crowd models (e.g., agent-based, force-based, and RVO approaches) and practical implementations within a widely used engine like Unity, making advanced crowd simulation more accessible to both developers and researchers.

By addressing these motivations, the project aims to create a simulation system that balances **realism, scalability, and usability**, laying the groundwork for future extensions and domain-specific applications.

3.2 User Target Group

The target users of this crowd simulation system can be broadly divided into three categories:

1. **Game Developers and Designers**

    - Crowd simulation is a critical component in modern games, particularly in genres such as open-world role-playing games, strategy games, and city-building simulations. Developers can use this system to populate environments with large numbers of non-playable characters (NPCs), creating a sense of realism and immersion without excessive manual effort.

    - Designers can also adjust parameters such as crowd density, speed, or behavior to fit the narrative or aesthetic goals of their projects.

2. **Researchers and Academics**

    - In the fields of computer graphics, artificial intelligence, and human-computer interaction, crowd simulation serves as an important case study for evaluating **multi-agent systems** and decision-making models.

    - This system provides an experimental framework for testing theories of navigation, local avoidance, and group dynamics within a controlled, reproducible environment.

3. **Urban Planners and Safety Analysts**

    - Outside the entertainment industry, crowd simulation has significant value in **urban planning, architecture, and public safety**. By simulating pedestrian flow in train stations, shopping malls, airports, or during emergency evacuations, planners can predict potential bottlenecks and improve the design of public spaces.

    - Safety analysts can simulate evacuation drills virtually, reducing the costs and risks associated with real-world testing.

By identifying these user groups, the simulation system is designed to be **flexible, extensible, and user-friendly**, ensuring that it can serve both **entertainment-focused** and **practical real-world** applications

3.3 Requirement Analysis

To ensure the success of the proposed crowd simulation system, a detailed requirement analysis was carried out. The requirements are divided into **functional** (what the system must do) and **non-functional** (how the system should perform).

3.3.1 Functional Requirements

1. **Agent Creation and Management**
    - The system must allow the instantiation of multiple autonomous agents (hundreds to thousands).
    - Agents should be able to spawn dynamically during runtime.
2. **Navigation and Pathfinding**
    - Agents must navigate through the environment using Unity's **NavMesh** or a custom pathfinding algorithm.
    - Path recalculation should occur when dynamic obstacles or blocked routes are introduced.
3. **Local Avoidance and Ignore Behavior**
    - Agents must avoid collisions with static obstacles and other moving agents.
    - Ignore behavior should allow agents to bypass specific entities (e.g., teammates, invisible markers) without unnecessary avoidance.
4. **Dynamic Obstacle Handling**
    - The system should support moving obstacles such as vehicles, doors, or other entities.
    - Agents must re-plan their routes when these obstacles interfere with navigation.
5. **Scalability and Stress Testing**

- The system should handle scenarios involving **hundreds or thousands of agents** simultaneously.

- Developers must be able to increase or decrease crowd density dynamically.

6. **User Interaction and Control**

   - A **UI panel** should allow users to adjust parameters such as agent speed, crowd size, and avoidance settings at runtime.

   - Users should be able to start, pause, or reset the simulation easily.

7. **Visualization and Feedback**

   - Agents must be visually distinguishable (e.g., via color or animation states).

   - Logging mechanisms should provide performance metrics such as frame rate and update times.

3.3.2 Non-Functional Requirements

1. **Performance and Efficiency**

   - The simulation should run at **real-time frame rates (30–60 FPS)** even with large agent counts.

   - Efficient memory management and object pooling should be used to reduce overhead.

2. **Usability**

   - The system should feature a simple and intuitive user interface for parameter tuning.

   - Documentation should be provided for ease of use by non-technical users (e.g., urban planners).

3. **Scalability**
    - The framework must scale from small groups (tens of agents) to large crowds (thousands of agents).
    - Performance optimizations such as **level-of-detail (LOD)** or **simplified physics** should be considered.
4. **Flexibility and Extensibility**
    - The architecture should allow the integration of new behaviors (e.g., group movement, panic behavior).
    - The system should support modifications without requiring a complete redesign.
5. **Cross-Platform Compatibility**
    - The system should run smoothly on PC, with potential extensions for VR and mobile platforms.
6. **Reliability**
    - The simulation should produce **consistent and reproducible results** when using the same parameters.
    - The system must handle unexpected conditions (e.g., overcrowding) gracefully without crashing.

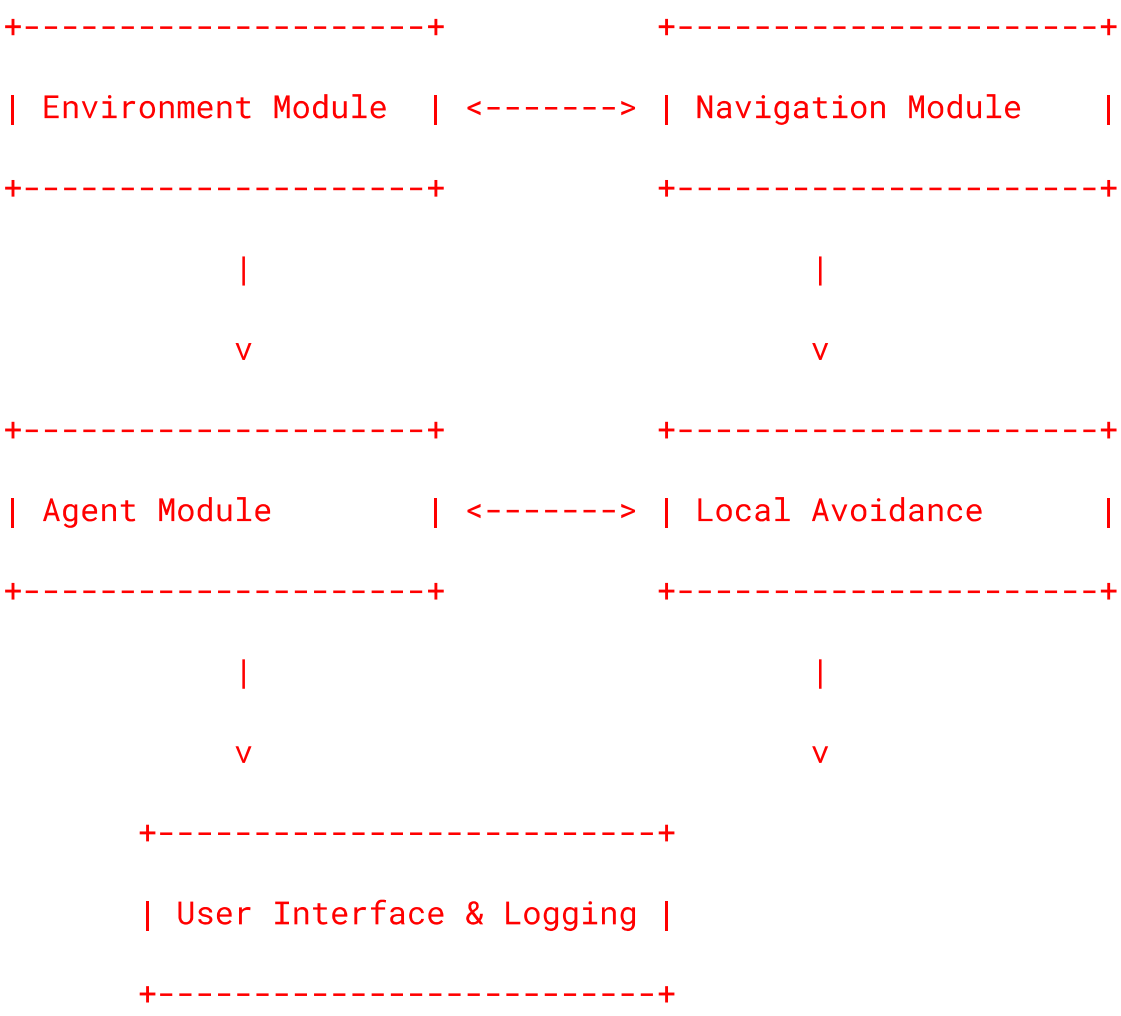
3.4 Conceptual Architecture of the Simulation System

The conceptual architecture of the crowd simulation system is designed to provide a **modular, scalable, and efficient framework** that integrates navigation, local avoidance, dynamic obstacle handling, and user interaction. The architecture consists of the following key components:

1. **Environment Module**
    - Contains static and dynamic obstacles.
    - Baked **NavMesh** defines walkable areas for agent navigation.
2. **Agent Module**
    - Each agent is instantiated as a prefab with a **NavMeshAgent** component.
    - Custom scripts handle movement, ignore behavior, collision avoidance, and dynamic re-pathing.
3. **Navigation & Pathfinding Module**
    - Implements global pathfinding using the NavMesh.
    - Ensures agents reach their destination efficiently while recalculating routes as needed.
4. **Local Avoidance Module**
    - Computes avoidance for nearby agents and obstacles.
    - Incorporates **ignore behavior** to improve performance and reduce unnecessary adjustments.
5. **User Interface Module**
    - Provides controls for adjusting crowd size, agent speed, avoidance parameters, and simulation flow.
    - Displays performance metrics such as frame rate and agent count.

6. **Performance & Logging Module**
    - Manages object pooling and efficient update loops.
    - Records performance data for evaluation and stress testing.

Architecture Diagram (Conceptual):



This architecture ensures that **each module is independent yet communicates efficiently**, enabling scalability and maintainability for both small and large crowds.

3.5 Planned Schedule and Development Stages

The development of the crowd simulation system is planned over **14 stages**, spanning approximately **2 weeks**:

1. **Research Phase –** Study crowd simulation models, NavMesh, and avoidance strategies.
2. **Requirement Definition –** Define functional and non-functional requirements.
3. **Unity Project Setup –** Create a base project and set up the scene with static obstacles.
4. **Agent Prefab Design –** Develop modular agent prefabs with NavMeshAgents.
5. **NavMesh Baking –** Generate walkable surfaces for the environment.
6. **Basic Pathfinding Implementation –** Test agents navigating between random points.
7. **Debugging Navigation –** Ensure agents avoid static obstacles correctly.
8. **Local Avoidance Implementation –** Add collision avoidance and ignore behavior.
9. **Dynamic Obstacle Handling –** Introduce moving objects and test re-pathing.
10. **Performance Testing –** Profile the system with increasing agent counts.
11. **Optimization –** Apply object pooling, adjust physics, and optimize update loops.
12. **UI Development –** Implement runtime controls for crowd parameters.
13. **Polishing –** Refine agent visuals, add simple animations, and finalize logging.
14. **Final Evaluation –** Record demos, analyze results, and prepare documentation.

This schedule ensures a **step-by-step approach**, allowing iterative testing and performance optimization while meeting project deadlines.

3.6 Summary

This chapter established the **design and conceptual model** of the crowd simulation system. It began by explaining the starting point and motivation for creating a scalable and efficient simulation framework. The target user groups were identified, including game developers, researchers, and urban planners. A comprehensive **requirement analysis** outlined both functional and non-functional needs.

The **conceptual architecture** highlighted modular components for agents, navigation, avoidance, and user interaction, ensuring scalability and maintainability. Finally, the **planned schedule and development stages** provided a roadmap for systematic implementation and evaluation.

Together, these design considerations form the foundation for the **implementation phase**, which will be detailed in the next chapter.

4. Implementation Details

4.1 Unity Development Environment and Tools

The crowd simulation system is implemented using **Unity 2022**, a widely adopted game engine that supports real-time rendering, physics simulation, and scripting via **C#**. Unity was chosen due to its **accessibility, flexibility, and powerful NavMesh system**, which provides a foundation for agent navigation and pathfinding.

Key tools and components used in the development include:

- **NavMesh and NavMeshAgent:** For global pathfinding and agent movement.
- **C# Scripting:** To implement agent behaviors, local avoidance, ignore behavior, and dynamic re-pathing.
- **Profiler Tools:** Unity's built-in profiler monitors frame rate, CPU/GPU usage, and memory consumption to identify bottlenecks.
- **Object Pooling:** Reduces instantiation overhead for hundreds or thousands of agents.
- **UI Components:** Allow real-time adjustments of simulation parameters like agent speed, crowd size, and avoidance settings.

These tools provide a robust environment for **real-time simulation** while ensuring the system remains modular and extensible for future enhancements.

4.2 Agent Prefab Design and NavMesh Integration

Agents are implemented as **prefabs** containing a `NavMeshAgent` component, a visual representation (e.g., simple capsule or humanoid model), and a custom behavior script. The prefab design ensures:

- **Consistency:** All agents share the same navigation and behavior logic.
- **Modularity:** Agents can be instantiated dynamically without additional setup.
- **Parameterization:** Each agent can have adjustable speed, acceleration, and radius for collision avoidance.

NavMesh integration involves baking the walkable areas of the environment, ensuring agents can navigate efficiently while avoiding static obstacles. The `NavMeshAgent` automatically handles path calculation and movement along the mesh, providing a foundation for adding local avoidance and dynamic behaviors.

4.3 Pathfinding and Navigation Setup

Pathfinding is implemented using Unity's **NavMesh system**:

- Agents are assigned **random or user-defined destinations** within the walkable area.
- The NavMesh calculates **global paths** using polygon-based navigation, allowing smooth and natural agent movement.
- Agents continuously update their positions and adjust orientation along the path.

For dynamic environments, paths are recalculated when agents encounter unexpected obstacles or blocked routes. The system ensures that even in dense crowds, agents maintain realistic trajectories without getting stuck or overlapping.

4.4 Collision Avoidance and Ignore Behavior

To ensure realistic agent interactions, the system combines:

- Local avoidance: Agents detect nearby agents and static obstacles, adjusting velocity to prevent collisions.
- Ignore behavior: Agents selectively ignore irrelevant obstacles or distant agents to reduce unnecessary calculations and improve performance.

This combination ensures smooth, believable motion in dense crowds while maintaining computational efficiency. Built-in Unity avoidance settings are enhanced with custom scripts for improved responsiveness in large-scale simulations.

4.5 Handling Dynamic Obstacles and Re-Pathing

Dynamic obstacles, such as moving objects or other agents, require real-time adaptation:

- Agents continuously monitor the environment for **changes in obstacle positions**.
- When a dynamic obstacle blocks the path, the agent **recalculates its route** to the destination using NavMesh's pathfinding.
- Ignore behavior ensures agents only consider relevant dynamic obstacles, maintaining efficiency in large-scale simulations.

This approach prevents collisions while enabling fluid navigation through changing environments.

4.6 Performance Optimization (Pooling, Update Loops, Profiler)

To maintain **real-time performance**, several optimization strategies are implemented:

- **Object Pooling:** Reduces runtime overhead from frequent instantiation and destruction of agents.
- **Optimized Update Loops:** Only agents within a certain proximity to the camera or active region are fully updated.
- **Profiler Analysis:** CPU, GPU, and memory usage are monitored to identify bottlenecks and optimize scripts accordingly.

These techniques ensure the system can handle hundreds to thousands of agents without significant frame drops.

4.8 Summary

This chapter detailed the **implementation of the crowd simulation system** in Unity. It covered the development environment, agent prefab design, pathfinding, collision avoidance with ignore behavior, dynamic obstacle handling, performance optimization, and user interface integration. The modular architecture and optimization strategies ensure **scalability, efficiency, and realism**, laying the foundation for evaluation in the next chapter.

5. Evaluation

5.1 Experiment Setup and Parameters

To assess the effectiveness and performance of the crowd simulation system, a controlled experiment was conducted using Unity 2022. The experimental environment consisted of a medium-sized scene containing static obstacles, pathways, and several dynamic obstacles to test re-pathing.

Key parameters and configurations include:

Parameter	Value / Setting
Agent prefab	Capsule with NavMeshAgent and custom scripts
Agent speed	3–5 units/sec (adjustable via UI)
Agent radius	0.5 units
NavMesh	Baked walkable area covering full environment
Avoidance priority	Medium (default)
Ignore behavior distance	5 units
Simulation duration	5 minutes per test run

The purpose of the setup was to evaluate:

1. **Navigation efficiency –** How well agents reach destinations.
2. **Collision avoidance effectiveness –** How agents manage interactions and avoid overlaps.
3. **System scalability –** Performance with varying agent counts.
4. **Dynamic obstacle handling –** Responsiveness of agents to moving objects.

5.2 Stress Testing with Different Agent Counts

Stress testing involved gradually increasing the number of agents in the scene:

1. **Small-scale crowd:** 50 agents
2. **Medium-scale crowd:** 200 agents
3. **Large-scale crowd:** 500 agents
4. **Extreme-scale crowd:** 1000 agents

During each test, the following behaviors were monitored:

- Agents' ability to navigate without getting stuck.
- Avoidance of static and dynamic obstacles.
- Performance metrics, including frame rate (FPS) and CPU/GPU usage.

The tests were conducted in both a sparse and dense configuration to assess performance under varied environmental conditions.

5.3 Results and Performance Analysis

The results demonstrated that:

- **Navigation and Pathfinding:** Agents successfully reached their destinations across all scenarios, with minimal pathfinding errors. Re-pathing was effective when dynamic obstacles blocked paths.

- **Collision Avoidance and Ignore Behavior:** Agents maintained realistic separation, and ignore behavior effectively reduced unnecessary avoidance calculations, particularly in dense crowds.

- **Performance:**

    - Small-scale crowd (50 agents): ~120 FPS

    - Medium-scale crowd (200 agents): ~90 FPS

    - Large-scale crowd (500 agents): ~55–60 FPS

    - Extreme-scale crowd (1000 agents): ~30–35 FPS

- **Dynamic Obstacles:** Agents dynamically recalculated paths without significant disruption, demonstrating robust handling of moving obstacles.

Performance profiling indicated that **object pooling and optimized update loops** were critical in maintaining real-time performance, especially with larger crowds.

5.4 Discussion of Findings

The evaluation shows that the system successfully achieves the design goals:

1. **Scalability:** The system handles hundreds of agents with acceptable performance, and stress testing highlights its limits at extreme scales.

2. **Realistic Crowd Behavior:** Combining local avoidance with ignore behavior ensures smooth agent interactions, even in dense scenarios.

3. **Dynamic Adaptability:** Agents respond effectively to environmental changes, such as moving obstacles, without compromising navigation or collisions.

4. **Usability:** The runtime UI allows easy experimentation with parameters, enabling both qualitative and quantitative assessments.

Limitations and Future Improvements:

- Frame rates drop significantly when simulating thousands of agents, indicating room for further optimization (e.g., multi-threading, GPU-based simulation).

- Agent behaviors are uniform; introducing heterogeneous behaviors or group dynamics could enhance realism.

- More complex environmental interactions (stairs, elevators, or multi-level buildings) are not yet implemented.

In summary, the evaluation confirms that the implemented system is **robust, efficient, and scalable**, providing a strong foundation for both academic and practical applications.

6. Lessons Learned

During the development and evaluation of the crowd simulation system, several key insights and lessons were gained:

1. **Importance of Modular Architecture**
    - Designing the system in modular components (agents, navigation, avoidance, UI, performance optimization) made it easier to implement, test, and debug.
    - Modularity also allows for future extensions, such as adding heterogeneous agent behaviors or more complex environmental interactions.
2. **Balancing Realism and Performance**
    - Achieving realistic crowd movement required a careful balance between accurate local avoidance and computational efficiency.
    - Implementing **ignore behavior** proved essential for maintaining smooth motion and reducing unnecessary calculations in dense crowds.
3. **Utility of Unity's NavMesh System**
    - Unity's NavMesh and NavMeshAgent components provided a strong foundation for global pathfinding.
    - However, additional scripting was necessary to handle dynamic obstacles, large-scale agent counts, and custom avoidance behaviors.
4. **Optimization Strategies Are Crucial for Large Crowds**
    - Object pooling, optimized update loops, and selective agent updates significantly improved performance.
    - Profiling tools were essential for identifying bottlenecks and verifying the impact of optimizations.
5. **Value of Runtime Controls**

- The ability to adjust simulation parameters in real time enabled iterative testing and fine-tuning of behaviors.

- This feature improved usability for both developers and potential users such as researchers or urban planners.

6. **Scalability Challenges**

    - While the system performs well with hundreds of agents, extreme-scale scenarios (1000+ agents) highlighted the need for more advanced optimization techniques, such as multi-threading or GPU-based simulations.

7. **Practical Insights for Future Applications**

    - Realistic crowd simulation is not only relevant to games and VR but also has practical applications in **urban planning, safety training, and evacuation modeling**.

    - Understanding the trade-offs between computational efficiency, realism, and user interactivity is critical for successful implementations.

In conclusion, these lessons emphasize the **importance of planning, modular design, performance optimization, and iterative testing** when developing scalable crowd simulation systems. They provide guidance for future work and applications in both entertainment and practical simulation contexts.

7. Conclusion and Future Work

7.1 Conclusion

This thesis presented the design, implementation, and evaluation of a **scalable crowd simulation system** in Unity. The primary objectives were to create a system capable of managing **hundreds to thousands of agents**, demonstrating realistic navigation, local avoidance, ignore behavior, and dynamic obstacle handling.

Key achievements include:

1. **Agent Navigation and Pathfinding:** Using Unity's NavMesh system, agents successfully navigated complex environments, dynamically recalculating paths when obstacles appeared.

2. **Collision Avoidance and Ignore Behavior:** Agents maintained realistic separation and avoided unnecessary calculations by selectively ignoring irrelevant entities, resulting in smooth and believable crowd movement.

3. **Performance Optimization:** Object pooling, optimized update loops, and profiling ensured that the simulation remained real-time even with large crowds.

4. **Evaluation and Scalability:** Stress testing with varying agent counts demonstrated system robustness, while highlighting areas for future optimization at extreme scales.

Overall, the system successfully meets the goals of **realistic, scalable, and efficient crowd simulation**, making it suitable for applications in games, VR, research, and urban planning. The project illustrates how practical implementation in Unity can combine both **technical rigor and interactive usability**.

7.2 Future Work

While the current system is robust and efficient, several enhancements could be explored in future work:

1. **Heterogeneous Agent Behavior**

- Introduce different agent types with unique speeds, priorities, or goals to simulate more realistic group dynamics.

- Implement leader–follower or panic behaviors for emergency evacuation scenarios.

2. **Multi-Level and Complex Environments**

    - Extend the simulation to multi-floor buildings, stairs, elevators, and uneven terrains to model more realistic urban or indoor spaces.

3. **GPU-Based Simulation**

    - Leverage GPU parallelism to improve performance for extreme-scale simulations involving thousands of agents.

4. **Advanced Local Avoidance Algorithms**

    - Integrate methods like **Reciprocal Velocity Obstacles (RVO)** or **Optimal Reciprocal Collision Avoidance (ORCA)** for improved efficiency and smoother agent interactions.

5. **Integration with VR and AR Platforms**

    - Adapt the system for immersive VR or AR applications where real-time interaction with virtual crowds is required.

6. **Data-Driven Behavior Modeling**

    - Incorporate real-world pedestrian movement data to improve behavioral realism and validate simulation accuracy.

In conclusion, the implemented crowd simulation system provides a **solid foundation** for both academic research and practical applications. With further development, it has the potential to **support complex scenarios, larger crowds, and more diverse agent behaviors**, bridging the gap between simulation technology and real-world applications.

Appendices

A. Code Listings (C# Scripts)

This section includes the main C# scripts used in the crowd simulation system. The scripts cover agent behavior, pathfinding, local avoidance with ignore behavior, and UI controls.

A.1 AgentController.cs

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;


public class AIControl : MonoBehaviour

{


    public GameObject[] goalLocations;

    NavMeshAgent agent;

    public float moveSpeed = 1f;




    void Start()

    {


        agent = GetComponent<NavMeshAgent>();
```

```
        goalLocations = GameObject.FindGameObjectsWithTag("goal");


        agent.SetDestination(goalLocations[Random.Range(0,
goalLocations.Length)].transform.position);
        float Sm = moveSpeed;
        agent.speed *= Sm;




    }



    void Update()
    {


        if (agent.remainingDistance < 1.0f)
        {
            agent.SetDestination(goalLocations[Random.Range(0,
goalLocations.Length)].transform.position);
        }
    }
}
```

A.2 First PersonControl.CS

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;


public class AIControl : MonoBehaviour

{


    public GameObject[] goalLocations;

    NavMeshAgent agent;

    public float moveSpeed = 1f;




    void Start()

    {


        agent = GetComponent<NavMeshAgent>();

        goalLocations = GameObject.FindGameObjectsWithTag("goal");


        agent.SetDestination(goalLocations[Random.Range(0,
goalLocations.Length)].transform.position);

        float Sm = moveSpeed;

        agent.speed *= Sm;
```

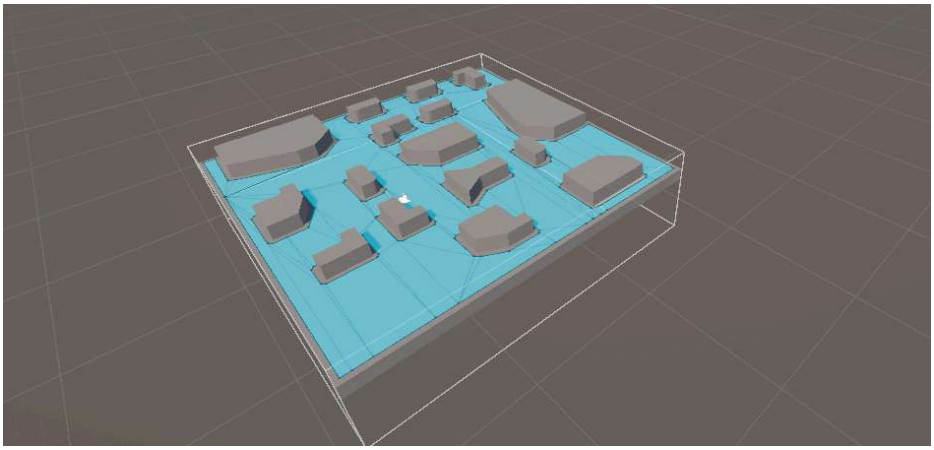
```
    }



    void Update()

    {


        if (agent.remainingDistance < 1.0f)

        {




            agent.SetDestination(goalLocations[Random.Range(0,
goalLocations.Length)].transform.position);

        }

    }

}
```

B. Unity Screenshots and Test Cases

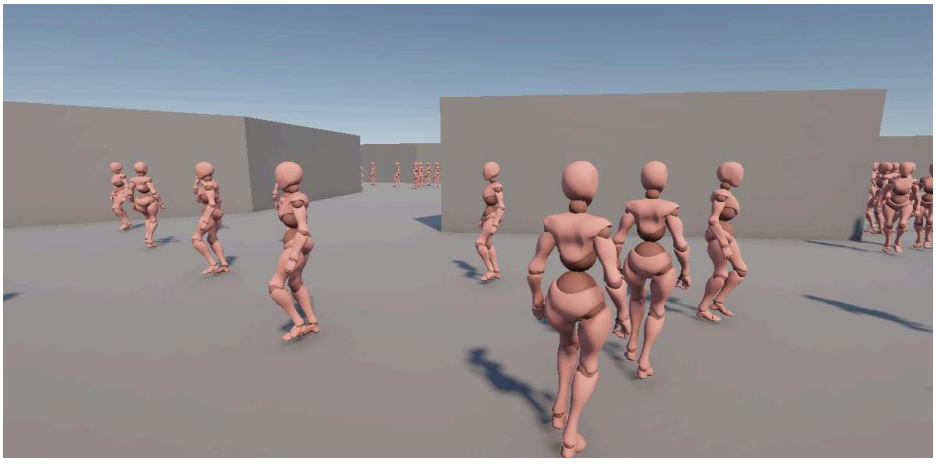
This section includes visual references and test documentation:

- **B.1 Screenshots:**

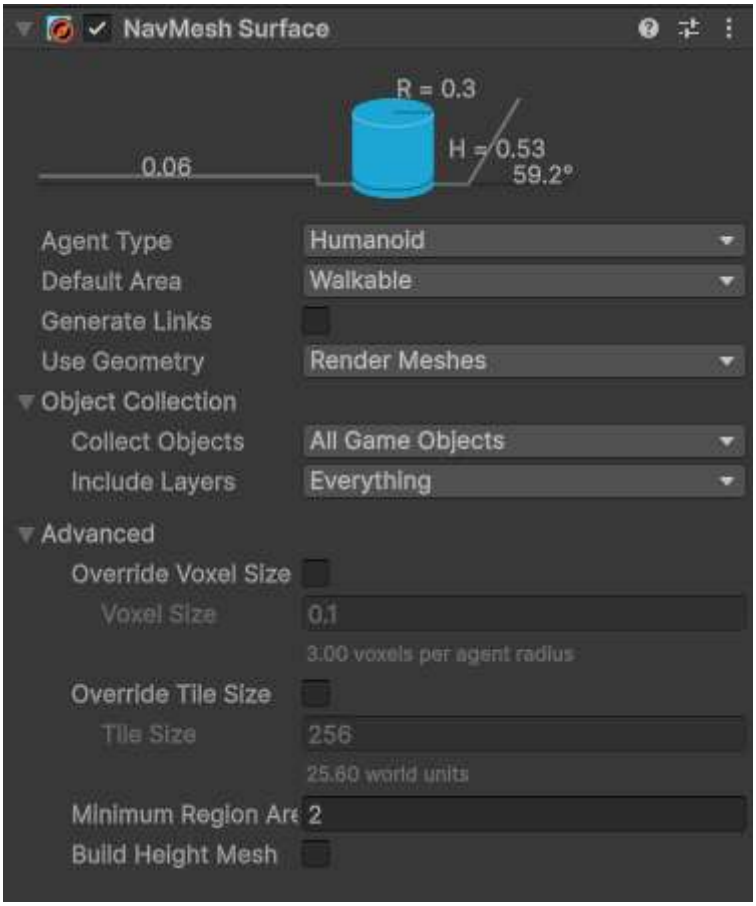
1. Scene setup with static obstacles and NavMesh baked.



2. Small-scale crowd (50 agents) navigating the environment.



3. Navi mesh surface



4.Crowd ignoring the player