Modeling the Spatiotemporal Spread of COVID-19 in Closed Transportation Environments in a Virtual Reality Setting

Bachelor Thesis

Author: Nikos Iliakis (Student ID: 4375)
Supervisor: Prof. Papagiannakis
Co-Supervisor: Prof. Kamarianakis

Department of Computer Science
School of Sciences & Engineering
University of Crete
Heraklion, Greece

Abstract

In response to the global COVID-19 pandemic, this research leverages Unity, a versatile game development engine, to implement the SEIR (Susceptible, Exposed, Infected, Recovered) model for simulating viral infections within an airport setting. We explore a range of virus containment measures, including social distancing, area-specific capacity limits, and mask mandates, evaluating their effectiveness in curtailing virus spread.

A notable aspect of our approach is the integration of crowd simulation within Unity. This allows agents within the simulation to execute custom behaviors tailored to specific areas, closely mirroring real-life dynamics in an airport environment. By incorporating crowd behavior into our simulations, we achieve a more realistic representation of how individuals move and interact.

The culmination of this study is the generation of data that enables a comprehensive evaluation of the various virus containment strategies. This data-driven approach, facilitated by Unity's capabilities, provides valuable insights for public health planning and policy formulation, particularly in the context of high-traffic locations like airports. Our research significantly contributes to the growing body of knowledge on effective measures for safeguarding public health and preventing the spread of infectious diseases in crowded environments.

Keywords: Crowd Simulation, Unity C#, Agents, 3D Airport, Virus Spread, Covid, Simulation, Data generation

Declaration of Authorship

I, Nikos Iliakis, declare that this thesis titled, "Modeling the Spatiotemporal Spread of COVID-19 in Closed Transportation Environments in a Virtual Reality Setting", and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signature: Nikos Iliakis

Date: March 31, 2024

Acknowledgements

I would like to express my gratitude to my supervising professors George Papagiannakis and Manos Kamarianakis for their continuous support and essential direction throughout the course of this project.

I would also like to thank George Kokiadis for their instructions and helpful advice.

Contents

1 Introduction

1.1 Problem Definition

The global community has witnessed a devastating toll as a result of the COVID-19 outbreak in 2019. With a staggering 7 million lives lost and counting, it is evident that this pandemic has left an unforgettable mark on societies worldwide. One of the notable epicenters of virus transmission has been international airports, which serve as vital transportation hubs connecting people from various corners of the world. This thesis attempts to address the pressing issue of virus transmission within these critical settings by providing a comprehensive tool for testing containment measures, assessing their effectiveness, and formulating strategies to mitigate the spread of the virus. Our approach leverages Unity and C# to create a versatile and powerful simulation environment tailored specifically for this purpose.

The primary objective of this research is to gain a profound understanding of how infectious diseases, such as COVID-19, propagate within the unique context of international airports. By simulating and analyzing various scenarios, we aim to develop valuable insights and recommendations that can aid policymakers, airport authorities, and public health officials in making informed decisions. Through the utilization of computer graphics and advanced modeling techniques, our work strives to contribute significantly to the improvement of airport safety and public health measures on a global scale.

1.2 Previous Related Work

Unity has been a key tool in various simulation studies, particularly those aimed at understanding crowd dynamics, which involves simulating how crowds behave in different scenarios. Prior research, exemplified by studies like [1, 2, 3, 4, 5, 6, 7], has harnessed the capabilities of 3D engines, with Unity being a popular choice, to create realistic crowd simulations.

Furthermore, Unity has also been utilized for testing epidemiological and virus transmission models, as seen in studies such as [8, 9, 10, 11]. In our study, we aim to combine elements from these various studies into a unified approach. We employ the SEIR model, which has been extensively studied in prior works [12, 13, 14, 15]

By incorporating an agent-based approach for crowd simulation, we endeavor to provide a more comprehensive understanding of virus mitigation and containment, especially in the critical context of international airports, which are considered hot-spots for virus transmission.

1.3 Thesis Overview and Novelty

This research distinguishes itself in several notable ways. Firstly, it selects the unique setting of an international airport, a choice that has not been extensively explored in previous studies. By focusing on modeling the complex airport environment, including passenger movement patterns, seating arrangements, and operational logistics, our research offers a fresh perspective on virus transmission dynamics.

Additionally, our approach introduces innovative data visualization techniques. In addition to traditional data representation through graphs and plots, we pioneer 3D data generation in the form of heatmaps. These heatmaps vividly illustrate the precise locations where infections occur within the airport, shedding light on critical areas for intervention. This novel visualization method provides a powerful tool for evaluating the effectiveness of virus containment measures, allowing for data-driven decision-making.

Finally, by combining the aspects mentioned and incorporating the SEIR model into an actual 3D Unity simulation with complex crowd dynamics, we present an innovative project.

2 Simulation Overview

In this section, we'll provide an insight into our simulation's purpose, its user-friendly interface, and the step-by-step flow from the initiation to the conclusion of the simulation. We'll also introduce the primary components of our simulation: crowd simulation, virus simulation, and data generation, which we'll explore in detail in the following sections.

2.1 User-Friendly Interface for Scenario Testing

Gone are the days when users had to wade through endless lines of code to experiment with different scenarios. We've crafted a straightforward user interface (UI) menu, making it easy for users to manipulate crucial simulation parameters.

For the crowd simulation, users can customize various factors via the first window of the UI menu (Figure 1). This includes the number of incoming flights, precise spawning times for agents, their numbers, boarding times, and assigned gates. The same goes for outgoing agents; users can tailor their parameters.

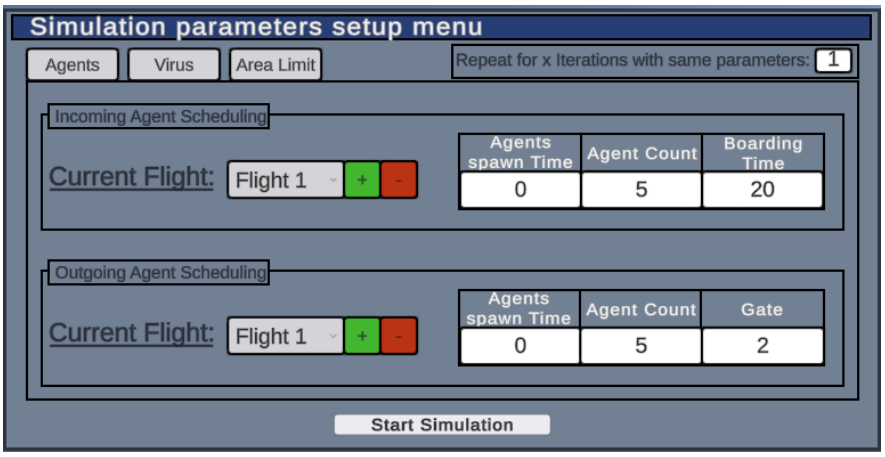

Figure 1: UI for Crowd Sim

In the virus simulation (see Figures 2,3), users can configure the virus's infectiousness, its range of infection, and the maximum number of infected agents randomly dispersed throughout the airport. Users can also easily set the specific virus countermeasures for the current run and define the number of agents in each area, facilitating social distancing.

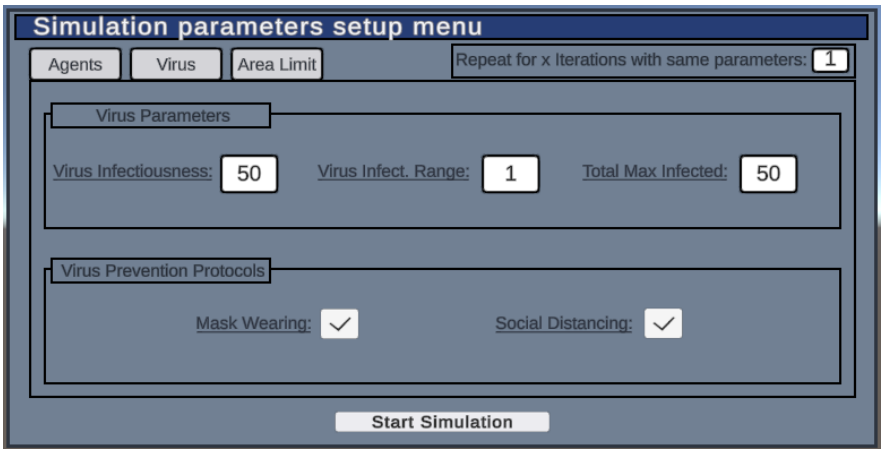

Figure 2: UI for Virus Sim

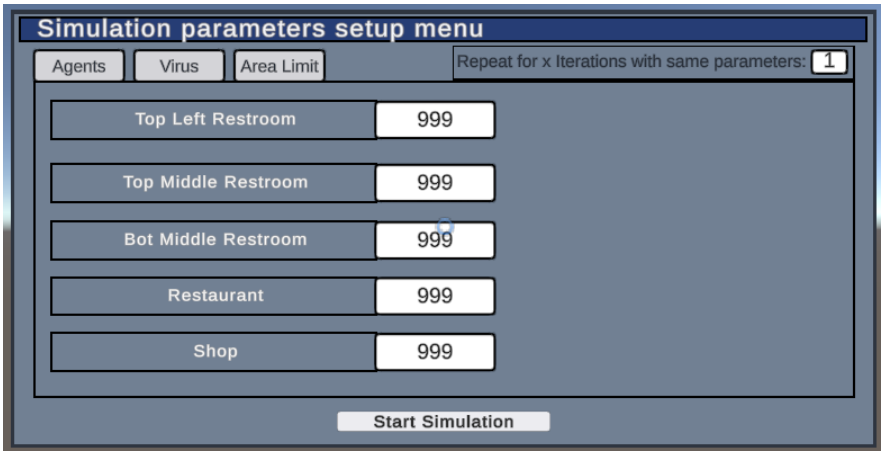

Figure 3: UI for Social Distancing (Limiting people per area)

2.2 The Simulation Flow

Let's dive into the detailed flow of our simulation, as outlined in Figure 4.

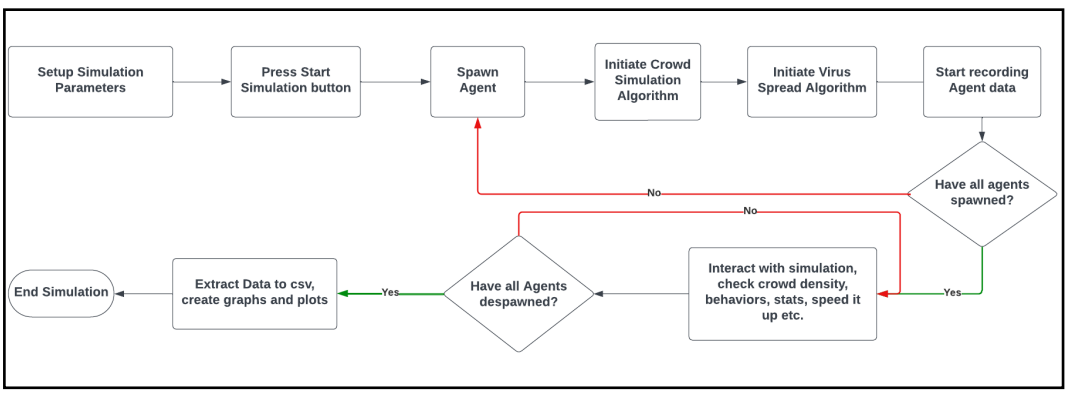

Figure 4: Simulation FlowChart

The moment the simulation is initiated, the UI menu pops up. The user then configures the simulation parameters for the Crowd Simulation algorithm and the Virus Spread Simulation algorithm. The user also specifies the number of times the simulation will run with the same parameters.

Once the user has finished setting up the parameters, the simulation begins spawning the agents. For each agent, both the crowd simulation and virus simulation algorithms are initiated. Simultaneously, the agents' data starts being recorded, which will later be exported and used to create plots, graphs, and 3D visualizations (we will delve deeper into this in the data-generation segment).

Meanwhile, as agents continue to spawn, the user can interact with the simulation. They can check statistics, observe behaviors, accelerate the simulation, and experiment with the crowd density visualization. The simulation continually checks if all agents have been despawned. When all agents have completed their tasks and have been despawned, the data is extracted into a CSV format, which will be used for subsequent visualizations.

Once the data is extracted, the simulation comes to an end.

Figure 5: In-Simulation screenshot 300 Agents

3 Crowd Simulation in an 3D virtual Airport

3.1 Airport digital twin

In our virtual airport project, we've taken inspiration from the layout of Fairbanks International Airport in Alaska. Our objective is to create a 3D digital replica, providing a platform for the exploration of COVID-19 spread dynamics and the testing of precaution strategies and to evaluate their effectiveness in various airport areas. This includes examination of confined spaces like bathrooms and high-traffic areas such as restaurants. By simulating diverse and intricate behaviors in these different areas, we aim to gain insights into the factors contributing to virus propagation in each setting.

To create this virtual airport, we've transformed the actual airport's design into a grid-based format using Unity's ProBuilder and ProGrids. This allows us to faithfully recreate the airport in 3D, offering a platform to simulate and understand real-life scenarios. Our focus is on studying how infections might propagate in different areas of the airport, shedding light on potential hot-spots and contributing to the development of effective preventive measures.

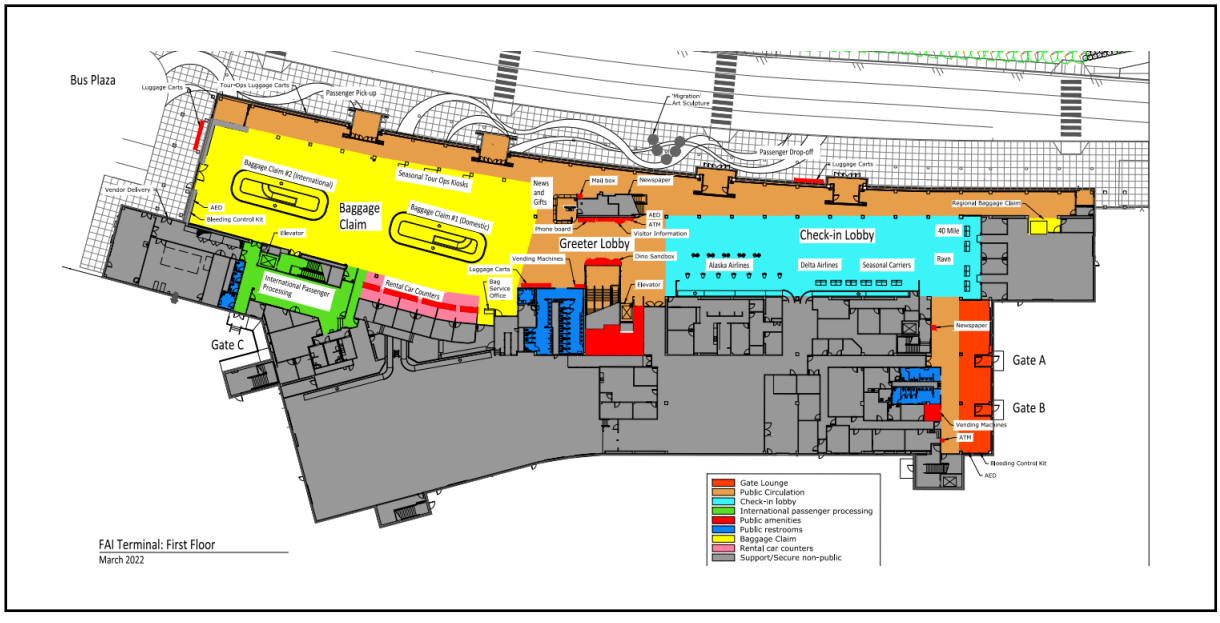

Figure 6: Fairbanks Airport Layout part 1

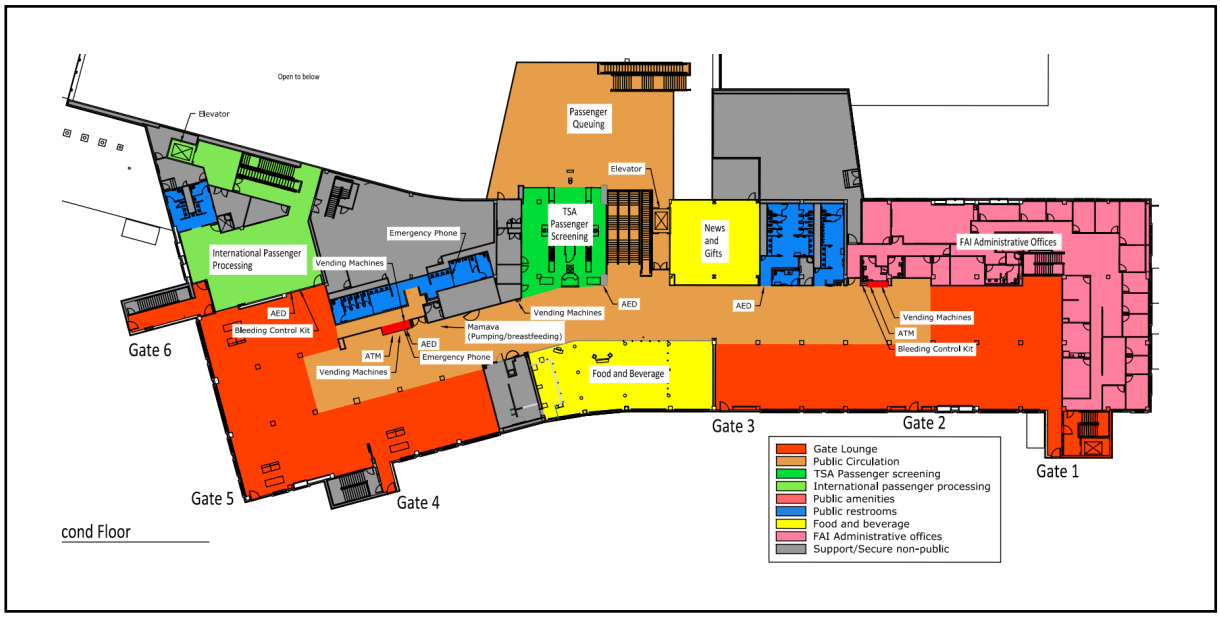

Figure 7: Fairbanks Airport Layout part 2

Figure 8: Complete virtual Airport in Unity 3D

3.2 Agent types, decision behavior tree

In our crowd simulation, we categorize agents into two primary groups: Incoming Agents and Outgoing Agents. Incoming Agents are individuals who arrive from outside the airport at a random time before their scheduled departure and intend to access the airport premises for their flight. Conversely, Outgoing Agents have just arrived inside the airport from another flight and are now looking to exit the airport.

The behavior of these agents is determined through the use of a decision behavior tree, as illustrated in Figures 9, 10. Here's a brief overview of how the decision tree operates: When an agent is spawned, a probability is assigned for each area available to that type of agent, a probability set by the user, which decides whether the agent will visit that area to perform the associated behaviors. Subsequently, the agent proceeds sequentially through each area, executing a list of tasks specific to that area. Once the agent has traversed all the areas and reached their final destination, the agent is terminated, and their data is recorded for statistical analysis.

The purpose of this algorithm is to provide a comprehensive mechanism for replicating real-life behaviors within our simulation, which is precisely what the behavior tree aims to achieve. For instance, in the bathroom area, the agent will first use an available toilet. There's a chance that they may wash their hands before leaving the area. In the baggage claim area, an agent arrives, waits for the baggage claim conveyor belt to bring their bag, collects their luggage, and proceeds to the next area to complete further tasks. Similarly, for Incoming Agents, one behavior is simply waiting in the gate area until it's time for their flight. By mimicking these real-life behaviors, we aim to gather valuable data about the transmission of COVID-19, which we will analyze in greater detail in the Virus Spread section.

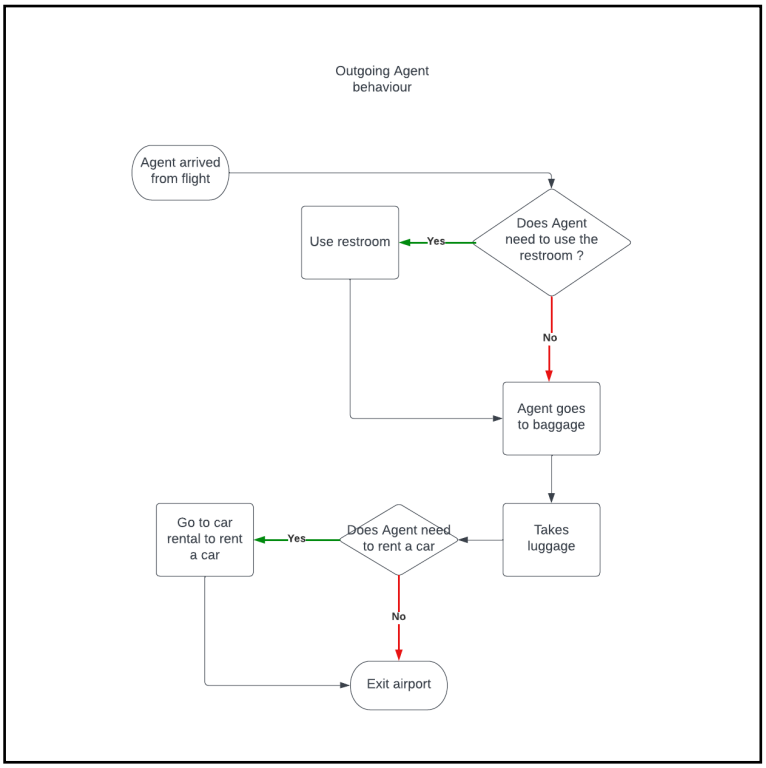

Figure 9: Tree graph for Outgoing Agents

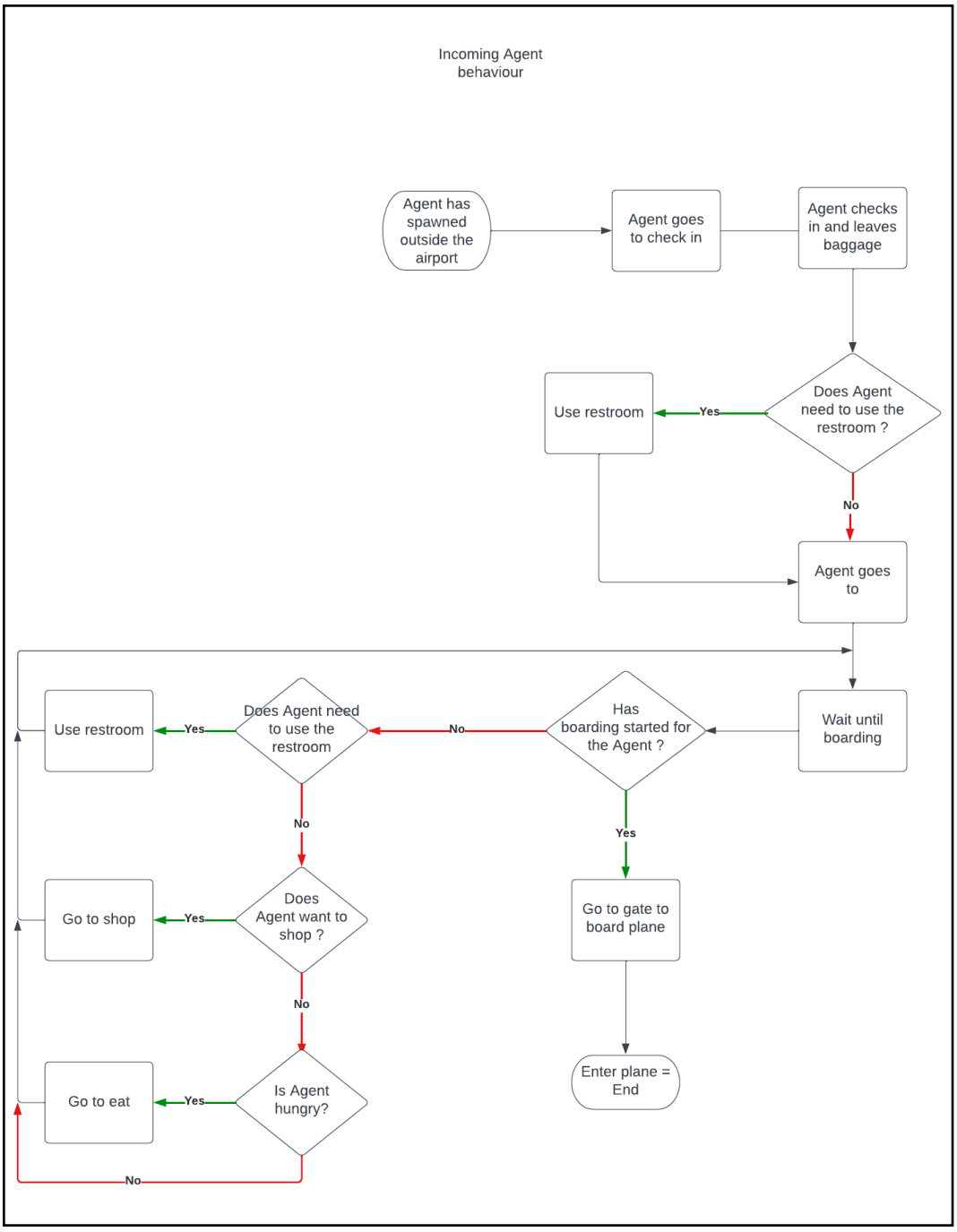

Figure 10: Tree graph for Incoming Agents

3.3 Shortest Path Algorithm, Agent Task Execution

In the previous section, we discussed how agents determine their destinations within the airport using a decision tree. Now, let's delve into the A* shortest path algorithm, a cornerstone of our simulation, which plays a pivotal role in guiding agents as they navigate the airport map and execute their tasks.

A* Shortest Path Algorithm:

The A* (pronounced "A-star") algorithm, widely recognized as a fundamental

pathfinding technique, is at the heart of our simulation. Its primary purpose is to compute the most efficient routes for agents to follow within the airport map.

To achieve this, A* considers a network of nodes and edges, effectively creating a graph of the airport environment. It relies on a heuristic function to estimate the cost from the agent's current position to the destination. By weighing the actual cost incurred so far against the estimated cost to reach the goal, the algorithm identifies the most promising path. This, in turn, ensures that agents reach their assigned tasks in the shortest possible time.

Now, let's examine how this pathfinding algorithm integrates with the agents' task execution process.

Once the decision tree assigns a destination to an agent, a list of tasks is generated for the agent to execute. Each task involves a random amount of time for completion, mimicking real-life activities and enhancing the authenticity of the simulation. The A* algorithm effectively directs agents throughout the airport to accomplish these tasks.

Additionally, a random waiting time is introduced before the completion of each task. This introduces nuance and variability in the time agents spend on their tasks, contributing to the realism of the simulation.

4 Virus Spread

4.1 SEIR model implementation in Unity

In our pursuit to model the spatiotemporal spread of COVID-19 in a closed transportation environment, we employed a modified version of the SEIR model within our Unity simulation. The SEIR model, which stands for Susceptible-Exposed-Infectious-Recovered, is a widely used mathematical framework for understanding the dynamics of infectious diseases. However, to adapt it to our specific scenario and make it more accessible to our virtual environment, we introduced a few simplifications and visual cues.

- **Simplified SEIR Model:**

  In our simulation, we tailored the SEIR model to our needs. We simplified it to focus primarily on the transition from the "Susceptible" state to the "Exposed" state. This simplification is motivated by the time constraints within the airport simulation, where the recovery phase doesn't align with the scenario's time frame.

- **Visual Representation:**

  To enhance the clarity and visual appeal of our simulation, we assigned distinctive colors to agents based on their disease states. Agents in the "Susceptible" state are represented in blue, "Exposed" individuals in yellow, and "Infected" agents in red. This color-coding system provides an intuitive visual representation of the disease dynamics and aids in tracking and observing the progression from susceptibility to exposure.

- **Rationale:**

  Our choice to focus on the transition from susceptibility to exposure aligns with the main objective of our research – understanding how infections can occur within the airport environment. By observing and tracking agents as they move from the "Susceptible" to "Exposed" state, we gain valuable insights into potential virus transmissions and critical points for implementing preventive measures.

  This adaptation of the SEIR model, coupled with the visual representation of agents, enables us to effectively study and simulate COVID-19 spread within the virtual airport environment. It streamlines the model to suit our research objectives while maintaining a clear and informative depiction of the disease's progression.

4.2 Virus Spread Algorithm

As mentioned in the previous section, the algorithm for this simulation is based on the SEIR model. Therefore, the simulation involves two types of agents in terms of virus state: Susceptible and Infected. The main parameters of the virus can be set by the user through a UI menu, as discussed in detail in Chapter 2.1. These parameters include the infectiousness of the virus, the range at which the virus can be transmitted, and the number of infected agents within the simulation.

The algorithm operates as follows: Each time an agent is spawned, there is a chance that they will be spawned in either the Susceptible or Infected state. This process continues until the maximum number of infected agents for the simulation is reached. Afterward, only Susceptible agents will be spawned. Subsequently, agents will move through the airport, completing their tasks as discussed in Section 3. Whenever a susceptible agent is within range of an infected agent, a probability is calculated. If the generated number is smaller than the InfectionChance, the agent's state is switched from Susceptible to Exposed.

InfectionChance is calculated as follows:

- First, we calculate the value of L, which is based on the distance and the virus range:
$$L = \frac{\text{distance} - \text{VirusRange}}{\text{distance}}$$
- Finally, we determine the InfectionChance by dividing the L value by the infectiousness of the virus:
$$\text{InfectionChance} = \frac{L}{\text{VirusInfectiousness}}$$

These are the basic functions for calculating the chance of transmitting the virus, and you can further modify the effect by multiplying the result with Mask or Antiseptic coefficients

4.3 COVID-19 Mitigation Measures

In our simulation, we have incorporated a range of countermeasures to assess their effectiveness and gain valuable insights into strategies for mitigating virus transmission within the dynamic environment of an international airport. Based on studies [2, 16, 17] the aim is to understand how these measures can collectively contribute to the reduction of virus spread and, ideally, inform strategies for minimizing transmission in high-risk settings like airports.

- **Mask-Wearing**

One key measure we've taken is ensuring that everyone wears masks. When this rule is in place, it significantly reduces the spread of the virus in the airport. In our simulation, all individuals are required to wear masks, and this greatly reduces the virus transmission. When masks are worn, we increase the risk of infection by 20-30%.. This measure becomes even more interesting when combined with other safety measures, which we'll explore further in the upcoming Data-Generation section.

- **Social Distancing and Crowd Control**

  This safety measure includes two important aspects: maintaining social distance and controlling the number of people in specific areas of the airport. By enforcing social distancing, individuals must keep a certain distance from each other in designated areas. This means they can't sit or stand too close to one another. Additionally, users can set limits on the number of people allowed in each area, effectively controlling how many individuals are in one place. This helps us understand how spreading people out in different areas can influence virus transmission dynamics.

- **Hand Sanitizers in Key Locations**

  To enhance our safety measures, we've strategically placed hand sanitizers in crucial spots within the virtual airport environment. These sanitizers play a vital role in reducing the likelihood of virus transmission. Studies have shown that having hand sanitizers in high-traffic areas significantly decreases the risk of virus transmission, reducing the chances of getting infected as effective sanitizer usage reduces the risk near 0. By strategically positioning hand sanitizers where individuals are most likely to use them, we directly decrease their chances of coming into contact with the virus. This measure adds an additional layer of protection to our simulation, making it more realistic and comprehensive in representing virus transmission dynamics within the airport.

In section 5 , we will delve deeper into the combined effects of these countermeasures and provide a thorough analysis of the data collected during the simulations.

5 Visualizations and Data Generation

5.1 Visualization of Data

Given that this project is developed within the Unity 3D simulation environment, the utilization of 3D visualizations plays a vital role in comprehending the dynamics of our simulation.

- **Crowd Density Shader:**

  To gain insights into agent clustering, a key factor in understanding the potential spread of the virus, we implemented a Crowd Density visualization. This visualization serves a dual purpose: it aids in identifying high-density agent areas, crucial for comprehending virus transmission patterns, and facilitates the identification and resolution of simulation-related issues. The Crowd Density shader operates by processing agent positions, assigning cooler colors (light blue) to areas with lower agent density and hotter colors (red) to areas with higher agent density.

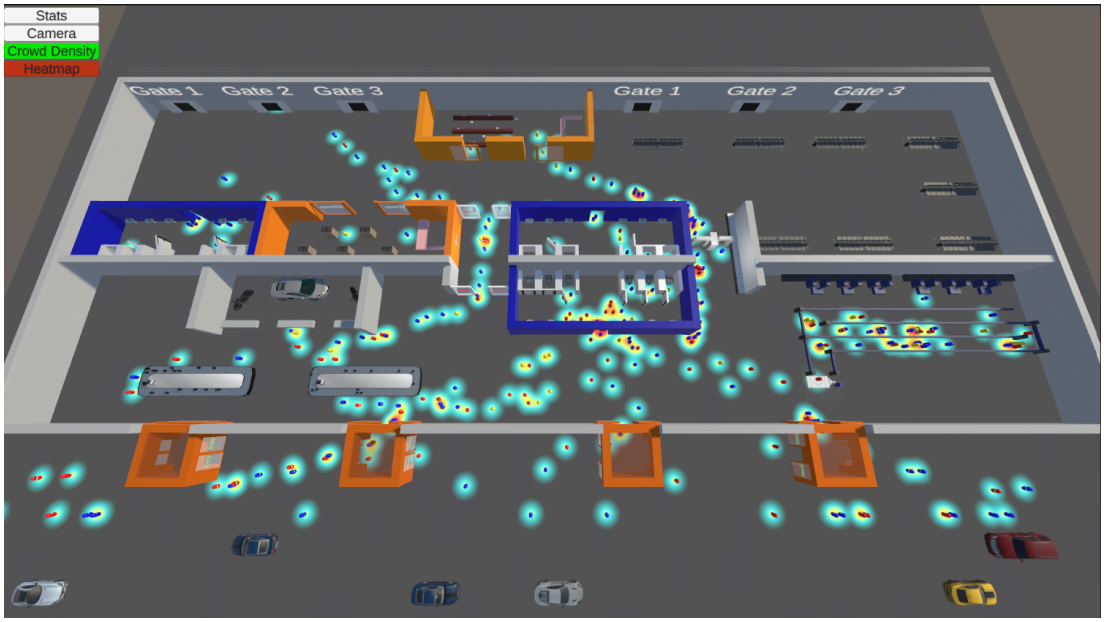

Figure 11: In-Simulation Screenshot of Crowd Density

- **Heatmap:**

  In our pursuit of pinpointing infection locations within our 3D airport, we introduced the Heatmap visualization—a novel addition to the field of crowd simulation. The Heatmap provides a detailed visualization of where infections occur, offering invaluable insights into the effectiveness of virus mitigation strategies. This visualization was implemented using Unity's particle system.

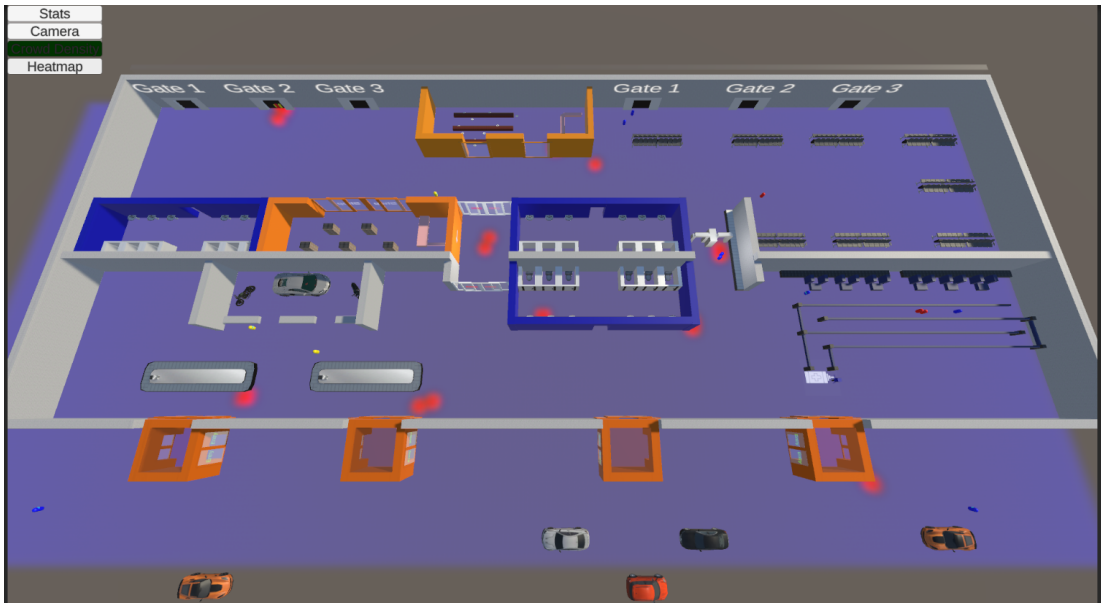

Figure 12: Heatmap Visualization of Infection Locations

5.2 Experiments

In our exploration of various scenarios within the COVID simulation, three experiments emerged as particularly notable.

Firstly, we implemented a straightforward scenario where only basic COVID countermeasures were enforced. This included average infectiousness of the virus and mandatory mask-wearing. Despite these measures, the number of individuals exposed to the virus continued to rise, albeit reaching a cap at 140 agents.

In the second experiment, we opted for more stringent countermeasures. Mandatory mask-wearing persisted, but we also enforced social distancing protocols by limiting the number of individuals per area to 4-7, depending on area size. Additionally, multiple antiseptics were strategically placed in crucial spots to further mitigate virus spread and encourage agent use. Remarkably, these measures resulted in a significant 48

Having tested these countermeasures both individually and in combination, we proceeded to compare their effectiveness against a scenario with no countermeasures at all. The results of our third experiment underscored the stark contrast: a 121% and 242% increase in the number of exposed individuals compared to experiments 1 and 2, respectively.

While these numerical outcomes may not perfectly mirror real-world circumstances, they nonetheless provide valuable insights into the effectiveness of COVID countermeasures and the potential for assessing them within game engines.

- **Experiment 1, Average Infectiousness, MASK-WEARING ONLY:**

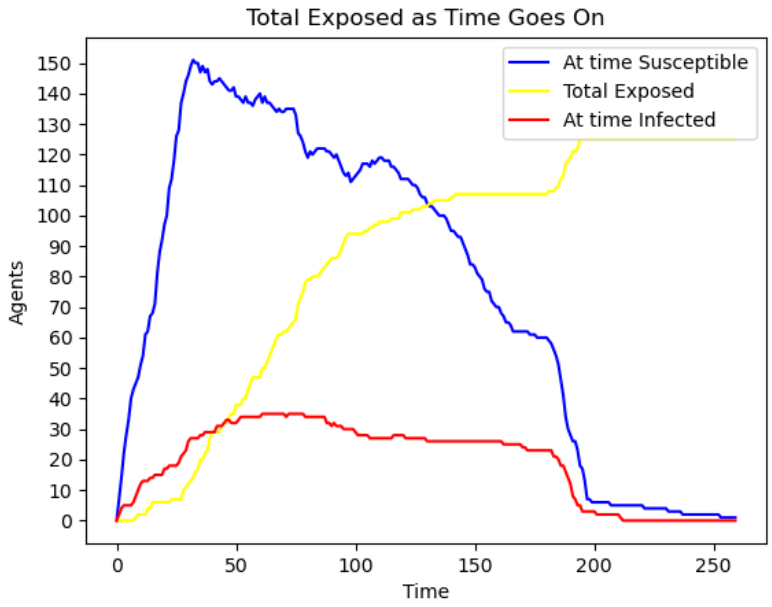

Figure 13: Experiment 1 Plot

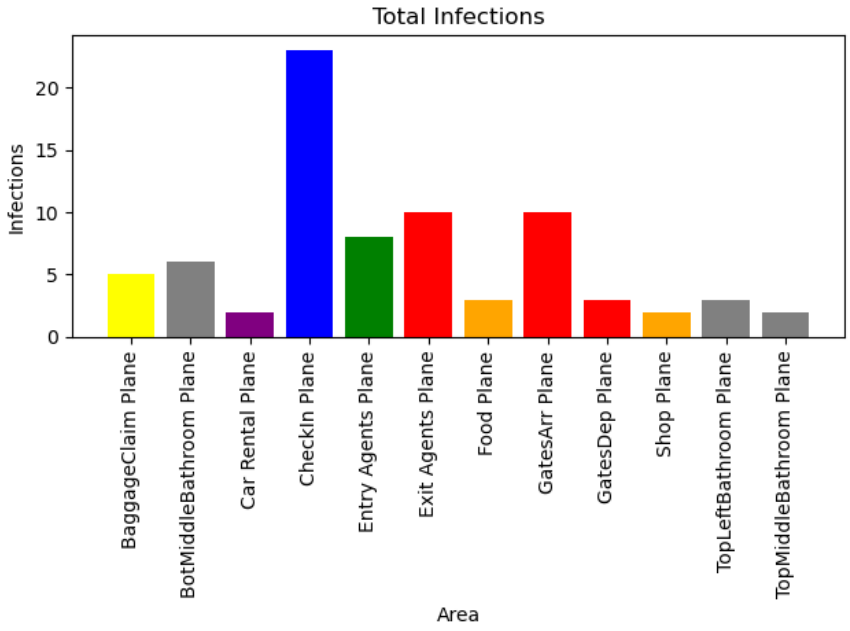

Figure 14: Experiment 1 Plot

- **Experiment 2, Average Infectiousness, MASK-WEARING ONLY + SOCIAL DISTANCING + ANTISEPTICS:**

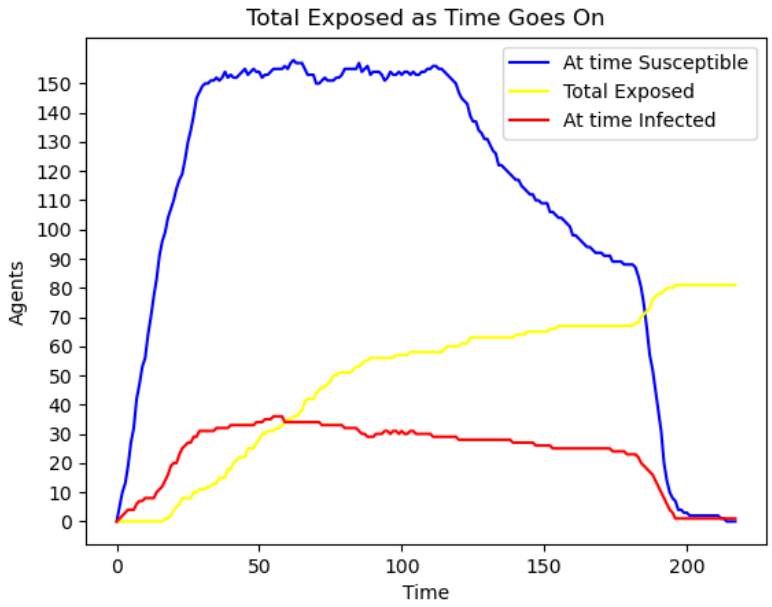

Figure 15: Experiment 2 Plot

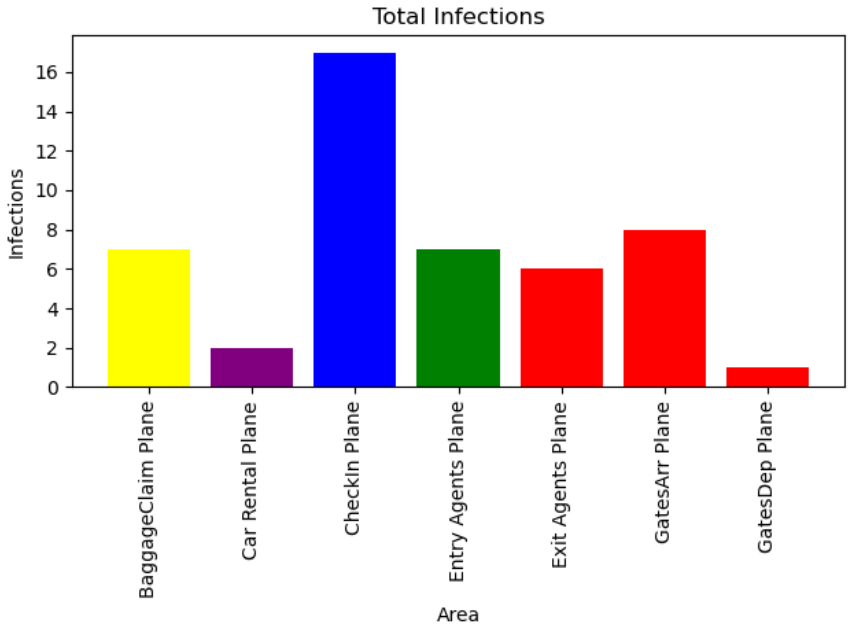

Figure 16: Experiment 2 Plot

- **Experiment 3, Average Infectiousness, No Covid Countermeasures**

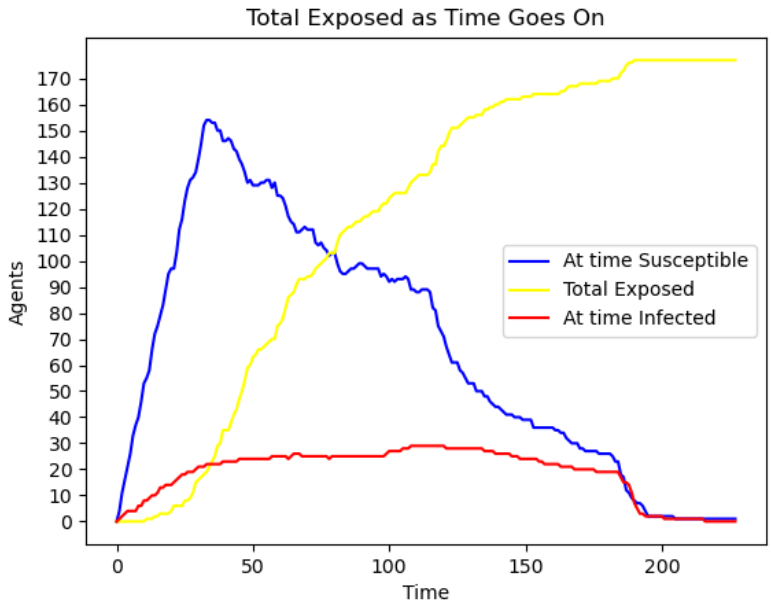

Figure 17: Experiment 3 Plot

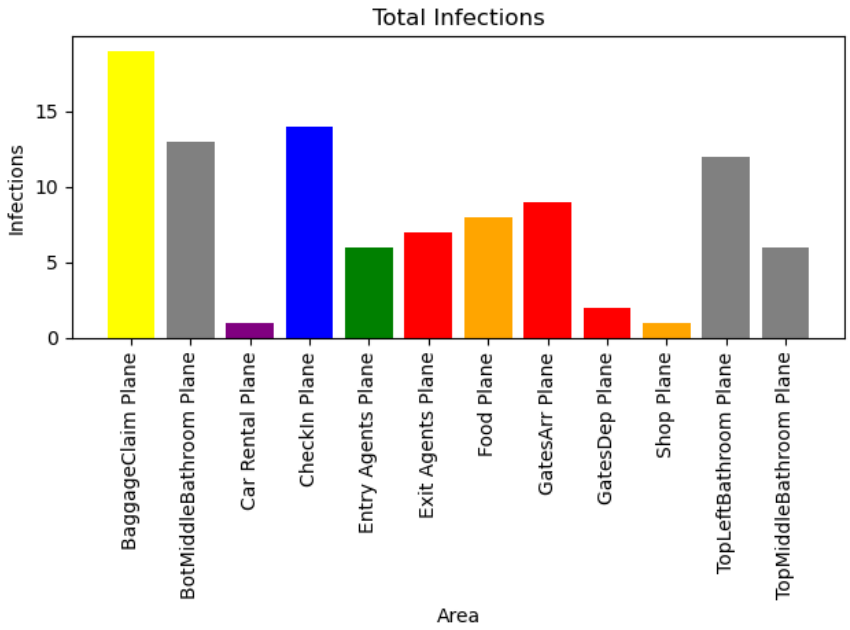

Figure 18: Experiment 3 Plot

6 Limitations and Future Work

In this section, we discuss the limitations of our project and potential avenues for future research.

6.1 Limitations

- **Airport Environment 3D Map**: While our project focuses on simulating an airport environment, there is room for enhancing the map design. Creating a bigger more realistic 3D virtual airport would enable to spawn more agents and have more realistic results.
- **Agent Behavior Nuance**: Although our AI algorithm is robust, introducing more randomness to the shortest path algorithm and incorporating complex behaviors, as well as indirect virus transmission modes, can add nuance and realism to the simulations.
- **Hardware Constraints**: The project's performance is dependent on hardware limitations, typically accommodating 300-450 agents for optimal operation. Future work could explore optimizations to handle larger scenarios or adapt to various hardware configurations.

6.2 Future Work

As we consider future work, we can explore these directions:

- **More agents**: Adding more types of agents including airport staff would greatly increase the realism and the accuracy of the results.
- **Dynamic Pathfinding**: Implementing dynamic pathfinding algorithms that adjust in response to real-time data and evolving conditions within the simulated environment.
- **Complex Virus Spread Models**: Expanding the virus spread model to include multiple modes of transmission, going beyond agent-to-agent contact, for example implementing air quality for open and enclosed areas.
- **Immersive Visualization**: Exploring more immersive and visually engaging data representations, including the integration of virtual reality-based visualizations for enhanced insights.
- **Large-Scale Scenarios**: Adapting the model for larger and more complex environments, such as busy city centers, public events, and other crowded settings.

By addressing these limitations and pursuing future research directions, we aim to create a more comprehensive and sophisticated simulation that offers valuable insights into virus transmission dynamics within crowded environments like airports.

7 Conclusion

In conclusion, our experiment using game engines to simulate crowd dynamics and virus spread within a 3D virtual airport has been quite revealing. We've used techniques like the A* algorithm to guide virtual people efficiently through the airport, making the simulation feel more realistic. Also, we've simplified a widely used disease model, the SEIR model, and added colors to represent different stages of infection. This helps us see how diseases might spread in places like airports.

We've also tested out different ways to reduce the spread of the virus, like wearing masks, social distancing, and having hand sanitizers around. These measures seem to work well in our virtual airport, showing promise for real-world applications.

Our visualizations, like the crowd density map and heatmap of infection locations, have given us a clearer picture of how the virus might spread in crowded places.

While our experiment has shown some interesting results, there are still some limitations. We could improve the design of the airport map, make the behavior of virtual people more realistic, and make the simulation work better with more virtual people.

Overall, this experiment shows how we can use game engines to study real-world problems like virus transmission. By continuing this research, we hope to find even better ways to prevent the spread of diseases in places like airports.

8 References

[1] Alicia Nic'as Miquel. Development of crowd simulation models using unity for immersive vr applications. Master's thesis, Universitat Politecnica de Catalunya, 2019.

[2] Ali Khumaidi. Crowd modelling and navigation in unity3d game engine: Crowd modelling and navigation in unity3d game engine. 2020.

[3] Songqiao Sun. Agent-based crowd simulation modelling for a gaming environment. Master's thesis, University of Windsor, 2017.

[4] Linbo Luo andSuiping Zhou, Wentong Cai, and Malcolm Y. H. Low Feng Tian andYongwei Wangand Xian Xiaoand Dan Chen. Agent-based human behavior modeling for crowd simulation. 2008.

[5] Walter Alan Cantrell, Mikel D. Petty, Samantha L. Knight, and Whitney K. Schueler. Physics-based modeling of crowd evacuation in the unity game engine. 2018.

[6] Walter Alan Cantrell, Mikel D. Petty, Samantha L. Knight, and Whitney K. Schueler. Autonomous agents in 3d crowd simulation through bdi architecture. 2021.

[7] Abhishek Nandy. Crowd simulation as a flocking behavior and windows mixed reality. 2018.

[8] Jordan Fernandes, Jack Li, Keye Li, Joseph Mirabile, and Gregg Vesonder. Exploring a plugin-based system for crowd and pathogen simulation in unity. 2021.

[9] Dina Jamal Hejji, Omar Mohammed Gouda, and Mohamamd S. Obaidat. Simulation and evaluation of precaution strategies for covid-19 pandemic: A case study. 2021.

[10] James Fort, Adam Crespi, Chris Elion, Rambod Kermanizadeh, Priyesh Wani, and Danny Lange. Exploring new ways to simulate the coronavirus spread. 2020.

[11] Ege Hosgungor. Create an epidemic simulation in unity i: Simulating an epidemic spread. 2020.

[12] Maheswari Rangasamy, Christophe Chesneau, Carlos Martin-Barreiro, and Víctor Leiva. On a novel dynamics of seir epidemic models with a potential application to covid-19. 2022.

[13] R. Devipriya, S. Dhamodharavadhani, and Selvi. S. Seir model for covid-19 epidemic using delay differential equation. Master's thesis, Periyar University, Salem, India, 2020.

[14] Sarah A. Al-Sheikh. Modeling and analysis of an seir epidemic model with a limited resource for treatment. 2012.

[15] Subrata Paul, Animesh Mahata, Uttam Ghosh, and Banamali Roy. Study of seir epidemic model and scenario analysis of covid-19 pandemic. 2019.

[16] Leah Boulos, Janet A. Curran, Allyson Gallant, Helen Wong, Catherine Johnson, Alannah Delahunty-Pike, Lynora Saxinger, Derek Chu, Jeannette Comeau, Trudy Flynn, Julie Clegg, and Christopher Dye. Effectiveness of face masks for reducing transmission of sars-cov-2: a rapid systematic review. 2023.

[17] Lufuno Muleba, Renay Van Wyk, Jennifer Pienaar, Edith Ratshikhopha, and Tanusha Singh. Assessment of anti-bacterial effectiveness of hand sanitizers commonly used in south africa. 2022.

A Appendix

Most crucial pieces of code.

A.1 Crowd Simulation:

ArrivingAgentBT

```
using BehaviorTree;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class ArrivingAgentsBT : BehaviorTree.Tree
{
    public NavMeshAgent navMeshAgent;
    public enum AgentAction { None, CheckIn, Restroom, Shop, Eat, WaitingUntilB
    public AgentAction Action;

    public ArrivingAgentsSpawner.ArrivingAgentSettings agentSettings;
    public int EntryGateNumber;
    public bool TimeToBoard = false;


    protected override Node SetupTree()
    {
        AgentData agentdata = gameObject.GetComponent<AgentData>();
        Node root = new Selector(new List<Node>
            {
                new Boarding(navMeshAgent, EntryGateNumber, gameObject),
                new Sequence(new List<Node>
                {
                    new CheckIfWeActivateBehavior("BathroomBool1", agentSetting
                    new BotMiddleBathroom(navMeshAgent, "BathroomBool1", agento
                }),
                new Sequence(new List<Node>
                {
                    new CheckIfWeActivateBehavior("CheckInBool", agentSettings.
                    new CheckIn(navMeshAgent,"CheckInBool")
```

```
                }),
                new Sequence(new List<Node>
                {
                    new CheckIfWeActivateBehavior("BathroomBool2", agentSetting
                    new TopMiddleBathroom(navMeshAgent, "BathroomBool2", agento
                }),
                new Sequence(new List<Node>
                {
                    new CheckIfWeActivateBehavior("ShopBool", agentSettings.Cha
                    new Shop(navMeshAgent, "ShopBool", agentdata)
                }),
                new Sequence(new List<Node>
                {
                    new CheckIfWeActivateBehavior("EatBool", agentSettings.Char
                    new Eat(navMeshAgent, "EatBool", agentdata)
                }),
                new WaitingUntilBoard(navMeshAgent, agentdata)
            });

        return root;
    }
}
```

ArrivingAgentsScheduler

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEditor;
using UnityEngine;

public class ArrivingAgentsScheduler : MonoBehaviour
{
    private int TotalFlights = 0;
    private float StartingTime = 0f;
    private List<OutgoingFlight> FlightList;
    private ArrivingAgentsSpawner AgentSpawner;
    private SimulationData sd;
```

```
public int getTotalFlights() { return TotalFlights; }
public float getStartingTime() { return StartingTime; }
public List<OutgoingFlight> getFlightList() { return FlightList; }

float GetTimeDelay(float AgentStartArrivingTime)
{
    return AgentStartArrivingTime - StartingTime;
}
public void InitiateAllFlights()
{
    foreach (OutgoingFlight flight in FlightList)
    {
        StartCoroutine(InitiateFlight(flight));
    }
}

public OutgoingFlight GenerateFlight(float agentStartArrivingTime, int ager
{
    TotalFlights++;
    OutgoingFlight newFLight = new(agentStartArrivingTime, agentNumber, boa
    FlightList.Add(newFLight);
    sd.IncreasedTotalFlightsGenerated();
    return newFLight;
}
private IEnumerator InitiateFlight(OutgoingFlight flight)
{
    yield return new WaitForSeconds(GetTimeDelay(flight.AgentStartArrivingT
    AgentSpawner.SpawnAgents(flight.AgentNumber, sd.StartingTime + flight.F
    sd.IncreasedTotalFlightsInitiated();
}
public void Init()
{
    StartingTime = Time.time;
    AgentSpawner = FindObjectOfType<ArrivingAgentsSpawner>();
    FlightList = new List<OutgoingFlight>();
    sd = FindAnyObjectByType<SimulationData>();
    sd.StartingTime = StartingTime;
}
```

```
    public void Reset()
    {
        TotalFlights = 0;
        StartingTime = sd.StartingTime;
    }

}
```

ArrivingAgentsSpawner

```
using System;
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using Random = UnityEngine.Random;

public class ArrivingAgentsSpawner : MonoBehaviour
{

    public GameObject agentPrefab; // Prefab of the agent to spawn
    public GameObject Parent;
    int agentGate;

    [Serializable]
    public class ArrivingAgentSettings
    {
        [Range(0f, 1f)]
        public float ChanceToUseRestroom;
        [Range(0f, 1f)]
        public float ChanceToCheckIn;
        [Range(0f, 1f)]
        public float ChanceToShop;
        [Range(0f, 1f)]
        public float ChanceToEat;
    }
    public ArrivingAgentSettings AgentSettings = new();

    private GameObject CrowdDensity;
```

```
SimulationData sData;

// Start is called before the first frame update
void Start()
{
    GameObject Visualizations = GameObject.Find("Visualizations");
    CrowdDensity = Visualizations.transform.Find("CrowdDensity").gameObject
    sData = FindObjectOfType<SimulationData>();
}

public void SpawnAgents(int AgentNumber, float BoardingTime, int FlightNumb
{
    StartCoroutine(AgentsArriving(AgentNumber, BoardingTime, FlightNumber))
}

void SpawnAgent(List<GameObject> agents, int FlightNumber, int i)
{
    Vector3 AgentPos = transform.position;
    AgentPos.x += Random.Range(-80f, 80f);
    GameObject newAgent = Instantiate(agentPrefab, AgentPos, Quaternion.ide
    agents.Add(newAgent);

    //Set Parent for agents so that its organized
    newAgent.transform.parent = Parent.transform;

    //Agent Number and Flight Number
    newAgent.name = "FlightNo" + FlightNumber.ToString() + "_AgentNo" + i.T

    //If crowd shader is active we need to detect collisions not triggers
    if (CrowdDensity.activeInHierarchy) newAgent.GetComponent<CapsuleCollid



    //Set color for this agent so all agents of this flight have the same c
    //Renderer agentRenderer = newAgent.GetComponent<Renderer>();
    //agentRenderer.material.color = randomColor;

    //Pass the settings to the agent
    ArrivingAgentsBT agentScript = newAgent.GetComponent<ArrivingAgentsBT>(
    agentScript.EntryGateNumber = agentGate;
```

```
        agentScript.agentSettings = AgentSettings;

        //Add agent to simulation data for general use
        sData.GetAirportData().InsertNewIncomingAgent(newAgent);
    }

    private IEnumerator AgentsArriving(int AgentNumber, float BoardingTime, int
    {

        //Color randomColor = Random.ColorHSV();
        List<GameObject> agents = new List<GameObject>();
        agentGate = Random.Range(1, 4);
        int i = 0;
        while (i < AgentNumber) {

            int j = Random.Range(2, 4);
            if(i + j < AgentNumber)
                for (int k = 0; k < j; k++) SpawnAgent(agents, FlightNumber, i+
            else
                SpawnAgent(agents, FlightNumber, i++);



            //Wait a little bit until next Agent
            float spawnDelay = Random.Range(0f, 1f);
            yield return new WaitForSeconds(spawnDelay);
        }

        float delay = BoardingTime - Time.time;
        if (delay > 0) yield return new WaitForSeconds(delay); //Wait X time un

        foreach (GameObject agent in agents)
        {
            ArrivingAgentsBT agentScript = agent.GetComponent<ArrivingAgentsBT>
            agentScript.TimeToBoard = true;
        }

        agents.Clear();
    }
```

```
}
```

OutgoingFlight

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OutgoingFlight
{
    public float AgentStartArrivingTime;
    public float BoardingTime;
    public int AgentNumber;
    public int FlightNumber;

    public OutgoingFlight(float agentStartArrivingTime, int agentNumber, float
    {
        AgentStartArrivingTime = agentStartArrivingTime;
        AgentNumber = agentNumber;
        BoardingTime = boardingTime;
        FlightNumber = flightNumber;
    }
}
```

DepartedAgentBT

```
using BehaviorTree;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class DepartedAgentsBT : BehaviorTree.Tree
{
    public NavMeshAgent navMeshAgent;
    public enum AgentAction { None, Restroom, BaggageClaim, CarRental, ExitAirp
    public AgentAction Action;

    public DepartedAgentSpawner.DepartedAgentSettings agentSettings;
```

```
    protected override Node SetupTree()
    {
        AgentData agentdata = gameObject.GetComponent<AgentData>();
        Node root = new Selector(new List<Node>
            {
                new Sequence(new List<Node>
                {
                    new CheckIfWeActivateBehavior("BathroomBool", agentSettings
                    new TopLeftBathroom(navMeshAgent, "BathroomBool", agentdata
                }),
                new Sequence(new List<Node>
                {
                    new CheckIfWeActivateBehavior("RandomizePath", 1f),
                    new RandomizePath(navMeshAgent, "RandomizePath")
                }),
                new Sequence(new List<Node>
                {
                    new CheckIfWeActivateBehavior("BaggageBool", agentSettings.
                    new Baggage(navMeshAgent, "BaggageBool", gameObject)
                }),
                new Sequence(new List<Node>
                {
                    new CheckIfWeActivateBehavior("CarBool", agentSettings.Char
                    new Car(navMeshAgent, "CarBool")
                }),
                new Exit(navMeshAgent, gameObject)
            });

        return root;
    }
}
```

DepartedAgentSpawner

```
using System;
using System.Collections;
using UnityEngine;
```

```
public class DepartedAgentSpawner : MonoBehaviour
{
    public GameObject agentPrefab; // Prefab of the agent to spawn
    public GameObject Parent;      // Parent to spawn the object below to

    [Serializable]
    public class DepartedAgentSettings
    {
        [Range(0f, 1f)]
        public float ChanceToUseRestroom = 0.33f;
        [Range(0f, 1f)]
        public float ChanceToHaveBaggage = 0.8f;
        [Range(0f, 1f)]
        public float ChanceToWantACar = 0.2f;
    }
    public DepartedAgentSettings AgentSettings = new();

    private GameObject CrowdDensity;
    SimulationData sData;

    // Start is called before the first frame update
    void Start()
    {
        GameObject Visualizations = GameObject.Find("Visualizations");
        CrowdDensity = Visualizations.transform.Find("CrowdDensity").gameObject
        sData = FindObjectOfType<SimulationData>();
    }

    public void SpawnAgents(int AgentNumber, int FlightNumber, int GateNumber)
    {
        StartCoroutine(PlaneWithAgentsArrived(AgentNumber, FlightNumber, GateNu
    }

    private IEnumerator PlaneWithAgentsArrived(int AgentNumber, int FlightNumbe
    {
        for (int i = 0; i < AgentNumber; i++)
        {
            //Spawn Agent
```

```
            Vector3 AgentPos = transform.GetChild(GateNumber - 1).GetChild(0).P
            //AgentPos.x = AgentPos.z + UnityEngine.Random.Range(-5f, 2f);
            GameObject newAgent = Instantiate(agentPrefab, AgentPos, Quaternion

            //Set Agents settings
            newAgent.GetComponent<DepartedAgentsBT>().agentSettings = AgentSett

            //If crowd shader is active we need to detect collisions not trigge
            if (CrowdDensity.activeInHierarchy) newAgent.GetComponent<CapsuleCo

            //Set Parent for agents so that its organized
            if (Parent != null) newAgent.transform.parent = Parent.transform;

            //Agent Flight Number and Number
            newAgent.name = "FlightNo" + FlightNumber.ToString() + "_AgentNo" +

            //Add agent to simulation data for general use
            sData.GetAirportData().InsertNewOutGoingAgent(newAgent);

            //Wait a little bit until next Agent
            float spawnDelay = UnityEngine.Random.Range(0.5f, 1f);
            yield return new WaitForSeconds(spawnDelay);
        }
    }
}
```

DepartedAgentScheduler

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DepartedAgentsScheduler : MonoBehaviour
{
    private List<IncomingFlight> FlightList;
    private int TotalFlights = 0;
    private float StartingTime = 0f;
    private DepartedAgentSpawner AgentSpawner;
```

```
private SimulationData sd;

public List<IncomingFlight> getFlightList() { return FlightList; }
public int getTotalFlights() { return TotalFlights; }
public float getStartingTime() { return StartingTime; }

float GetTimeDelay(float AgentStartArrivingTime)
{
    return AgentStartArrivingTime - StartingTime;
}

public void InitiateAllFlights()
{
    foreach (IncomingFlight flight in FlightList)
    {
        StartCoroutine(InitiateFlight(flight));
    }
}

public IncomingFlight GenerateFlight(float agentStartArrivingTime, int ager
{
    TotalFlights++;
    IncomingFlight newFlight = new(agentStartArrivingTime, agentNumber, Tot
    FlightList.Add(newFlight);
    sd.IncreasedTotalFlightsGenerated();
    return newFlight;
}

public IEnumerator InitiateFlight(IncomingFlight flight)
{
    yield return new WaitForSeconds(GetTimeDelay(flight.AgentStartArriving
    AgentSpawner.SpawnAgents(flight.AgentNumber, flight.FlightNumber, fligh
    sd.IncreasedTotalFlightsInitiated();
}

public void Init()
{
    StartingTime = Time.time;
    AgentSpawner = FindObjectOfType<DepartedAgentSpawner>();
```

```
        FlightList = new List<IncomingFlight>();
        sd = FindAnyObjectByType<SimulationData>();
    }

    public void Reset()
    {
        TotalFlights = 0;
        StartingTime = sd.StartingTime;
    }
}
```

IncomingFlight

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class IncomingFlight
{
    public float AgentStartArrivingTime;
    public int AgentNumber;
    public int FlightNumber;
    public int Gate;

    public IncomingFlight(float agentStartArrivingTime, int agentNumber, int fl
    {
        AgentStartArrivingTime = agentStartArrivingTime;
        AgentNumber = agentNumber;
        FlightNumber = flightNumber;
        Gate = gate;
    }
}
```

A.2 Virus Transmission:

Virus Transmission:

```
using System.Collections;
using System.Collections.Generic;
```

```
using Unity.VisualScripting;
using UnityEngine;

public class VirusTransmission : MonoBehaviour
{
    SimulationData sd;
    AgentData Data;
    AgentVirusData vData;
    private void Start()
    {
        sd = FindAnyObjectByType<SimulationData>();
        Data = GetComponent<AgentData>();
        vData = GetComponent<AgentVirusData>();
    }

    float CalculateL(Vector3 InfectedAgentPosition)
    {
        float distance = Vector3.Distance(InfectedAgentPosition, transform.posi
        return (distance - sd.GetVirusData().GetInfectionRange()) /  ( - sd.Get
    }

    float VirusTransmissionProbabilityGen(Vector3 InfectedAgentPosition)
    {
        float L = CalculateL(InfectedAgentPosition);

        if (sd.GetVirusData().GetMaskWearing()) return L / (sd.GetVirusData().(
        else return L / sd.GetVirusData().GetVirusInfectiousness() * vData.natu
    }

    bool InRangeOfInfected(Collider other)
    {
        if (other.CompareTag("Agent"))
        {
            AgentVirusData other_vData = other.GetComponent<AgentVirusData>();
            if (other_vData.AgentViralState == AgentVirusData.SEIRMODEL.Infecte
                && vData.AgentViralState == AgentVirusData.SEIRMODEL.Susceptibl
                return true;
        }
        return false;
```

```
    }

    private IEnumerator AddPointToHeatMap()
    {
        GetComponent<CapsuleCollider>().isTrigger = false;
        yield return new WaitForSeconds(1.0f);
        GetComponent<CapsuleCollider>().isTrigger = true;
    }

    private void OnTriggerStay(Collider other)
    {
        if (InRangeOfInfected(other) == true)
        {
            float chance = VirusTransmissionProbabilityGen(other.transform.posi
            //Debug.Log(chance);
            if(Random.value < chance)
            {
                vData.ChangeViralState(AgentVirusData.SEIRMODEL.Exposed);
                sd.GetRecordedDataBar().Add(new RecordedData(Data.CurrentAreaIn
                StartCoroutine(AddPointToHeatMap());
                Debug.Log("Exposed");
            }
        }
    }
}
```

Virus Data Generation:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class GenerateAgentVirusData
{
    const int SUSCEPTIBLE = 0;
    const int INFECTED = 2;

    Color SUSCEPTIBLECOLOR = Color.blue;
```

```
    Color EXPOSEDCOLOR = Color.yellow;
    Color INFECTEDCOLOR = Color.red;
    Color RECOVEREDCOLOR = Color.green;


    // IMPORTANT WE DONT GENERATE RECORVERED AGENTS FOR THE TIME BEING OR EXPOS
    public int GenerateViralStateValue(SimulationData sd)
    {

        if (Random.value < 0.15)
        {
            sd.counter = 0;
            return INFECTED;
        }
        else
        {
            sd.counter++;
            return SUSCEPTIBLE;
        }
    }

    public Color GenerateViralStateColor(int ViralStateValue) {
        if (ViralStateValue == 0) return SUSCEPTIBLECOLOR;
        else if (ViralStateValue == 1) return EXPOSEDCOLOR;
        else if (ViralStateValue == 2) return INFECTEDCOLOR;
        else return RECOVEREDCOLOR;
    }

}
```

Agent Virus Data

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Analytics;

public class AgentVirusData : MonoBehaviour
```

```
{
    public enum SEIRMODEL { Susceptible, Exposed, Infected, Recovered }

    [Header("Data")]
    public SEIRMODEL AgentViralState;
    public Color ViralStateColor;
    public float naturalSusceptibility;

    [Header("Quarantine Measures")]
    public bool MaskWearing;
    public bool SocialDistancing;


    GenerateAgentVirusData VirusDataGen = new();
    SimulationData sd;

    void SetAgentVirusData()
    {
        int StateValue = VirusDataGen.GenerateViralStateValue(sd);

        AgentViralState = (SEIRMODEL)Enum.GetValues(typeof(SEIRMODEL)).GetValue
        ViralStateColor = VirusDataGen.GenerateViralStateColor(StateValue);
        naturalSusceptibility = 1f;
        MaskWearing = sd.GetVirusData().GetMaskWearing();
        SocialDistancing = sd.GetVirusData().GetSocialDistancing();
    }


    void EnableMask()
    {
        transform.GetChild(0).gameObject.SetActive(MaskWearing); // Enable or d
        transform.GetChild(1).gameObject.SetActive(MaskWearing); // Enable or d
    }

    public void PassSimulationData()
    {

        if (sd != null)
        {
```

```
            if (AgentViralState == SEIRMODEL.Susceptible) sd.GetVirusData().Inc
            else if(AgentViralState == SEIRMODEL.Infected) sd.GetVirusData().In
        }
    }

    private void Start()
    {
        sd = FindAnyObjectByType<SimulationData>();

        SetAgentVirusData();

        GetComponent<Renderer>().material.color = ViralStateColor; //Pass it to

        EnableMask();

        PassSimulationData();
    }

    //should fix this ugly ass of a function
    public void ChangeViralState(SEIRMODEL newState)
    {
        if (GetComponent<AgentVirusData>().AgentViralState == SEIRMODEL.Suscept
            sd.GetVirusData().DecreaseCurrentNumberOfSusceptible();
        else if (GetComponent<AgentVirusData>().AgentViralState == SEIRMODEL.Ex
            sd.GetVirusData().DecreaseCurrentNumberOfExposed();
        else if (GetComponent<AgentVirusData>().AgentViralState == SEIRMODEL.In
            sd.GetVirusData().DecreaseCurrentNumberOfInfected();

        AgentViralState = newState;
        if (newState == SEIRMODEL.Susceptible)
            sd.GetVirusData().IncreaseCurrentNumberOfSusceptible();
        else if (newState == SEIRMODEL.Exposed)
        {
            sd.GetVirusData().IncreaseTotalNumberOfExposed();
            sd.GetVirusData().IncreaseCurrentNumberOfExposed();
        }
        else if (newState == SEIRMODEL.Infected)
            sd.GetVirusData().IncreaseCurrentNumberOfInfected();
```

```
        SetViralStateColor(VirusDataGen.GenerateViralStateColor(Convert.ToInt32
    }

    public void SetViralStateColor(Color newColor)
    {
        ViralStateColor = newColor;
        GetComponent<Renderer>().material.color = ViralStateColor;
    }
}
```

AntiSeptics:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AntiSeptics : MonoBehaviour
{
    [SerializeField] int ImmunityDuration = 8;

    IEnumerator TemporaryImmunity(Collider other)
    {

        other.GetComponent<AgentVirusData>().naturalSusceptibility = 0;
        yield return new WaitForSeconds(ImmunityDuration);
        other.GetComponent<AgentVirusData>().naturalSusceptibility = 1;
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Agent"))
        {
            StartCoroutine(TemporaryImmunity(other));
        }
    }
}
```

A.3 Visualizations

Heatmap, used and modified from https://github.com/kDanik/heatmap-unity repo

Controller:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using UnityEngine;
using Debug = UnityEngine.Debug;

public class HeatmapController : MonoBehaviour
{
    [Serializable]
    public class Settings
    {
        public int heightInParticles;
        public float maxColoringDistance;
        public bool ignoreYforColoring = false;
        public Gradient gradient;
        public float particleDistance;
        public float colorMultiplier;
        public float particleSize;
        public float colorCutoff;
        public Material particleMaterial;
        public int maxParticleNumber = 50000;

        public string pathForReadingData;
    }
    public Settings settings = new();
    private HeatmapVisualisation heatmapVisualisation;

    private bool eventsAreLoaded = true;
    private bool particleSystemIsInitialized = false;

    public List <Vector3> Points;
    private void Awake()
    {
        heatmapVisualisation = new HeatmapVisualisation(settings);
```

```
}

private void Start()
{
    InitializeParticleSystem();
    AddSelectedEventsToHeatmap();
}

/// <summary>
/// Loads events from file into events property (that also makes them displ
/// </summary>
public void LoadEvents()
{
    /*Stopwatch stopwatch = new();
    stopwatch.Start();

    eventReader = new JSONEventReader(settings.pathForReadingData);

    if (eventReader.ReaderIsAvailable())
    {
        events = eventReader.ReadEvents();
        eventsAreLoaded = true;
    }
    else
    {
        eventsAreLoaded = false;
        Debug.Log("Error while trying to read events. Event reader is not a
    }

    stopwatch.Stop();
    Debug.Log("LoadEvents - Elapsed Time is " + stopwatch.ElapsedMillisecon
}

/// <summary>
/// Creates and configures particle system (and particle array)
/// </summary>
public void InitializeParticleSystem()
{
    Stopwatch stopwatch = new();
```

```
        stopwatch.Start();

        heatmapVisualisation.InitializeParticleSystem(gameObject);
        heatmapVisualisation.InitializeParticleArray();
        particleSystemIsInitialized = true;

        stopwatch.Stop();
        Debug.Log("InitializeParticleSystem - Elapsed Time is " + stopwatch.Ela
    }

    /// <summary>
    /// Resets heatmap color(color values) to default
    /// </summary>
    public void ResetHeatmap()
    {
        heatmapVisualisation.ResetParticlesColor();
        heatmapVisualisation.UpdateParticlesInParticleSystem();
    }

    /// <summary>
    /// Adds selected (in Editor window) events to heatmap and updates heatmap
    /// </summary>
    public void AddSelectedEventsToHeatmap()
    {

        Stopwatch stopwatch = new();
        stopwatch.Start();

        heatmapVisualisation.ResetParticlesColor();

        heatmapVisualisation.AddEventToHeatMap(Points.ToArray());


        heatmapVisualisation.UpdateParticlesInParticleSystem();

        stopwatch.Stop();
        Debug.Log("AddEventsToHeatMap  - Elapsed Time is " + stopwatch.ElapsedM
    }
```

```
    /// <summary>
    /// Status of Load Events action. (see HeatmapGUI.cs for usage))
    /// </summary>
    public bool IsLoadEventsActive()
    {
        return !string.IsNullOrEmpty(settings.pathForReadingData);
    }

    /// <summary>
    /// Status of Initialize Particle System action.  (see HeatmapGUI.cs for us
    /// </summary>
    public bool IsInitializeParticleSystemActive()
    {
        return GetComponent<BoxCollider>() != null;
    }

    /// <summary>
    /// Status of Add Events to Heatmap action.  (see HeatmapGUI.cs for usage)
    /// </summary>
    public bool IsAddEventToHeatMapActive()
    {
        return eventsAreLoaded && particleSystemIsInitialized;
    }

    /// <summary>
    /// Status of Reset Heatmap action. (see HeatmapGUI.cs for usage)
    /// </summary>
    public bool IsResetHeatmapActive()
    {
        return particleSystemIsInitialized;
    }
}
```

ParticleSystem:

```
using System.Collections.Generic;
using UnityEngine;
using static UnityEngine.ParticleSystem;

/// <summary>
```

```
/// Controls interactions with particle system of heatmap
/// </summary>
public class HeatmapParticleSystem
{
    private ParticleSystem particleSystem;

    /// <summary>
    /// This array is NOT representing directly real particles of particle syst
    /// Particles in ParticleSystem can't be modified directly, so each change
    /// </summary>
    private Particle[,,] particles;

    private Bounds particleSystemBounds;
    private Vector3Int sizeInParticles;

    /// <summary>
    /// Creates and configures particle system for heatmap visualisation.
    /// </summary>
    /// <param name="parent">Object that will contain ParticleSystem component<
    public void InitializeParticleSystem(GameObject parent, HeatmapController.S
    {
        particleSystemBounds = parent.GetComponent<BoxCollider>().bounds;
        particleSystem = CreateAndConfigureParticleSystem(parent, settings);
    }

    /// <summary>
    /// Creates and populates particle array
    /// </summary>
    public void CreateParticleArray(HeatmapController.Settings settings)
    {
        sizeInParticles = CalculateSizeOfParticleSystemInParticles(settings);

        particles = new Particle[sizeInParticles.x, sizeInParticles.y, sizeInPa

        for (int x = 0; x < sizeInParticles.x; x += 1)
        {
            for (int y = 0; y < sizeInParticles.y; y += 1)
            {
                for (int z = 0; z < sizeInParticles.z; z += 1)
```

```
            {
                Particle particle = new();

                Vector3 position = ConvertParticleGridPositionToGlobal(new
                particle.position = position;

                particle.startSize = settings.particleSize;
                particle.startColor = settings.gradient.Evaluate(0);

                particle.remainingLifetime = 1000;
                particle.startLifetime = 1000;

                particles[x, y, z] = particle;
            }
        }
    }
}

/// <summary>
/// Checks particles and adds its particles to particle system.
/// Depending  on colorCutoff and their alpha, particles will be added or i
/// </summary>
public void UpdateParticlesInParticleSystem(float[,,] particleColorValues,
{
    List<Particle> particleList = new();

    for (int x = 0; x < sizeInParticles.x; x += 1)
    {
        for (int y = 0; y < sizeInParticles.y; y += 1)
        {
            for (int z = 0; z < sizeInParticles.z; z += 1)
            {
                // the invisible particles / particles with color value low
                if (settings.colorCutoff <= particleColorValues[x, y, z])
                {
                    Color particleColor = settings.gradient.Evaluate(partic
                    if (particleColor.a > 0.001f)
                    {
                        particles[x, y, z].startColor = particleColor;
```

```
                        particleList.Add(particles[x, y, z]);
                    }
                }
            }
        }
    }

    particleSystem.SetParticles(particleList.ToArray());
}


/// <summary>
/// Calculates size (bounds) of particle system in particles
/// </summary>
private Vector3Int CalculateSizeOfParticleSystemInParticles(HeatmapControll
{
    Vector3Int calculatedSizeInParticles = Vector3Int.FloorToInt((particleS

    if (settings.heightInParticles != 0)
    {
        calculatedSizeInParticles.y = settings.heightInParticles;
    }

    return calculatedSizeInParticles;
}

/// <summary>
/// Converts global position to closest index in particle grid (particles a
/// </summary>
public Vector3Int ConvertGlobalPositionToParticleGrid(Vector3 globalPositio
{
    Vector3Int convertedPosition = Vector3Int.RoundToInt((globalPosition -

    return convertedPosition;
}

/// <summary>
/// Converts position in particles array to global position
/// </summary>
```

```
public Vector3 ConvertParticleGridPositionToGlobal(Vector3Int positionInPar
{
    Vector3 convertedPosition;
    convertedPosition.x = (positionInParticleGrid.x * settings.particleDist
    convertedPosition.y = (positionInParticleGrid.y * settings.particleDist
    convertedPosition.z = (positionInParticleGrid.z * settings.particleDist

    return convertedPosition;
}

public Vector3Int GetSizeOfParticleSystemInParticles()
{
    return sizeInParticles;
}

private ParticleSystem CreateAndConfigureParticleSystem(GameObject parent,
{
    ParticleSystem newParticleSystem = parent.AddComponent<ParticleSystem>(

    EmissionModule emission = newParticleSystem.emission;
    emission.enabled = false;

    ShapeModule shape = newParticleSystem.shape;
    shape.enabled = false;

    ParticleSystemRenderer renderer = parent.GetComponent<ParticleSystemRen
    renderer.sortMode = ParticleSystemSortMode.Distance;
    renderer.allowRoll = false;
    renderer.alignment = ParticleSystemRenderSpace.Facing;

    MainModule main = newParticleSystem.main;
    main.loop = false;
    main.simulationSpace = ParticleSystemSimulationSpace.World;
    main.maxParticles = settings.maxParticleNumber;
    main.playOnAwake = false;


    renderer.material = settings.particleMaterial;
```

```
        return newParticleSystem;
    }
}
```

Visualization:

```
    using UnityEngine;

public class HeatmapVisualisation
{
    private readonly HeatmapController.Settings settings;

    private HeatmapParticleSystem heatmapParticleSystem;

    private float[,,] particleColorValues;

    public HeatmapVisualisation(HeatmapController.Settings settings)
    {
        this.settings = settings;

        heatmapParticleSystem = new HeatmapParticleSystem();
    }

    /// <summary>
    /// Creates and configures particle system for heatmap visualisation.
    /// </summary>
    /// <param name="parent">Object that will contain ParticleSystem component<
    public void InitializeParticleSystem(GameObject parent)
    {
        if (parent.GetComponent<ParticleSystem>() != null)
        {
            Debug.Log("There is particle system present on parent object alread

            return;
        }

        if (settings.particleMaterial == null)
        {
            Debug.LogError("Particle material is not defined in settings!");
```

```
            return;
        }

        heatmapParticleSystem.InitializeParticleSystem(parent, settings);
    }

    /// <summary>
    /// Initializes particle array and populates it with particles with default
    /// </summary>
    public void InitializeParticleArray()
    {
        heatmapParticleSystem.CreateParticleArray(settings);

        Vector3Int sizeInParticles = heatmapParticleSystem.GetSizeOfParticleSys
        particleColorValues = new float[sizeInParticles.x, sizeInParticles.y, s

        UpdateParticlesInParticleSystem();
    }

    /// <summary>
    /// Adds all positions from eventData to heatmap (by calculating new color
    /// </summary>
    public void AddEventToHeatMap(Vector3[] eventData)
    {
        foreach (Vector3 eventPosition in eventData)
        {
            AddOnePositionToHeatmap(eventPosition);
        }
    }

    /// <summary>
    /// Resets color value of all particles to default (0f)
    /// </summary>
    public void ResetParticlesColor()
    {
        Vector3Int sizeInParticles = heatmapParticleSystem.GetSizeOfParticleSys
        particleColorValues = new float[sizeInParticles.x, sizeInParticles.y, s
    }
```

```
public void UpdateParticlesInParticleSystem()
{
    heatmapParticleSystem.UpdateParticlesInParticleSystem(particleColorValu
}

private void AddOnePositionToHeatmap(Vector3 eventPosition)
{
    Vector3Int eventPositionInParticleGrid = heatmapParticleSystem.Convert(
    Vector3Int sizeInParticles = heatmapParticleSystem.GetSizeOfParticleSys

    // calculate bounds in which particles can be affected by eventPositior
    Vector3Int minBound = CalculateMinBound(eventPositionInParticleGrid);
    Vector3Int maxBound = CalculateMaxBound(eventPositionInParticleGrid);

    // checking all particles in this bounds, and updating their color valu
    for (int x = minBound.x; x <= maxBound.x; x += 1)
    {
        for (int y = minBound.y; y <= maxBound.y; y += 1)
        {
            for (int z = minBound.z; z <= maxBound.z; z += 1)
            {
                if (IsInBoundsOfParticleArray(x, y, z, sizeInParticles))
                {
                    UpdateColorAddValue(new Vector3Int(x, y, z), eventPosit
                }
            }
        }
    }
}

private void UpdateColorAddValue(Vector3Int particlePositionInGrid, Vector3
{
    float distance = CalculateDistanceBetweenTwoParticleGridPoints(particle

    if (distance < settings.maxColoringDistance)
    {
        // calculate colorAddValue, depending on how close is distance to m
        float colorAddValue = settings.colorMultiplier * (1 - distance / se
```

```
            particleColorValues[particlePositionInGrid.x, particlePositionInGri
        }
    }

    private float CalculateDistanceBetweenTwoParticleGridPoints(Vector3Int poin
    {
        float distanceSquare = (point1.x - point2.x) * (point1.x - point2.x) +

        if (!ignoreHeightInCalculation)
        {
            distanceSquare += (point1.y - point2.y) * (point1.y - point2.y);
        }

        return Mathf.Sqrt(distanceSquare) * settings.particleDistance;
    }

    private bool IsInBoundsOfParticleArray(int x, int y, int z, Vector3Int size
    {
        return x >= 0 && z >= 0 && y >= 0 && x < sizeInParticles.x && y < sizeI
    }

    private Vector3Int CalculateMinBound(Vector3Int positionInParticleGrid)
    {
        Vector3Int min = new();

        min.x = (int)(positionInParticleGrid.x - settings.maxColoringDistance /
        min.z = (int)(positionInParticleGrid.z - settings.maxColoringDistance /

        if (!settings.ignoreYforColoring)
        {
            min.y = (int)(positionInParticleGrid.y - settings.maxColoringDistan
        }
        else
        {
            min.y = 0;
        }

        return min;
```

```
    }

    private Vector3Int CalculateMaxBound(Vector3Int positionInParticleGrid)
    {
        Vector3Int max = new();

        max.x = (int)(positionInParticleGrid.x + settings.maxColoringDistance /
        max.z = (int)(positionInParticleGrid.z + settings.maxColoringDistance /

        if (!settings.ignoreYforColoring)
        {
            max.y = (int)(positionInParticleGrid.y + settings.maxColoringDistan
        }
        else
        {
            max.y = positionInParticleGrid.y - 1;
        }

        return max;
    }
}
```

Crowd Density Shader

```
Shader "Unlit/CrowdDensityShader"
{
    Properties
    {
        _MainTex("Texture", 2D) = "white" {}
        _Color0("Color ", Color) = (0, 0, 0, 1)
        _Color1("Color ", Color) = (0, .9, .2, 1)
        _Color2("Color ", Color) = (.9, 1, .3, 1)
        _Color3("Color ", Color) = (.9, .7, .1, 1)
        _Color4("Color ", Color) = (1, 0, 0, 1)
    }
        SubShader
    {
        Tags { "RenderType" = "Opaque" }
        LOD 100
```

```
Pass
{
    CGPROGRAM
    #pragma vertex vert
    #pragma fragment frag
    // make fog work
    #pragma multi_compile_fog

    #include "UnityCG.cginc"

    struct appdata
    {
        float4 vertex : POSITION;
        float2 uv : TEXCOORD0;
    };

    struct v2f
    {
        float2 uv : TEXCOORD0;
        UNITY_FOG_COORDS(1)
        float4 vertex : SV_POSITION;
    };

    sampler2D _MainTex;
    float4 _MainTex_ST;

    v2f vert(appdata v)
    {
        v2f o;
        o.vertex = UnityObjectToClipPos(v.vertex);
        o.uv = TRANSFORM_TEX(v.uv, _MainTex);
        UNITY_TRANSFER_FOG(o,o.vertex);
        return o;
    }

    float4 colors[5];
    float pointranges[5];

    float _HitsX[1000];
```

```
float _HitsY[1000];
float _ValidHits[1000];
int _HitCount = 0;

float4 _Color0;
float4 _Color1;
float4 _Color2;
float4 _Color3;
float4 _Color4;

void init() {
    colors[0] = _Color0;
    colors[1] = _Color1;
    colors[2] = _Color2;
    colors[3] = _Color3;
    colors[4] = _Color4;

    pointranges[0] = 0;
    pointranges[1] = 0.25;
    pointranges[2] = 0.50;
    pointranges[3] = 0.75;
    pointranges[4] = 1.0;
}

float distsq(float2 a, float2 b)
{
    float area_of_effect_size = 0.05;
    float d = pow(max(0.0, 1.0 - distance(a, b) / area_of_effect_s

    return d;
}

float3 getHeatForPixel(float weight)
{
    if (weight <= pointranges[0])
    {
        return colors[0];
    }
```

```
        if (weight >= pointranges[4])
        {
            return colors[4];
        }

        for (int i = 1; i < 5; i++)
        {
            if (weight < pointranges[i])
            {
                float dist_from_lower_point = weight - pointranges[i -
                float size_of_point_range = pointranges[i] - pointrange

                float ratio_over_lower_point = dist_from_lower_point /

                float3 color_range = colors[i] - colors[i - 1];
                float3 color_contribution = color_range * ratio_over_lo

                float3 new_color = colors[i - 1] + color_contribution;

                return new_color;
            }
        }

        return colors[0];

    }

    fixed4 frag(v2f i) : SV_Target
    {
        init();

        // sample the texture
        fixed4 col = tex2D(_MainTex, i.uv);

        float2 uv = i.uv;
        uv = uv * 4.0 - float2(2.0, 2.0); //change uv coordinate range

        float totalWeight = 0;
        for (float i = 0; i < _HitCount; i++) {
```

```
                    if (_ValidHits[i] == 1) {
                        float2 work_pt = float2(_HitsX[i], _HitsY[i]);
                        float pt_intensity = 1;

                        totalWeight += 0.5 * distsq(uv, work_pt) * pt_intensity
                    }
                }

                float3 heat = getHeatForPixel(totalWeight);

                return col + float4(heat, 0.5);
            }
            ENDCG
        }
    }
}
```