UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Crowd Simulation Software:

Game/Software Application Development

Crowd Simulation Software Framework

Student Name: Stephen King

Student ID: 15167771

Course: LM051 Computer Systems

Supervisor: Conor Ryan

Abstract:

Crowd simulation is the process of simulating the movement (or dynamics) of multiple entities or characters. Crowd Simulation has attracted significant research interest due to its broad application in a variety of fields including military simulation, safety science, sociology, training systems, robotics etc.

This project will provide users such as architects, engineers, urban planners and emergency service personnel with a framework to build their own simulations. They will use pre-defined programs both to control the agents and how they interact and in addition to create environments and events, e.g. a fire breaking out in a building causing all the agents to panic and flee.

In short, scenario testing will allow architects, engineers, emergency planners and emergency service personnel to develop and improve their emergency preparedness and response procedures.

Specifically, this will enable emergency services to simulate how crowds react to different events within a specified environment. Emergency services can establish the evacuation time and any bottleneck or choke points that occur within the specified environment.

The following are covered by the report:

- A description of the project introduction, objectives and motivation.
- A description of the current progress of the development of the formal framework that will contain the crowd simulations.
- Analysis of how the artificial intelligence (AI) agents will respond to multiple events.
- Explanation of the implementation of the NavMesh system that allows AI agents to walk around freely and navigate towards a goal while simultaneously avoiding collisions.

Acknowledgements:

I would like to thank Conor Ryan, my project supervisor, for giving me ample support throughout this project and for giving me the freedom to develop the project to exacting standards.

I would also like to thank my family and friends for putting up with me over the last year and for their thoughts and input throughout the project. Especially my parents (Mary and Michael King) and friends (Daniel Lavin, Trevor Sherin, Michael Ryan, Michael Coyne, Daire McDonald, Vilius Drumsta, Ben Smith, Conor Moloney, Kennan Brennan and James Hurley) who have provided input and kept me sane during this time.

Finally, I would like to thank anyone who has tested my product either at demo day or throughout its development.

Declaration:

This final year project is presented in part fulfilment of the requirements for the degree of Bachelor of Science in Computer Systems. It is entirely my own work and has not been submitted to any other University or higher education institution, or for any other academic award in this University.

Where there has been use made of the work of other people it has been fully acknowledged and fully referenced.

All brands, product names, or trademarks are properties of their respective owners.

Signature: _____ Date: _____

Table of Contents:

1) Introduction

1.1 Motivation:

Every year natural disasters kill around 90 000 people and affect close to 160 million people worldwide. (Environmental Health in Emergencies)

Natural disasters include earthquakes, tsunamis, volcanic eruptions, landslides, hurricanes, floods, wildfires, heat waves and droughts. Even more people are killed each year when you also include man-made disasters such as explosions, shootings and fires. These disasters have an immediate impact on human lives and often result in the destruction of the physical, biological and social environment of the affected people, thereby having a longer-term impact on their health, well-being and survival. So, imagine if we could minimize the overall impact of these disasters by successfully imitating real world environments, AI agents and events. Allowing for the possibility to improve natural and man-made disaster prevention by allowing emergency services to access knowledge about crowd behaviour (navigating towards a single goal and avoiding collisions on the path), evacuation times and choke points at exits when evacuating, within a real-life environment where an event has occurred. This project will provide the framework for the user to create these crowd simulations.

The nature of these real-life environments, the human dynamic behaviours, complex agent interactions, realistic simulated disasters and AI learning algorithms makes these simulations scalable to complex situations. The issue with current crowd simulations is that it is difficult to create a realistic environment that contains a representative quantity of characters, appearance variety, animation variety, realistic motion planning and crowd behaviours, while maintaining enough performance rate and without being extremely CPU intensive.

1.2 Challenges with current Crowd Simulations:

The challenges that are faced when creating an evacuation crowd simulation with realistic characters, realistic disasters and realistic environments are shown below:

- **Variety:**

  When generating crowds, we need to take a reasonable approach that will not require multiple designers or an unreasonable amount of memory.

- **Appearance Variety:**

  As the name suggests, these individuals will have to have distinctly different in appearance.

- **Animation Variety:**

  Similarly, to the Appearance Variety, these individuals will have to have distinctly different animations, as if they all had the same animation, it would not be realistic.

- **Motion Planning:**

  To provide real-time crowd planning, we need methods that allow for path planning and obstacle avoidance. The challenge is to find methods that are both efficient and realistic and deal with known issues such as herding, arching and clogging that appear during emergency situations.

- **Crowd Behaviour:**

  It is hard to create realistic crowd behaviours, due to a various amount of challenges. Due to the primitive research and technology in this area, we can only generate very simple behaviours, causing a lack of realistic interaction between characters. Behaviours such as emotions are very primitive and mostly not present in current crowd technology, making is difficult to consider crowd behaviours such as panic in emergency situations. The main challenge of creating such a technology would be acquiring enough real-world panic crowd data for crowd model learning.

1.2.1 Challenges in Crowd Behaviour in Emergency Situations:

- **Modelling the Intelligent Behaviour of Rescue Workers or the Authority:** There is difficulty when simulating the behaviours of rescue workers or authority as we will face different technical challenges such as evacuating crowds safely and providing help to those who have been affected by the emergency.

- **Modelling Confrontational Crowd Behaviours:** Without emotions, it is very difficult to recreate realistic confrontational crowd behaviours such as riots, demonstrations, prison breaks etc.

- **Modelling Extreme Cases Such as the Squeezing and Trampling in the Crowd:** Building a model to initiate and evolve squeezing, pushing, shoving, crushing, tumbling, trampling and piling up will be of paramount significance to emergency services.

1.3 Research Objective:

This project will provide users such as architects, engineers, urban planners and emergency service personnel with a framework to build their own simulations. They will use pre-defined programs both to control the agents and how they interact and in addition to create environments and events, e.g. a fire breaking out in a building causing all the agents to panic and flee. Other objectives include:

- To open people's eyes and minds about the idea of crowd simulations.
- To research aspects of crowd simulations that has grown rapidly and to provide implementation to resolve the challenges discussed above.
- To create an evacuation crowd simulation framework that can be used by architects, engineers, emergency planners, and emergency services personnel to complete scenario testing to develop and improve their emergency preparedness and response procedures.
- To encourage users to create their own crowd simulations based on the framework created in this project, even if they have no prior knowledge of the key languages used such as Unity.
- Another key objective will be to implement the risk management of the environment by having a consequence – likelihood matrix that will tell the user if the event would be likely an event will occur given the environment structure and the exits that the user has entered. It will appear like the shown below: (Figure 1)

		Consequences →				
		A	B	C	D	E
Likelihood →	V	Medium 5	High 10	High 15	Extreme 20	Extreme 25
	IV	Medium 4	Medium 8	High 12	High 16	Extreme 20
	III	Low 3	Medium 6	Medium 9	High 12	Extreme 15
	II	Low 2	Low 4	Medium 6	High 8	Extreme 10
	I	Negligible 1	Low 2	Medium 3	High 4	Extreme 5
Consequence scale 1		Description 1	Description 2	Description 3	Description 4	Description 5

C

If an environment has a likelihood of 5 and the event is major (E), suppose the environment has one exit and a fire occurs, resulting in clogging at the exit. The event could result in multiple permanent disabilities or fatalities.

2. Research: State of the Art:

In this chapter, I will delve into the research I did before and during the project. This includes my research into software which was similar to the project I intended to make, into software which I might use to develop the project and research regarding challenges within crowd behaviour.

2.1 Common Aspects of Crowd Simulations:

Before I began this project, I looked at several crowd simulations. During my research, I found that for the emergency disaster software I planned to develop, there were a few functions that were common in all these crowd simulations. Firstly, there was a realistic disaster environment that the disaster would take place in. In these crowd simulations Navmesh agents and exit points were central features, while crowd behaviours based on events was integral.

The most important aspect of these crowd simulations was getting the AI Navmesh agents to walk around the map in a realistic manner through waypoints, for the user to be able to use an in-game UI to create disaster events on command and finally for the AI agents to react to these events and react accordingly.

I recognised the smooth transition between menus, maps, animations and the reaction of the crowds was vitally important, and if this smooth transition is not apparent the user would feel disconnected. This informed me that I needed to ensure that my animation transition and scene management must be consistent and of good standard.

2.2 Research into Development Processes and Practices:

To begin research into how to develop crowd simulations, I had to consider the software which facilitated development in that area.

Both Unity and Unreal Engine are free game engines and contain most of the features required to develop a crowd simulation. Unity game engine uses a tiered model which can be priced monthly depending on which tier you want to gain access too, for example Plus and Pro allows multiple PCs to work on the same project through cloud building. However, there is a tier that is available for beginner developers which contain most of the same features as Plus or Pro but is free and widely accessible. Being widely accessible, the makers of this game engine had to consider the low-end machines that would be running Unity to make it more accessible to beginner developers. Unreal engine's realistic graphics were impressive; however for a crowd simulation although graphics are important, the main functionality of the crowd simulation was the key aspect. Having these realistic graphics required high-end computers which meant that Unreal Engine was not as easily accessible than Unity.

Considering all the information above, I made the decision to use the Unity Engine, as it was more accessible being able to run on low-end computers, it provides many of the same features as Unreal Engine, it provides access to the Unity Asset Store which can be used to download plugins, materials and prefabs, as well as being able to export the Unity Project on multiple platforms such as Android, IOS or as a Windows application.

2.3 Challenges in Crowd Behaviour in Emergency Situations:

Variety:

The challenge of generating thousands of individuals for a crowd simulation was mentioned in Section 1.2, this was a major area of research due to the impact the number of individuals would have both on memory and GPU processing power. The best approach to create a large crowd was mentioned in the article (D.Thalmann, 2009) which suggested that we should instantiate a group of characters several times to create a large crowd. Each character will have a different texture and each character will contain Idle, Walking, Running and Dying states that will be called throughout the life of the crowd simulation.

Appearance Variety:

Colour Variety:

The challenge of colour variety was mentioned in Section 1.2, it is vitally important for the appearance of each character to be varied to allow for a realistic crowd simulation. Previous work in this area of research involved dividing a character into several body parts and assigning each body part a colour to modulate the texture. "Tecchia et al. extended this method by allowing several passes to render each impostor body part, while Dobbyn et al. extended this method further to 3D meshes avoiding multipass allowing the body parts to render with programmable graphics hardware" (D.Thalmann, 2009). Based on this research, I decided to implement the extended method by Dobbyn et al. to allow for minimal load times and a realistic appearance. We should also implement a height and shape for each character to create a realistic crowd, the user should be able to modify the height of the character by scaling the character's skeleton in Unity and should be able to modify the shape in the blender application.

Animation Variety:

The challenge of generating thousands of individuals for a crowd simulation was mentioned in Section 1.2, as discussed previously in this section under the Variety Section due to the small scale of this project, I used the Animator Controller in Unity for animations; each character contained the Idle, Walking, Running and Dying states. Once the character enters the crowd simulation the Walking state is called and the character is designated a waypoint destination, once this character reaches the waypoint the Idle state is called until a new waypoint destination is designated and the Walking state is called. A character enters either the Running or Dying state based on a natural disaster occurring and the panic level that each character was given according the in-game UI menu. Although, not mentioned as the best approach in the article (D.Thalmann, 2009), it provided minimal load times and consumed a low amount of GPU processing power at runtime. In this article a motion-capture-based locomotion engine was mentioned as the best approached to use for animation variety; and although it would provide a substantial range of different animations, this would not be feasible due to the amount of memory and GPU processing power it would consume.

The motion-capture-based locomotion engine developed by Glardon et al. This engine would be used to generate a large amount of locomotion cycles (walking and running) and idle cycles (standing, talking, sitting etc.) and would adapt these animations for each character. In his approach the Principal Component Analysis (PCA) would be used to represent motion capture data in a new, smaller space. (D.Thalmann, 2009)

There are three parameters that would allow for the modulation for these motions:

- **Personification Weights:** People of different heights and gait will be captured while walking or running, to allow the user to choose how he wishes to parameterized these different styles. (D.Thalmann, 2009)
- **Speed:** the subjects will be captured at many different speeds. Allowing the user to choose at which velocity the walk/run cycle that should be generated. (D.Thalmann, 2009)
- **Locomotion Weights:** this parameter defines whether the cycle is a walk or run animation. (D.Thalmann, 2009)

Motion Planning:

Navigation Graphs:

The challenge of motion planning was mentioned in Section 1.2 and mentioned in article (D.Thalmann, 2009), one approach that could solve the problem of real-time crowd planning, allowing for path planning and obstacle avoidance. We can use one approach that would automatically extract a topology from scene geometry and handle path planning using a navigation graph. This approach would allow for the handling of uneven and multi-layered terrains, however this approach would not be able to handle collisions between characters and other obstacles.

Scalable Motion Planning:

Another approach created recently by Treuille et al. proposed realistic motion planning for crowds, this method produces a field for each position with a waypoint to avoid all obstacles. This technique allows the characters to simulate the behaviours of pedestrians; however it creates less believable results. (D.Thalmann, 2009)

Based on my research of both these methods, I wanted to achieve the benefits of both motion planning techniques, I achieved this by using the in-built Unity Navmesh motion planning which allowed for the handling of uneven and multi-layer terrains, while dealing with obstacle avoidance and allowing of characters to travel throughout the map as pedestrians by set the destination of each character to a random waypoint.

Crowd Behaviour:

The psychology of people in a crowd is fascinating subject: the behaviour of a crowd can sway without just cause and it is a common idea that people act differently in crowd situations. In the article (D.Thalmann, 2009), Musse et al. describes a model that describes this crowd behaviour through relationships. In this model he suggests that a crowd's parameters should be defined by group goals allowing for the group to choose the destination that the crowd should reach. The group will also take other behaviours such as the emotional status of the group and the individual interests to reach the common goal, when creating crowd behaviour.

Group Behaviour:

Modelling the Intelligent Behaviour of Rescue Workers or the Authority:

The challenge of modelling the Intelligent Behaviour of Rescue Workers or Authority was mentioned in Section 1.2, with the goal of creating a realistic evacuation system, rescue workers and authority is needed to solve this problem; I will create a sub template when creating the simulations that will give a percentage chance that a citizen will become a guard. A guard or rescue worker, would create an alert area around them once they notice an event, and would be able to lead the citizens in this area to the nearest safety exits. A guard template will be able to deal with the danger, assist/inform others and gather valuables.

Modelling and Simulation of Panic Crowds:

The challenge of modelling and simulation of panic crowds was mentioned in Section 1.2, "panic is a protective psychological reaction when we face sudden and irresistible disasters." Panic patterns such as herding and squeezing are evident during natural disasters, based on this research I decided I would try to create similar behaviours using the Navmesh model and would assign a panic level to each character that would cause the character to either die or flee from the environment. (Ming Liang Xu, 2014)

Modelling Confrontational Crowd Behaviours:

The challenge of modelling and simulation of confrontational Crowd Behaviours was mentioned in Section 1.2, there are many kinds of confrontational crowd events in the real world such as riots and demonstration. I will try to recreate this methods taking into account factors such as crowd evacuation problems and violence. (Ming Liang Xu, 2014)

Modelling Extreme Cases Such as the Squeezing and Trampling in the Crowd:

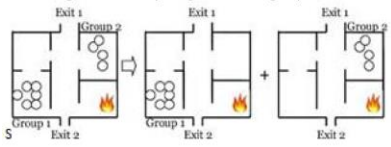
The challenge of modelling and simulation of confrontational Crowd Behaviours was mentioned in Section 1.2, as discussed under the Modelling and Simulation heading of this section, I will try to recreate the behaviours such as squeezing, pushing, crushing and trampling accidents that occur in a crowd during an emergency evacuation due to the result of accumulation of multiple characters in one area using the Navmesh model, this will be of paramount significance to public safety. (Ming Liang Xu, 2014)

Another approach found during my research in article (Noor Akma Abu Bakar, 2017) was in dealing with the three previous challenges; to introduce an emotion-based model that can deal with panic situations. Crowd evacuations can prove difficult in some circumstances due to the phenomena such as panic escape, arching and congestion.

The main goal in the crowd evacuation is to have a plan evacuation route and set the destination of the crowd towards emergency exit. The figure below depicts a crowd simulation where there is only one emergency exit for the characters to flee through. "It depicts the time taken for the characters to escape in 4, 14, 22 and 28 seconds; depicted as a, b, c, d in the figure." (Noor Akma Abu Bakar, 2017) This figure (Figure 2) also shows the clogging and choke points that can occur within a simulation if there is only a limited amount of exits and can prove vital information for the emergency services.

Figure 4 depicts different scenarios of the location of crowds, the location for the fire and the paths for each character with only one emergency exit.

Fig. 4. The theory design of fire emergency scenario



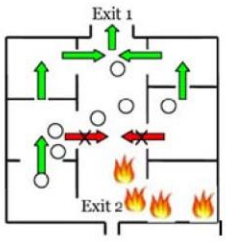


Fig. 5. Design of panic scenario with obstacle

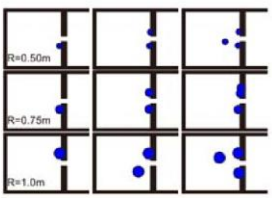
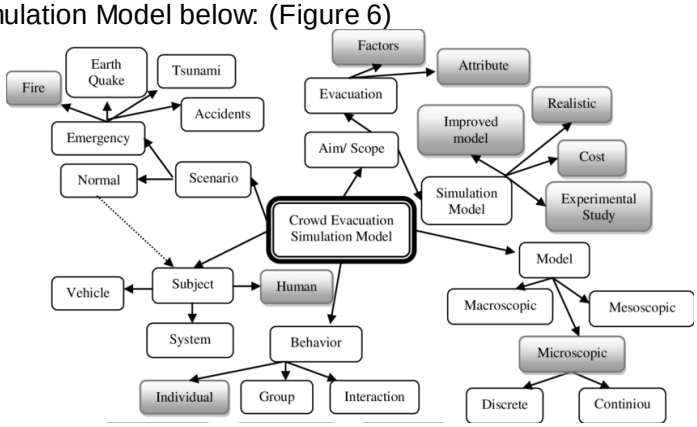


Figure 5 provides the number of the obstacle which is set from 1 to 3. R is the radius of the obstacle and represented as the blue circle (blue dots). The regular flow of a crowd evacuation movement is as in this figure and describes a typical scenario that could happen when the escape slows down, and the characters fall or are injured and in worst case could cause death. These deaths should be accounted for in the risk management matrix in this project discussed later. (Noor Akma Abu Bakar, 2017)

The illustration of the evacuees' movement is started from the black circle; firstly, the character will touch the wall/ any obstacle. Then the character starts to turn to the left-hand side or right-hand side direction to where the opening occurs. And the third one, the character will move along the wall and finally the character will leave the room and escape from the environment. (Noor Akma Abu Bakar, 2017)

This leads us to the conclusion that the individual will move faster than the crowd, and then starts pushing until they begin moving and passing the bottleneck. However, the exit is jammed by the crowd and causes trampling and squeezing which results in the escape slowing down and AI agents being injured. The AI agents exit one by one until the crowd has fully been evacuated.

Overall, we can see that this crowd evacuation simulation model can not only be used for fires but also multiple disasters such as earthquake, tsunamis and accidents using the crowd evacuation Simulation Model below: (Figure 6)



3. Design Requirements:

In this chapter, I will discuss the requirements I imposed on the project. I will also detail the reasoning that went into the design and the methodologies I included when developing the crowd simulation, as well as including a description of each script used in the project and how they interact.

3.1 Design Requirements: Time Constraint:

From my knowledge of the modules Project Management, Software Quality and Requirements Engineering, it is vitally important to consider Design Requirements before beginning a project. There are numerous factors that could influence Design Requirements but the largest that I have faced when creating this project were time constraints and quality. This project was developed in fourth year which left me with very limited time to conjure up the scope and requirements of my project, especially when I consider such factors as other college projects and associated deadlines and family events.

Therefore, with the help of my supervisor, I set out a detailed plan for my project, splitting each week of the semester into detailed features that needed to be completed before the next FYP meeting, as well as leaving time to debug and test the project.

3.2 Design Requirements: Quality Constraint:

Quality was another major factor on the project. I wanted to ensure that the level of quality of the coding and product was on par with current crowd simulations. I did not want my project to either be unfinished or have important features missing. This meant I had to ensure my code was up to required standard through testing at design.

3.3 Design Requirements: Informed Design Requirements:

From my research into crowd simulations, I have acquired several design requirements:

- **Ease of Use:** the crowd simulation must have easy to use controls, to allow an architect or engineer to use the program regardless of technical background.
- **Accuracy:** the crowd simulation must depict a realistic crowd and crowd behaviours. As well as depicting a realistic reaction to the events.
- **Look and Feel:** the crowd simulation should feel smooth in transition between scenes, animations, reactions and events and the menus in the crowd simulations should maintain a consistent theme.

3.4 Design Requirements: UI, Sprite and Map Design:

At the beginning of this project, I chose to design my own map and in-game user interface (UI). The thinking behind this decision: I wanted to gain knowledge about these areas for future projects and I wanted my project to feel unique relative to other crowd simulations. This led me to create multiple maps using the terrain editor in Unity 9f1, once I created a reasonable number of maps, Lake, Forest, Street, Beach, School, Park and Island scenes, I decided to narrow down my crowd simulations to three maps. This was to ensure that I focused on the functionality of the crowd simulations and integrating this functionality on each map perfectly, instead of extending my scope and ending up with unfinished integration on multiple maps.

The next step in my design was the player's avatar and the characteristics of each character. During my design phase, I realised that I wanted the characters to be realistic depiction of an environment, instead of a low poly design. I followed the following Udemy courses to apply character modelling (https://www.udemy.com/blender-character-modeling-for-beginners-hd/) and rigging (https://www.udemy.com/blender-character-rigging-for-beginners-hd/) for the character in Blender. With animations such as Walking, Running and Dying. Clothing was provided by the clothing collection pack on Unity Assets Store (https://assetstore.unity.com/packages/3d/props/clothing/clothing-collection-47133).

Similarly, I wanted to create realistic environments rather than a cartoon or low poly design, to make the scenes more realistic, I created particle effects for the Waterfall effect on the Forest scene, as well as creating Water effects for both the Forest Scene and Island scene.

For the UI, I decided to create easily viewable text and buttons for each menu.

For the in-game UI, I created a canvas, panel and buttons like the canvas on the MainMenu scene. The user can use a slider in Unity to change the values of the Movements Speed, Detection Radius, Panic Level and Flee Radius. I also attached this panel to a toggle in Unity, to toggle whether the panel is active or not. To allow the scripts to access whether a button has been clicked or to make changes according to the slider, I used the event system in Unity which allows game objects within the game to be clicked.

3.5 Design Requirements: Scene Descriptions:

There are 10 scenes in the project. Each scene equates to either a menu or an environment for the crowd simulation to take place in. I will detail what scene is and in what order the scenes are called in.

1. "Start" is the scene shown the first time the user opens the project. The scene only serves the purpose of redirecting the user to the "MainMenu" when the start button is clicked.

2. "MainMenu" is the scene where the user can either select the environment that they want the crowd simulation to take place in or quit the application. The "MainMenu" contains multiple buttons which redirect the user to the corresponding scene.

3. "Example1" is an example environment scene which it contains an open corridor for which the user can use to escape to the exit. The user can use the menu to initiate the crowd, change behaviours and characteristics of the crowd, cause events, reload the menu and return to the main menu.

4. "Example2" is an example environment scene which it contains a narrow corridor for which the user can use to escape to the exit. The user can use the menu to initiate the crowd, change behaviours and characteristics of the crowd, cause events, reload the menu and return to the main menu.

5. "Example3" is an example environment scene which it contains two open corridors for which the user can use to escape to the exit. The user can use the menu to initiate the crowd, change behaviours and characteristics of the crowd, cause events, reload the menu and return to the main menu.

6. "Example4" is an example environment scene which it contains two narrow corridors for which the user can use to escape to the exit. The user can use the menu to initiate the crowd, change behaviours and characteristics of the crowd, cause events, reload the menu and return to the main menu.

7. "Hotel" is an environment scene which replicates a hotel floor, it contains narrow corridors and multiple bedrooms, some of which contain one exit and others which two exits to contrast the choke points in both. The user can use the menu to initiate the crowd, change behaviours and characteristics of the crowd, cause events, reload the menu and return to the main menu.

8. “Forest” is an environment scene which as the name suggests replicates a forest scene with a pathway from the AI agents to walk through and look upon the waterfall, it contains a barrier on one side to enter through one side of the pathway and on open pathway on the other side, in order to contrast the difference it takes to escape the environment. The user can use the menu to initiate the crowd, change behaviours and characteristics of the crowd, cause events, reload the menu and return to the main menu.

9. “Island” is an environment scene which as the name suggests replicates an island scene with a villa that contains four rooms and a lobby. The user can use the menu to initiate the crowd, change behaviours and characteristics of the crowd, cause events, reload the menu and return to the main menu.

10. “Street” is an environment scene which as the name suggests replicates a Street scene, it tries to replicate a scenario where a bus is broken down on the street causing a traffic jam, the AI agents can either chose to you a pedestrian crossing or cross through a make shift narrow area created by the buses breaking down. The road on the map was created using the EasyRoad3D plugin in Unity (https://assetstore.unity.com/packages/3d/characters/easyroads3d-free-v3-987).

4. Development:

In this chapter, I will discuss the software and plugins I used to develop this project, with a detailed breakdown of the major scripts used.

4.1 Development: Software:

In section 2.2, I discussed my reasoning for choosing Unity over Unreal Engine. Once I had made this decision to use Unity, I began to develop sample projects under the Unity Learn section of the Unity program and on Udemy courses such as (https://www.udemy.com/unitycourse2/) to learn how to make games in 3D. Also, I watched numerous Unity tutorial videos online about how to use Unity such as videos by Youtuber Brackeys (https://www.youtube.com/watch?v=IlKaB1etrik).

For Characters and Animations, I used software called Blender. This was since I bought the Udemy courses that were linked above for modelling and rigging.

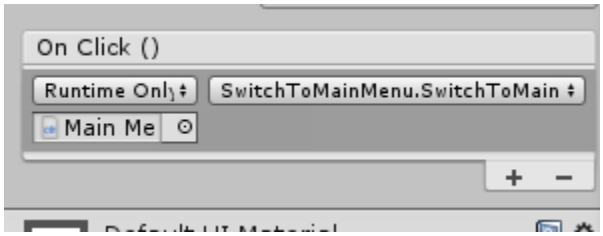
4.2 Development: Scripting:

4.2.1 Moving Between Screens:

The user needs to navigate easily between scenes with low load times to make a smooth transition, I used C# to navigate between this scene, this created a smooth transition. Each button that redirected to a scene, had a C# script attached to redirect to the desired scene. For example, to redirect to the Main Menu scene a user clicked the Main Menu button and was redirected to the scene via the C# script. Each C# script was similar to the shown below.

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

public class SwitchToMainMenu : MonoBehaviour
{
    public void SwitchToMainMenuMethod()
    {
        SceneManager.LoadScene("Main Menu");
    }
}
```



4.2.2 Character Movement: AIControl script:

Each character has an AI control script attached to it, once it enters into the script, it calls the isItFirstTime () method which sets the destination of a character to a random waypoint on the AreaController of the map, once the character reaches the waypoint, it will call the isItFirstTime () method and if it is false, the character will assign a new waypoint as their destination.

This script contains the set and gets values for Movement Speed, Detection Radius, Flee Radius and Panic Level, the set methods are called once a value is changed on the in-game UI. The get values are called throughout the program to gain knowledge to what value the characteristics of each characters is, as well as to update the text value of the Movement Speed.

The ResetAgent () method is called once the user clicks the reset button on the in-game UI, this resets all agents Movement Speed, Detection Radius, Flee Radius and Panic Level to default values.

The PanicButton () method is called once the user clicks the panic button on the in-game UI, the exitModeEntered Boolean is set to true and the characters find a random exit from the list of exits.

The DetectNewObstacle () method is called once an object is placed using the PlaceObject script. If the distance is within the detection radius and if the panic level is above 40, the character will flee to the nearest exit in the opposite direction of the event. Otherwise if the panic level is equal to or below 40 the character will enter the dying state and will be deleted from the crowd simulation.

```
public void DetectNewObstacle(Vector3 position)
{
    if (Vector3.Distance(position, this.transform.position) < detectionRadius)
    {
        if (getPanicLevel() > 40)
        {
            Vector3 fleeDirection = (this.transform.position - position).normalized;
            Vector3 newgoal = this.transform.position + fleeDirection * fleeRadius;

            NavMeshPath path = new NavMeshPath();
            agent.CalculatePath(newgoal, path);

            if (path.status != NavMeshPathStatus.PathInvalid)
            {
                exitModeEntered = true;
                isExitModeEntered();
                anim.SetTrigger("isRunning");
                agent.speed = 10;
                agent.angularSpeed = 500;
            }

            if (path.status == NavMeshPathStatus.PathInvalid)
            {
                agent.SetDestination(waypoints[Random.Range(0, exits.Count)].transform.position);
            }

        }

        if (getPanicLevel() <= 40)
        {
            agent.isStopped = true;
            agent.speed = 0;
            anim.SetTrigger("isDying");
            dead = true;
        }
    }
}
```

```
public void setMoveSpeed(float number)
{
    movementValue = number;
    int value;
    value = (int)number;
    switch (value)
    {
        case 0: agent.speed = 0f; break;
        case 1:  agent.speed = 1f; break;
        case 2:  agent.speed = 2f; break;
        case 3:  agent.speed = 3f; break;
        case 4:  agent.speed = 4f; break;
        case 5:  agent.speed = 5f; break;
        case 6:  agent.speed = 6f; break;
        case 7:  agent.speed = 7f; break;
        case 8:  agent.speed = 8f; break;
    }
}

public float getMoveSpeed()
{
    return movementValue;
}

public void setDetectionRadius(float value)
{
    detectionRadius = value;
}

public float getDetectionRadius()
{
    return detectionRadius;
}

public void setFleeRadius(float value)
{
    fleeRadius = value;
}

public float getFleeRadius()
{
    return fleeRadius;
}
```

```
public void setPanicLevel(float value)
{
    panicLevel = value;
}

public float getPanicLevel()
{
    return panicLevel;
}

void ResetAgent()
{
    speedMult = getMoveSpeed();
    agent.speed = speedMult;
    agent.angularSpeed = 120;
    anim.SetFloat("speedMult", speedMult);
    anim.SetTrigger("isWalking");
    agent.ResetPath();
}

public void isDead()
{
    anim.SetTrigger("isDying");
}
```

```
public void panicButton()
{
    exitModeEntered = true;
    isExitModeEntered();
}

public void isItFirstTime()
{
    if (firstTime)
    {
        agent.SetDestination(waypoints[Random.Range(0, waypoints.Length)].transform.position);
        firstTime = true;
    }

    if(!firstTime)
    {
        while (found)
        {
            int value;
            value = Random.Range(0, waypoints.Length);
            agent.SetDestination(waypoints[value].transform.position);
            found = true;
            ResetAgent();
        }
    }
}

public void isExitModeEntered()
{
    if (exitModeEntered)
    {
            agent.SetDestination(exits[Random.Range(0, exits.Count)].transform.position);

    }
}
```

4.2.3 Environment: InitiateCharacters script:

Once the user enters the environment and clicks to add 2 groups button, the InitiateCharacters script is entered. Once the script is entered, a list of characters is activated and the whichScene () method is called which determines where the location of the exits and characters is instantiated.

Once the 2 groups' button is clicked, the instantiateCharacters method is called and 2 groups are instantiated and each character in the group is entered the characters list.

```
public void whichScene()
{
    Scene currentScene = SceneManager.GetActiveScene();
    string sceneName = currentScene.name;

    if (sceneName == "Example 1")
    {
        x = 190;
        y = 6;
        z = 120;
        x1 = 210;
        y1 = 1;
        z1 = 128;
        addFirstExit();
    }
    else if (sceneName == "Example 2")
    {
        x = 190;
        y = 6;
        z = 120;
        x1 = 210;
        y1 = 1;
        z1 = 128;
        addFirstExit();
    }
```

```
public void instantiateCharacters(int numberOfCharacters)
{
    for (int i = 0; i <= 1; i++)
    {
        GameObject persons = Instantiate(people, new Vector3(x, y, z), Quaternion.identity);

        for (int j = 0; j < persons.transform.childCount; j++)
        {
            characters.Add(persons.transform.GetChild(j).gameObject);
            AIControl aic;
            aic = characters[characters.Count - 1].GetComponent<AIControl>();
            aic.ac = ac;
            aic.waypoints = waypoints;
            aic.exits = exits;
            isInitiated = true;
        }
    }
}
```

The addFirstExit () and addSecondExit () adds one or two exits to the co-ordinates (given in the WhichScene () method) given which environment the scene is set it.

```
public void addFirstExit()
{
    firstExit = Instantiate(exit, new Vector3(x1, y1, z1), Quaternion.identity);
    EmergencyTriggers fe = firstExit.GetComponentInChildren<EmergencyTriggers>();
    fe.ic = this;
    exits.Add(firstExit);
}

public void addSecondExit()
{
    secondExit = Instantiate(exit1, new Vector3(x2, y2, z2), Quaternion.identity);
    EmergencyTriggers se = secondExit.GetComponentInChildren<EmergencyTriggers>();
    se.ic = this;
    exits.Add(secondExit);
}
```

The setNumberOfDeadAgents () method goes through the list of characters and removes the characters if they are dead by calling the removeCharacter method with that character's game object, the getNumberOfDeadAgents () returns the value of the number of characters who are dead which will be used in the matrix discussed later on.

```
public void setNumberOfDeadAgents()
{
    for(int i = 0; i<characters.Count; i++)
    {
        AIControl aic;
        aic = characters[i].GetComponent<AIControl>();
        if(aic.dead == true)
        {
            numberOfDeadAgents++;
            removeCharacter(characters[i].gameObject);
        }
    }
}

public float getNumberOfDeadAgents()
{
    return numberOfDeadAgents;
}

public void removeCharacter(GameObject objectToDelete)
{
        characters.Remove(objectToDelete);
        Destroy(objectToDelete);
}
```

The setExit () method is called when a new Exit is placed when the PlaceExit script is activated and an exit is placed on the environment and that exit is then added to the exits list.

The getNumberOfExits () returns the number of exits within the environment, which will be used in the matrix which will be discussed later on.

```
public void setExit(GameObject exit)
{
    numberOfExits += 1;

    if (numberOfExits == 2)
    {
        exits.Add(exit);
        for (int i = 0; i < characters.Count; i++)
        {
            AIControl aic;
            aic = characters[i].GetComponent<AIControl>();
            aic.exits = exits;
        }
    }

    if (numberOfExits == 3)
    {
        exits.Add(exit);
        for (int i = 0; i < characters.Count; i++)
        {
            AIControl aic;
            aic = characters[i].GetComponent<AIControl>();
            aic.exits = exits;
        }
    }

    if (numberOfExits == 4)
    {
        exits.Add(exit);
        for (int i = 0; i < characters.Count; i++)
        {
            AIControl aic;
            aic = characters[i].GetComponent<AIControl>();
            aic.exits = exits;
        }
    }
```

```
public float getNumberOfExits()
{
    return numberOfExits;
}
```

4.2.4 Menu: PlaceExplosion script:

The in-game menu contains a Lightning, Fire, Tornado, Cube, Exit, and Explosion buttons which once clicked toggle on and off a corresponding manager (e.g. ExplosionManagerToEnable) which toggles on and off a corresponding script (e.g. ExplosionScript) as shown below. This script places an explosion at the location the mouse has been clicked on and calls the DetectNewObstacle () method for each character with that location.

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

public class PlaceExplosion : MonoBehaviour
{
    public GameObject obstacle;
    public InitiateCharacters ic;
    AIControl aic;
    // Use this for initialization
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            RaycastHit hitInfo;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray.origin, ray.direction, out hitInfo))
            {
                Instantiate(obstacle, hitInfo.point, obstacle.transform.rotation);
                foreach (GameObject a in ic.characters)
                {
                    aic = a.GetComponent<AIControl>();
                    aic.DetectNewObstacle(hitInfo.point);
                }
            }
        }
    }
}
```

4.2.5 Environment: EmergencyTriggers script:

Once a character enters the trigger surrounding the exit, the OnTriggerEnter () method will be called, which searches for characters within the trigger and removes them from the crowd simulation.

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

public class EmergencyTriggers : MonoBehaviour
{
    public InitiateCharacters ic;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "agent")
        {
            ic.removeCharacter(other.gameObject);
        }
    }
}
```

4.2.6 Environment: Timer script:

The timer script is entered once the InitiateCharacters script is activated and will begin counting the minutes and seconds until the character count is 0 and the crowd simulation has ended.

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

public class Timer : MonoBehaviour
{
    public Text mytext;
    public InitiateCharacters ic;
    private float startTime;
    // Start is called before the first frame update
    void Start()
    {
        startTime = Time.time;
    }

    // Update is called once per frame
    void Update()
    {
        if(ic.isInitiated == true && ic.characters.Count != 0)
        {
            float t = Time.time - startTime;

            string minutes = ((int)t / 60).ToString();
            string seconds = (t % 60).ToString("f2");
            mytext.text = minutes + ":" + seconds;
        }

        if (ic.isInitiated == true && ic.characters.Count == 0)
        {
            mytext.color = Color.yellow;
            return;
        }
    }
}
```

4.2.7 Environment: EndGame script:

The end game count is called once the initiateCharacters script is entered, the script calls the getNumberOfExits () and getNumberOfCharacters () methods and given the number of people that have died and the number of exits in the environment, it assigns the lethality and likelihood values a number, multiples these numbers and assigns this value to the total score and shows these values on text on screen on the canvas.

```
void Update()
{
    likelinessToLeave = ic.getNumberOfExits();
    deadPeople = ic.getNumberOfDeadAgents();

    if (deadPeople >= 0 && deadPeople <= 5)
    {
        lethality = 1;
    }

    if (deadPeople > 5 && deadPeople <= 10)
    {
        lethality = 2;
    }

    if (deadPeople > 10 && deadPeople <= 15)
    {
        lethality = 3;
    }

    if (deadPeople > 15 && deadPeople <= 20)
    {
        lethality = 4;
    }

    if (deadPeople > 20 && deadPeople <= 25)
    {
        lethality = 5;
    }

    if(deadPeople >=25)
    {
        lethality = 6;
    }

    if (likelinessToLeave == 1)
    {
        likehood = 1;
    }

    if (likelinessToLeave == 2)
    {
        likehood = 2;
    }

        if (likelinessToLeave == 3)
        {
            likehood = 3;
        }

        if (likelinessToLeave == 4)
        {
            likehood = 4;
        }

        if (likelinessToLeave == 5)
        {
            likehood = 5;
        }

        if (likelinessToLeave == 6)
        {
            likehood = 6;
        }

        if (ic.characters.Count == 0 && ic.isInitiated == true)
        {
            totalScore = lethality * likehood;
            myText.text = "Lethality score: " + lethality.ToString()  + "\r\n" + "Likeliness score: " + likehood.ToString() + "\r\n" + "Total score: " + totalScore.ToString();
        }
    }
}
```

4.2.8 Environment: Events:

Fire:

The Fire prefabs and fire propagation scripts were found on the asset store in Unity (https://assetstore.unity.com/packages/tools/fire-propagation-92187).

This plugin contains the WildFireNoLight, Torch, FireManager and WildFireMedium and FireNode prefabs used in the project, as well as the FireNode, FireManager and FireNodeChain scripts.

Tornado:

Each child of the Tornado prefab has a TornadoVertex script attached to it. Once an object (which has the PullObject script) enters the TornadoVertex trigger, it calls the OnTriggerStay () method and finds objects under the tag Pullable in the trigger and moves that object towards the centre of the tornado.

```
using UnityEngine;
using System.Collections;

public class TornadoVortex : MonoBehaviour {

    private GameObject PullOBJ;
    public float PullSpeed;

    public void OnTriggerStay(Collider coll)
    {
        if (coll.gameObject.tag == "Pullable") {
            PullOBJ = coll.gameObject;

            PullOBJ.transform.position = Vector3.MoveTowards (PullOBJ.transform.position, this.transform.position, PullSpeed * Time.deltaTime);
        }
    }

}
```

Once the object enters the centre of the Tornado, the death enumerator is entered, and the object is deleted.

```
embly-CSharp                    PullObject
using UnityEngine;
using System.Collections;

public class PullObject : MonoBehaviour {

    public void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Tornado") {
            StartCoroutine ("Death");
        }
    }

    public IEnumerator Death()
    {
        yield return new WaitForSeconds (3);
        Destroy (this.gameObject);
    }
}
```

Lightning:

Once a character enters the trigger of the lightning prefab, the OnTriggerEnter () method is called and the runAway () method in initiateCharacters is called. Which does the same as the DetectNewObstacle () method in AIControl.

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

public class RunAway : MonoBehaviour
{
    public InitiateCharacters ic;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "agent")
        {
            ic.runAway(other.gameObject);
        }
    }
}

```

Grenade:

Once the grenade is placed in the environment a countdown is initialized. Once the countdown exceeds the delay, the Explode () method is called, the grenade explodes at that location and each object with the Destructible script is destroyed.

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

public class Grenade : MonoBehaviour
{
    public float delay = 3f;
    public float radius = 5f;
    public float force = 700;

    public GameObject explosionEffect;

    float countdown;
    bool hasExploded = false;

    // Start is called before the first frame update
    void Start()
    {
        countdown = delay;
    }

    // Update is called once per frame
    void Update()
    {
        countdown -= Time.deltaTime;

        if(countdown <= 0f && !hasExploded)
        {
            Explode();
            hasExploded = true;
        }
    }
```

```
    void Explode()
    {
        GameObject explosionObject = Instantiate(explosionEffect, transform.position, transform.rotation) as GameObject;
        Destroy(explosionObject, 1.9f);

        Collider[] colliders = Physics.OverlapSphere(transform.position, radius);

        foreach (Collider nearbyObject in colliders)
        {
           Destructible dest = nearbyObject.GetComponent<Destructible>();
            if(dest != null)
            {
                dest.Destroy();
            }
        }

        Collider[] collidersToMove = Physics.OverlapSphere(transform.position, radius);

        foreach(Collider nearbyObject in collidersToMove)
        {
            Rigidbody rb = nearbyObject.GetComponent<Rigidbody>();
            if (rb != null)
            {
                rb.AddExplosionForce(force, transform.position, radius);
            }
        }

        Destroy(gameObject);
    }
}
```

The Destructible script on the object, firstly replaces the object with a destroyed version of the object and then deletes that game object.

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

public class Destructible : MonoBehaviour
{
    public GameObject destroyedVersion;

    public void Destroy()
    {
        Instantiate(destroyedVersion, transform.position, transform.rotation);

        Destroy(gameObject);
    }

}
```

5. User Review and Evaluation:

In this chapter, I will discuss the user reviews and evaluation for this project.

5.1 Reviews from users:

Most of the user reviews were throughout the process and especially at demo day.

The first user's review was "There are errors causing the AI agents to walk into walls and other obstacles and getting stuck." (Dean, Demo Day 2019). Given this feedback, I decided to do some testing and found there was an error for the AI agents NavMesh path causing the AI agents to select paths outside the given environment. To fix this, I gave all the walls and objects a non-walkable Navmesh and put in a check that if the path is invalid or blocked by a non-walkable object and if so to select another waypoint within the environment.

The second user's review was "There is a bug that if the user clicks on to instantiate a tornado or lightning on the menu, it causes the lightning or tornado to fill up the screen." (Jack, Demo Day 2019). Given this feedback, I decided to research this problem and although there were similar errors reported by other users in their projects, there was no information on how to fix this problem. My solution to this problem was to create a check to see the position of the location that is clicked and if the y coordinate of the location clicked is the same y coordinate of the menu, then the object will not instantiate, otherwise the object will instantiate on the environment.

The third user's review was "Would other AI agents not run away from the event if they see other AI agents running" (Siobhan, Demo Day 2019). While this was not what I intended the AI agents in the project to do, it would be interesting to take this into account into the future and make AI agents panic if other AI agents are panicking to mimic a real evacuation crowd behaviour. This would require the construction of an Emotional Contagion Model.

6. Conclusions:

In this chapter, I will discuss the topics covered by both the report and project, my thoughts on the projects, its issues and success and the problems I faced and overcame. As well as my plans for the future of the project.

6.1 Conclusions:

During my research phase of the project, I considered existing crowd simulations and technology and I delved into the psychology of crowd behaviours.

There are multiple projects which cover the spectrum of crowd simulations, for example most AI agents in games such as "Assassin's Creed Brotherhood" (2007-2014), "WatchDogs" (2014), Grand Theft Auto V (2013) and most Hollywood disaster movies such as "Pompeii" (2014) or "The Day After Tomorrow" (2004). With all these similar range of disaster simulations, it is very hard to make a project stand out without an unusual niche or executing the disaster simulation perfectly.

Most projects in this area nowadays focus on both an unusual niche and executing the simulation perfectly. This area of disaster simulations had become increasingly popular in games and movies as suggested above, however there is plenty of new research to be capitalised on reference models such as the Emotion Contagion Model, Fuzzy Cognitive Model etc. The niche that I have identified, and I have addressed is creating a simulation framework specifically designed to cater to architects, civil and mechanical engineers, construction companies and emergency services to find choke points in their buildings or blueprints. I believe this will make my project stand out from the rest.

The main aim of my project is to generate interest in crowd simulations and more importantly, to allow users to create their own crowd simulations, as well as allowing builders and architects to use the framework and in-game risk management matrix to evaluate their crowd simulation and environment.

Using the risk management matrix, the user is empowered to mathematically compare the risk rating for their chosen scenario and most importantly to redesign their environment (e.g. building design) to reduce the risk rating to low. The ultimate safety goal of any designer is 'Safety First' and 'Zero Harm' and this crowd simulation software framework is a key tool to enable him/her to achieve his/her goal. (Figure 1)

		Consequences →				
		A	B	C	D	E
Likelihood →	V	Medium 5	High 10	High 15	Extreme 20	Extreme 25
	IV	Medium 4	Medium 8	High 12	High 16	Extreme 20
	III	Low 3	Medium 6	Medium 9	High 12	Extreme 15
	II	Low 2	Low 4	Medium 6	High 8	Extreme 10
	I	Negligible 1	Low 2	Medium 3	High 4	Extreme 5
Consequence scale 1		Description 1	Description 2	Description 3	Description 4	Description 5

6.2 Reflections on the Project:

Now that I have finished the project and had a chance to reflect. I am satisfied with my progress so far, but I am not totally happy with the final product and with some of the things that went wrong during production.

The first problem that I faced was my indecision when selecting a development environment to develop the framework in. I spent the first few weeks of the first semester debating whether to use either Unity, Unreal Engine or any other software that has an in-game framework for Crowd Simulations. I then researched and used a selection matrix to allow me to come to a clear decision. I decided to select Unity engine, because I wanted to allow the user to create their own environment without the boundaries of some other crowd simulation software.

Another setback or problem I faced was my focus on getting the characters and maps perfect during a large majority of the first semester instead of focusing on the main functionality of the project.

Finally, the last problem I faced was my novice experience of Unity. Although I did create sample Unity projects and followed the Unity basics course during the first semester, I still did not feel fully competent in Unity and as such spent the majority of my type researching how to use NavMeshAgents and how to create an in-game UI for example.

Overall, I am pleased with the project however I have identified the following opportunities for improvement for the future.

6.3 Plans for the Project in the Future:

I plan to continue working on the project into the future, as I feel the current edition of the project, although satisfactory, is in no way perfect and needs updating. I feel if I develop this framework and follow environment safety guidelines, it could prove useful for emergency services and developers and could save lives (safety first).

Other features I would like to add would be:

- **Emotion Contagion Model:** to allow agents to be affected by the crowd emotion or behaviour.
- **Fuzzy Cognitive Map:** to allow for events such as riots, shootings and demonstrations for terrorists or other to chase after Emergency Services Personal or other citizen characters.
- Consider deaths caused by characters trampling and crushing each other
- New events such as Whirlpools, Tsunamis, Flooding etc.
- New detailed environments and scenarios such as School evacuation, Stadium evacuation.
- Different templates for characters such as Emergency Services Personal, Rescue Workers, Terrorists etc.
- To add music and sound effects to the crowd simulation based on the environment selected

In addition to these changes, I would like to release this project as an open source project to allow other programmers or developers to change the framework based on different requirements or niches.

7. References:

Bibliography

D.Thalmann, H. J. (2009). Challenges in Crowd Simulation. *2009 International Conference on CyberWorlds*.

Daniel Thalmann, H. G. (January 2009). Challenges in Crowd Simulation.

Environmental Health in Emergencies. (n.d.). Retrieved 22 November, 2018, from World Health Organization: https://www.who.int/environmental_health_emergencies/naturalevents/en

Ming Liang Xu, H. J. (2014). Crowd Simulation and Its Applications: Recent Advances. *School of Information Engineering, Zhengzhou University*.

Noor Akma Abu Bakar, K. A. (2017). A simulation model for Crowd Evacuation of Fire Emergency Scenario. *Faculty of Computer System and Software Engineering, University Malaysia Pahang*.

(Figure 1 URL: https://i.pinimg.com/originals/5d/2c/48/5d2c48991ebaa6ead54931e3b279ebc8.jpg)

(Figure 2 URL: https://journals.plos.org/plosone/article/figure?id=10.1371/journal.pone.0115463.g001)

(Figure 4 URL: https://ai2-s2-public.s3.amazonaws.com/figures/2017-08-08/465942c394221aafaa1623994b161ba1b3d760da/4-Figure4-1.png)

(Figure 5 URL: https://ai2-s2-public.s3.amazonaws.com/figures/2017-08-08/465942c394221aafaa1623994b161ba1b3d760da/4-Figure4-1.png)

(Figure 6 URL: https://www.researchgate.net/profile/Noor_Akma_Abu_Bakar2/publication/318340411/figure/fig1/AS:516027517595648@1500041955813/A-perspective-of-crowd-evacuation-simulation-model.png)

(All code snippets and unity snippets are from Unity or Virtual Studio)