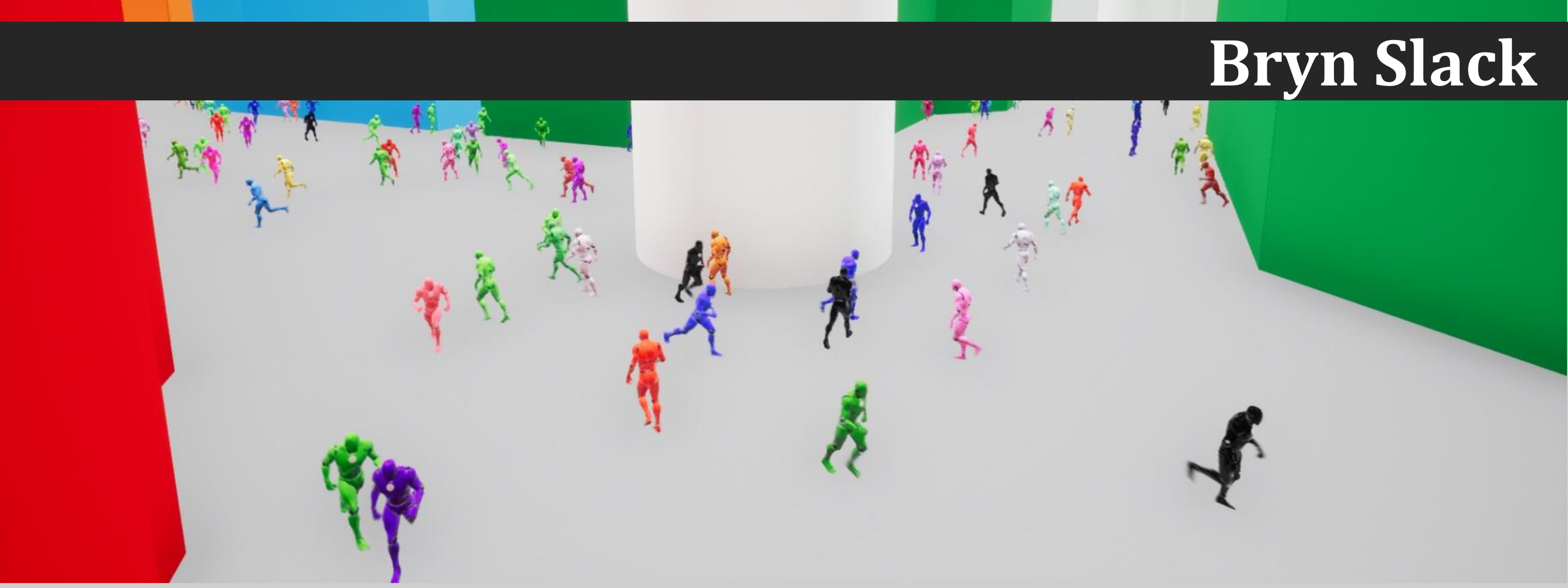
Bryn Slack



Creating a 'Threat Focused' Crowd Simulation

Can we realistically simulate real world crowds? This underlying question for my crowd simulation project comes from the initial reason for the project, which is the creation of a crowd analysis system. This crowd analysis system is intended to be used as a real-time threat detection system by analysing live CCTV footage, detecting threats before they become a more significant problem and, ideally, preventing them from even happening.

Creating a Crowd Simulation

The need for making a simulation is down to the difficulty in both obtaining and finding the ideal test scenarios needed to create and test the analysis system. With a simulation, the ideal test scenario can be hand crafted and repeated over and over with as many variances as needed to ensure it is working correctly. However, with any realistic simulation comes complexity and creating complexity takes time, more time than I personally had to work on this project.

My time on this project is intended as the starting stages and will serve almost as a proof of concept for the various components needed for this crowd simulation. Therefore, the goal of this project was to determine what aspects were required and to implement the fundamental components of them, allowing for each of them to be expanded and improved in the future.

The more obvious areas are pathfinding and responses to threats but, even with those, I wanted to make space to allow for every aspect of the project to become more complex if needed. This led to the inclusion of a personality model, giving each person in the crowd unique parameters that can affect their decision making. Individual actions are an important aspect of crowds in general but they become even more important in a danger scenario. Peoples individual personalities can cause entirely different decision making given an identical scenario. When stress is then factored into that decision making, such as in a danger scenario, it can become very illogical. Stress and individual personality values are two important aspects for the simulation that can be used to determine decision making.

The personality model component can be used in almost all areas of the project in some way. For example, someone who is extremely shy may take a path that avoids large crowds, even potentially waiting for a lack of people before proceeding forward. This is a simplified description of just one very specific scenario unique to a specific personality type. There is a huge amount of space for these kinds of personality based extensions to the project with the only factor standing in the way of these improvements being time. It will take a significant amount of time to create behaviour that accounts for many different personalities but, given time, there is no limit to the amount of improvements that can be added using the personality model.

The project was created using Unreal Engine 4 (UE4) and used various existing features of the engine. The project was created in UE4 to make use of these features and to be able to quickly and easily test them.

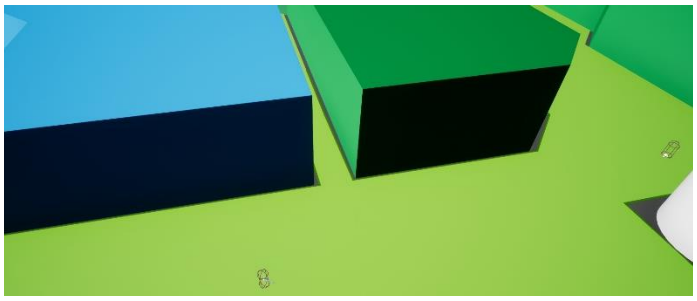
Pathfinding

The pathfinding was the fundamental starting point for the project and was split into two areas; global pathfinding and local pathfinding. The overall location that a person is walking towards is what I defined as the global pathfinding.

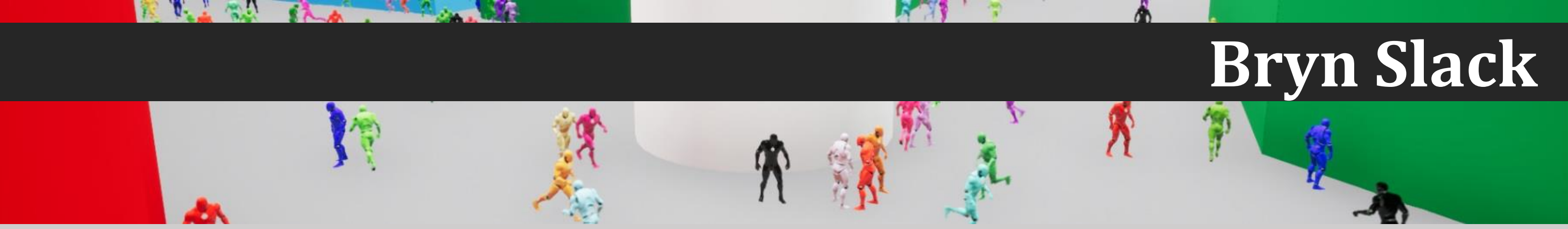
The global pathfinding is the system that defines the route to a desired location based on the static objects in the scene, such as buildings. The local pathfinding is the avoidance system which will be looking at all the dynamic objects in the scene that either can move or are moving, such as other people. The avoidance system will allow the crowd to move between and around each other as they are moving towards their destination.

Global Pathfinding

The global pathfinding is achieved using a Navigation Mesh (NavMesh) and simply defines an area in which an agent can navigate. This area is then cut into by any static object overlapping the defined zone. This leaves just the areas around the objects that make up the NavMesh. This is the core of the navigation system and has a few requirements to work correctly.



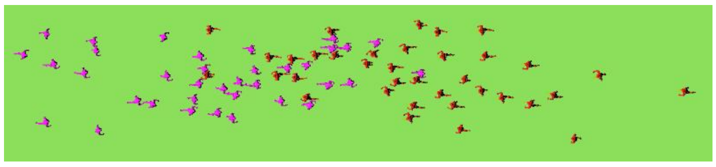
Once the NavMesh has been set up, we need a character and an AIController. The character is what will navigate the NavMesh and the AIController will, put simply, control that navigation. I defined my own AINavCharacter and AINavController based on the existing UE4

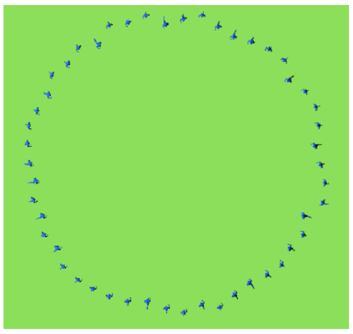


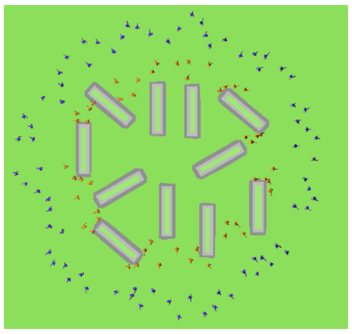
default components for each to add my own functionality. For now, when the character spawns it simply creates a controller and sets a target location using that controller. The main job of the controller is to take information from the character and send it to the Blackboard to be used in the Behaviour Tree. The Behaviour Tree is essentially a flow of logic that decides the current behaviour of the character.
The Blackboard is where values are stored that can then be used in the behaviour tree. At this point, the Behaviour Tree and the Blackboard are not priority and have very simple implementation. The Blackboard holds a value for the Target Location and the Behaviour Tree simply moves to a Target Location once a location has been set. Now the AINavCharacter can be placed into the world, somewhere on the NavMesh, and it will move to a target along the NavMesh.

Spawning Agents

Now that we have a character that can move to a location we need to have multiple characters to create the crowd. I created some spawners for this, mainly for testing, with the intention of producing a simpler spawner further down the line. The two spawners I created here were a circle spawner and a grid spawner. The circle spawner spawned a certain number of agents in a circle of defined radius and set the target to the opposite position in the edge of the circle. The grid spawner spawned the agents in a grid using a defined width and length. Both spawners were also created so that a predefined target could be used for all the agents being spawned. I set up the spawners in a variety of ways so that I could test both the Global Pathfinding and the Local Avoidance (Which, at this point, is not implemented)







After setting up these various spawners it becomes clear that the next step is to set up the local avoidance as almost all the agents are colliding with each other and getting stuck. The agents can navigate through the obstacle layout since that is NavMesh navigation but as soon as other as they cross path with another agent, which happen a lot with a crossing circle, they will get stuck.

Local Pathfinding

By default, the pathfinding in UE4 has no local avoidance but there are a couple of options implemented for the avoidance. The first and simplest option is RVO (Reciprocal Velocity Obstacle) avoidance. RVO avoidance is force based and will stop the agents from colliding by pushing them away from each other. This is not ideal as these forces can potentially push the agents off the NavMesh. Even more important is that it looks very unrealistic as the agents will slide sideways while walking forward due to the forces pushing them. Since this was not a suitable choice I moved onto the second option, Detour Avoidance.

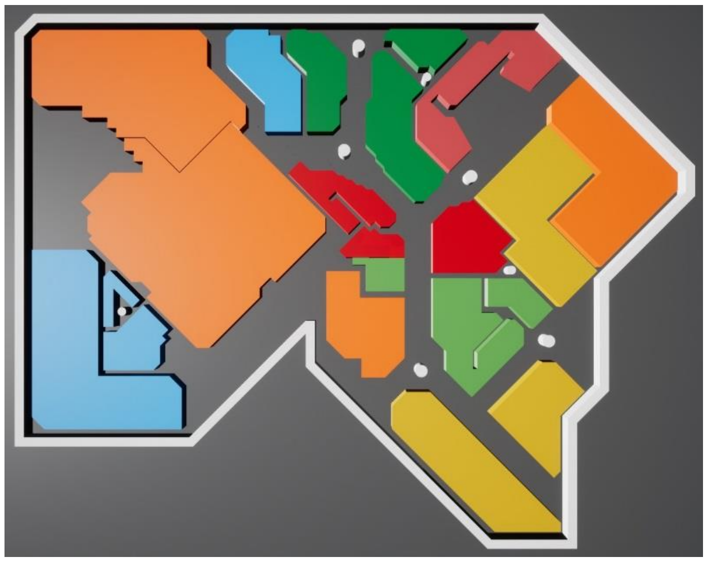
Detour Avoidance

To implement Detour Avoidance I had to make a new AIController which inherited from the DetourCrowdAICharacter instead of the standard AIController. I also decided to make a new AICharacter to go with it so replacing the AINavController and AINavCharacter I now had an AIDetourController and an AIDetourCharacter.

Instead of being force based like RVO, Detour Avoidance make use of the NavMesh to try and avoid obstacles. This means that the agents will never leave the NavMesh and instead of being pushed the agents will move in different directions to avoid obstacles. This is a much more realistic looking approach and when testing this against the RVO avoidance there was an instant noticeable improvement. However, the agents were still getting stuck in places. Looking through the settings for the Detour Avoidance I decided to increase the sideways bias and forward bias for the movement, with sideways bias being the highest value. This change was to make the agents ‘want’ more space in front of and at the side of them and, given an option, they will tend towards trying to make space sideways by walking around things. This change made the crowd more spread out with each individual agent seemingly desiring more personal space. In tests with many agents, the number of agents getting stuck was significantly reduced and in all cases the agents would eventually continue to their destination.

A Larger Level Layout

Now that the core components for the navigation were implemented and the small-scale testing had been done it was time to make a much larger level and test in a more realistic layout of objects. I created the simpler spawner I mentioned earlier that would take a defined number of agents and randomly spawn each them at a navigable location on the NavMesh. I then created a large level layout based on a shopping mall.



I also created a simple camera system to be used during the simulation to give the view from a CCTV camera and to allow switching between cameras.

The larger level was to be used for testing the agent’s movement and spawning on a larger scale and after that to be split up into smaller sections for individual levels. This testing did show some problems with the FindRandomNavigableLocation function on larger scales which would sometimes spawn agents inside buildings. All the objects create NavMesh both around and inside the object meaning that there is NavMesh in all the buildings. This NavMesh should not be reachable

through the function FindRandomNavigableLocation however so I assumed this was a bug with the function itself. With a smaller spawn radius, this bug does not occur and since the main levels will be smaller sections this wasn't a significant problem. But since I wanted to test the agents on the larger level I decided to fix this by adding a NavModifier to all the objects in the scene with NavArea_NULL so that it could not spawn NavMesh inside the object.

Behaviour Tree

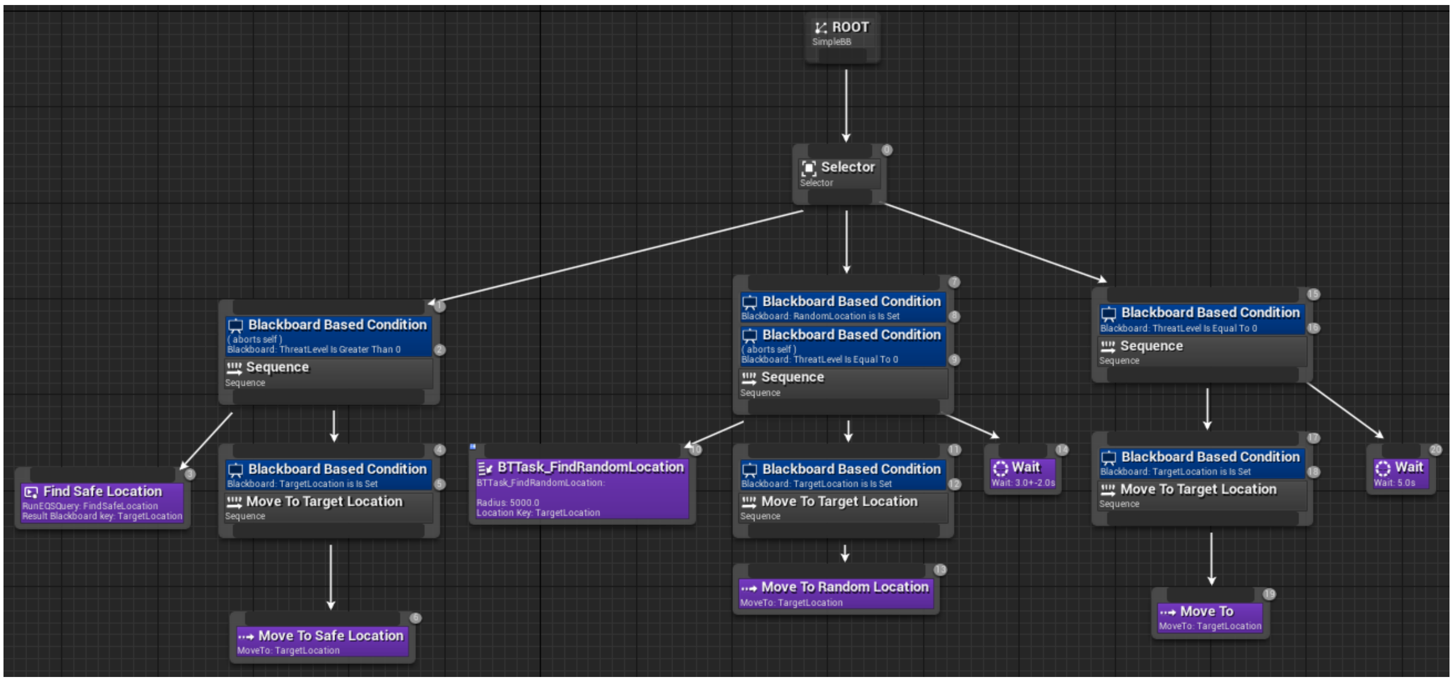
The Behaviour Tree is the most important aspect in creating a realistic crowd simulation. There are a huge number of potential behaviours that could be added based on real world behaviour and so it is important that there are systems in place for any of these to be added. When this project is continued in the future, this area is the main area that will be added to. At a fundamental level, the simulation must be able to simulate behaviour on an individual level because, as mentioned previously, this is an important for creating a threat detection system. The small individual details between agents will be vital in the creation and testing of the crowd analysis system. Alongside individual behaviours, there is also a requirement that the agents have a perception of their environment and the dangers within it. The system must allow for perception of various or even multiple dangers which will then influence the agent's actions.

are not necessarily traits themselves. They provide a simpler way of describing a large portion of personality related terms. This kind of model allows for a huge number of personality traits to be used in a much simpler way than defining every trait. Each agent contains the 5 OCEAN parameters which are defined when the AIDetourCharacter is spawned. These values are then sent to the controller and in turn sent to the Blackboard where they can be used in the Behaviour Tree. For example, a test could be made against the Extraversion value and if decided that it was very low then some behaviour could be implemented to stay away from people where possible. This area is easily the area with the largest room for further development, with potentially thousands of behaviours that could be implemented.

I used the OCEAN model along with the Big Five Inventory (BFI) to create realistic values for each of OCEAN parameters. The BFI is a set of 44 questions that are answered 1 to 5 and each answer corresponds to one of the five OCEAN parameters either positively or negatively. Using the BFI I determined the minimum and maximum for each OCEAN value and used these to randomly set an agents personality when spawned.

Stress Model

The Stress Model is to be used as a way of simulating more panicked decision making. The Stress Model will act as a sort of multiplier to the


ROOT
SimpleBB
Selector
Selector
Blackboard Based Condition
(aborts self)
Blackboard: ThreatLevel Is Greater Than 0
Sequence
Sequence
Find Safe Location
RunEQSQuery: FindSafeLocation
Result Blackboard key: TargetLocation
Blackboard Based Condition
Blackboard: TargetLocation is Is Set
Move To Target Location
Sequence
Move To Safe Location
MoveTo: TargetLocation
BTTask_FindRandomLocation
BTTask_FindRandomLocation:
Radius: 5000.0
Location Key: TargetLocation
Blackboard Based Condition
Blackboard: RandomLocation is Is Set
Blackboard Based Condition
(aborts self)
Blackboard: ThreatLevel Is Equal To 0
Sequence
Sequence
Blackboard Based Condition
Blackboard: TargetLocation is Is Set
Move To Target Location
Sequence
Move To Random Location
MoveTo: TargetLocation
Wait
Wait: 3.0+-2.0s
Blackboard Based Condition
Blackboard: ThreatLevel Is Equal To 0
Sequence
Sequence
Blackboard Based Condition
Blackboard: TargetLocation is Is Set
Move To Target Location
Sequence
Move To
MoveTo: TargetLocation
Wait
Wait: 5.0s

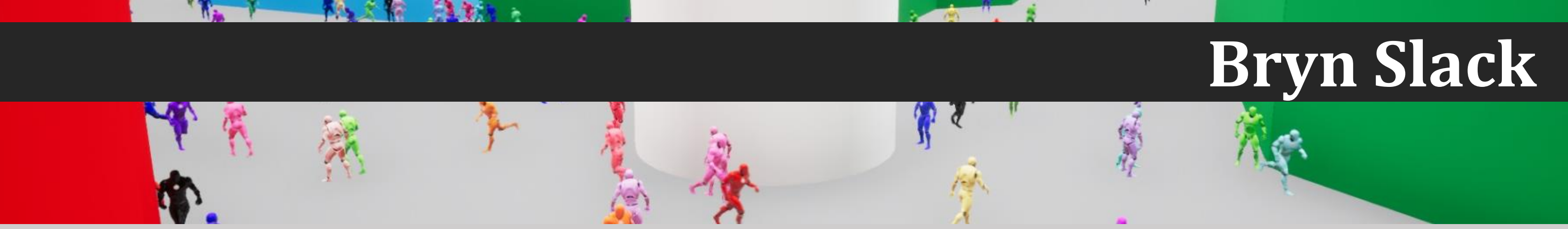
These requirements for the agent behaviour create the set of requirements for the Behaviour Tree implementation. To achieve these kinds of behaviours with the Behaviour Tree, many different systems need to be implemented and used by the Behaviour Tree. The first area that I looked at was the individual behaviours and there were two systems that I wanted to make use of for this. The first was a Personality Model and the second was some form of Stress Model and each would work alongside each other in changing the way the agents make their decisions.

OCEAN Personality Model

The OCEAN personality model, sometimes referred to as The Big 5, is a well-respected personality model that defines a person's personality in 5 areas; Openness, Conscientiousness, Extraversion, Agreeableness and Neuroticism. Each of these 5 areas define many traits of personality and

base OCEAN personality values defined. It will essentially make a person's personality more extreme and create more illogical decision making. If a person is only somewhat shy then, when panicked, they may become extremely shy or maybe not shy at all causing decision making to potentially by very different. This kind of implementation can be done with the Stress Model.

The Stress Model implementation will be based on the number of threats detected by the agent. Every danger that is perceived adds to the overall ThreatLevel and this value can be used in whatever way required in the Behaviour Tree. As a baseline use, the agents will move quicker depending on their current ThreatLevel with each new threat adding to this value. In future extensions to this area, the ThreatLevel could be used, as previously mentioned, to affect the OCEAN-based decision making creating more extreme decisions in attempt to imitate a level of panic in the agent.



Dangers

The Danger system is another core component of the simulation and will allow for simulation of panic scenarios. Anything from fires and explosions to fights in the street can be defined as a threat and the system should allow for all levels of danger. It is also important that the amount of danger can be acknowledged by the agent since not all danger holds the same amount of threat. The action taken during a dangerous scenario will be defined by the total level of threat, potentially consisting of multiple different threats.

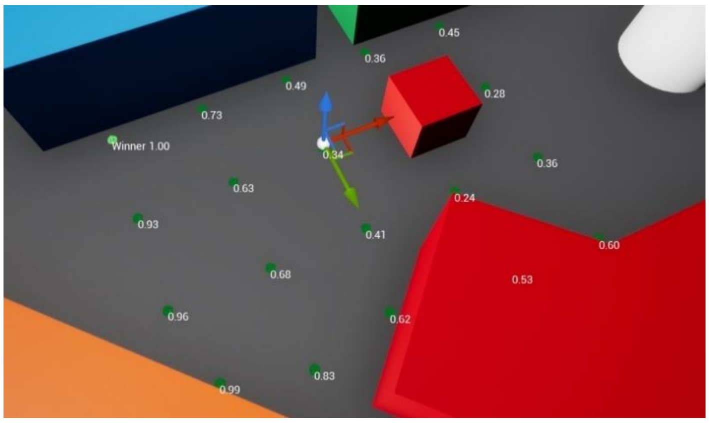
The first part of the danger implementation is giving the agents a way to perceive a danger. In UE4 there is a perception system that I made use of which allows anything with a perception stimuli source to be perceived. The perception system can be configured to use a variety of senses and new senses can be defined if needed. The main sense that the agents use is sight but hearing was also added as there are many dangers that could potentially be heard but not seen.

The next step to the perception system is defining a danger. For this I created a danger interface that can be added to any object. The danger interface has a set of functions that define what information an agent takes from this danger upon perceiving it. The main bit of information here is the threat level, which simply quantifies the danger level of this danger. It is the interface that contains the threat level so that any kind of danger that needs adding can be added by using this generic interface.

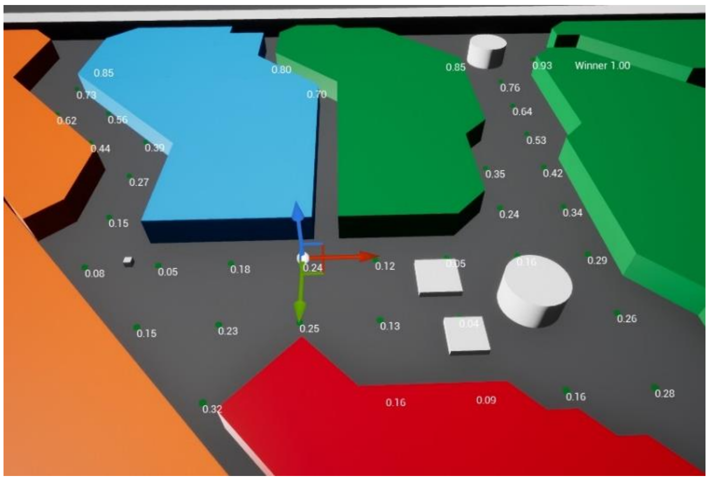
The danger object was then created as a blueprint that contained both the danger interface and a perception source stimuli. At this point, the danger can be spawned and an agent can perceive the danger. Once the danger is perceived some checks are made to verify that it hasn’t already been perceived. If it hasn’t been perceived, then the threat level is added to the agent’s current ThreatLevel and the agent’s blackboard is updated with the new ThreatLevel. This ThreatLevel is the value that defines the Stress Model mentioned previously.

Environment Query System

Now that dangers are being perceived by the agent’s there needs to be some behaviour implementation for after a danger has been perceived, Although the dangers are being perceived, the behaviour of the agents is currently not taking that into account. To implement this behaviour, I make use of the Environment Query System (EQS). The EQS is a system in UE4 that can query multiple points on the NavMesh and, based on a set of defined parameters, can find an optimal point. The EQS is used for the crowd simulation after an agent has perceived a danger. At the point an agent perceives a danger they must determine a safe route away from the danger, which is where the EQS comes in.



There are two queries that I use for the EQS. The first Query, and the most important one, is how near this point is to a danger. The EQS will check every point, within a given grid size and point spacing, and will evaluate how far that point is from one or more dangers giving the furthest point away the highest scoring. The second Query is about how far the point is away from the agent. The ideal point to escape to is both away from a danger and far from the point you are currently at. This query is important as it stops situations where the agent decides that a location next to a wall is best because it is far from a danger where actually it’s best to run right along the wall to escape from the danger instead of staying in the same place. Using both queries ensures that agents can identify a safe route away from one or more dangers.



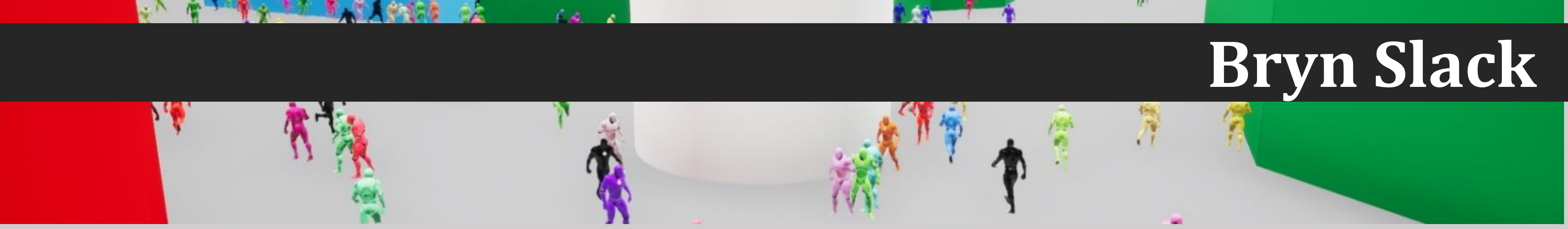
There are many more queries that could be added in the future to improve the EQS further. Things such as the threat level of the danger, giving points near to less threatening dangers a higher score, and the number of people near that point. If there are many people escaping down a specific path, then it could become very congested. Using the EQS system will allow for these things to be used in the agent’s decision making.

The EQS is used from directly within the Behaviour Tree and is only used once the ThreatLevel inside the Blackboard is not 0. Again, there are many changes and improvements that can be made here to add additional behaviour functionality. Multiple different EQS checks could be used depending on ThreatLevel amount or personality values, giving each agent even more individuality. Perhaps only certain types of people will think about how congested an area is when escaping.

Usability

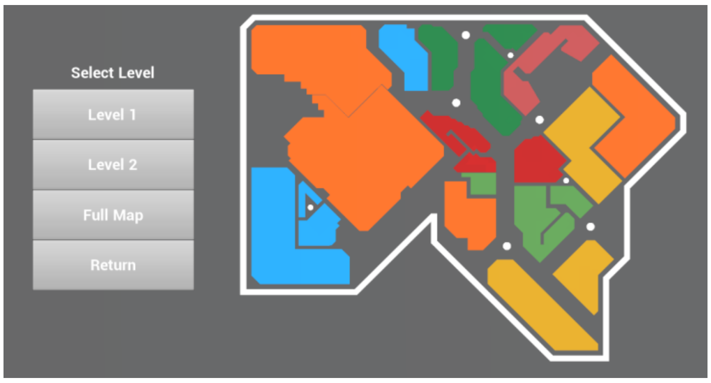
As well as being a functional crowd simulation there is also some user requirements for the system. The focus of my time on this project was to implement the core components needed so that it could be expanded on in the future with a good base for expansion. At this point in the project, these features were implemented and, in terms of the simulation, almost all the future improvements will be implementing new behaviours. This alone is an extremely large task and months, if not years, could be spent making improvements in this area. For this reason, my focus shifted towards some usability aspects for the project.

The way that the simulation was edited was almost entirely within the UE4 editor which was not ideal. For example, dangers were placed into the scene with a value to be used as a timer. Once the simulation begins, the timer will start and when finished will spawn the danger at the location it was placed in. With this method, the functionality was all there but making changes to the scenario was a little tedious. I changed the way that dangers were spawned so that dangers could be added

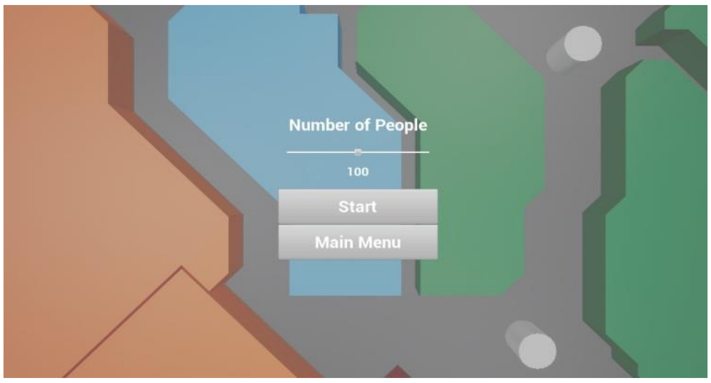


during the simulation with a mouse click. The user just clicks somewhere on the screen and a danger spawns at that location. This makes it much easier to make changes to the scenario. With this method, you can simply run it over and over clicking in different places to spawning dangers and seeing how it affects the simulation.

The other usability component I felt the project needed was a simple menu system with a few simple features. There are multiple levels or areas that the simulation can be run in so clearly some form of level selection is required. I chose to include the Full Map as a playable level along with two smaller sections of the map that I had chosen. A huge improvement to this would be to allow the user to define an area of the map to use for the simulation. This would remove the need for a level selection as such and would give the user much more freedom in creating the simulation.



Inside the editor, there is also the option to set the number of agents to spawn which is also something I felt that the user should have control over. Again, this gives the user a lot more freedom in choosing what they want from the simulation.



This menu system is only a simple start to what user functionality could be added but already its clear how much more user friendly the system is. Not only will is it more user friendly but it makes it much quicker and easier to test different scenarios and with further improvements in the future it will get even quicker which will be very important towards the completion of this crowd simulation. When this system is to be used to for the creation of the crowd analysis system it must be both quick and simple to create a new scenario.

The Project Goals

Overall, the project went according to plan and once a bit of research into the project had been undertaken the goals for the project became a lot clearer. The goals for the project, as a whole, remained the same which is to create a realistic crowd simulation that can be used to create a crowd analysis system. After research, my individual goal for my input into the project was decided. That goal was to implement the various features at a fundamental level so that a baseline set of features existed for the simulation. Before the research, it was part of my goal to implement a variety of complex behaviours but it became apparent that behaviour implementations to that level could have been a project of my length on its own. However, a project like that would require a base crowd simulation to start from. My project could easily provide that for someone in the future and would be a perfect starting point for a project solely focused on behaviour implementation. I believe that my goal for the project was achieved and each component was implemented at a basic or higher level leaving sufficient space for improvements and further implementation. The personality model is fully implemented for each agent and is usable in the behaviour tree for any personality based behaviours. The perception system provides the ability for every agent to perceive any kind of danger and the EQS provides a system for a danger response.

The usability implementations became a smaller secondary goal as I progressed through the development. Since my original goals, that were purely focused on the actual simulation, were essentially fulfilled, I decided to make a start on some of the user requirements for the project. Although this is quite a simple start into the potential UI features, it will still provide a good foundation for adding more UI and user based features in the future. It also made my work on the project a much more complete and finished package.

The Future of the Project

Going forward there are only two main areas that should be worked on and those areas are the behaviours and the usability. For the behaviours, it will mainly be work on the Behaviour Tree with perhaps some changes being made to the EQS. The Behaviour Tree is much more simple than I had anticipated it would be before starting the project. But, as previously mentioned, after both researching the project and starting the project, it was clear that the Behaviour Tree would not be as complex as I previously thought. It would be far too time consuming and would be impossible to do that as well as the fundamentals that it relies on. Making use of the OCEAN model inside the Behaviour Tree is a very interesting and exciting prospect that will vastly improve the variation in agent responses and make the crowd feel much more realistic. The usability improvements should be made alongside the behaviour improvements as they can be implemented separate from the behaviour. Things like being able to set the type of danger to spawn or defining a specific area for the level, perhaps even defining the exact locations for the CCTV cameras will all provide a better user experience. Being able to then save a scenario created to use later could also be a potential idea which would further contribute towards being able to use this for the crowd analysis system.

Summary

In summary, using UE4 to create the crowd simulation proved to be an excellent choice. UE4 provided an excellent platform for all the features required and allowed for quick testing of those features. There are also many guides on each of the various features I used which is very helpful when implementing lots of features for the first time. Generally, the guides do not explain the features in terms of crowds but more in terms of a single AI. This knowledge however, is still very helpful in gaining an understanding of the feature which can then be used in making your own variation on the implementation seen. The main issue with using UE4 for this project was that, much like most engines, it comes with bugs here and there. Some implementation bugs, such as the random navigable location function, and some more annoying bugs like the development environment crashing. With that being said, I think Unreal Engine is a great choice for creating any crowd simulation application.