Thresholding Closed Structure Capacity to Facilitate Efficient Evacuation using Crowd Simulation

by

Saurabh Pathak, B.Tech

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2017

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Saurabh Pathak

August 30, 2017

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Saurabh Pathak

August 30, 2017

Acknowledgments

The author would like to thank his family members for their constant support and love during implementation and writing of this thesis. The author would also like to thank Dr. Carol O'Sullivan for her support and guidance during the project. Without these people, this thesis would not have been possible.

Saurabh Pathak

University of Dublin, Trinity College

September 2017

Thresholding Closed Structure Capacity to Facilitate Efficient Evacuation using Crowd Simulation

Saurabh Pathak, M.Sc.
University of Dublin, Trinity College, 2017

Supervisor: Carol O'Sullivan

The paper here discusses methods to improve the interactions between the agents and environment to facilitate them reach their target with minimum effort. This will help design a closed system that responds to emergencies by helping agents evacuate the space as early as possible. The end result for this thesis will be a framework which will help us find the maximum capacity of people that may be accommodated in a structure to facilitate fastest evacuations. Path-finding will be implemented for helping an agent to reach its goal by following the shortest path and avoiding all the collisions that come its way. This paper will likewise talk about the crossing point of AI and VE. It considers the utilization of AI as a part of a VE and Intelligent Virtual Agents as a noteworthy application region, covering movement, sensing, behavior, and control architectures. A smart environment which the agent needs to interact or respond to like walls (which can't be passed), another agent (avoid collision with them), etc.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Why Crowd Simulation

Crowd Simulation is the way toward reenacting the development (or flow) of a substantial number of elements or characters. It is regularly used to make virtual scenes for visual media like movies and computer games, and is likewise utilised as a part of emergency preparing, engineering and urban arranging, and departure reproduction.
Group reenactment may concentrate on angles that objective distinctive applications. For reasonable and quick rendering of a group for visual media or virtual cinematography, lessening of the intricacy of the 3D scene and picture based rendering are utilised, while varieties in appearance help exhibit a practical populace.

In diversions and applications expected to recreate genuine human group development, as in departure reenactments, mimicked specialists may need to explore towards an objective, stay away from crashes, and show other human-like conduct. Many group guiding calculations have been created to lead reenacted group to their objectives reasonably. Some more broad frameworks are inquired about, that can bolster various types of agents(like autos and people on foot), diverse levels of abstraction(like individual and continuum), specialists interfacing with shrewd articles, and more mind boggling physical and social elements.

1.2 Motivation

Year	Location	Scale of the incident	Failure Elements
1989	Hillsborough, UK	96 dead, 400 injured (Overcrowding)	Design (Throughput)
1990	Mina Valley, Saudi Arabia	1426 pilgrims crushed (Overcrowding)	Design (Throughput)
1991	Orkney, South Africa	42 dead, many injured (Overcrowding/riot)	Design (Capacity + Crazing)
1994	Jamarat, Saudi Arabia	266 pilgrims crushed, 98 injured (Overcrowding)	Design (Throughput)
1996	Guatemala City	83 crushed, 180 injured (Overcrowding)	Design (Capacity + Forged Tickets)
1997	Jamarat, Saudi Arabia	22 pilgrims crushed, 43 injured (Overcrowding)	Design (Throughput)
1998	Jamarat, Saudi Arabia	118 pilgrims crushed, 434 injured (Overcrowding)	Design (Throughput)
1999	Kerala, India	51 killed, 150 injured in stampede (Reaction)	Information (Weather + Running)
1999	Minsk, Belarus	53 killed, 190 injured in stampede (Reaction)	Information (Weather + Running)
2001	Ellis Park, South Africa	43 dead, 200+ Injured (Over capacity - Overcrowding)	Design (Capacity + Crazing)
2001	Aracaju, Brazil	4 dead, including 3 children (Reaction)	Information (Handouts + Crazing)
2002	Yokohama, Japan	10 trampled, Mall crowd craze (Reaction)	Information (Handouts + Crazing)
2001	Jamarat, Saudi Arabia	35 pilgrims crushed, 179 injured (Overcrowding)	Design (Throughput)
2004	Jamarat, Saudi Arabia	249 pilgrims crushed, 252 injured (Overcrowding)	Design (Throughput)
2004	Beijing, China	37 dead, 15 injured in crowd crush (Overcrowding)	Design (Throughput)
2006	Jamarat, Saudi Arabia	363 dead, 389 injured crowd crush (Overcrowding)	Design (Throughput)
2006	Manila, Philippines	74 dead, 300 injured crowd crush (Reaction)	Design (Throughput)
2006	Yemen, Middle East	51 dead, 238 Injured crowd crush (Reaction)	Information (Political Rally)
2008	Himachal Pradesh, India	146 dead, 50 injured in stampede (Narrow road)	Design (Throughput)
2008	Pasuran, Java	23 dead, dozens injured in Ramadan (Reaction)	Information (Handouts)
2009	Abidjan, Ivory Coast	22 dead, 132 injured Football (Reaction)	Design (Throughput)
2009	Birmingham (JLS), UK	60 injured, 4 hospitalised (JLS) (Reaction)	Design (Capacity + Arrival Profile)
2010	Timbuktu, Mali, West Africa	26 dead, 55 injured (Overcrowding)	Design (Throughput)
2010	Kunda, North India	63 dead, 44 injured (Overcrowding - narrow road)	Design (Throughput)
2010	Brick Lane, London	10 police officers injured (Reaction)	Design (Capacity + Crazing)
2010	Johannesburg, South Africa	14 injured (Overcrowding at entry gates)	Information (Forged Tickets)
2010	Duisburg, Germany.	21 dead, 511 injured (Overcrowding)	Design (Throughput)
2010	Bihar, India	10 dead, dozens injured (Reaction)	Information (Over Reaction)
2010	Nairobi, Kenya	7 dead, 70 injured (Reaction)	Information (Rain Stopped)
2010	Phnom Penh, Cambodia	347 dead, 395 injured (Overcrowding)	Design (Capacity)
2011	Kerala, India	102 dead, 44 injured (Overcrowding)	Design (Capacity)
2011	Port Harcourt, Nigeria	11 dead, 29 injured (Reaction)	Information (Shots Fired)
2011	Bamako, Mali	36 dead, 70 Injured (Overcrowding)	Design (Capacity)
2011	Brazzaville, Congo	7 dead, 30 Injured (Overcrowding)	Design (Capacity)
2011	Jakarta, Indonesia	2 dead, 13 Injured (Overcrowding)	Information (Poor Ticket Allocation)
2012	Port Said, Egypt	74 dead, over 1,000 Injured (Overcrowding + rioting)	Design (Throughput + Riot)
2012	Cairo (copic), Egypt	3 dead, dozens injured (Overcrowding)	Design (Throughput)
2013	Abidjan, Ivory Coast	62 dead, dozens injured (Overcrowding)	Design (Capacity + Crazing)
2013	Allahabab, Northern India	36 dead, 31 injured (Overcrowding, railway platform)	Design (Capacity + Crowd management)
2013	Hubei, China	4 Dead, 14 injured (Overcrowding on stairs - school)	Crowd Management
2013	Shanghai, Beijing	7 Injured (Crowd Crazing, David Beckham)	Crowd Management
2013	Datia, India	50 dead, 100+ injured (Overcrowding)	Design (Capacity + Crowd management)
2013	Anambra, Nigeria	28 dead, 200+ injured (Overreaction/Design - call of "fire")	Design (Capacity and crazing)
2014	Ningxia, China	14 Dead, 10 Injured (Design/Overcrowding/Capacity - food handouts)	Design (Crowd flow, Crowd management)
2014	Mumbai, India	20 dead, 40 injured (Overcrowding, narrow streets)	Design (Capacity, Crowd Flow)
2014/15	Shanghaii, China	36 dead, 46 injured - crowd crushing (overcrowding)	Design (Capacity, Crowd Flow)

Figure 1.1: Past data where accidents occurred due to bad structural architecture. Retrieved from http://www.gkstill.com/Support/WhyModel/index.html

The image has some highlighted figures which represent fiascos that have a plan issue (appeared as red below) as the central (or distal) causality. In particular, the outline (space) was not adequately vigorous for the expected group. Neglecting to comprehend the outline limit and throughput, and additionally neglecting to foresee the group numbers is a formula for fiasco, and a central component of group hazard examination. This highlights instructing the standards and utilizations of how to investigate swarm stream rates, fill times and limit would have had a huge changes to swarm well-being and spared lives.

Computer Simulation is an intense instrument for investigating unpredictable and dynamic situations. It gives an engaging way to deal with, think about and dissect circumstances which would be exceedingly costly to actualize in this present reality. This

capable strategy helps the basic leadership handle. Agent based crowd simulation has risen as a productive device to research prevention of above fiascos due to architectural shortcomings of a building. The advancement of viable person on foot recreation frameworks is critical in understanding danger and swarm security in spots of open get together.

We manufacture models to comprehend a specific wonders or normal for a framework. We manufacture models to comprehend the connection amongst circumstances and end results. We manufacture models to enhance our comprehension of an unpredictable domain. We do as such in light of the fact that, amid the way toward building a model, we experience the procedure of revelation, with disclosure comes understanding and with understanding comes control. A mind boggling issue can be examined and unraveled by a procedure of displaying.

1.3 This Dissertation

This paper talks about implementing the agent-based model for crowd simulation. The root of the issue here is the manner by which the basic decision making may be modeled for every individual in the group. There are numerous decision-making frameworks that are accessible like Bayesian systems, fuzzy logic, neural networks, BDI, and decision networks. Every one of these techniques has been broadly utilized as a part of different applications for decision making.

However, these techniques are more disposed on the mathematical and calculative structure as opposed to reproducing human's decision-making process. So in this paper, the attempt is to expect a human behavior impersonation framework that shows how people process while settling on choices in various situations. This is known fact that the external stimulus has an equal impact on a human's physical condition, feeling, and social dealing. These impacts, in turn, have effects on a human's decision-making process. Accordingly, this system goes for handling the two phases of the insightful strategy associated with decision making - knowledge of a person of the circumstance and the changes

it causes on their internal characteristics. This system makes a suitable designation of individual efforts involving connection amongst agents and obstruction engine which plays a critical part in the genuine reproduction of gigantic crowds in complex situations.

1.4 Road-map

The design of this thesis is as per the following: Chapter 2 surveys the present cutting edge research going on inside this field, including clarifying the various types of activity reenactment frameworks depicted as agent based, flow based, and so on. Chapter 3 plots the first outline and plan for building up the Crowd Simulation framework. The execution is explained point by point in Chapter 4. Chapter 5 basically investigates the usage and talks about system advantages and downsides of the framework and the outcomes it offers. Chapter 6 exhibits the conclusions and highlights conceivable future work important to the venture.

Chapter 2

State of the art

2.1 Approaches to Crowd Simulation

One of the real objectives in Crowd Simulation is to control swarms practically and reproduce human dynamic practices.

There exists a few larger ways to deal with swarm reenactment and AI, every one giving preferences and burdens in light of group size and time scale. Time scale alludes to how the goal of the recreation likewise influences the length of the reproduction. For instance, looking into social inquiries, for example, how philosophies are spread among a populace will bring about an any longer running recreation since such an occasion can traverse up to months or years. Utilising those two qualities, specialists have endeavoured to apply arrangements to better assess and compose existing group test systems.

2.2 Crowd Dynamics

2.2.1 Flow-based Approach

Flow based crowd simulations centre in light of the group overall as opposed to its segments[22]. All things considered people don't have any unmistakable practices that happen because of contribution from their environment and behavioural components are to a great extent lessened [12]. This model is for the most part used to assess the stream of

development of an extensive and dense crowd in a given situation. Best utilised as a part of concentrate vast group, brief time destinations. [18] [6]

2.2.2 Entity-based Approach

Models that actualise an arrangement of physical, predefined, and worldwide laws intended to mimic social/mental elements that happen in people that are a part of a group fall under this classification [21]. Entities for this situation don't have the ability to, it might be said, have an independent perspective. All developments are controlled by the worldwide laws being enforced on them[14]. Simulations that utilise this model frequently, do as such, to look into swarm dynamics, for example, jamming and flocking. Little to medium-sized group with short term goals fit this approach best.[25] [2]

2.2.3 Agent-based Approach

Described via independent, associating people. Every agent of a crowd in this approach, is given a level of insight; they can respond to every circumstance all alone, in light of an arrangement of choice principles[17]. Data used to choose an activity is gotten locally from the agent's' environment[23]. Frequently, this approach is utilised for reproducing sensible group conduct as the researcher is given full opportunity to actualise any practices. [8] [15]

2.3 Particle Systems

One approach to mimic virtual crowds is to utilize a particle system. A particle system is a gathering of various individual components or particles. Every particle can act self-rulingly and is relegated an arrangement of physical characteristics, (for example, color, size and speed).

A particle system is dynamic, in that the developments of the particles change after some time. A particle system's development is the thing that makes it so attractive and

simple to actualize. Ascertaining the developments of these particles takes next to no time. It just includes material science: the whole of the considerable number of strengths following up on a particle decides its movement. Forces, for example, gravity, friction and force from an impact, and social forces like the alluring force of an objective. [3]

Normally every particle has a speed vector and a position vector, containing data about the particle's present speed and position separately. The particles next position is ascertained by adding its speed vector to its position vector. An exceptionally basic operation (again why particle systems are so alluring). Its speed vector changes after some time, in light of the powers following up on the particle. For instance, a crash with another particle will make it alter course. [1]

Particles systems have been generally utilized as part of movies for impacts, for example, blasts, for water effect. [10]

2.4 Individual behavior modelling

One arrangement of procedures for AI-based group simulation is to show crowd conduct by cutting edge reproduction of individual specialist inspirations and decision making. For the most part, this implies every operator is allotted some arrangement of factors that measure different attributes or statuses, for example, stress, identity, or diverse objectives. This outcomes in more practical group conduct however might be more computationally escalated than more straightforward systems. [11] [16]

2.4.1 Personality-based models

One strategy for making individualistic conduct for crowd agents is using identity attributes. Every agent may have certain parts of their identity tuned in view of an equation that partners viewpoints, for example, forcefulness or lack of caution with factors that oversee the operators' conduct. One way this affiliation can be found is through a subjective report in which specialists are randomly allocated values for these factors and

members are made a request to portray every operator regarding these identity qualities. A regression may then be done to decide a connection between these qualities and the agent personality traits. The identity qualities would then be able to be tuned and appropriately affect agent conduct. [4] [5]

2.4.2 Stress-based model

The conduct of crowds in high-push circumstances can be displayed utilising General Adjustment Disorder theory. Agent conduct is influenced by different stressors from their condition sorted into four models: time weight, range weight, positional stressors, and relational stressors, each with related numerical models. [20] [19]

Time weight alludes to stressors identified with a period constrain in achieving a specific objective. A case would be a road crossing with a timed walk signal or boarding a train before the entryways are shut. This model is displayed by the accompanying recipe:

$$I_t = max(t_e - t_a, 0) \tag{2.1}$$

where I_t is the intensity of the time pressure as a function of the estimated time to reach the goal t_e and a time constraint t_a.

2.5 Background Study

Apart from talking about state of the art, here are some of the papers that inspired this thesis. Majority of these have incorporated agent based model to achieve their goals in crowd simulation. So these helped build a strong base for the current theses. Also, these papers were very helpful throughout the implementation of the current thesis as they have done a very thorough research on agent based simulations and they know how to tackle challenges. So this made the implementation very smooth and that's why these are included as a part of the current state of the art.

1. **A Synthetic-Vision Based Steering Approach for Crowd Simulation [13]**

   This paper presents a very innovative way of avoiding collisions between agents using computer vision as a primary tool. The agents are assumed to have the feature of interacting with each other and the virtual system. The agents detect future accidents and their severity from visual stimuli similar to any human. The response for such scenarios is layered. Firstly the agent tries to avoid a collision if possible. If not they try to reduce their speed to reduce the impact of the collision.

   The paper also presents some insights about why so much research is being done around crowd simulation. The field of interactive entertainment is different from security and architecture. So it desires frameworks or systems which may include complex behaviors of crowds and pictorially represent the communication between them. That is the reason for devoting computer animation resources for this field.

   Collision avoidance

   Discussing further, the paper gives some insights about collision avoidance using agent based methods based on geometrical prototypes. The way this is achieved is by calculating acceptable velocities and assigning them to agents so they may dodge imminent collisions. The next step would be to route the agents over probable paths along with having acceptable velocities.

   The two main approaches to achieve this are Rule-based technique and Particle systems and the inference drawn is that the former fails to simulate the beginning walkers while the latter may fail to achieve workable motion, for example, feasible accelerations and velocities.

2. **Controlling Individual Agents in High-Density Crowd Simulation [15]**

   This paper also tries to study the behavior of individual agents in a high-density crowd. To solve the problem of cogently simulating the behavior of individual

agents, the combination of psychological and physiological rules together with physical forces may be used. If the same attributes are assigned to all the agents it may lead to identical behaviors. Hence all the agents should be provided with different psychological and physical properties that may generate unique behaviors within the agents. Apart from these attributes, sensing is also a viable option that is being used in the current paper to consider the cases of the agent's reaction to other moving and stable objects.

3. **Agent-based human behavior modeling for crowd simulation**
   [9]

   The is paper was the best inspiration to proceed with the agent based model for crowd simulation. It's trying to simulate crowds using a layered architecture to show the human like decision-making process. This means considering the human reaction to various external situations and changes in internal attributes of the agent. The social psychology of an agent is what drives the activities done in a crowd or social gathering. Further, these attributes when provided to the agent they behave uniquely when presented with different scenarios by making very real life like decisions.

   The study on crowd simulation may be divided into two categories. The first one is where the crowd is considered a collection of agents which respond to any external changes governed by simple rules. Some of the standard methods for doing this are cellular automata model and particle system model. But these methods are not considered as adequate to study complex crowd behaviors.

   The second approach this paper refers to is the one where a crowd is treated as a set of individual agents who have attributes for making solid individual decisions. This is the approach that has been implemented in the current thesis. Here each agent is capable of making distinct and individual decisions when putting in a crowd.

2.6 Analysis

Going through all this research it was very clear that the agent based approach will be the best and an easy way to simulate crowds in a closed structure and study further about evacuation times in emergencies. The agent based method is very simple and streamlined. The only target here is to provide each agent with all the attributes required to make decisions and interact with various other objects in the system. Then these attributes may be provided to N number of agents and they all will behave individually provided they are programmed with unique values for these attributes. This will accomplish the first part of crowd simulation[7].

The next part is to introduce the environment with some emergency and notify the agents about that. Although there is not much material available online still there is an understanding that if the agents are provided with some extra features like a panic speed and sensing they may respond to these emergencies taking good decisions. So this is the current base on which the framework in development proceeds. [24]

Chapter 3

Design

By modelling correctly the attributes of humans in a closed structure with obstructions, when an explosion is introduced, a number of interesting evacuation observations will emerge will all collectively put together will provide the maximum capacity for an efficient evacuation.

3.1 Plan

The original plan for this system was to get the threshold capacity for a closed structure which may facilitate efficient evacuation in case of any emergency. To get this capacity it was needed to simulate the exact scenario using some game engine.

Start with getting a simple agent up and moving over a ground which may be simulated by a plane. An agent should mock the behaviour of humans in physical and sensing perspectives. So initially the agent should try and avoid other agents.

Next step would be to introduce some obstructions like a wall or blocked areas where agents cannot walk. These areas should be provided with some flag and never be considered in our navigation areas. While the agents should not pass through them we also don't want them to jump over this.

Extending agent behaviour, it also needs to have some feature to jump so that it may walk over stairs. Another attribute would be define a threshold inclined plains which the agents may use for navigating to areas which are at some height.

After these features have been assigned to agents, it is also required to put in a set of destinations which our agents need to reach in order to keep moving in the structure. Always we find in a mall or a museum people keep moving from one place to other, so as soon as the agents reaches a particular target the framework should update the goal of the agent with another destination so that the agent doesn't stop at one destination. To simulate a real-time scenario it is a must that agents keep moving to different targets.

Apart from destination it is also required to put in exits in the structure. In case of any emergency the agents should stop moving to their destinations and should find the nearest exit and should start moving towards it.

Finally since the evaluation of evacuation capacity requires an evacuation event, it is also required to simulate some kind of an explosion or an emergency which may cause the agents to evacuate the structure. This event should be sensed by our agents and should trigger a panic state in them to make them move faster to the nearest exits.

3.2 Requirements

The principal necessity of this framework is make a structure in which the entire framework could be examined.

The prerequisites for the system were that it should -

- **Agent should not pass through each other**

  When an agent is moving and it encounters another agent it shouldn't pass through it. Instead the agent should look for a new path to its destination. This functionality also helps in considering the fact that during emergencies people step over each other. So each agent should behave like an obstruction such that there is a consideration that people who fall are being picked up and not walked over.

- **Agents should not pass through obstructions**

  One of the very basic requirements that this framework needs is avoiding obstruc-

tions and excluding their position when calculating paths to a particular destination. In real world, humans come across a lot of obstructions while commuting from one place to another. They avoid these obstructions either by looking for a new path to their destinations, jumping over the obstruction, etc. In this simulation also we want our agents to have similar properties like find paths across the obstructions and jumping over some of them. These attributes are mandatory for our framework as only then will we be able to evaluate the evacuation properly.

- **Agents should not jump over each other**

  We are providing physical attributes to our agents to jump over some obstructions but we do not want to be jumping over each other. If we consider any real world scenario, we will come to a conclusion that there are very slim chances that we will have to jump over other person when moving from one place to another. So we need to keep a threshold height for our agents that keeps in check that our agents only walk through a stair height or wall with same height.

- **Agents should find the shortest path to their destination**

  A very valid attribute would be to find shortest path to the destination. Humans use their senses and calculations to find quickest paths to their destinations. Nowadays Google maps generally solves that problem but humans don't use that when they are in a museum or a mall. Similarly for the agents to reach their destination we can't use maps API, instead we need to provide them this feature so that they reach their destinations using the shortest path. Also, apart from mocking a human's behaviour we also want to avoid the case that our agents keep moving across the exit or boundaries and in case of evacuation they simply run to the next exit. This never happens. Agents should be moving inside the closed structure where they have their destinations.

- **Agents should react quickly to an emergency event**

  Agents should also have some kind of events to which they may react to and perform

some actions. Just like when humans see some danger they try to run away run it. Similarly agents should also get notified about some kind of emergency happening in the closed structure they are in. Also we are simulating evacuation so it makes sense to provide an attribute to our agents to react quickly to events.

3.3 Methodology

The methodology would include:

- Use of a robust game engine where simulating 3D environment and agents would be seamless and won't be an overhead to the memory of the computer.
  - Unity 3D
- Use of a path-finding library which may handle all the fuss of where agents may walk and finding shortest path to their destinations.
  - Unity's inbuilt Navmesh Library
- Using an IDE for development of code to achieve the desired functionality
  - Microsoft Visual Studio 2015
- SVN repository for Version control, backup, system roll-back and seamless transfer to multiple computers
  - Github
  - Sourcetree

A logic utilized in the outline and execution was to use an iterative way to deal with smooth advancement by learning and refining the steps from past strides.

The framework was implemented in C#. The primary reason this was done was that C# is a robust object oriented programming language that is quick, mature and all around

bolstered. It would encourage future mix with a tremendous measure of work being done in the interactive entertainment industry.

3.4 Goals

- To develop agents with capabilities of moving to an assigned destination using shortest path
- Provide the environment with smart obstructions which the agent may avoid and complicated paths like a ramp or a stair which the agent may use to reach their destination in real-time.
- To have ability to adapting and reacting to the movement of other agents.
- To respond to an emergency and evacuation the structure.

Chapter 4

Implementation

This Chapter sets out points of interest of the implementation of the project, the issues that emerged, the way to deal with settling them and a portrayal of their answers. Issues for which worthy arrangements were not accomplished or finished are likewise featured. These issues were not coordinated into the last framework. The whole source code in addition to an assembled rendition of the program is available on GitHub.

4.1 Approach - Overview

This is a known fact that all individuals behave differently when they are present in a crowd as compared to when they are alone. So this current implementation focuses on exploring physical and social group attributes which directly affect an agent's decision-making process when they are present in a crowd during normal or emergency situations. The framework further tries to integrate social and crowd related assumptions and conclusions from the field of social psychology to mock actual social interactions that happen in real life. The framework that is being developed will be capable of generating efficient and vivid behaviors required for large scale crowd simulations. By vivid, it doesn't mean that in current simulation this framework will produce life like behaviors but instead, its decision making capability will be like a human brain. This means that emphasis will be on design and procedural accuracy.

4.2 Why Unity

Unity is a game engine which is used to develop games for mobiles, desktops and consoles.

The reason for choosing Unity over other Game Engines was that it's really simple to use. The UI provided for building game environment is pretty straight forward. It has a huge developer community that provides support. So there is always a solution to any problem that arises. Apart from these, it uses JavaScript or C# as the core languages. Both of them are pretty easy to learn. It also comes with an IDE Mono-Develop, so that takes another overhead away of installing an IDE for programming purpose. Another feature that makes it stand apart is creating 3D simulations. The engine handles creating 3D objects and structures with a lot of ease. The API for unity is also very rich. It comes with a build in Navigation framework - NavMesh which is being used as a core dependency in the current simulation. One of the most unique features that it provides is exposing public variables on the inspector to assign them values there itself either it's a primitive type or a GameObject, refer Figure 4.13. Last and the most important of all is that unity comes for free. So it may be used to build any of the projects, simulations. etc without worrying about the wallet.

4.3 Framework - Crowd Simulation

A framework was a necessary requirement as there needed to be separation between different components in the system as not all of them would be fully developed. Each section should be 'plug and playable' and switchable at a later stage if a better solution came along. There were four main design issues that needed to be resolved;

- The world in which the humans operate
- The path-finding for humanoid agents
- The rules by which the agents operated and interacted, and

- The agent evacuation trigger

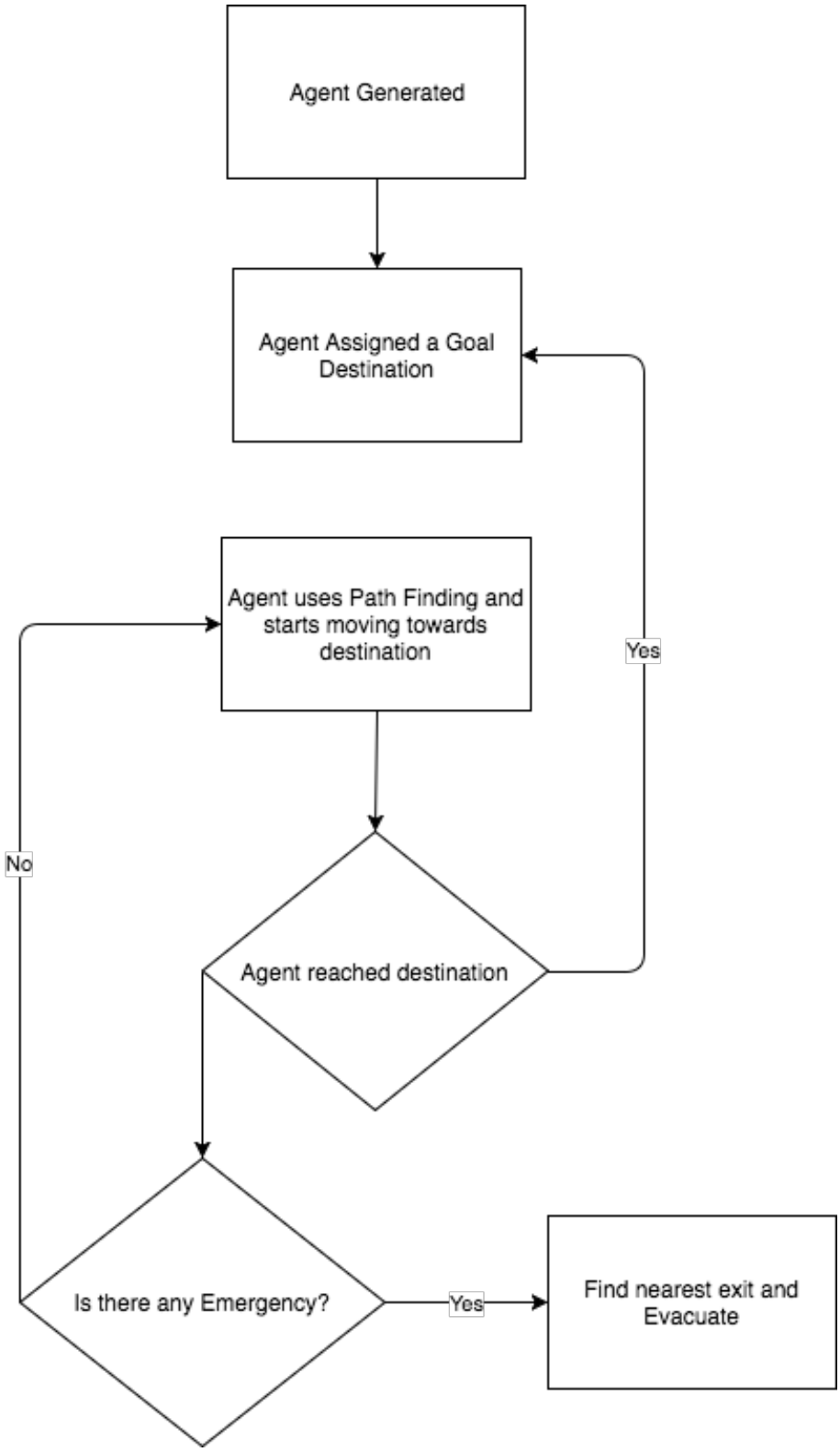

Figure 4.1: State Diagram for an Agent Lifecycle

4.4 World

There were two issues that should have been considered when choosing how to represent the world. The first was what should have been represented, and the second was the way that data will be utilized by the agent to explore the world over. After a considerable measure of understanding, I found that larger part of the papers out there speak to the world in networks and keep the coordinates of these networks in memory. So when the agent needs to move to a point, they apply some path-finding algorithm utilizing grid coordinates and continue refreshing the position of the agent until the point that it achieves the goal coordinates. So this was the first approach to consider while implementing the world for Crowd Simulation.

But there was a lot of scope for research in this field. There were some papers which considered sensing as one of the approaches to make agents reach their targets. So to begin with making agents move, they were provided with sensing attribute. The world was not divided into any grid, it was simply making agents move towards the target and avoiding obstructions using sensing. But this didn't work. Agents moved in a very weird way using sensing. While moving towards their target the started moving exactly in direction of the destination and when they encountered any wall or obstruction they moved away from it. This didn't make any sense while simulating. Humans never move like that. They find feasible paths to their destinations and move through them. So because of this reason this approach was dropped (discussed in detail in later sections).

The next thing was the obvious choice of representing the world in some form of grid and applying path-finding to make agents reach their targets. Since the simulation was done on unity there were a lot of frameworks provided by the engine to facilitate path-finding. So some research was done to choose the best fit for simulating crowds.

4.5 Deciding on a Path Finding Library

A couple of words about my encounters with a portion of the path finding libraries out there.

- **Aron Granberg's A* Project**

  The first library that I tried was Aron Granberg's A* Project. While I was looking for a good library to use for pathfinding this was a very straightforward choice for many game developers. I went through the features of this library it felt that it had pretty much everything but it was for a price of $99. A free version was also available but it didn't have any local avoidance, so it was no good. I found it pretty good while going through it.

  So I integrated it into the projects and it worked fine but it had some essential problems.

  - **Scene Loading**

    The library creates a huge overhead on the CPU performance plus the scene loads a lot slower. With this library, the scene loading time went to 5-10 seconds. Just out of curiosity removed this library from the project and the loading time dropped to 1-2 seconds. Since this was a pretty dramatic difference, the simulation was better off with this library.

  - **RVO Local Avoidance**

    Throughout reading about the library it was highlighted everywhere that this is the library's strong point. For instance, agents were getting stuck at random places where they should have gone through. Also around the corners, they were not able to go through. I assumed that there was some setting buried but I couldn't get it right. However, the important thing to notice about avoidance in this library is that it uses RVO algorithm and the performance of the agents in a huge crowd was impeccable. the agents never went through each other

or intersect, but when walls and corners were introduced the performance got worse.

- **Licensing issues**

  It's not available for free. UNC (University of North Carolina), owns the copyright for the RVO algorithm. So it was asked to remove RVO from the library or pay the licensing fees.

- **UnitySteer**

  Unitysteer was free and an open source, but I was not able to implement it properly. On reading about it it sounded really good, even on the demos and videos, but I couldn't get it to work. I assume it's for more advanced users and I chose not to proceed with this.

- **Unity's built in NavMesh Navigation**

  Since a lot of time was spent on exploring third party libraries. I decided to try Unity's built in Navigation System. It used to be a part of Unity Pro only but sometime around 2013 Unity added it to their free version. So this was also a reliable choice and it worked really well. The current simulation uses Unity's Navmesh. There are some good and bad sides of this library.

  - **The Good**

    It's quick. Initial response when running it for the first time was that it was too fast. The simulation was easily able to support 4-5 times more agents than I anticipated without any lags in pathfinding. The paths were getting updated really quick. Also, there was no issue with FPS as well. The simulation was done starting with 200 agents and went as high as 1200, still, the simulation was running smoothly.

    Easy to setup. Getting Unity's NavMesh to run was really easy. What effec-

tively did require was a single line of code.

$$agent.destination = target.position; \tag{4.1}$$

The best thing about Unity's NavMesh was it may be easily utilized by anyone with little or no experience with game development. All that was required for generating a NavMesh was importing the component to the scene, providing it with walkable and un-walkable areas and finally setting the properties of all agents to navigation static and providing with whether they are walkable or not.

Good pathfinding quality. By good quality what I mean is that agents were navigating throughout the world without getting stuck at places and didn't have any problems moving through tight spaces as well like they did for Aron's A* library. The library worked as the simulation needed, plus it the agents moved really swiftly without any extra work for smoothing the paths.

– **The Bad**

Well, not exactly a bad side of the library but there is something that needs to be taken care of when using this library. The local avoidance for the library is not that great. In case of a large number of agents, they start intersecting and jiggling around each other. Fortunately, the library is very easy to extend and I found a fix for these problems, so it wasn't that bad.

This sums up pretty much everything encountered during the search for a pathfinding library. It's better to stick with Unity's NavMesh (inbuilt navigation library) as it works really well with an RPG or RTS games, it's easy to use and it's free.

4.6 NavMesh

The Unity Navmesh framework depends on the following characteristic features:

- NavMesh

  It is a representation of the walkable surfaces present in the game world using which the agent may commute from place to other. The mesh is created after we specify which areas are walkable and which are not. The framework then creates a mesh over the surface from the level geometry. This is referred to as Baking.

- NavMesh Agent

  It is another component that the framework provides us for implementing movable objects in the game. They use the information provided by the NavMesh to navigate in the game world from one location to other. Apart from finding paths using NavMesh, they are also capable of evading each other (other agents) and dodging any moving obstacles which are defined before baking the Navmesh.

- Off-Mesh Link

  This component helps define routes to reach a destination which can't be reached using paths provided by the NavMesh. These are the cases when it's needed to implement a hole or a fence and the agent should have the ability to jump of these structures. This can be achieved using Off-work links.

- NavMesh Obstacle

  These are again a very important component of the framework. For any simulation, it is required to implement some obstacles so the agent navigates around it to reach the destination. These components need to be dodged or avoided by the agent while moving around the game world. In case of the moving obstacles, the agent tries to dodge it as much as possible. But for the case of stationary obstacles, the baking process creates holes in place of stationary objects so that those regions are not considered while finding paths to a destination as those areas can't be navigated.

4.6.1 Building a NavMesh

Before explaining how a Navmesh is created its required to understand what navigation static objects mean and how they are helpful. In Unity, while building any game or simulating any use case, the engine needs to know which objects are movable and which are static to facilitate optimizations. This information can often be pre determined to the game engine such that it won't be invalidated by changes in the game object's position. Considering a simple example, rendering may be a lot optimized if we make a single object by cumulating various static objects in a single one which is known as a batch.

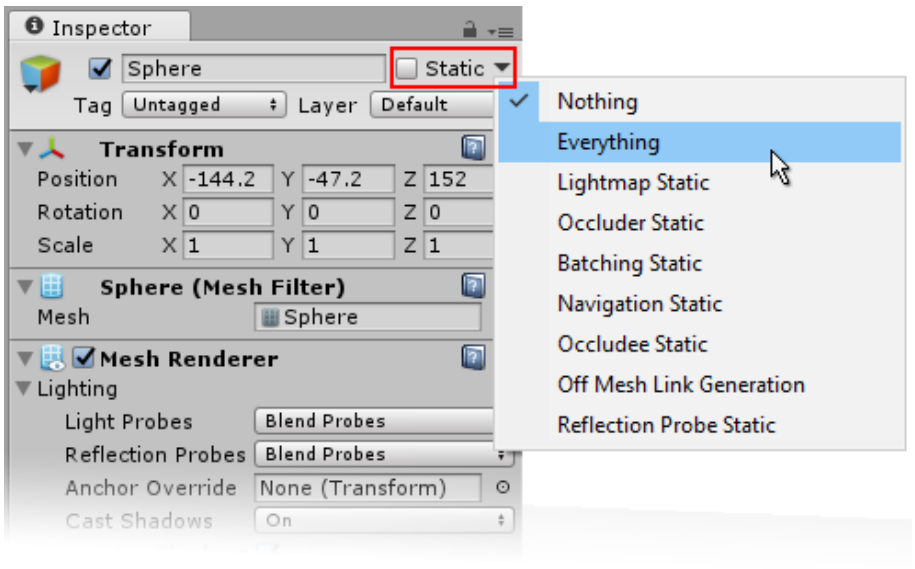

Figure 4.2: Navigation Static Property.
Retrieved from https://docs.unity3d.com/Manual/StaticObjects.html

Making a game object Navigation Static is very simple. The inspector toolbar in Unity has a static check box on the extreme top-right which holds a boolean that represents navigation static property of objects. It informs all the corresponding systems whether an object will move or not.

Creating a Navmesh from the base level corresponding to its geometry is called NavMesh Baking. The process starts with collecting the Render Meshes and Terrains for all the Game Objects which are Navigation Static. Then the next step is to process these Navigation Static objects and create a navigation mesh which provides unity with

all the walkable spaces over the surface.

To generate a NavMesh, select navigation option under Window menu option. Then set the bake properties required as per the agent that is being deployed. Following are the details of properties that are available to set.

- *Agent Radius* - Threshold distance from agent's center to any other game object.
- *Agent Height* - Used to determine how lower the structure can be for the agent to enter it.
- *Max Slope* - Determines the max inclination for a ramp to be walkable for the agent.
- *Step Height* - Determines the height for obstructions which may be walkable by the agent.

Following are the images representing steps to build a Navmesh in Unity.

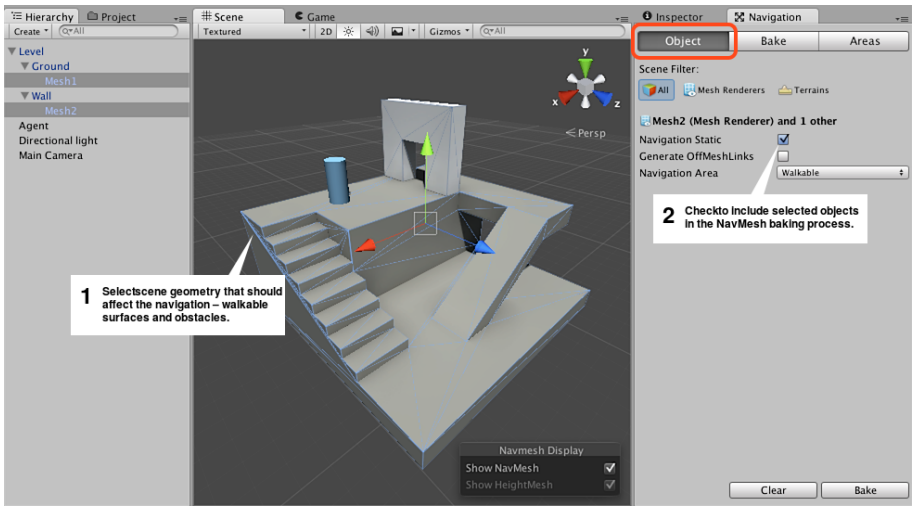

Figure 4.3: Building NavMesh - Steps 1 & 2.
Retrieved from https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html

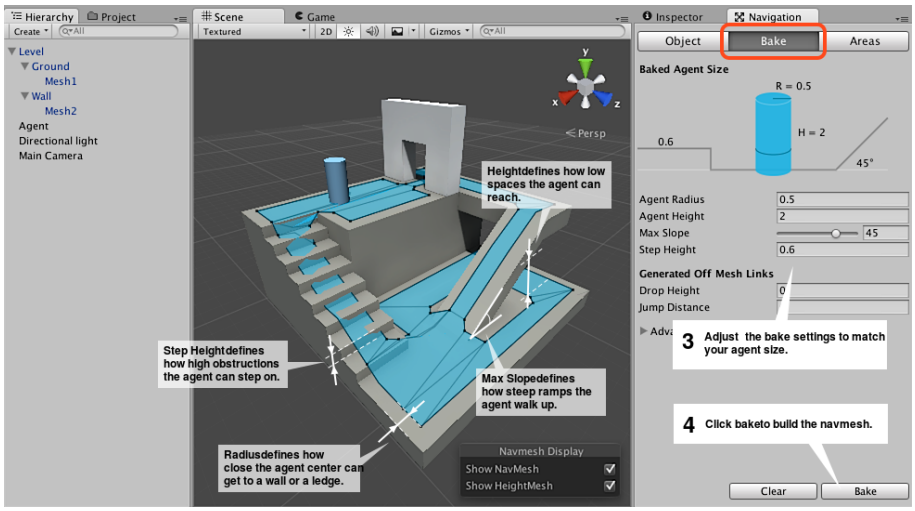

Figure 4.4: Building NavMesh - Steps 3 & 4.
Retrieved from https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html

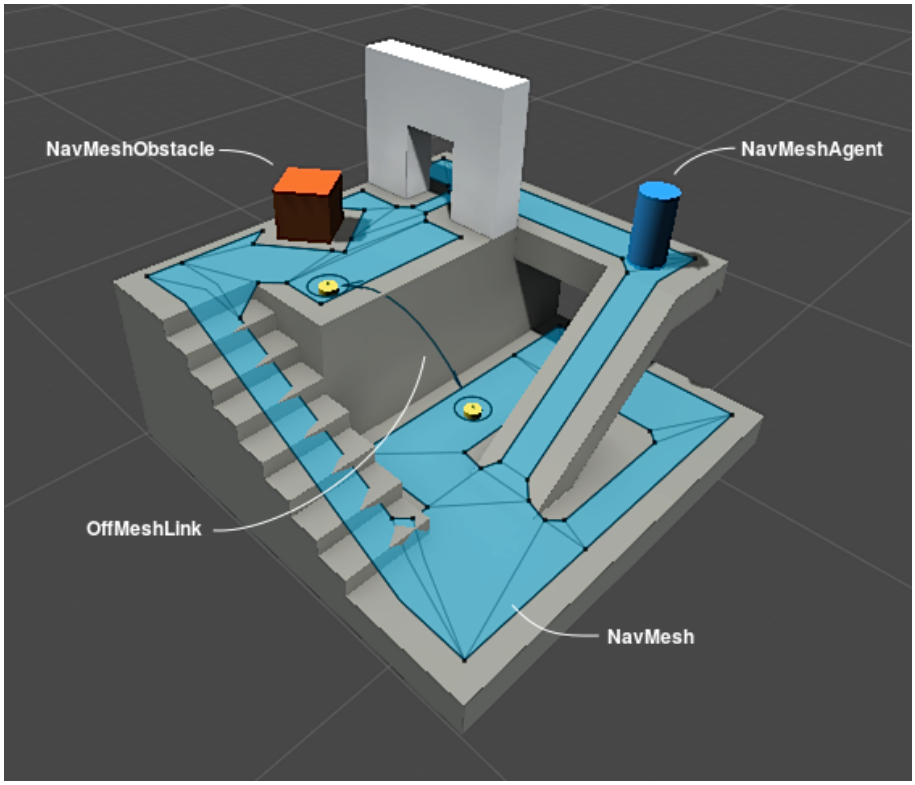

Figure 4.5: Final baked NavMesh.
Retrieved from https://docs.unity3d.com/Manual/nav-NavigationSystem.html

After Baking is complete, Unity represents the mesh as a blue surface over the base where we want our agents to walk. This is only shown when the navigation window is

opened.

4.6.2 NavMesh - Working

When you need to wisely move characters in your game (or agents as they are called in AI circles), you need to take care of two issues: how to reason about the level to discover the goal, then how to move there. These two issues are firmly coupled, yet very unique in nature. The issue of thinking about the level is more worldwide and static, in that it considers the entire scene. Moving to the goal is more local and dynamic, it just considers the direction to move and how to forestall impacts with other moving agents.

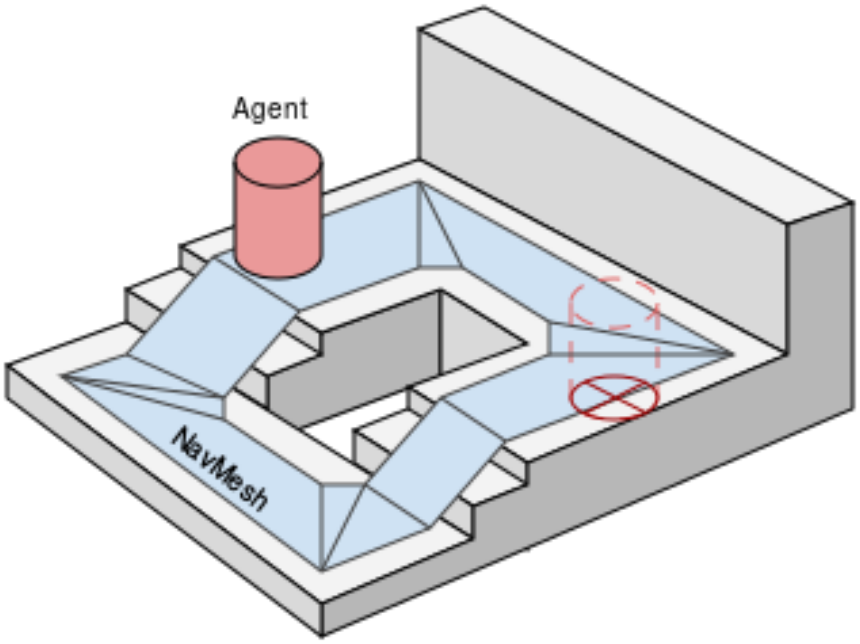

Figure 4.6: Walkable Areas.
Retrieved from https://docs.unity3d.com/Manual/nav-InnerWorkings.html

The navigation system needs its own data to speak to the walkable territories in a game scene. The walkable zones characterize the spots in the scene where the agent can stand and move. In Unity, the operators are depicted as cylinders. The walkable range is constructed consequently from the geometry in the scene by testing the areas where the agent can stand. At that point, the areas are associated with a surface laying over the

scene geometry. This surface is known as the navigation mesh (NavMesh for short).

The NavMesh stores this surface as convex polygons. Convex polygons are a valuable portrayal since we realize that there are no obstructions between any two points inside a polygon. In addition to the polygon limits, we store data about which polygons are neighbors to each other. This enables to reason about the entire walkable range.

Also, it is noticeable that the walkable are represented by the mesh is a little shrunk. This is because of the fact that the mesh only shows those areas where the agent's center may be placed. So even if you place a cube, cylinder or a sphere they are all treated same by the mesh. This point depiction of the NavMesh also provides better runtime efficiency as the data stored is small.

4.7 Finding Paths

To discover a path between two areas in the scene, we initially need to delineate the start and destination areas to their closest polygons. At that point we begin looking from the start location, going to every one of the neighbors until the point when we achieve the destination polygon. Following the visited polygons enables us to discover the succession of polygons which will lead from the start to the destination. A typical calculation to discover the path is A* (articulated "A star"), which is the thing that Unity employs.

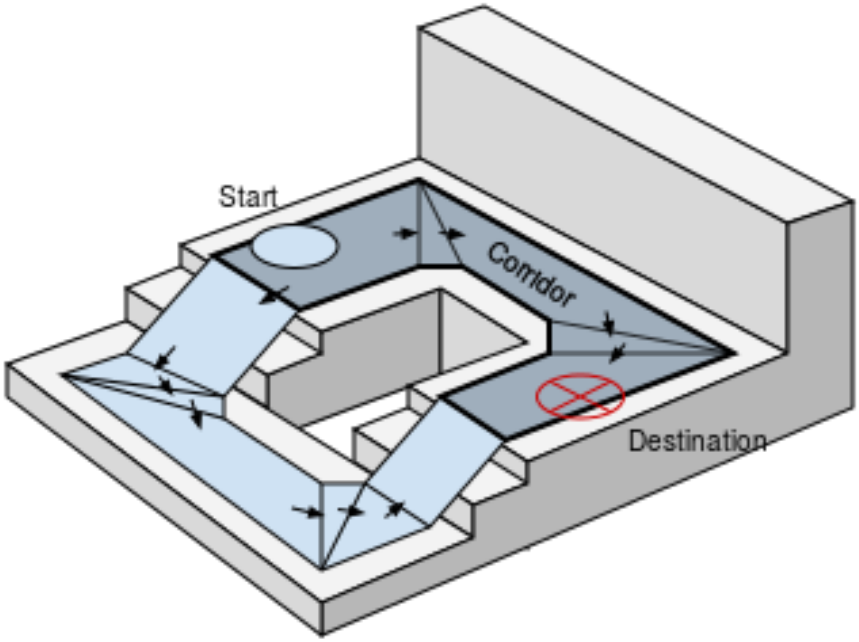

Figure 4.7: Finding Paths.
Retrieved from https://docs.unity3d.com/Manual/nav-InnerWorkings.html

4.7.1 Path Following

The succession of polygons which portray the path from the start to the goal polygon is known as a corridor. The agent will reach the goal by continually guiding towards the next visible corner of the corridor. In the event that you have a straightforward game where just a single agent moves in the scene, it is fine to discover all the sides of the corridor in one swoop and animate the character to move along the line segments connecting the corners.

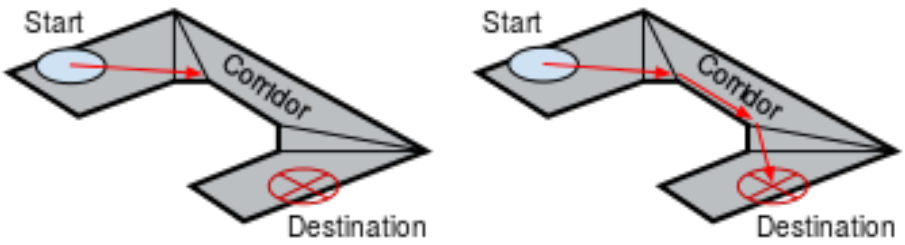

Figure 4.8: Following Path provided by NavMesh.
Retreived from https://docs.unity3d.com/Manual/nav-InnerWorkings.html

When managing various agents moving at the same time, they should go amiss from the original way while staying away from each other. Endeavoring to right such deviations utilizing a path comprising of line segments soon turns out to be exceptionally troublesome and error prone.

Since the agent movement in each frame is very little, we can utilize the network of the polygons to repair the corridor on the off chance that we have to take a little detour. At that point, we rapidly locate the next visible corner to direct towards.

4.7.2 Obstacle Avoidance

The guiding logic takes the position of the following corner and in light of that figures out a coveted direction and speed (or velocity) expected to reach the destination. Utilizing the coveted speed to move the agent can prompt impact with other agents.

Obstacle avoidance picks another speed which adjusts between moving the coveted direction and averting future impacts with different agents and edges of the navigation mesh. Unity is utilizing reciprocal velocity obstacles (RVO) to foresee and prevent crashes.

4.7.3 Moving An Agent

At long last, after steering and obstacle evasion the last speed is calculated. In Unity, the agents are reproduced utilizing a basic dynamic model, which additionally considers acceleration to permit more normal and smooth movement.

At this stage, it is conceivable to bolster the speed from the simulated agent to the Mecanim animation framework to move the character or let the navigation framework deal with that.

Once the agent has been moved utilizing either technique, the simulated agent location is moved and compelled to NavMesh. This last little stride is essential for a robust route.

4.7.4 Global vs Local Navigation

One of the most important things to understand about navigation is the difference between global and local navigation.

Global navigation is utilized to discover the corridor over the world. Finding a path over the world is an exorbitant operation requiring a considerable amount of power and memory.

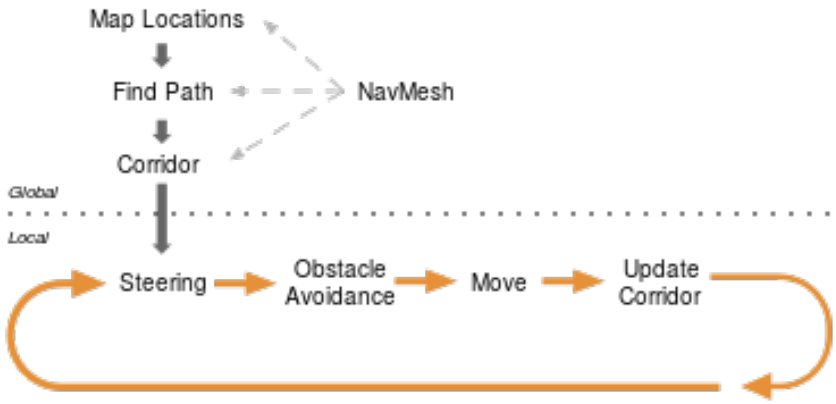

Figure 4.9: Navigation Loop.
Retrieved from https://docs.unity3d.com/Manual/nav-InnerWorkings.html

The linear list of polygons depicting the path is an adaptable data structure for guiding, and it can be privately balanced as the agent's position moves. Local navigation tries to figure out how to effectively move towards the following corner without slamming into different agents or moving articles.

4.7.5 Obstacle Scenarios

Numerous uses of navigation require different sorts of obstructions instead of simply different agents. These could be the typical boxes and barrels in a shooter game or vehicles. The obstructions can be dealt with utilizing local obstacle avoidance or global pathfinding.

At the point when an obstacle is moving, it is best dealt with utilizing local obstacle avoidance. Thusly the operator can presciently stay away from the obstacle. At the

point when the obstacle ends up noticeably stationary and can be considered to obstruct the way of all agents, the hindrances should influence the global navigation, that is, the navigation mesh.

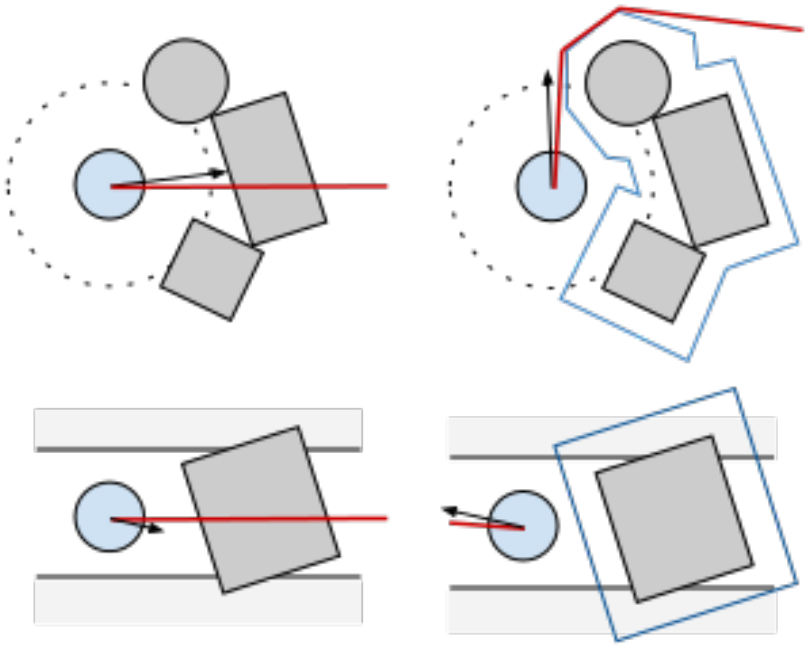

Figure 4.10: Two cases of Obstacles.
Retrieved from https://docs.unity3d.com/Manual/nav-InnerWorkings.html

Changing the NavMesh is called carving. The procedure recognizes which parts of the obstacles touch the NavMesh and cuts gaps into the NavMesh. This is a computationally costly operation, which is yet another convincing reason, why moving hindrances ought to be taken care of utilizing collision avoidance.

Local collision avoidance can be frequently used to control around sparsely scattered hindrances as well. Since the algorithm is local, it will just consider the following quick impacts, and can't direct around traps or handle situations where the obstacle hinders a path. These cases can be tackled utilizing carving.

4.8 Destinations

The destinations in the current simulation are the objects where the agents need to reach in order to accomplish their goal. These destinations are represented by the red colored cubes which have been placed throughout the world. Once an agent reaches its goal destination, the framework updates its destination to a new destination object by dynamically selecting one from the list of destinations. A sample of how a destination looks in the current scenario is shown in the figure 4.11.

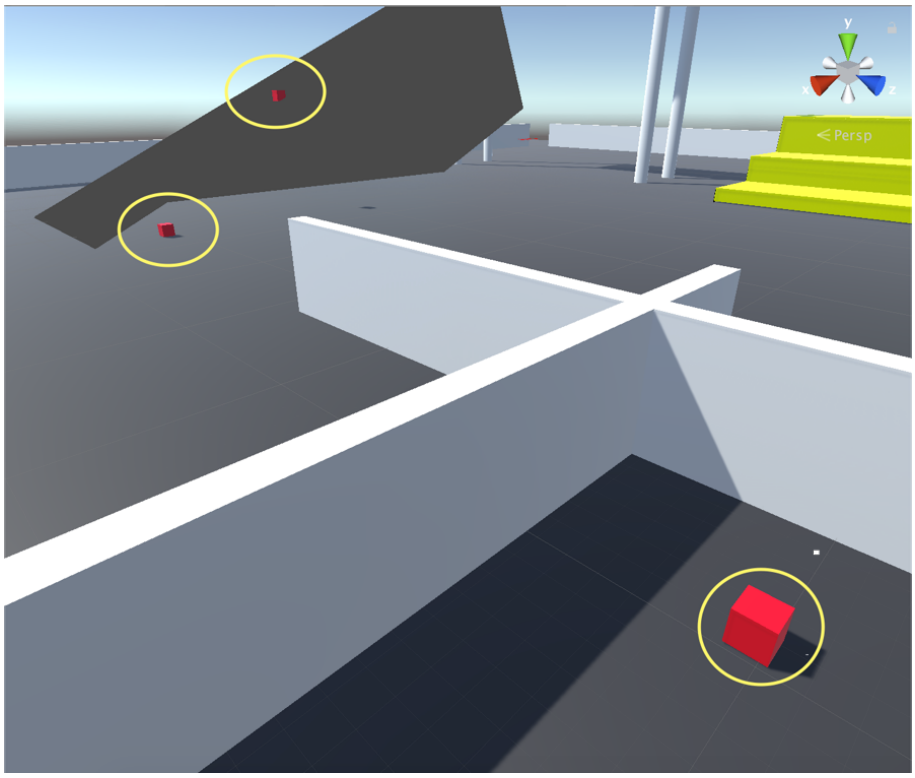

Figure 4.11: Destinations represented by red cubes in the simulation

4.9 NavMesh Agent

We have our NavMesh ready to use. Now it's time to introduce some agents to the scene which may navigate through the world. For this simulation, a Unity 3D object Capsule is being used as a sample agent and is being set in motion. This may be accomplished using the NavMesh Agent component and some scripts.

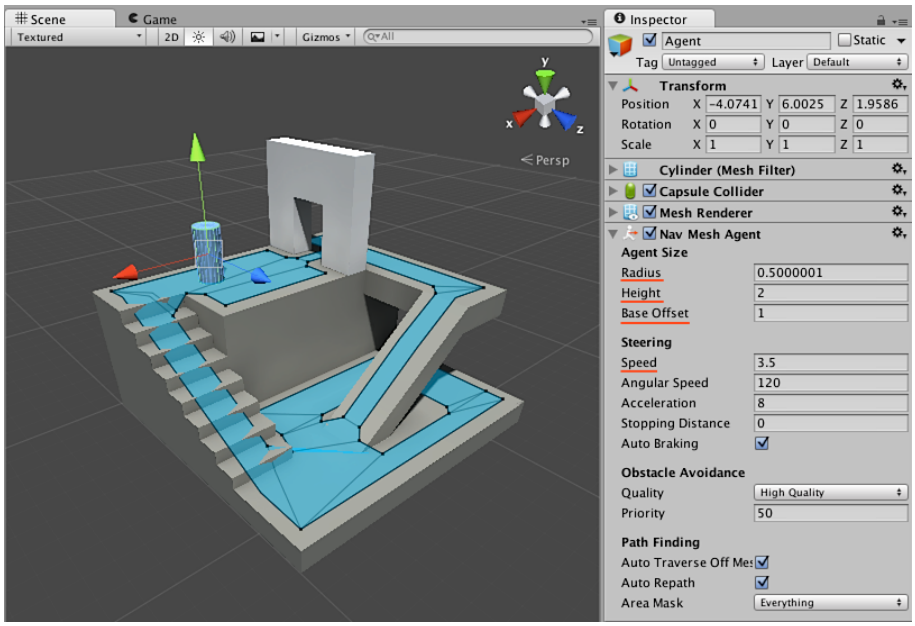

Figure 4.12: NavMesh Agent Setup.
Retrieved from https://docs.unity3d.com/Manual/nav-CreateNavMeshAgent.html

Following are the steps I followed to create an agent.

- Create a 3D Game Object capsule.
- The default Capsule dimensions were height: 2 and radius: 0.5. I left them as they were.
- Add the NavMesh Agent component from: Components - Navigation - NavMesh Agent

Now the agent is ready with all the attributes required to move over Navigation mesh. Besides this the agent needs another script which will provide the agent with destination coordinates, event handlers and resetting destinations on reaching a destination. So a new script is created by the name **NavMeshAgentSample**.

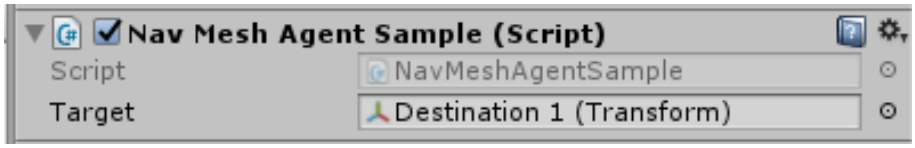

Figure 4.13: Script added to provide additional functionality to the Agent

4.10 Theory Underlying the Implementation

The agents are provided with all the attributes and functionalities needed to move in a closed structure as well as the feature to get notified about any emergency happening inside the structure. These features are achieved programmatically. These will be discussed a little later but before that, it is required to discuss the underlying theory behind these features. To model the agents correctly a lot of theoretical studies were done so that the agents mock human behavior as good as possible.

4.10.1 Behavior Modelling

The objective here is to illustrate human behavior which may be considered as outlining human's understanding, mental and physiological capacities in some form of a mathematical matrix which can be represented computationally to a certain level of accuracy. Diving a little deeper, this includes numerous modeling issues such as group organization, situation information, perception, and multi-agent interaction. To achieve such real behaviors for crowd simulation, the framework will be developed in a procedural fashion for mocking human like decision making. This includes making agent aware of the changes made in the world and consequently changing goals and attributes of the agent.

4.10.2 Situation Awareness

Situation awareness means providing the agent a property about being aware of all the external incidences being made currently and in the past in the simulation world. But, for this simulation, the agent won't be storing any information about the past occurrences. The agent will be provided with the attributes like sensing by developing an event management system which notifies our agents of emergencies and obstructions. Also, there is an attribute, walking speed, which may vary depending on the circumstances like in case of an emergency the walking speed will increase. Precisely, in the simulation, situation awareness is achieved by sensing and event triggers.

- **Sensing**

  The agent can obtain sensory information by continuously executing triggers which may use detection functionality and direct agents away from obstructions and dangers. Such detection feature will look for all the possible events occurring in the agents near by space including other agents, surrounding obstacles, and external events.

- **Reasoning**

  The agent may deduce some inferred knowledge about the current happenings of the environment established on the basis of unambiguous sensing of the virtual world. Hence some situational variables may be obtained subsequently. Threat level may be one of these variables. Second may be in-danger duration. The reasoning mechanism is provided with the navigation mesh and knows the position of static obstacles plus it provides the feature to avoid other moving agents as well. For instance, an agent will either get notified using sensory attributes or by the panic state of the surrounding agents.

4.10.3 Agent Attributes

Internal parameters of an agent which has a direct influence on their decision-making process are referred to as agent attributes. These may be of any type - static or dynamic. Static attributes of an agent are those properties which stay constant and separate its fundamental properties over a longer period of time. While dynamic attributes are more of the changing attributes, like emotional and group dynamics. Dynamic attributes contribute to the decision making process of an agent. Emotion attributes contribute to the personal behavior of the agent while group dynamics determine how an agent will behave in a crowd.

4.11 Programming the Agent

4.11.1 Making an Agent Move

With the NavMesh feature of Unity there is not much required programmatically to make an agent move. The script added to the agent - "Nav Mesh Agent Sample" is represented by a class which has a public property of

```
public Transform target;
```

Initially this property is set by dragging and dropping one of our destinations to the Target placeholder available in the script window present in figure 4.13

Then in the Update method provided by the **MonoBehaviour** interface, which runs every frame, we keep setting the Target property using

```
agent.SetDestination(target.position);
```

where **agent** object represents the NavMesh agent component of the agent.

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

4.11.2 Updating the Destination

Following code updates the destination once the agent reaches its current destination. A check is made for whether there is any evacuation state or not and the distance between the agent and destination is withing touching range. If this test passes, a random destination if captured and assigned to the target.

```
if (!evacuationState &&
    Vector3.Distance(agent.transform.position,
    target.transform.position) < 1.5) {
    int rDest = r.Next (1, 11);
    target = GameObject.Find ("Destination " + rDest)
        .GetComponent<Transform> ();
```

```
}
```

4.11.3 Populating Exits

The world had been provided with few exits which the agents will try to reach. These exits are populated dynamically using the following code.

```
ArrayList exits;
void populateExits(ArrayList exits)
{
    int i = 1;
    GameObject exit = GameObject.Find ("Exit " + i);
    while (exit != null) {
    exits.Add (exit);
    i++;
    exit = GameObject.Find ("Exit " + i);
    }
}
populateExits(exits);
```

4.11.4 On Evacuation scenario getting the nearest Exit

The discussion about the Event Manager will be done in the next section. For now, the only thing that's needed to be explained is that there is an event named "evacuation" defined in the EventManager script which gets triggered in case of any emergency. The following code gets the nearest neighbor and sets it as the destination for the agent. hence the agent stops moving towards the current destination and starts moving towards the nearest exit.

```
Transform getNearestExit()
```

```
{
float minDistance = Mathf.Infinity;
    Transform nearestExit = new GameObject ().transform;
    for (var i = 1; i <= exits.Count; i++) {
    GameObject currExit = (GameObject)exits [i - 1];
    float distance = Vector3.Distance (agent.transform.position,
                currExit.transform.position);
    if (distance < minDistance) {
    minDistance = distance;
            nearestExit = currExit.GetComponent<Transform> ();
    }
    }
    return nearestExit;
}
if (eventManager.getEvent('evacuation')) {
    evacuationState = true;
    agent.speed = 0;
    target = getNearestExit ();
    agent.speed = speedCache;
}
```

4.12 Evacuation

The next and final part of the current implementation was programming the evacuation module. Evacuation may be done in a number of ways. But for developing the current framework it was necessary to use the most efficient way as it should not put a lot of overhead on the CPU. Besides increasing the number of agents was already a heavy task so the evacuation process needed to be light.

The process for implementing the whole evacuation process was divided into parts.

4.12.1 Creating an Event Manager

The system needed an interface, which when attached to any game object gave it the feature of listening to the events broadcasted in the world and triggering some functionality based on the event triggered. If we compare this to the real world, it would be providing senses to the agents. For instance, if a person sees an explosion, eyes tell the brain that an explosion is happening and then the brain reacts with the running away reaction. Similarly, the event manager will behave as a sense to the agent which tells it that an event has occurred and the Event Trigger like a brain, in turn, replies with a reaction.

For creating an Event Manager, a class will be created which will have methods for initializing the EventManager, if it's not initialized, starting to listen to events, stop listening to events and triggering event like evacuation so the agents know what it needs to do. The following code snippet was originally taken from unity Event Messaging document and updated as per requirement. (https://unity3d.com/learn/tutorials/topics/scripting/events-creating-simple-messaging-system)

```
public class EventManager : MonoBehaviour {
private Dictionary <string, UnityEvent> eventDictionary;
private static EventManager eventManager;
public static EventManager instance
{ get {} }
void Init () {}
public static void StartListening (string eventName,
UnityAction listener) {}
public static void StopListening (string eventName,
UnityAction listener) {}
public static void TriggerEvent (string eventName) {}
```

```
}
```

4.12.2 Creating an Event Trigger

The EventTrigger class behaves as the central management for all the events captured. This class listens to all the events being triggered and takes some action based on the event. For now, the only event that the trigger is listening to is 'evacuation'. As soon as the event gets broadcasted, the event trigger for each agent forces agent to do some tasks.

First, it makes the agent stop by setting the speed to zero as a symbol of processing the explosion. The next thing it does is removing the current destination from the memory and starts finding the closest exit. Upon finding the closest exit it sets that exit to the new destination for the agent. The agent's speed is updated to "panic speed" variable which is being maintained for each agent. After all this is done the agent starts moving to the current destination, that is the exit with a higher speed which is the panic speed.

```
public class EventTest : MonoBehaviour {
private UnityAction someListener;
void Awake ()
{
someListener = new UnityAction (SomeFunction);
}
void OnEnable ()
{
EventManager.StartListening ("test", someListener);
EventManager.StartListening ("evacuation", evacuteAgent);
}
void OnDisable ()
{
EventManager.StopListening ("test", someListener);
```

```
EventManager.StopListening ("evacuation", evacuteAgent);
}
void evacuteAgent () {}
```

The evacuation process for one of the scenes is shown in figure 4.14 and rest of the evacuation scenes may be found under Appendix in Chapter 5 (Figure 5.16 and Figure 5.15)

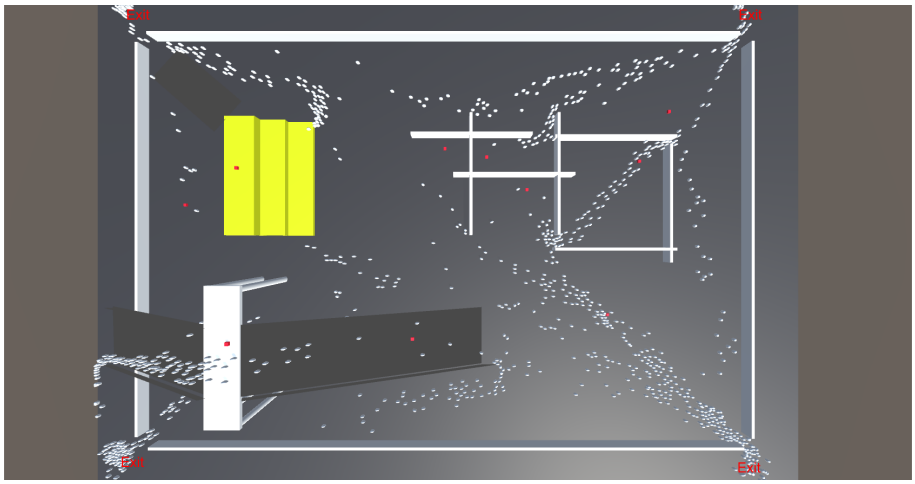

Figure 4.14: Scene 3: Evacuation in Process

This concludes with explaining the implementation of the system. The reason behind choosing any library or implementing any new functionality has been covered in depth. Now let's move ahead towards explaining the simulations that were done to complete the study.

Chapter 5

Simulations

As explained in the previous chapter that a number of libraries were used to initialize the simulation and these libraries may be configured and extended to change the simulations. For the purpose of testing the framework 3 sets of structures were developed in unity. These structures were different in the percept of placing obstructions, destinations and complicated paths at different positions. The simulations were run on these structures to analyze the long and short term performance. To get the performance of each section of the simulation a number of tests were run which tested different features of the system. Overall the crowd simulation worked as desired. There were no cases where agents were stuck or were not able to find a way in a dense crowd. The cases where a lot of agents accumulated near a single destination, tolerance was increased for threshold distance which decides whether an agent has reached the destination. The cases where there were emergencies, were giving the results as expected. The ability to see the results immediately for these alterations allowed for a very quick process to determine the best capacity for a closed structure for most efficient evacuation.

5.1 The Simulation System

Once all the libraries are set up and the simulation is started, the scene runs with all the agents being generated randomly from the positions of one of the destinations. The world is composed of a flat plane that represents ground, a number of red cubes that

represent different destinations, wider cubes combined to represent stairs, inclined plane representing the ramp and a number of capsules representing human models (agents). There are 8 displays (refer Figures 5.3 - 5.10) available which the user may use to view the scene from different locations. For example, the user may view from a position close to stairs to see if the agents are walking up the stairs properly or not as seen in Figure 5.11. The user may switch over to near the ramp to see whether the agents are walking up or not. This is shown in Figure 5.12. There are many other views available but the most important of all is the view from the top from where the user can see the whole structure along with the exits so that they may study the evacuation times to know what capacity might be the threshold for the current structure. One of the snapshots is depicted in Figure 5.1.

5.2 Scenes

To confirm the flexible nature of the framework three different structures were developed. The first map as seen in figure 5.1 is a structure where all the complicated walkable areas are separated from one another. For example, the stairs, the ramp, and the maze structure composed of walls are separated from each other. This scenario which looks simple is the most complicated one. The reason for this is that none of them are connected to each other or the exits. So this use needs to be studied for the cases where exits are not navigable from each of the stores in a mall, for instance. Also, the case where the agent has to reach down the floors to access any exit.

The second structure is illustrated in Figure 5.3. The map is now expanded to have more stairs and walls. Now we may see that the structure has started to become more cluttered. Again this is a scenario where the architects don't restrict the number of small rooms in a structure and apparently, this leads to obstructing the agents while evacuating during an emergency. this is a very simple fact but in this simulation, it was observed that the number of smaller structures inside a building should also be restricted as they

have a direct impact on the evacuation times.

The third simulation was done to see if providing connectivities to the exits from each part of the structure helps or not. Figure 5.2 shows that the structure has a comfy arrangement of smaller structures inside it. Apart from this, an attempt has been made to every destination, which is farther from the exits, to be connected to the nearest exit in any way. This scenario is being considered here to check if the evacuation times decrease for the same number of agents or not in comparison to the previous simulations where the structure distribution was complicated.

5.3 Appendix - Simulation Images

All the below images have been captured for different Simulations with an agent population of 1000.

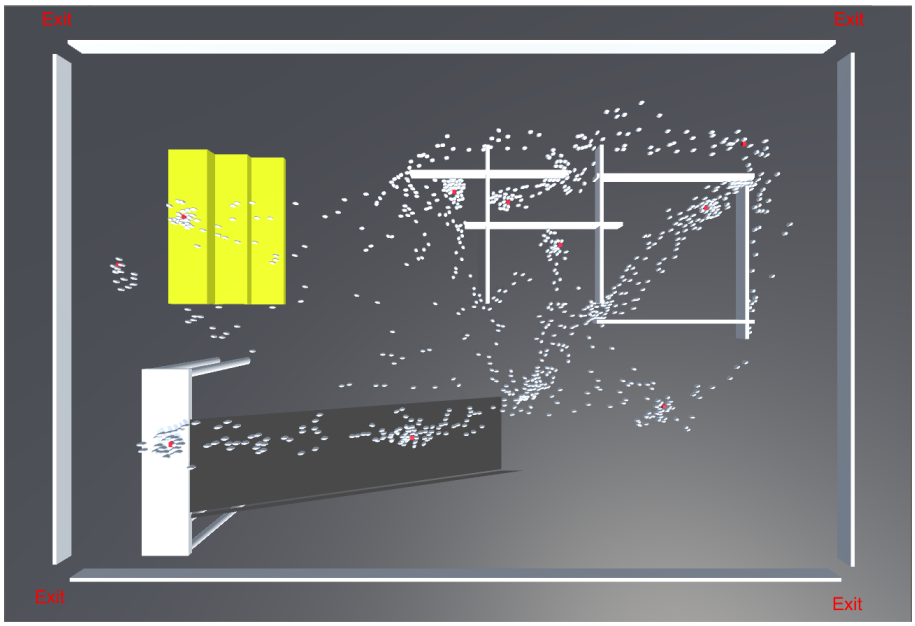

Figure 5.1: Scene 1: Top View of the Structure

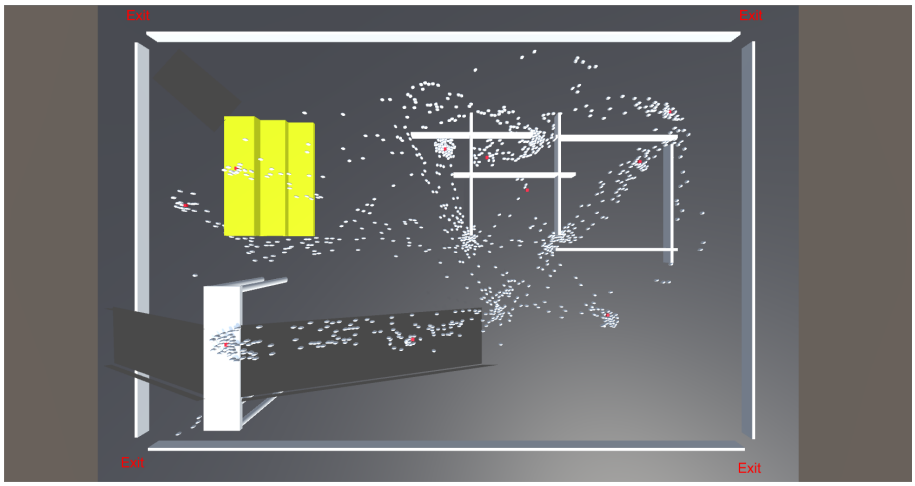

Figure 5.2: Scene 3: Top View of the Structure

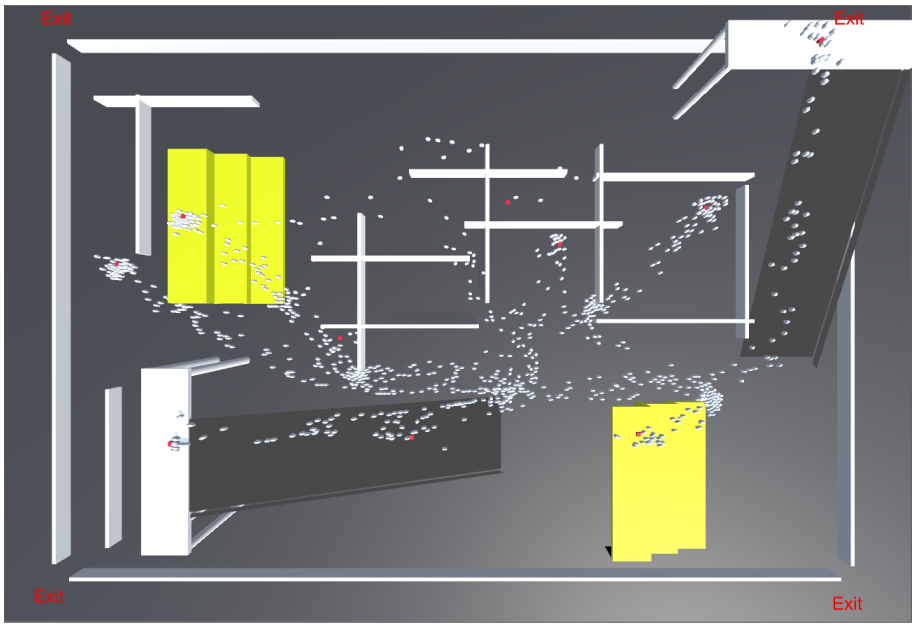

Figure 5.3: Display 1: Scene 2

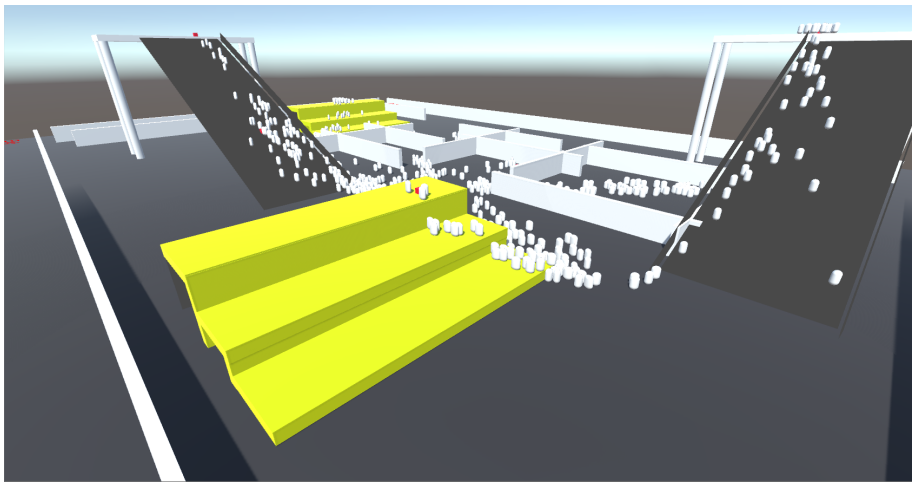

Figure 5.4: Display 2: Scene 2

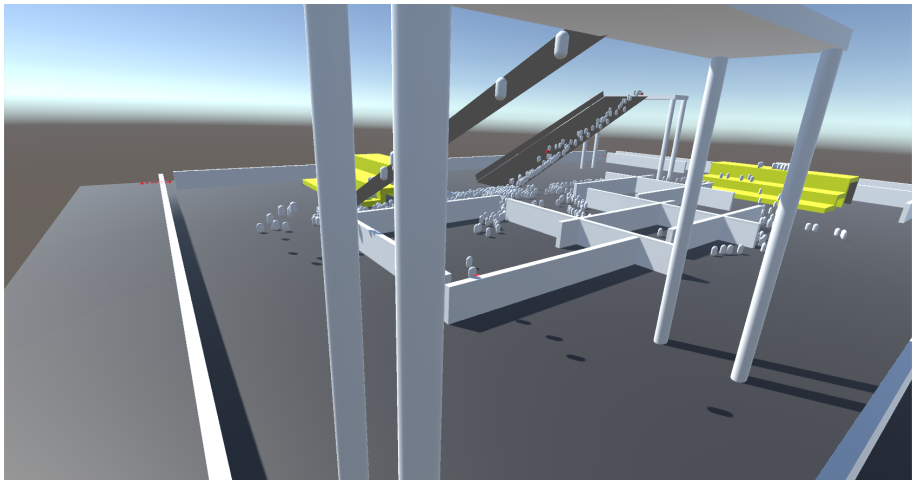

Figure 5.5: Display 3: Scene 2

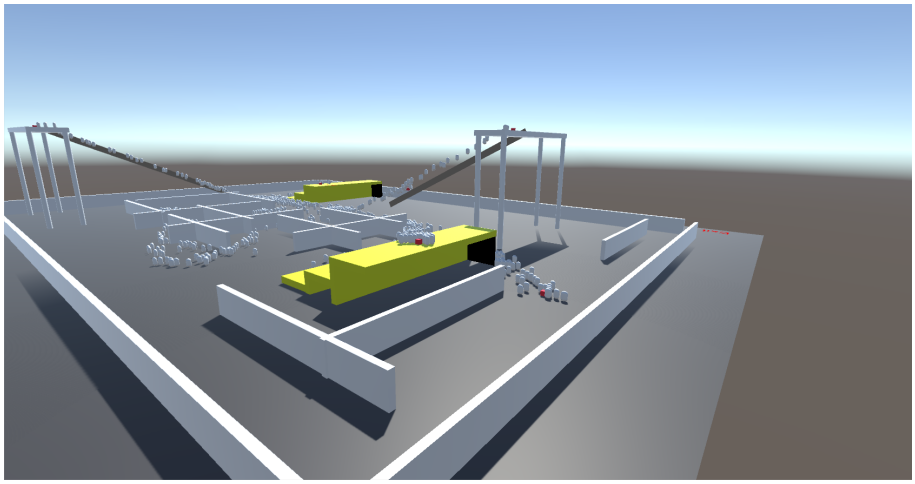

Figure 5.6: Display 4: Scene 2

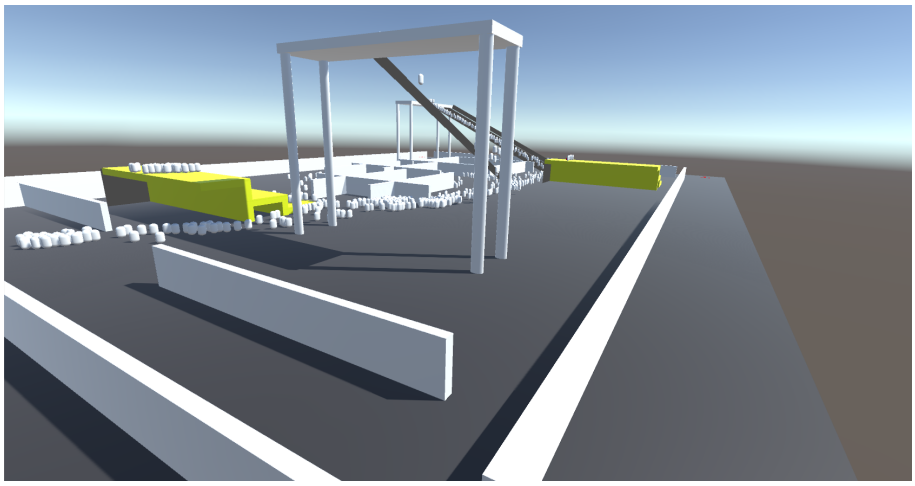

Figure 5.7: Display 5: Scene 2

Figure 5.8: Display 6: Scene 2

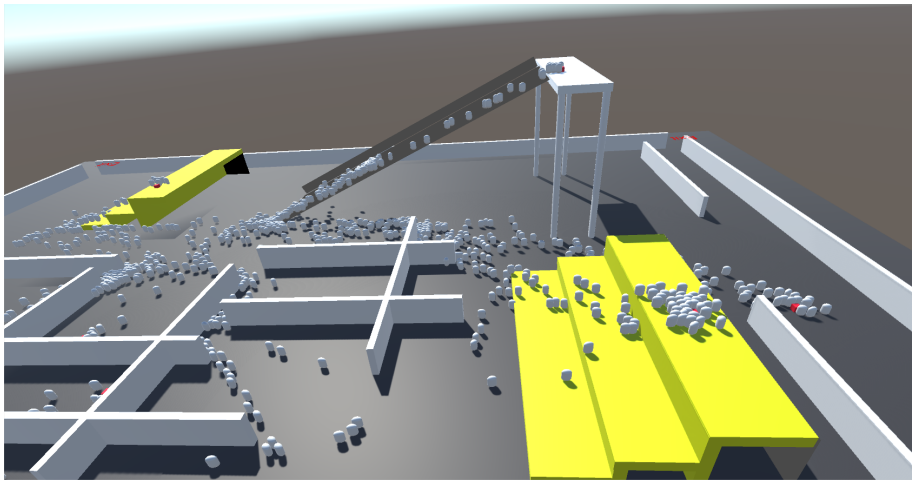

Figure 5.9: Display 7: Scene 2

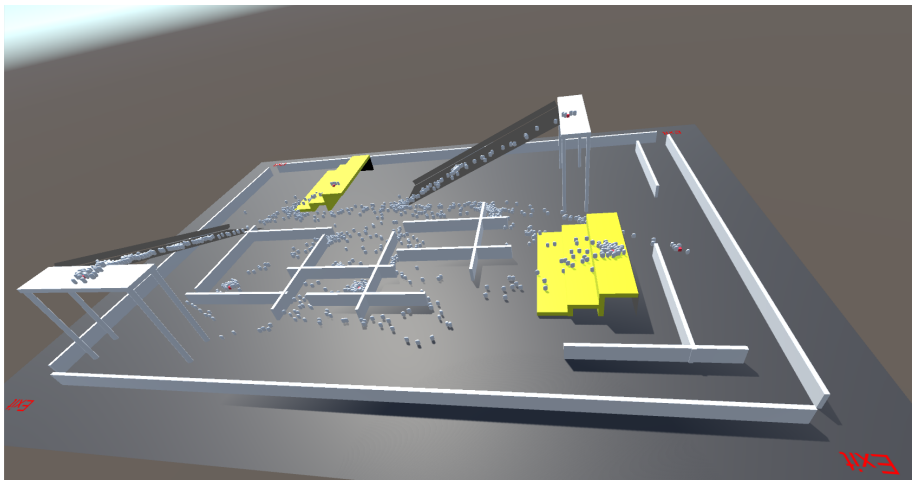

Figure 5.10: Display 8: Scene 2

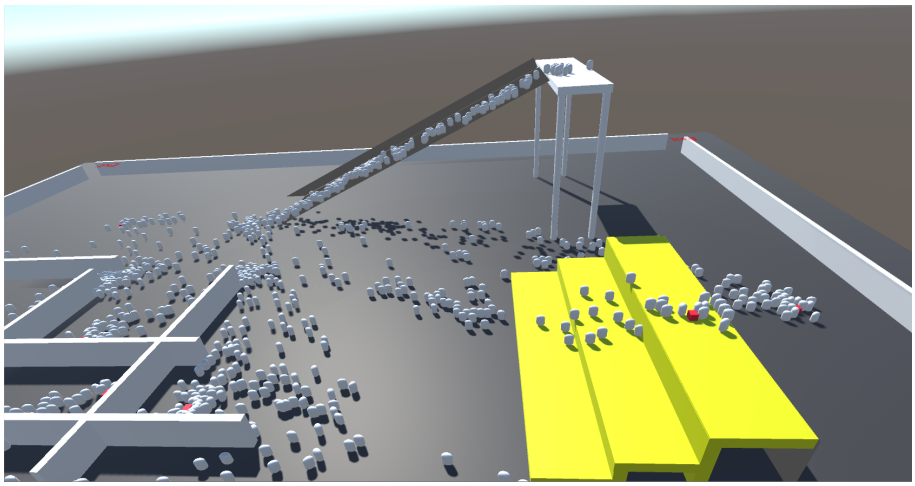

Figure 5.11: Scene 1: From Near the Stairs

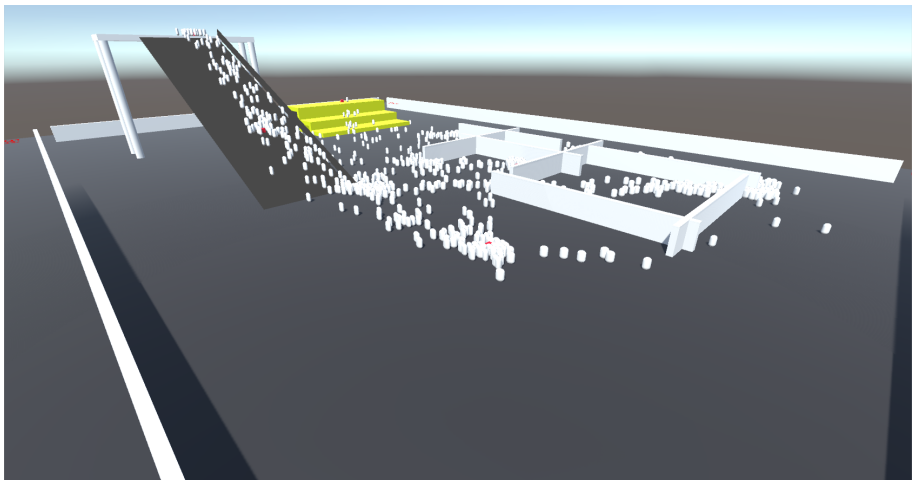

Figure 5.12: Scene 1: Front of the Ramp

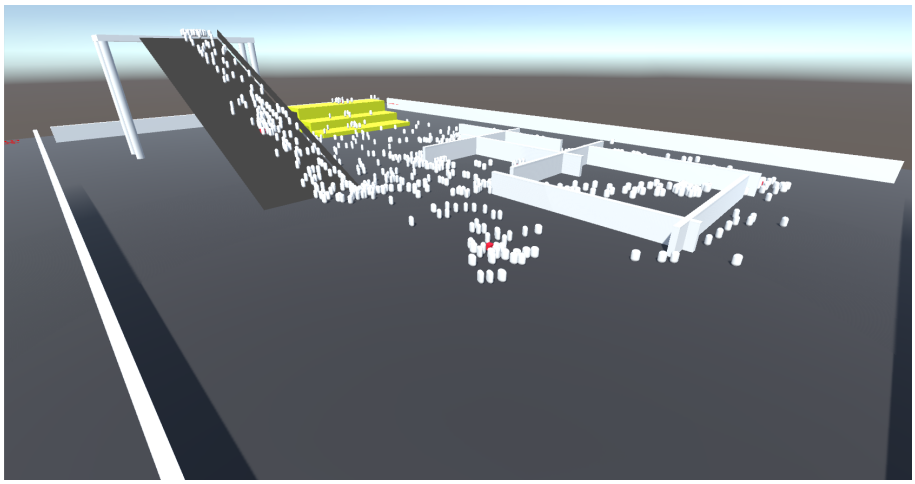

Figure 5.13: Scene 3: Front view of the Ramp

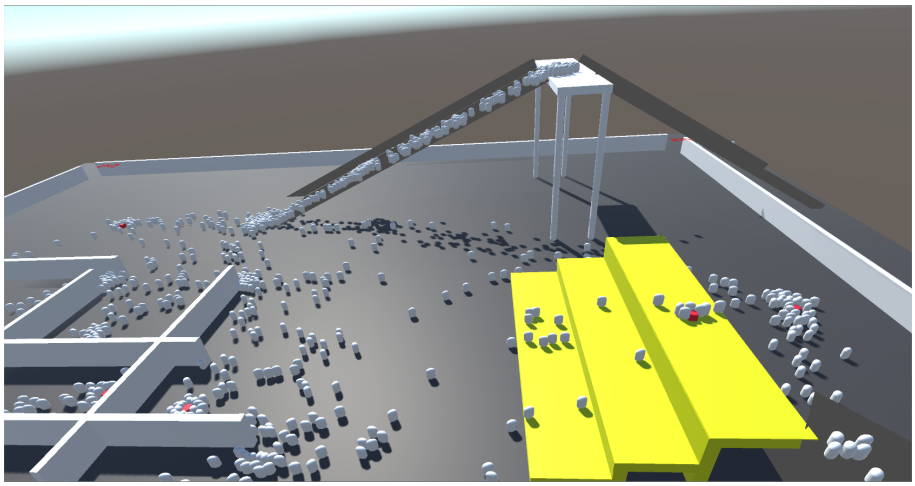

Figure 5.14: Scene 3: Stairs View of the Scene

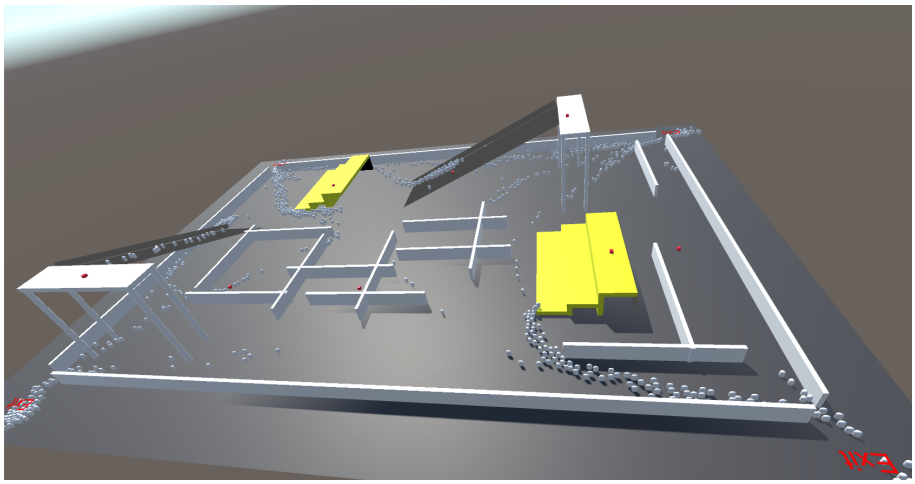

Figure 5.15: Scene 2: Evacuation in Process

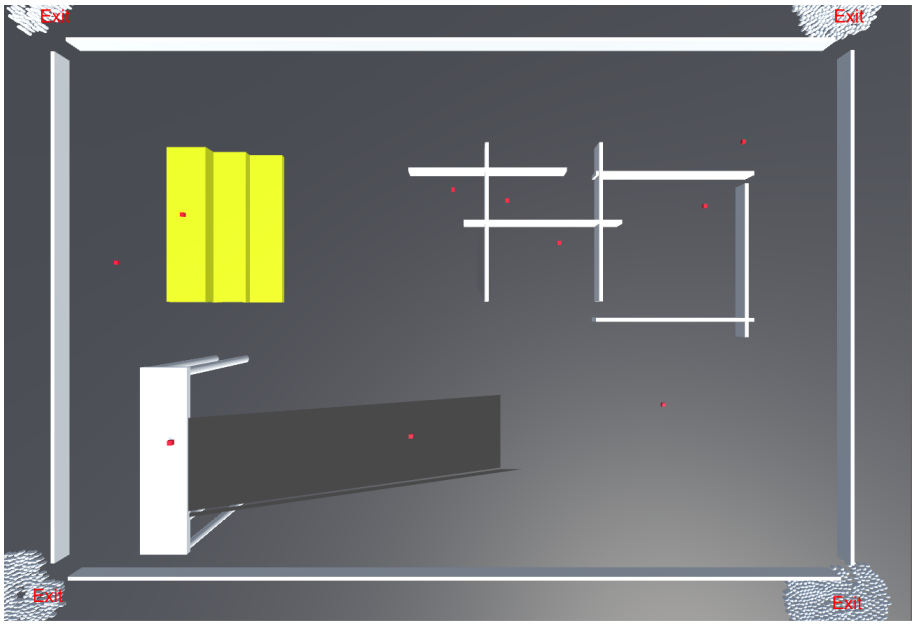

Figure 5.16: Scene 1: Evacuation Complete

Chapter 6

Results

This section talks about the results and outcomes from the current framework. The simulation had 3 variants. Each variant was put to test to get the threshold capacities using the evacuation times vs number of agents graph. We further talk in detail about the outcomes, performance of the current system and how much the outcomes resembled reality.

6.1 Breakdown of Performance

To determine how each section of the system was performing the evacuation times for each of the scenes were recorded. The results are shown in table 6.1. Each of the scenes was tested against an explosion which triggers the world to broadcast an evacuation event. The first scenario was a simple placement of all the structures in the center and none of them are connected to each other. Hence the performance of the system was satisfactory. This scene represents the case where the exits are not placed as per the design of the structure. So the performance should increase as we move to the next structures.

Number of Agents vs Evacuation Time	
Number of Agents	Evacuation Time (sec)
100	20.8
150	20.7
200	20.99
250	24.34
302	24.75
350	24.81
450	24.93
550	27.83
650	28.0
750	28.98
850	31.70
1200	43.38

Table 6.1: Results obtained from Scene 1

The second scene was a worse case scenario setup. This scene was only evaluated to scrutiny our system, whether it's performing as per the expectations or not. So the expectations from this scene were that the evacuation times will be greater when compared to the first scene. Exactly as we expected happened. The evacuation times for the same number of agents was more for this scene as compared to the first scene. The results are shown in table 6.2. The reason for this is pretty simple. When a structure is cluttered with a lot of smaller structures there is no space for the movement of people. So when the explosion happened and agents started moving towards the exits they got jammed. hence the evacuation tie was higher as compared to scene 1.

Number of Agents vs Evacuation Time	
Number of Agents	Evacuation Time (sec)
100	22.3
150	22.56
200	23.49
250	24.67
300	25.6
350	25.92
450	26.9
550	28.45
650	31.23
750	34.34
850	37.77
1200	49.02

Table 6.2: Results obtained from Scene 2

Talking about the third scene now, this scene was the perfect example of how rooms should be placed in a big structure to facilitate faster evacuation. In this case, we placed all the smaller structures very cozy. Apart from their placement, all the upper floor locations were connected to the ground floor by some kind of a ramp or stairs. So when the explosion happened, all the agents started moving towards the exits. Since the structure had a lot of space all the agents were able to evacuate the structure comfortably. Besides the upper floors were connected to the ground floor with some kind of passage. hence the agents were found using them to reach the closest exits. The results for this scenario are shown in table 6.3.

Number of Agents vs Evacuation Time	
Number of Agents	Evacuation Time (sec)
100	20.8
150	20.82
200	21.02
250	24.85
300	24.90
350	24.93
450	25.02
550	26.72
650	27.32
750	27.81
850	29.75
1200	35.04

Table 6.3: Results obtained from Scene 3

6.2 Framework

The framework was developed from the start. This was not an extension to any of the previous frameworks. As per now, all the components are working fine and the framework performs efficiently for as many as 1200 agents. The pathfinding for the agents is also working correctly. All agents reach their destinations using the shortest possible path. The agents do not pass through each other but make their way through the crowd. Agents have the capability of walking up an inclined plane and stairs as well. On introduction of any emergency, for this simulation an explosion, the agents are notified using a messaging system which seems to be working effectively. The messaging system was built in place of an alarm system whose purpose is to notify people about any danger. the messaging system achieves that. As soon as the agents are notified by the environment about any danger they stop moving towards their current destinations, find the closest exit and start moving towards it at their panic speed.

The times recorded for Evacuation versus the Number of Agents is plotted for all the three simulations below. Looking at the plots it's clear that as the number of agents increases over the threshold capacity of the structure the evacuation times starts rising.

So we may decide using the framework what should be the capacity of a closed structure for which it will facilitate the fastest evacuation.

Some debate may be done for the threshold capacity. The point at which the evacuation time starts climbing the graph may be considered as the ideal capacity. But if the threshold capacity comes very low then there may be a fault in setting up of the structure as well. As in the second simulation, the arrangement was faulty due to which the evacuation time went higher. So the end result obtained from this framework will be the best for the provided structure but not for the area on which the structure is built as the architecture of the building is a very important factor in deciding the evacuation strategies.

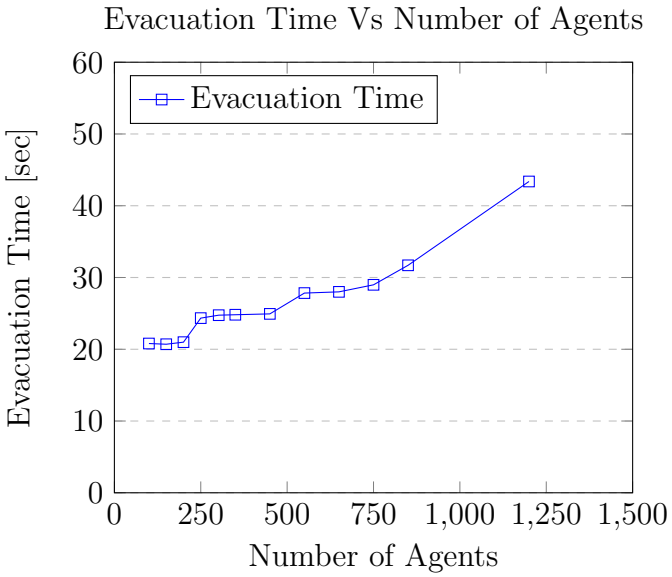

Figure 6.1: Evaluation for Simulation 1

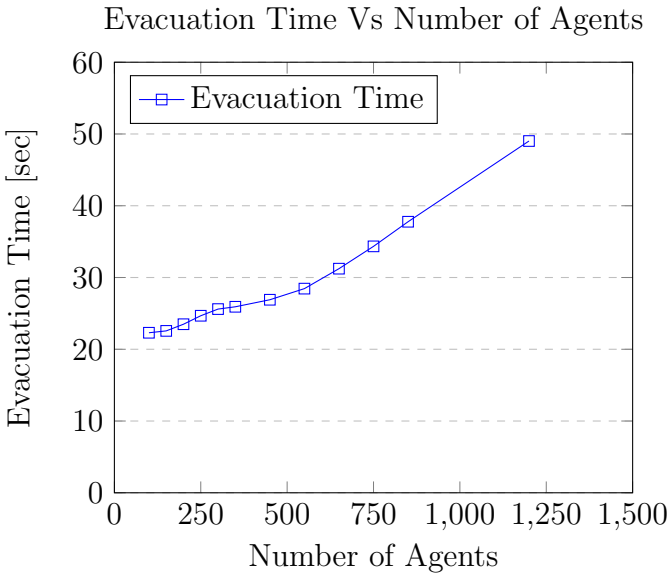

Figure 6.2: Evaluation for Simulation 2

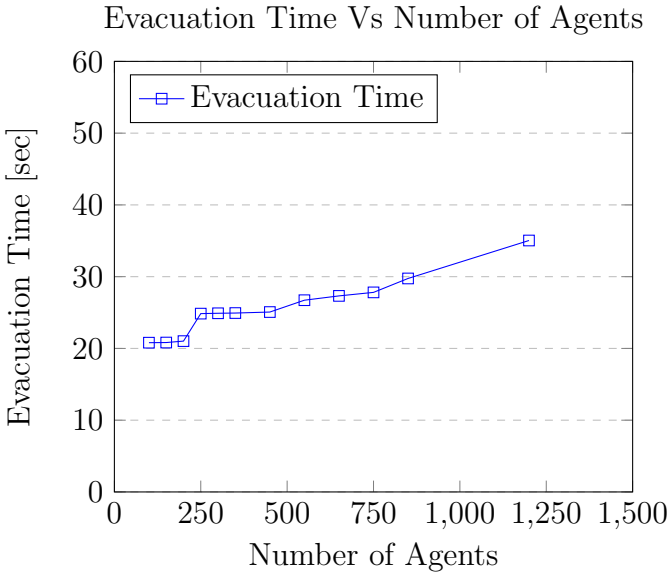

Figure 6.3: Evaluation for Simulation 3

6.3 Validity of Simulation

Looking at the results from the simulation it may be considered that the framework worked as expected. As per the understanding, if a closed structure has an enormous number of agents and there arises a case of emergency, then it will be very difficult to

evacuate the structure because the exits cannot accommodate more than 2 or 3 persons at the same time. Hence there will be a queue near the exit and the density of that queue depends on how many people are present in the building. So it may be inferred that more the number of people present more the time it will take for them to evacuate.

The current framework gives the same output. With increasing number of agents, the evacuation times increase. Hence it may be concluded that the framework is performing as expected and the results obtained from it are apt.

The accuracy of the framework is something that needs to be verified. This can only be judged if we plug in our framework with some of the architectural software where we may provide a structure to the framework and it gives us the output. Unfortunately, this part couldn't be done as these softwares are mostly paid.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

This thesis outlines a flexible tiny crowd simulation system. The user may provide a design of the structure to the system and it will provide the evacuation time based on when the emergency occurs. The system has a pre located set of destinations which the agents try to reach. The system allows dynamic generation of agents capable of navigating through the ground, stairs and the ramp. the system uses Unity's inbuilt navigation library to provide agents the feature of navigating to their destinations. the system also has a feature of introducing an emergency case to the simulation. The user may introduce an emergency at any point of time while the simulation is running. The structure has the capability of detecting any case of emergencies and broadcasting events to the agents that an evacuation should be done. On listening to this event, agents stop moving towards their destination and start moving towards the nearest exit. Overall the system is flexible enough to be easily plugged into any other system and be adapted to provide the threshold capacity for it.

The system presents a viable method to study emergent crowd flow levels and see how they would react to emergencies with a closed structure. Modifications can be made very quickly to the structure layout again to observe how changes make an impact on the crowd flow. The ability to place destinations is also provided to scatter crowd as desired while

building a model of a closed structure. This may facilitate the study of how a new destination added to the structure affects the crowd congestion areas. Overall the agent based crowd simulation system creates an environment where emergent crowd phenomenon can be investigated.

7.2 Future Work

7.2.1 Behavior

To improve the realistic nature of the agent's behavior, it would be desirable to extend upon the current behavior aspects of the agent. The goal would be to add more attributes to the agent apart from speed and distance calculation. Then by linking the variations together under concepts of aggression, patience, competency, and urgency and maybe using fuzzy logic algorithms an even greater level of realism might be achieved.

7.2.2 Create and Destroy Agents using FSM

In the current implementation, the agents keep roaming inside the structure until any emergency occurs. From the point of realism, it's not that sensible. There should be a functionality where the agents enter the structure, complete some goals and leave the premises. Another functionality should be there to introduce new agents to the scene randomly to balance out the departing agents. So there may be a Finite State Machine representing agent states where it directs them to there next states while roaming in the structure. This would make the agent's goal much clear and the simulation would make good sense.

7.2.3 Adding more floors to the Structure

Currently, the implementation only considers the structures with a ground and a 1st floor. The core problem being addressed here is the evacuation time. For such a small structure

the evacuation problem won't be that big. It's the structures which are 15-20 stories long which would really benefit from this framework. So the model currently being studied can be made to include many more stories so that the study becomes more beneficial. There may be some updates required in the framework for studying such tall structures, but the current framework will be able to handle such inputs as well as it is made very flexible.

7.2.4 Implement IVRS to facilitate faster Evacuation

The world (VR environment) here in the current system is intelligent or we may put it as it has the ability to notify the agents whenever there is a case of an emergency. Apart from this the current implementation also recognizes the obstacles where the agent cannot walk plus some complicated walkable surfaces like a ramp or a stair. But there is still a lot of scope for improvement in the current VR system. Currently, the VR space only notifies the agents using some alarm to notify them to evacuate the space. To extend this, the current VR system may also show the paths to the agents to the nearest exits. Visual stimuli is a faster way of communicating things. So if the VR system shows the path to the agents, they do not need to think in their panic state. As in panic, people are bound to make mistakes. So this may take an overhead away from the agent's memory and they do not need to worry about finding the nearest exit. They may simply follow the path shown to them by the IVRS.

7.2.5 Put in more Agents with slower speed to consider for Handicapped People

The current simulation only considers two kinds of speeds - Normal and Panic. None of them ever goes lower than 11kmps. To make the implementation more fair to handicapped people there should be a consideration of much lower speeds. People with walking disabilities should be the first consideration when designing any evacuation strategy. It's

normal human behavior that during panic they can't think straight. Even though they want to help a fellow handicapped person they can't because of their concern for their own lives. But the handicapped can't help themselves. So the next extension to the current framework may be considered very low speeds for handicapped people and then calculating the evacuation times to get the consideration for the disabled ones.

Appendix A

Abbreviations

Short Term	Expanded Term
IVRS	Intelligent Virtual Reality Systems
VR	Virtual Reality
AI	Artificial Intelligence
FSM	Finite State Machine
VE	Virtual Environment
BDI	Belief-Desire-Intention
API	Application Programming Interface
NavMesh	Navigation Mesh
UI	User Interface
IDE	Integrated Development Environment
FPS	Frames Per Second
RPG	Rle Playing Game
RTS	Real Time Strategy
RVO	Reciprocal Velocity Obstacles
CPU	Central Processing Unit

Bibliography

[1] A. Braun et al. "Modeling individual behaviors in crowd simulation". In: *Proceedings 11th IEEE International Workshop on Program Comprehension.* May 2003, pp. 143–148. DOI: 10.1109/CASA.2003.1199317.

[2] D. Chen et al. "Parallel Simulation of Complex Evacuation Scenarios with Adaptive Agent Models". In: *IEEE Transactions on Parallel and Distributed Systems* 26.3 (Mar. 2015), pp. 847–857. ISSN: 1045-9219. DOI: 10.1109/TPDS.2014.2311805.

[3] Eyal Cohen and Laurent Najman. "From crowd simulation to airbag deployment: particle systems, a new paradigm of simulation". In: *Journal of Electronic lmaging* 6.1 (1997), pp. 94–107.

[4] G. David Garson. "Computerized Simulation in the Social Sciences". In: *Simulation & Gaming* 40.2 (2009), pp. 267–279. DOI: 10.1177/1046878108322225. eprint: http://dx.doi.org/10.1177/1046878108322225. URL: http://dx.doi.org/10.1177/1046878108322225.

[5] Stephen J. Guy et al. "Simulating Heterogeneous Crowd Behaviors Using Personality Trait Theory". In: *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA '11. Vancouver, British Columbia, Canada: ACM, 2011, pp. 43–52. ISBN: 978-1-4503-0923-3. DOI: 10.1145/2019406.2019413. URL: http://doi.acm.org/10.1145/2019406.2019413.

[6] J. C. Silveira Jacques Junior, S. R. Musse, and C. R. Jung. "Crowd Analysis Using Computer Vision Techniques". In: *IEEE Signal Processing Magazine* 27.5 (Sept. 2010), pp. 66–77. ISSN: 1053-5888. DOI: 10.1109/MSP.2010.937394.

[7] Ioannis Karamouzas, Roland Geraerts, and Mark Overmars. "Indicative Routes for Path Planning and Crowd Simulation". In: *Proceedings of the 4th International Conference on Foundations of Digital Games*. FDG '09. Orlando, Florida: ACM, 2009, pp. 113–120. ISBN: 978-1-60558-437-9. DOI: 10.1145/1536513.1536540. URL: http://doi.acm.org/10.1145/1536513.1536540.

[8] Kang Hoon Lee et al. "Group Behavior from Video: A Data-driven Approach to Crowd Simulation". In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '07. San Diego, California: Eurographics Association, 2007, pp. 109–118. ISBN: 978-1-59593-624-0. URL: http://dl.acm.org/citation.cfm?id=1272690.1272706.

[9] Linbo Luo et al. "Agent-based human behavior modeling for crowd simulation". In: *Computer Animation and Virtual Worlds* 19.3-4 (2008), pp. 271–281. ISSN: 1546-427X. DOI: 10.1002/cav.238. URL: http://dx.doi.org/10.1002/cav.238.

[10] Bernard Moulin et al. "MAGS Project: Multi-agent GeoSimulation and Crowd Simulation". In: *Spatial Information Theory. Foundations of Geographic Information Science: International Conference, COSIT 2003, Kartause Ittingen, Switzerland, September 24-28, 2003. Proceedings*. Ed. by Walter Kuhn, Michael F. Worboys, and Sabine Timpf. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 151–168. ISBN: 978-3-540-39923-0. DOI: 10.1007/978-3-540-39923-0_11. URL: https://doi.org/10.1007/978-3-540-39923-0_11.

[11] S. R. Musse and D. Thalmann. "Hierarchical model for real time simulation of virtual human crowds". In: *IEEE Transactions on Visualization and Computer Graphics* 7.2 (Apr. 2001), pp. 152–164. ISSN: 1077-2626. DOI: 10.1109/2945.928167.

[12] Rahul Narain et al. “Aggregate Dynamics for Dense Crowd Simulation”. In: *ACM SIGGRAPH Asia 2009 Papers*. SIGGRAPH Asia ’09. Yokohama, Japan: ACM, 2009, 122:1–122:8. ISBN: 978-1-60558-858-2. DOI: 10.1145/1661412.1618468. URL: http://doi.acm.org/10.1145/1661412.1618468.

[13] Jan Ondřej et al. “A Synthetic-vision Based Steering Approach for Crowd Simulation”. In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH ’10. Los Angeles, California: ACM, 2010, 123:1–123:9. ISBN: 978-1-4503-0210-4. DOI: 10.1145/1833349.1778860. URL: http://doi.acm.org/10.1145/1833349.1778860.

[14] Sébastien Paris, Julien Pettré, and Stéphane Donikian. “Pedestrian Reactive Navigation for Crowd Simulation: a Predictive Approach”. In: *Computer Graphics Forum* 26.3 (2007), pp. 665–674. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2007.01090.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2007.01090.x.

[15] N. Pelechano, J. M. Allbeck, and N. I. Badler. “Controlling Individual Agents in High-density Crowd Simulation”. In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’07. San Diego, California: Eurographics Association, 2007, pp. 99–108. ISBN: 978-1-59593-624-0. URL: http://dl.acm.org/citation.cfm?id=1272690.1272705.

[16] Nuria Pelechano et al. *Crowd simulation incorporating agent psychological models, roles and communication*. Tech. rep. PENNSYLVANIA UNIV PHILADELPHIA CENTER FOR HUMAN MODELING and SIMULATION, 2005.

[17] Ameya Shendarkar et al. “Crowd Simulation for Emergency Response Using BDI Agent Based on Virtual Reality”. In: *Proceedings of the 38th Conference on Winter Simulation*. WSC ’06. Monterey, California: Winter Simulation Conference, 2006, pp. 545–553. ISBN: 1-4244-0501-7. URL: http://dl.acm.org/citation.cfm?id=1218112.1218216.

[18] Nirajan Shiwakoti et al. “Animal dynamics based approach for modeling pedestrian crowd egress under panic conditions”. In: *Transportation Research Part B:*

Methodological 45.9 (2011). Select Papers from the 19th ISTTT, pp. 1433–1449. ISSN: 0191-2615. DOI: http://dx.doi.org/10.1016/j.trb.2011.05.016. URL: http://www.sciencedirect.com/science/article/pii/S019126151100066X.

[19] Barry G. Silverman et al. "Constructing Virtual Asymmetric Opponents from Data and Models in the Literature: Case of Crowd Rioting". In: 2002.

[20] Barry G. Silverman et al. "Human Behavior Models for Agents in Simulators and Games: Part I: Enabling Science with PMFserv". In: *Presence: Teleoperators and Virtual Environments* 15.2 (2006), pp. 139–162. DOI: 10.1162/pres.2006.15.2.139. eprint: http://dx.doi.org/10.1162/pres.2006.15.2.139. URL: http://dx.doi.org/10.1162/pres.2006.15.2.139.

[21] Mankyu Sung, Michael Gleicher, and Stephen Chenney. "Scalable behaviors for crowd simulation". In: *Computer Graphics Forum* 23.3 (2004), pp. 519–528. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2004.00783.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2004.00783.x.

[22] Branislav Ulicny and Daniel Thalmann. "Crowd simulation for interactive virtual environments and VR training systems". In: *Computer Animation and Simulation 2001: Proceedings of the Eurographics Workshop in Manchester, UK, September 2–3, 2001*. Ed. by Nadia Magnenat-Thalmann and Daniel Thalmann. Vienna: Springer Vienna, 2001, pp. 163–170. ISBN: 978-3-7091-6240-8. DOI: 10.1007/978-3-7091-6240-8_15. URL: https://doi.org/10.1007/978-3-7091-6240-8_15.

[23] Branislav Ulicny and Daniel Thalmann. "Towards Interactive Real-Time Crowd Behavior Simulation". In: *Computer Graphics Forum* 21.4 (2002), pp. 767–775. ISSN: 1467-8659. DOI: 10.1111/1467-8659.00634. URL: http://dx.doi.org/10.1111/1467-8659.00634.

[24] Xiaoping Zheng, Tingkuan Zhong, and Mengting Liu. "Modeling crowd evacuation of a building based on seven methodological approaches". In: *Building and Environment* 44.3 (2009), pp. 437–445. ISSN: 0360-1323. DOI: http://dx.doi.org/

10.1016/j.buildenv.2008.04.002. URL: http://www.sciencedirect.com/science/article/pii/S0360132308000577.

[25] Suiping Zhou et al. "Crowd Modeling and Simulation Technologies". In: *ACM Trans. Model. Comput. Simul.* 20.4 (Nov. 2010), 20:1–20:35. ISSN: 1049-3301. DOI: 10.1145/1842722.1842725. URL: http://doi.acm.org/10.1145/1842722.1842725.