Noach Vandekerckhove

Heterogenous crowd simulation through individual traits

Graduation work 2021-2022

Digital Arts and Entertainment

Howest.be

CONTENTS

ABSTRACT

In this paper an attempt will be made to create a more heterogenous crowd simulation using trait theory. These traits will be split into 2 types of traits namely anthropometric traits that will have an influence on the visual representation of the characters and personality traits that will primarily influence the behavior of each individual character. To achieve this, connections will be made between a trait and 1 or several underlying parameters. These parameters would then in turn influence the visuals of the character while other parameters would influence the steering behaviors of the character. While this trait theory approach does succeed in diversifying the crowd to some extent the personality traits are not that noticeable seeing as the differences in behavior are not that evident unless close attention is paid.

INTRODUCTION

Crowd simulation is the process of simulating the dynamics of numerous entities. It is a technique that is commonly used to create virtual scenarios for video games and films. It is not only used for entertainment purposes, but it also has significant uses in the public safety and architecture departments where it could be used to simulate evacuation scenarios for example.

This paper will focus on the individual behavior modeling of each agent in the crowd. The technique that this paper will investigate further will be the technique that incorporates personality traits for each agent to achieve individualistic behavior. This means that for this project each individual agent will have certain variables that directly correlate to personality traits such as aggressiveness. Theses variables will govern the agents' behavior.

Anthropometric variables will also be added to each agent to further diversify each agent and add to the overall realism of the crowd simulation. Therefore, the main purpose of this paper will be to examine if a heterogenous crowd could be simulated using both personality and anthropometric traits that improve the diversity in the crowd. This diversity will be measured on both behavior and appearance.

During the process of constructing and evaluating the effectiveness of our artifact this paper will also investigate which type of traits are more effective in achieving this diversity since both Personality and Anthropometric traits are used.
Lastly this paper will go over several existing frameworks and analyze if this framework could be a suitable starting point to create the crowd simulation with the traits.

RESEARCH

In this section, Related work in crowd simulation and behavior modelling for crowd is discussed. Because of the complexity of crowd simulation, there is a broad spectrum of problematic aspects that various studies have assessed. One of those studies is [1] that describes the most problematic aspects and challenges of crowd simulation. These challenges included: Performance, Motion planning, Animation variety, Appearance Variety, Crowd behavior and many more. In this paper the focus will be on the Appearance Variety and behavior modelling.

In a GDC talk [14] where they explain some of the crowd mechanics in the game Hitman: Absolution they go over how they achieved more visual variety in their crowds. This talk was useful to determine the challenge of Appearance Variety and give insight into possible solutions. One of the techniques they used is a unique scaling factor for each agent. It is only a 5% factor, but they explain that it does wonders for the perceived diversity of the crowd and helps to soften up the horizon. Another technique they used is diffuse texture overrides. This serves as a cheap way of reusing the same shirt for example but still diversifying it with different colors.

And for the challenge that is behavior modelling there are also a lot of approaches that have been taken before. In a GDC talk [13] they talk about the game Watch dogs 2 where they combine 3 systems to create diversity in their crowd. These systems consist of Attractors that are spawned activities that are manually placed in the world. Secondly there are Dynamic attractors that use the same logic as the normal Attractors, but these ones are dynamically spawned in the world. An example of a Dynamic Attractor would be a crime occurring. The last and most intricate system they have implemented would be a Reaction system. This is a stimuli-based system where each stimulus will cause a certain reaction out of the agents that are affected by the stimuli. Each agent could have multiple possible reactions to a stimulus the decision will be affect by some criteria such as the current behavior and what the stimuli was and other criteria. The behavior that results from this stimulus will in turn also be a stimulus for other agents thus creating a chain reaction of emergent behavior.

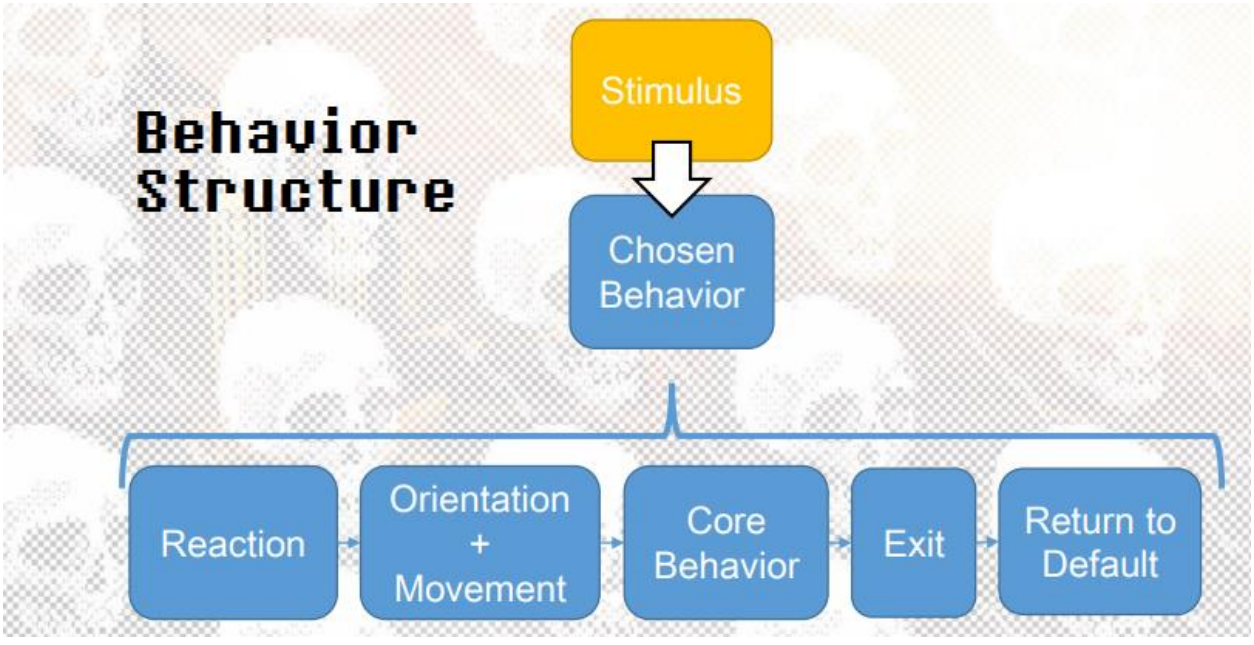

Figure 1: Behavior structure of an NPC in the watch dogs franchise

1. BEHAVIOR MODELING WITH THE TRAIT SYSTEM

1.1. TRAIT THEORY

As stated in [4] psychologist have proposed various ways of characterizing the spectrum of personalities exhibited by humans. Several of those theories focus on aspects of personality that are relatively consistent over time and across various situations. While there are many sources of variety in behavior, psychologist have proposed methods to categorize and organize these variations.

A personality trait is a habitual pattern of behavior, thought or emotion. While humans display a vast number of different traits, a small number of these traits are believed to be central to an individual's basic personality. Trait theories identify these primary traits, which can be used to describe variations in personality; an individual's personality is described based on a score of how strongly or weakly they exhibit each of these primary traits.

Some traits such as introversion <-> extroversion work on a spectrum but there are also other traits that researchers categorize as traits a person either has or does not. So, this will need to be kept in mind when choosing traits to implement and when implementing them.

1.2. PERSONALITY MODELS

Before looking into personality models, it might be beneficial to investigate personality a bit more. While there is no generally agreed upon definition of personality, a definition for Personality could be "Personality is the dynamic organization within the individual of those psychophysical systems that determine his characteristics behavior and thought" (Allport, 1961, p. 28). This definition really emphasizes the uniqueness of the individual.

Now that a better understanding of what the word personality entails is established, the time has come to delve into some Personality models. One of the most well-established trait theories is the Eysenck 3-factor model. This model identifies three major factors which categorize personality: Psychoticism, Extraversion, and Neuroticism.

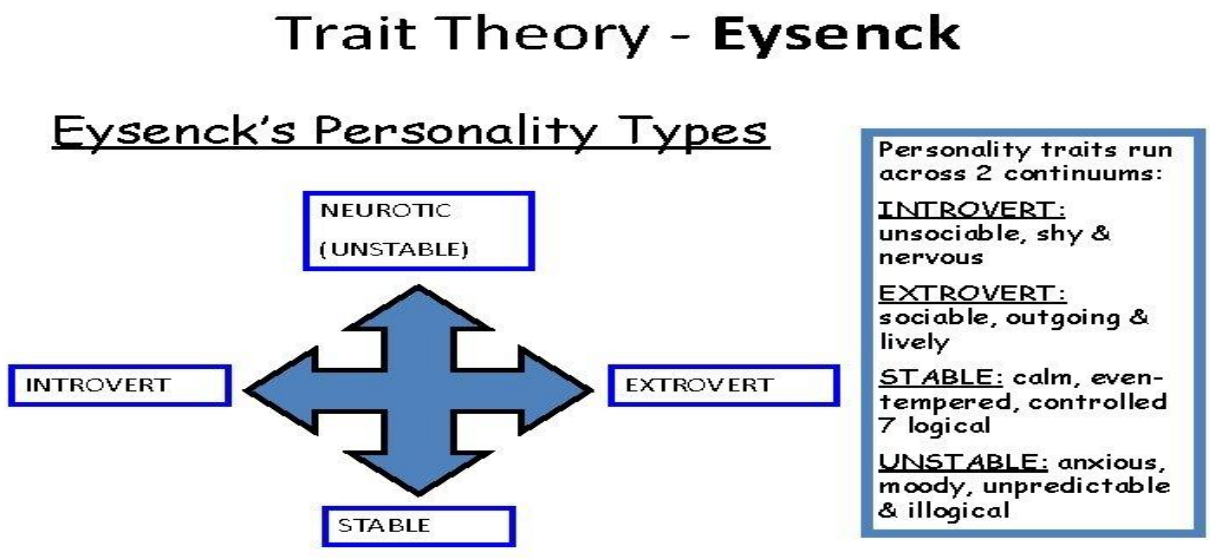

Figure 2: Illustration of Eysenck's personality trait model

1.3. OCEAN MODEL

Another personality trait model that could be interesting to use has already been successfully implemented by both [3] and [5], There they decided to use the OCEAN model and had successful results. The OCEAN model which is also known as the Big Five personality trait model is a suggested grouping for personality traits. It is divided in 5 factors (see Figure 3).

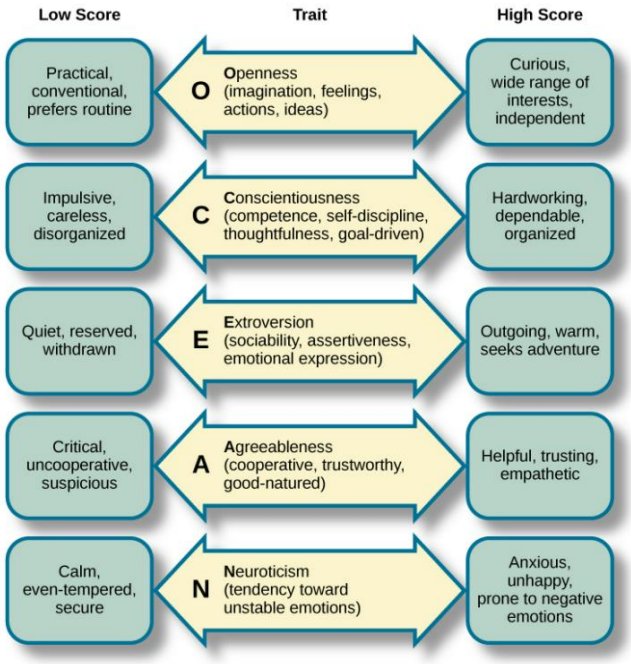

Figure 3: Illustration of the OCEAN model

They map each of these key factors to one or several low-level parameters. Then they used these parameters and some of the built-in behaviors to change the agents' behavior based on the personality traits of that agents. The strength of each trait is amplified by how many parameters are tied to it. In the research examples they set each trait to be either Positive, Negative or in some cases Neutral.
The 5 traits in question are:

- openness to experience (inventive/curious vs. consistent/cautious)
- conscientiousness (efficient/organized vs. extravagant/careless)
- extraversion (outgoing/energetic vs. solitary/reserved)
- agreeableness (friendly/compassionate vs. critical/rational)
- neuroticism (sensitive/nervous vs. resilient/confident)

In [3] they give a concrete example (see Figure 4) of how each of these traits are directly connected to a different variable that will have a direct impact on the behavior on the behavior of each individual agent. They do use some fuzzy logic however that decided how each of these variables are mapped to each other. Implementing a Fuzzy logic interface will be out of scope for this Paper.

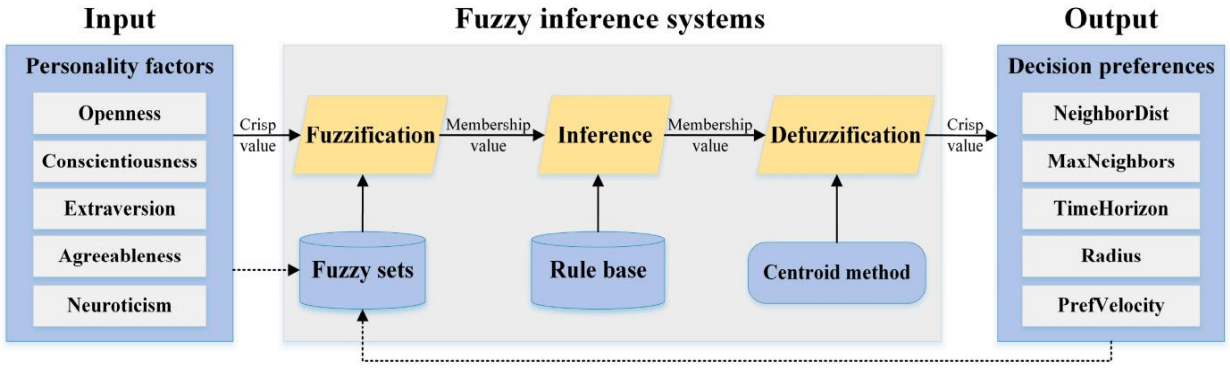

Figure 2. Structure of the proposed fuzzy logic system.

Figure 4: Example of the fuzzification of the personality traits in [3]

1.4. PHYSICAL TRAITS MODELS

There are a lot of differences between the body dimensions of different people, because of their height, age, weight, and other measurements. These differences should not only affect the behavior of people but also how they are visualized. This in turn makes it necessary for us to define the most important measurements to help further improve the realism of our simulation model.

In [2] they introduce anthropometric measurements. The core elements of anthropometry are height, weight, head circumference, body mass index (BMI), body circumferences to assess for adiposity (waist, hip, and limbs), and skinfold thickness. In the image below you can see a full list of the variables that were used in the previously mentioned research paper. They also specify which measurement unit is used for each of these variables.

Variables	Measurement Unit	Variables	Measurement Unit
Age	Years	Height	Centimeters
Weight	Kilograms	Body Mass Index (BMI)	(kg/m^2)
Total Arm Length	Centimeters	Upper Arm Length	Centimeters
Lower Arm Length	Centimeters	Wrist Circumference	Centimeters
Upper Arm Circumference (REST)	Centimeters	Upper Arm Circumference (contracted)	Centimeters
Chest Circumference	Centimeters	Chest Circumference (Elevated)	Centimeters
Hip Circumference	Centimeters	Thigh Circumference	Centimeters
Calf Circumference	Centimeters	Hand Width	Centimeters
Shoulder Width	Millimeters	Hip Width	Millimeters
Bi-Humerus Diameter	Millimeters	Bi-femur Diameter	Millimeters
Wrist Diameter	Millimeters	Body Density (BD)	(gm/cc)
Percentage Body Fat	%	Total Body Fat (TBF)	Kilograms
Lean Body Mass (LBM)	Kilograms	Grip Strength (Rigth Hand)	Kilograms
Grip Strength (Left Hand)	Kilograms		

Figure 5: Example of Anthropometric measurements and how they are measured in [2]

These measurements are a good starting point to further define which traits would be worth implementing.

2. CREATING A HETEROGENOUS CROWD

2.1. BEHAVIOR OF THE INDIVIDUAL AGENTS

The behavior of the agents will be split it up into 2 aspects. The first aspect being a Finite State Machine or a behavior tree. In [12] they based the behavior of the agent on a Finite State Machine. Each state would impart a unique and observable behavior on the agent. For this aspect the agents could already use their traits to define which states are accessible to the agent and how easily they go from a specific state to the other.

The other major aspect for the behavior modeling will be the steering behaviors. Blended or priority steering behaviors where some of the traits will be the deciding factor for how to blend or when to prioritize one steering behavior over the other would be the base for this aspect. On the right side you can see an example of the Finite State machine used in [12].

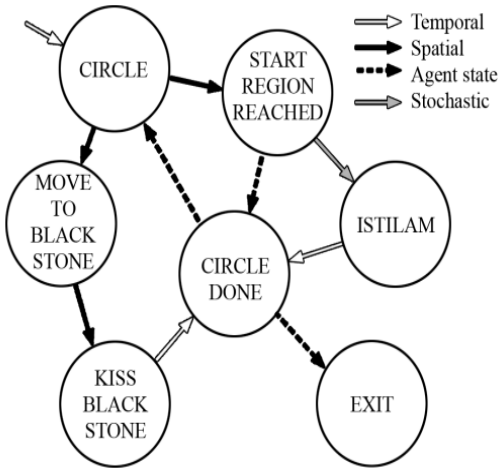

Figure 6: The Finite state machine in [12]

2.2. VISUAL VARIETY OR DIVERSITY IN THE CROWD

One of the problematic aspects in crowd simulation is visual variety or diversity of the crowd since it affects the overall perception of realism in many crowd simulation scenarios. Owing to computation and resource limitations, most real-time simulation systems must repeat agent appearances or motion patterns for efficient performance, with a corresponding sacrifice of crowd diversity. Researchers have proposed several approaches for enriching the appearance.

Figure 7: Variation of height

In the research paper [1] they go over how to create variety in appearances. There they divide the Appearance variety into 2 key challenges. The first one being Color variety and the second one being Shape variety. They then proceed to subdivide these aspects further. For the latter aspect for example they subdivide it further into shape and height variety. This information can be used to choose how the traits can influence the variety in appearance. For the antropometric traits they would logically be closely linked with the Shape aspect while for the personality traits it might make it easier to link them to the Color aspect which if possible would be linked to clothes.

The Watch dogs' franchise was also on the top of the investigation list since they appear to be a leading franchise when it comes to diversity in the AI. The NPC diversity in the Watch dogs' franchise has always been an important aspect of the gameplay. In the first two games you could "Profile" any NPC you can across, and they would have some unique text popup with some information about them. In their latest game Watch dogs Legion, they took things a step further and made any NPC roaming around the world playable. Naturally to make this mechanic fun and they needed to make every NPC feel as unique as possible. Another interesting GDC talk is [11] where they explain to some extent how they approached this challenge. In the image below you can see how they went about generating unique looking NPC's. They randomly generated some data for each of these NPC's then using the given data they started filling in the rest. For the example given below it could have selected the age range first. Then after this was selected it selected an occupation that is suitable for someone in this age range. Then using this occupation, it would select a fashion style and so on.

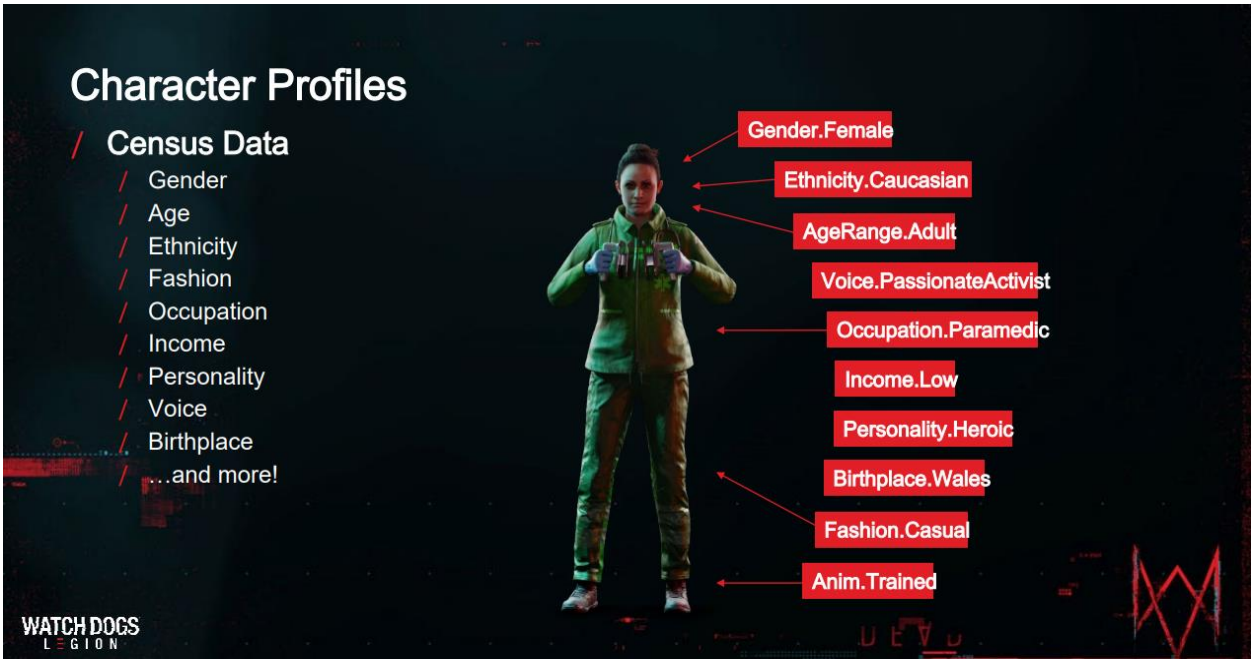

Figure 8: Example of a character profile in the watch dogs franchise

3. EXISTING CROWD SIMULATION FRAMEWORKS

3.1. UNREAL ENGINE 4 CROWDSYSTEM

The first framework that was considered was the Unreal engine 4 framework. This engine was the first to be considered for a couple of reasons. The first being the experience that had already been gathered with the engine since this would significantly speed up the development process. Secondly, in the documentation it is stated that there is already a preexisting UCROWDMANAGER class in this engine. This class might be worth looking further into to decide if this would be a good starting point for this project.

Other than the UCROWDMANAGER class the Unreal engine 4 also has other starting options available in its asset store. An example of this would be the AtomsUnreal – Crowd simulation asset. This is a more advanced asset that takes care of a lot of the aspects that aren't the focus of this project but is still easily extendable by both C++ and blueprints. This asset does require a license which might be hard to obtain without paying for it.

3.2. OPENSTEER

Another possible framework that was discovered during our research was the OpenSteer framework. This framework was encountered when looking into [6]. The OpenSteer framework is a C++ library that helps construct steering behaviors for autonomous characters in both games and animation.

When looking further into the documentation of OpenSteer [7] it became clear that some steering behaviors such as Flee, Seek. Cohesion and many more have already been implemented. This would be convenient since creating the steering behaviors would take a considerable amount of time that could otherwise be spend on refining the trait system.

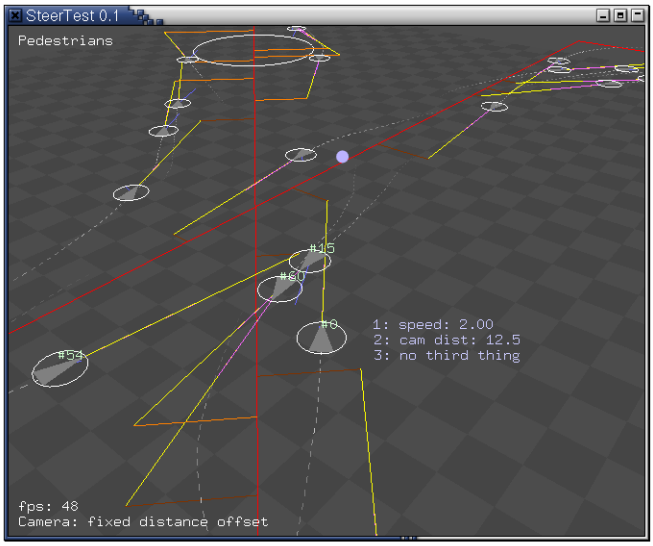

Figure 1: typical OpenSteerDemo window

Figure 9: Example of an opensteer application

Unfortunately, when looking at the section in the documentation that is supposed to explain how to integrate your own code with the engine it specifies that this section is only a preliminary placeholder.

3.3. THE OPEN AGENT ARCHITECTURE

The Open Agent Architecture was used in [8] were they divided the system in 3 main components: The group of agents, The Java application and lastly the Web application. After taking a closer look at the documentation [9] it became clear that this framework while useable might not be the best fit for this project. This conclusion was made after taking into consideration that the framework is rather outdated with its latest version dating back in 2007.

CASE STUDY

1. INTRODUCTION

This chapter will cover the steps taken to create the crowd simulation model and will explain some of the reasoning behind several of the decision that were made. The first subject that will be discussed will be the reason as to why the Unreal Engine 4 was chosen as the engine for this specific crowd simulation and the many benefits and possibilities that this engine provides.

Secondly the logic behind the AI will be explained. This subject will be dived into 2 major parts. The first of which being the pathfinding. In that pathfinding section the details about the path creation and navigation mesh alterations will be explained.

After the pathfinding logic has been reviewed the next subject will be the various steering behaviors that have been implemented into the AI. In short, the steering behaviors that will be discussed are the Seek, Wander, Cohesion, Separation, and the Avoidance steering behavior. The blending of the steering behaviors will not be explained in this section but instead in the next one since the blending is directly linked to the traits.

The last part that will be covered is an explanation of the individual traits and how they were implemented. Like the previously mentioned pathfinding section this will again be split up into 2 parts. This time the parts will be split up by which type of traits namely anthropometric traits and personality traits. For each of these traits there will be a description as to what underlying parameters and behaviors they relate to and how they come into fruition in the crowd simulation.

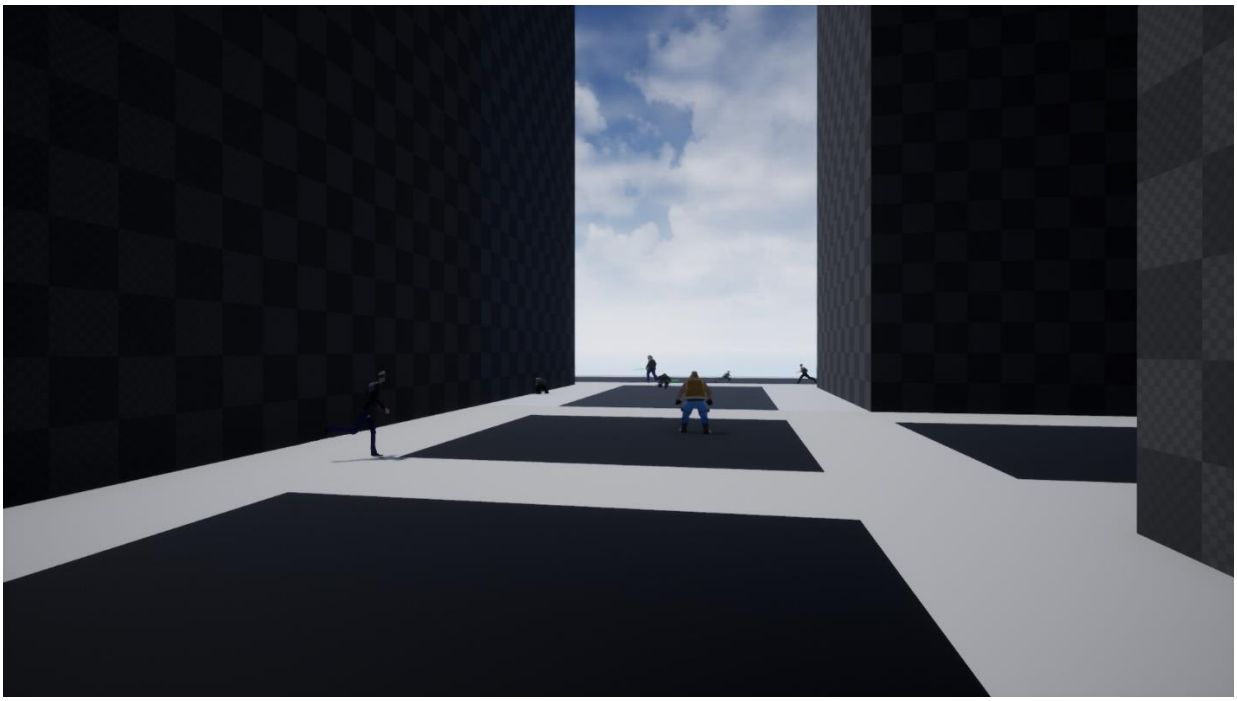

Figure 10: Screen capture of the crowd simulation in progress

2. WHY UNREAL ENGINE 4 WAS CHOSEN AS THE FRAMEWORK

2.1. OPENSTEER

The Opensteer framework looked promising with its already implemented steering behaviors and the fact that this framework was specifically designed to help construct steering behaviors for autonomous characters in games would also be beneficial to the project. However, the lack of documentation [7] for implementing custom C++ code in the Opensteer framework (see Figure 11) became the main reason that hindered it being chosen as the framework. The lacking visual capabilities was also considered as a good argument against using this framework.

Integrating with Your Code

Note: this section is so preliminary that it is really just a placeholder...

This section will eventually talk about the structure of OpenSteer's classes, and how they are designed to provide you with flexibility as you integrate OpenSteer with your existing code. It will mention that if you were writing new code, you could always base your classes on OpenSteer's. More typically you will already have an existing code base, a game engine or a procedural animation system, into which you want to integrate some of OpenSteer's facilities.

OpenSteer's classes have been designed to allow you freedom to either inherit its functionality, or to layer its functionality on top of your existing classes. The latter approach is supported by the concept of mixin classes (essentially a class with templated superclass) which allows its functionality to be "mixed in with" (or "layered on top of") your existing classes. A similar effect can be obtained by multiple inheritance of your preexisting base classes and OpenSteer's classes, but most C++ programmers prefer to avoid multiple inheritance. Many of OpenSteer's classes are defined in three parts: an abstract protocol (aka interface or pure virtual class: "AbstractLocalSpace"), an implementation expressed as a mixin ("LocalSpaceMixin") and an instantiable class made by layering the mixin on top of the abstract protocol ("LocalSpace"). For more detail see **LocalSpace.h** and the discussion at the top of **SimpleVehicle.h**

Finally OpenSteer anticipates that it may be used in class hierarchies based on either the IS-A or the HAS-A architecture. That is, you might want an agent in your game to be structured so that it IS-A OpenSteer Vehicle or you may want to structure it so that it HAS-A Vehicle as a component part. (But no sample of using an OpenSteer Vehicle in a HAS-A architecture is provided [see "Known Bugs" below].)

Figure 11: The section of the Opensteer framework documentation that specifies how to integrate code

2.2. THE OPEN AGENT ARCHITECTURE

The second framework that was considered but eventually not chosen was the Open Agent Architecture. It's main feature that was appealing to this project was its ability to easily add or replace agents at runtime.

Nevertheless, this positive point was swiftly overshadowed by the many negative elements that came with choosing this framework for this specific use case. A couple of the most concerning negative points summed up are the fact that when looking further into the documentation [9] and the Open Agent Architecture based applications it seems that this framework isn't meant for game related projects (see Figure 12).

Figure 12: Example of an Open Agent Architecture Application

Secondly when looking into what platforms it supports (see Figure 13) it reads "SunOs/Solaris, Linux, Windows 9x/NT/2000/XP". So, as you can see the supported platforms for C++ based applications seem outdated or at least haven't been tested.

OAA 2.x agent libraries exist for the following languages, and have been used on (at least) the following platforms:

Quintus Prolog	SunOs/Solaris, Windows 9x/NT/2000/XP, other Quintus-supported platforms
Sicstus Prolog	SunOs/Solaris, Linux, Windows 9x/NT/2000/XP, other Sicstus-supported platforms
ANSI C/C++ (Unix, Microsoft, Borland)	SunOs/Solaris, Linux, Windows 9x/NT/2000/XP
Java	Any Java platform
Compaq's Web Language	Any Java platform

Figure 13: Screen capture of the documentation of the Open Agent Architecture framework

2.3. UNREAL ENGINE 4

After all the comparisons were made the framework that eventually got chosen is the Unreal Engine 4. This engine had a lot of positives going for it. For example, the user-friendliness of this engine is very high with its custom blueprint language and extensive documentation. To add onto that since it is a well-known engine there are many other sources such as guides, tutorials and forums that can provide additional knowledge or even propose solutions to any eventual problems. Performance wise this engine also performs great with being able to simulate many Ai controlled characters at once without a heavy performance hit (see Figure 14).

Figure 14: Example of performance in unreal with crowd instancing

Visually this framework also outperforms the other two by quite a large margin. So, not only would this make it possible to visually the traits better for each character but there would be a possibility to make a city scene to add to the realism of the simulation. Additionaly for character visuals or the cityscene there also exists an asset store that could have the assets needed to build a city or make the characters seem more realistic.

Figure 15: Example of city scene from the Unreal engine 4 asset store

2.3.1. UCROWDMANAGER

Unreal Engine 4 also comes with a preexisting UCROWDMANAGER class. Unfortunately, the UCROWDMANAGER class only works as a C++ class and cannot be used as the base for a blueprint. When looking at the documentation [10] for this UCROWDMANAGER class it says “If you want to create a custom crowd manager implement a class extending this one and set UNavigationSystemV1::CrowdManagerClass to point at your class”. This has been considered when creating a derived class to test if this parent class is a suitable starting point for the crowd manager class that will be needed in this project.

Some of the benefits that this class provided would be the functionality that is already in the parent class. For example, to move the agents one of 3 pre-existing functions could be used the: SetAgentMoveDirection, SetAgentMovePath and lastly the SetAgentMoveTarget functions. As made clear by the naming convention of each of these functions there will be an opportunity to pick the way the agents will move. Either by giving them a direction that the agent should go towards or giving the agent an entire path that they would then follow. Another option would be able to simply give a target that the agent would automatically go towards.

For the more complex steering behaviors there already exist some useful functions like the GetNearbyAgentLocations function that would be helpful when creating a Cohesion steering behavior.

It would also be possible to update the existing avoidance config (more on that in the next section) using the SetAvoidanceConfig. There is also a function to set the NavMesh data.

Regrettably, when continuing to examine the documentation [10] and the source code of this class it became clear that to register new agents to this manager it would need to have an agent with the ICrowdAgentInterface. Looking at the documentation of this interface class it lacked info about what exactly this class entailed.

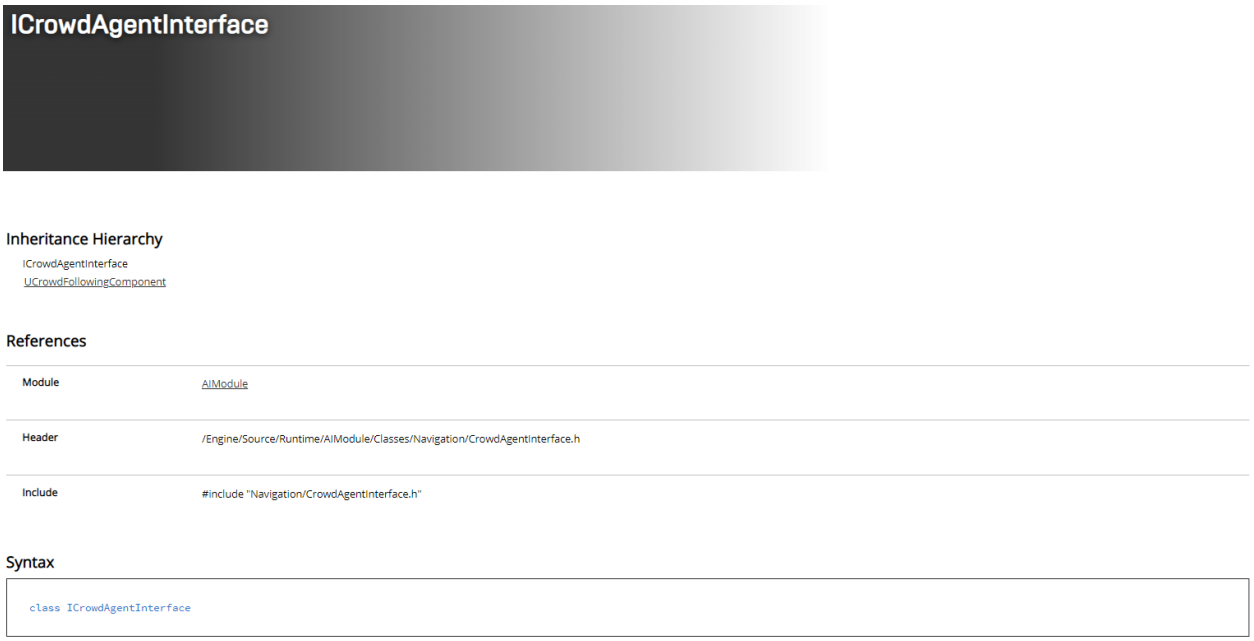

Figure 16: The documentation for the ICrowdAgentInterface

This necessary interface made it difficult to implement the custom agents into the derived CrowdManager class, so the decision was made to create a new CrowdManager that does not inherit from a parent class.

2.3.2. AI CROWD MANAGER

As mentioned in the previous section there is an avoidance config system in Unreal Engine 4 that gets handled by the AI Crowd Manager class. To achieve this effect the class will use Detour which is a Recast library in Unreal Engine 4. The parameters for this manager can be set using either the Project settings (see Figure 17) or by creating a FCrowdAvoidanceConfig object in C++ code and setting it as the active one by calling the previously discussed function SetAvoidanceConfig in the CrowdManager class.

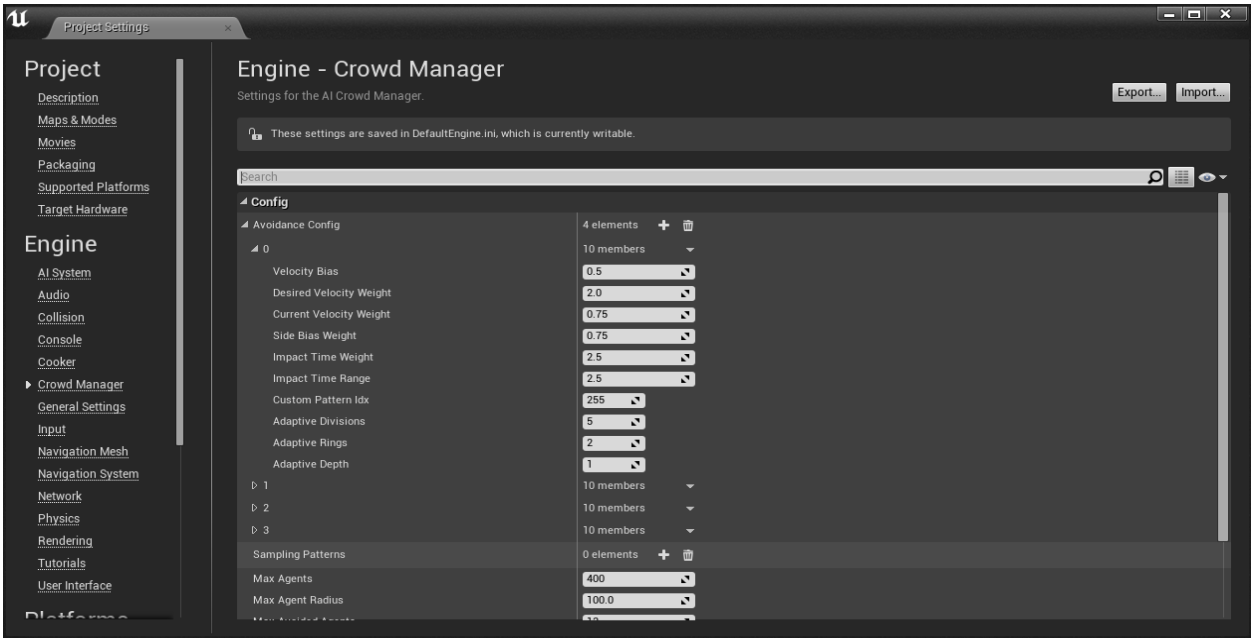

Figure 17: Section of the parameters of the Crowd Manager project settings

This can be easily implemented by inheriting from the DetourCrowdController class. A simple tutorial can be found in the Unreal Engine 4 documentation [19]. When this controller or a derived controller class is assigned to the NPCs, they will automatically try to avoid each other depending on the parameters that were set.

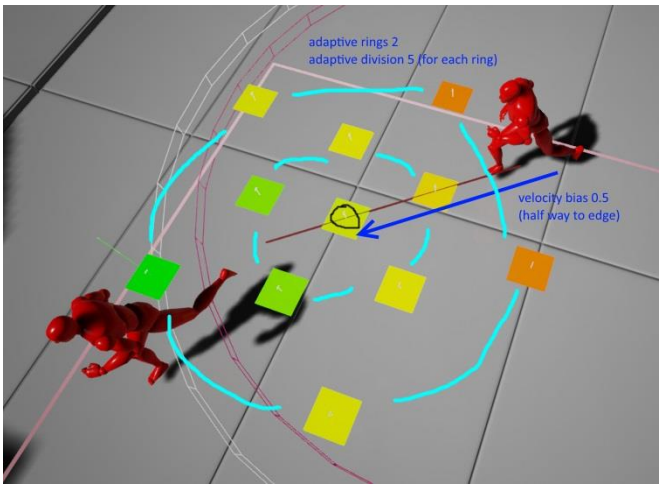

Figure 18: Illustration from [16] that helps visualize some of the Crowd manager project settings

Some of these parameters are self explanatory like the Max Avoided Agents parameter or the Max Agent Radius variables. But there are also ones that are not as clear. Such as the Desired/Current-VelocityWeight variables. These variables take weigh the direction of the avoidance against the desired or current veleociy respectivly. Another parameter that was not clear at first is the Adaptive depth parameter (see Figure 18). This variable defines how many times it will create a sample circle. The higher this variable to more refined the avoidance behavior would be.s

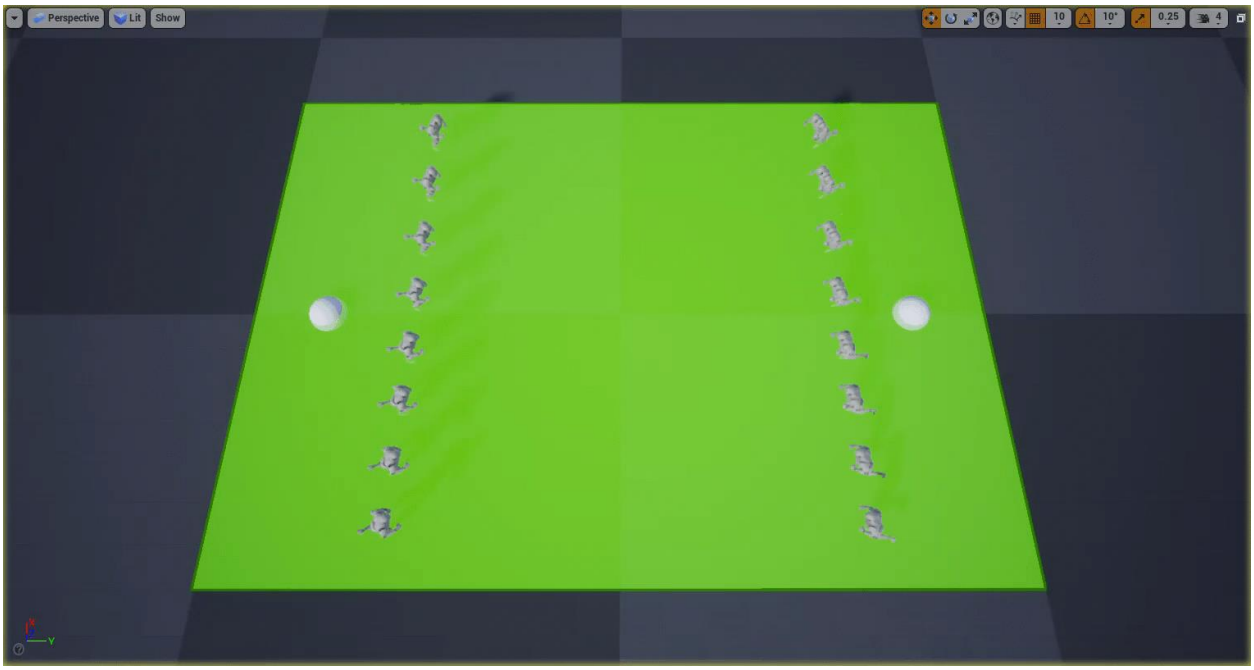

Figure 19: Example without any avoidance logic from [19]

Figure 20: Example with the same logic but with use of the DetourCrowdController from [19]

The main reason for this Detour AI crowd manager not being used in the final project Is because it is not possible to have different managers or parameters for each agent. This would result in the agents behaving the same which is not the intended result.

3. PATHING

3.1. PATHING SYSTEM

The base for the pathing system was derived from [17]. Their implementation uses blueprints which was converted into C++ for this project. It works by creating a custom blueprint that has a spline component that will spawn another custom blueprint at an interval along the spline. You can move the spline and add points to it while working in the editor to adjust it to the level. The blueprint that it spawns is a simple blueprint that just consist of a box collision. The collision for this box collision was set to a custom object channel that was defined in the engine project settings.

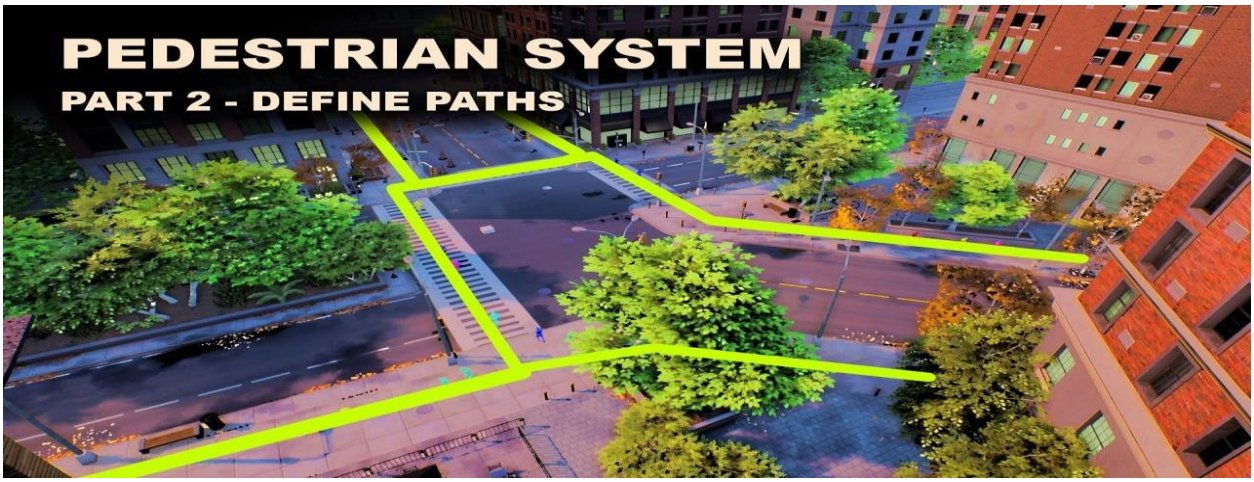

Figure 21: Example of the NPC doing a multi sphere cast

Now with the path creation out of the way the next step was to create a custom AI character with a custom AI controller. At first the capabilities of this AI were very limited but further in this paper he well gains more features but at this point the only feature he had was following the path. This path following was done by doing a MultiSphereRayCast a bit in front of the NPC. This ray cast only checked for the custom object channel that was created so that only the path points that were spawned by the spline would be hit. If it hit one or more path points it would randomly select one and move towards it. If no path points were found it will increase the radius of the sphere until it eventually does find one.

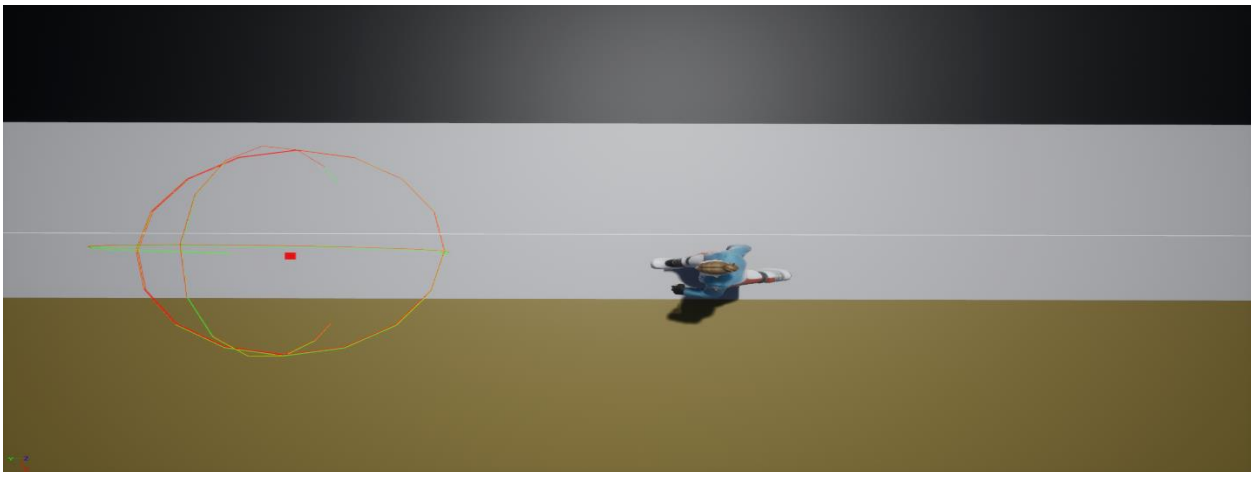

Figure 22: Example of the NPCsdoing a multi sphere cast (red point is the path point)

3.2. CUSTOM NAVIGATION AREAS

Another big part of the pathing logic would be with accomplished with the custom navigation areas native to the Unreal Engine 4 as seen in [20]. This will work in tandem with the previously explained pathing system.

Figure 23: The different options for a custom navigation area from [20]

This project will only work with 3 possible navigation types:

- Green areas: These are the main walkable areas. The thinner section represents sidewalks where the main activity will be held. There are also 2 bigger bright green areas these signify parks or open sections that are also walkable at no additional cost but don't have any reason the be walked on.
- Red areas: This represents a street. While it would be possible for the NPCs to traverse a street it would not be a common occurrence seeing as the cost for the street would be a lot higher than the sidewalks.
- Brown areas: These areas represent the spots where buildings will be or other obstacles that would make it impossible to walk on these parts.

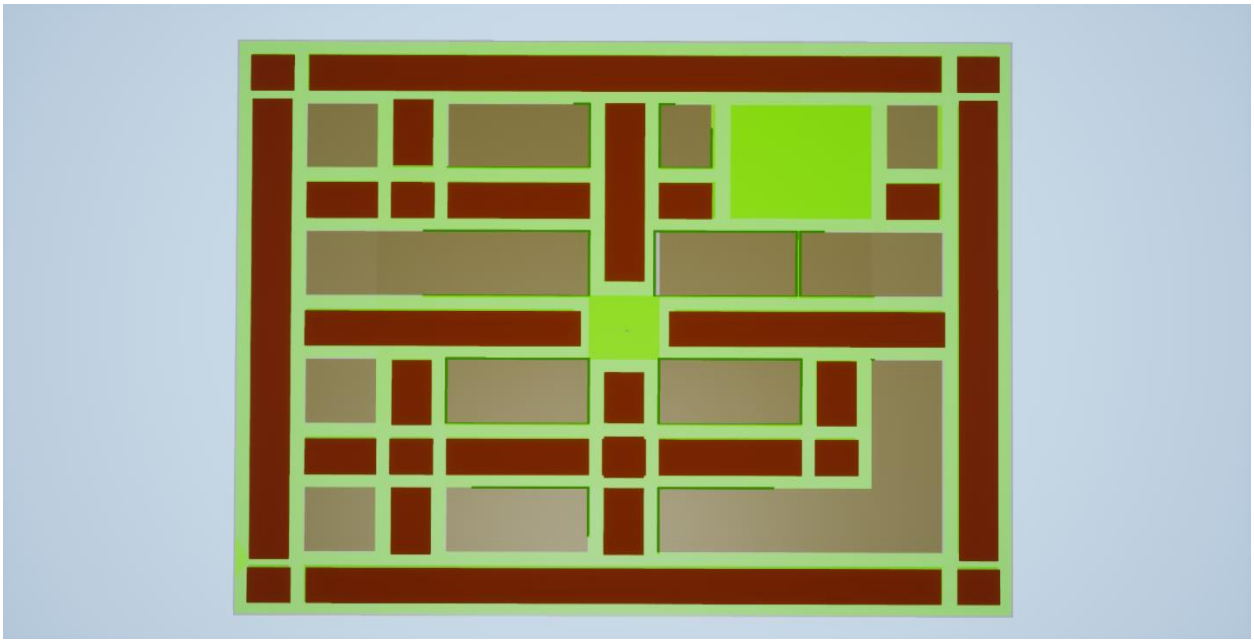

Figure 24: The nav mesh of the level

3.3. NAVIGATION QUERY FILTERS

Working in relation to the custom navigation areas are the navigation query filters. Like the previously discussed custom navigation areas they are native to the Unreal Engine 4 and are explained in [20]. These custom navigation query filters allow us to overwrite the base cost for any area class. This can be used by creating several different queries and depending on the traits we set the default filter class to the corresponding one.

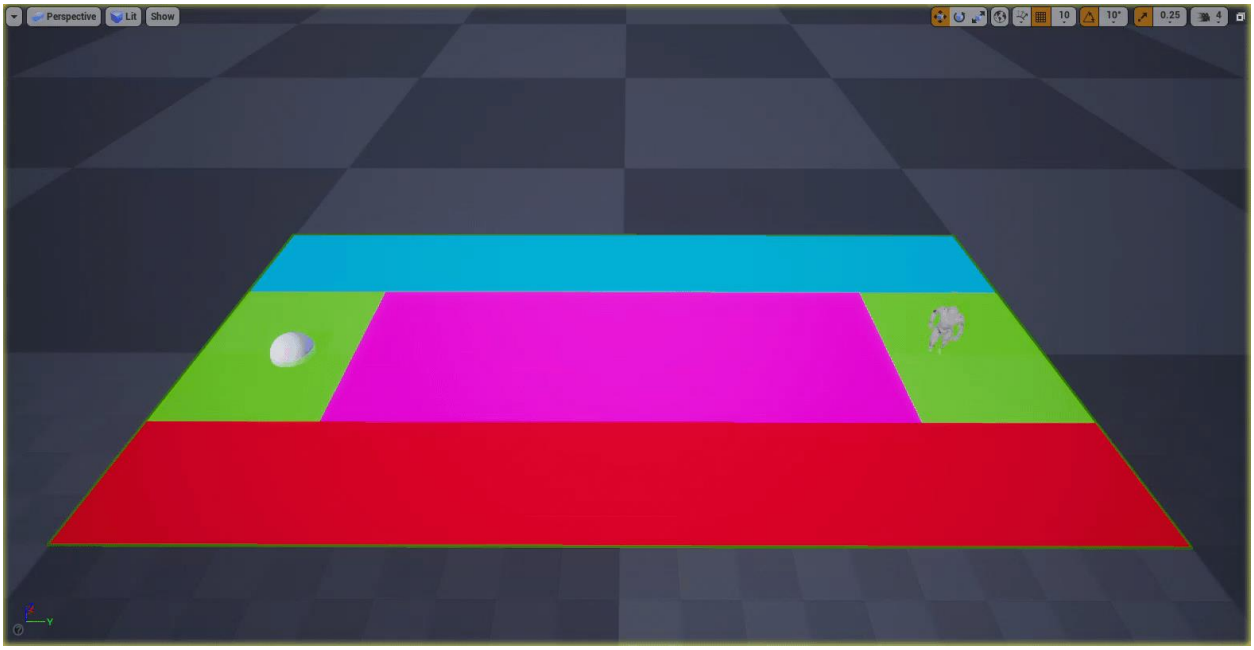

Figure 25: Example of behavior with the query making the red area lower in cost from [20]

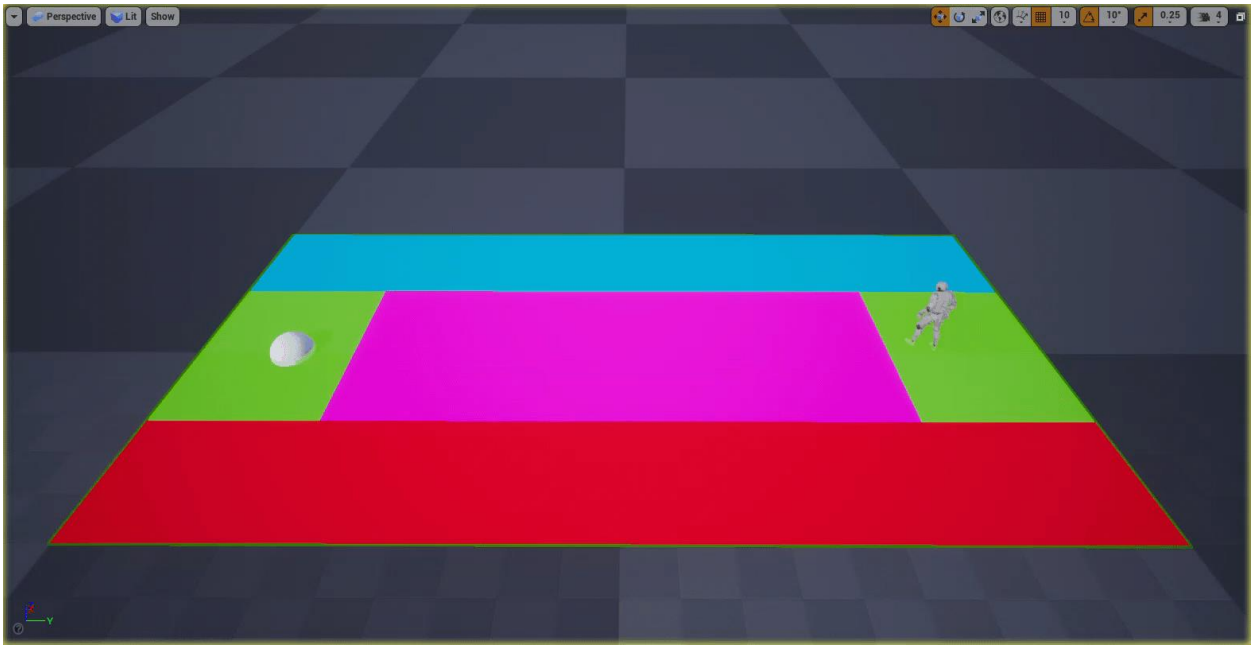

Figure 26: Example of behavior with the query making the blue area lower in cost from [20]

4. STEERING BEHAVIOURS

After that the desired path point has been found by the pathing logic described in the Pathing system chapter it is now time to calculate the steering behaviors. Blending of these behaviors will be explained with the traits.

4.1. SEEK

The first steering behavior that was created was the Seek behavior. The first step for this steering behavior would be to calculate the direction between the path point and the NPC. Once this directional vector has been calculated the next step included normalizing this vector to then multiply it again by a predefined float value that decided how far the distance of the seek would be. While normally the next step would be to calculate the steering by subtracting the current velocity from the desired velocity this project takes a slightly different approach. Instead, the seek direction vector (see Figure 27 the red vector) is added to the actor location. This operation results in a vector that defines end point of the seek. When the traits are implemented, this seek target point will be used for some blending logic but at this point this value would be used as the parameter for the MoveTo function that exist in Unreal Engine 4.

4.2. WANDER

At certain intervals a new wander direction (see Figure 27 blue vector) will also be calculated. It is only calculated at certain intervals to prevent the NPC from jittering too much. Another benefit of this is that it will not calculate this wander direction every frame so there is a slight performance gain. The wander steering behavior is accomplished by getting the desired direction from the previous seek behavior and then rotating it with a small value. This also a bit different from the normal approach that requires a circle to be made in front of the character. Then calculating a displacement vector with its origin being the same as the circle. Then rotating this displacement value with a random angle. Then with the displacement and velocity you could calculate the wander direction.

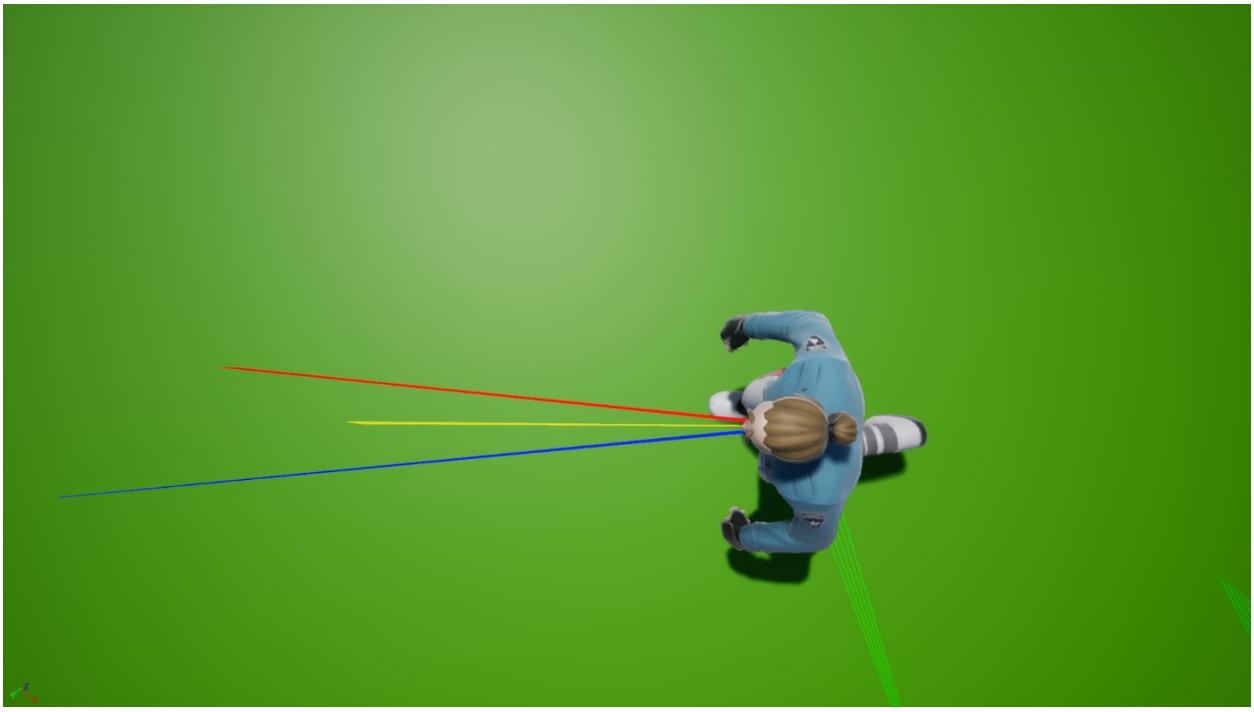

Figure 27: Visualization of the seek and wander behavior

4.3. COHESION AND SEPARATION

Cohesion and separation are in the same section because instead of calculating them separately this project approached the separation steering behavior a bit differently. Instead of calculating a genuine separation steering behavior the decision was made to fake this behavior by inverting the cohesion direction if the traits defined that this character would shy away from other characters.
As for the cohesion steering behavior this was calculated with the help of the custom crowd manager class. First off, some functionality was implemented that allowed the character to check if there are other character nearby and if so, it would return the center point between all the nearby characters. With this center point the final step would then be to calculate the cohesion direction (see Figure 27 green vector).

4.4. AVOIDANCE

The last steering behavior that will be discussed is the avoidance steering behavior. Instead of implementing this behavior this project will make use of the RVO avoidance that exists in the Unreal Engine 4.

4.4.1. RVO AVOIDANCE

RVO (Reciprocal Velocity Obstacles) avoidance is an existing avoidance behavior in Unreal Engine 4 [19]. Its effect is a bit like the previously discussed DetourController. Unfortunately, the RVO avoidance is not quite as good as the DetourController. One benefit of using RVO avoidance is that unlike the DetourController parameters the variables for the RVO avoidance can be changed per character and they are also able to be altered at runtime.

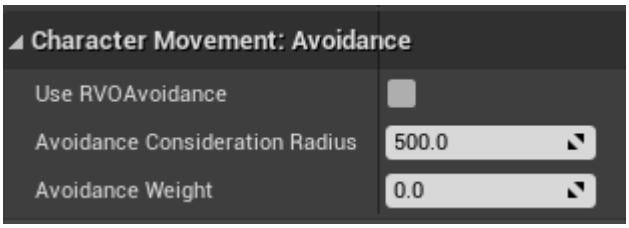

Figure 28: The default parameters for RVO avoidance

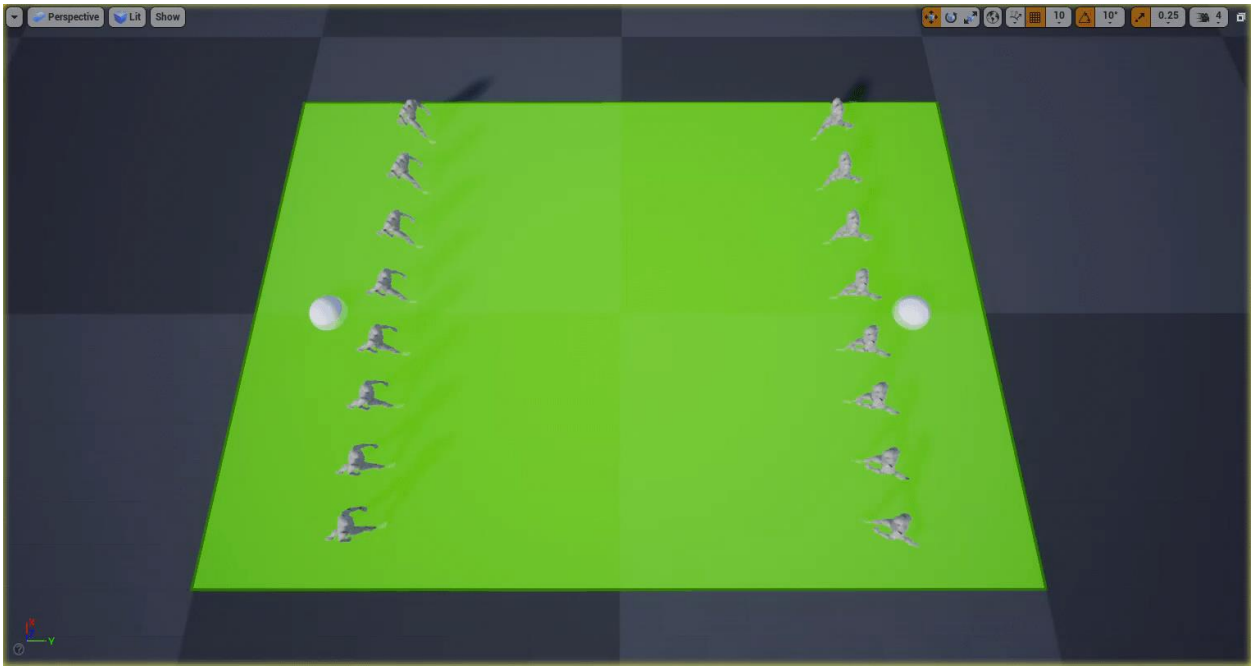

Figure 29: Demonstration of RVO avoidance from [19]

5. PERSONALITY TRAITS

5.1. CHOOSING THE PERSONALITY MODEL

The fist type of trait that will be implemented into the crowd simulation will be personality traits. These traits will have a significant impact on the behavior of each individual character on the other hand since personality traits do not have a visual indication, they will not have any impact on the visuals for the character.

Before implementing the personality traits, the decision needed to be made as for what personality model to use. After some thought, the decision was made to use the OCEAN model instead of the Eysenck model. This decision was made because of several reasons some of which being: It had been successfully used in other crowd simulation models such as [3] and [5], Secondly 5 is a good number of traits and unlike the traits in the Eysenck model the traits are separate.

The connections between traits (see Figure 30) are based on other implementations such as [3] and [5] and also on the definition of the traits [21] and what they would logically be linked to.

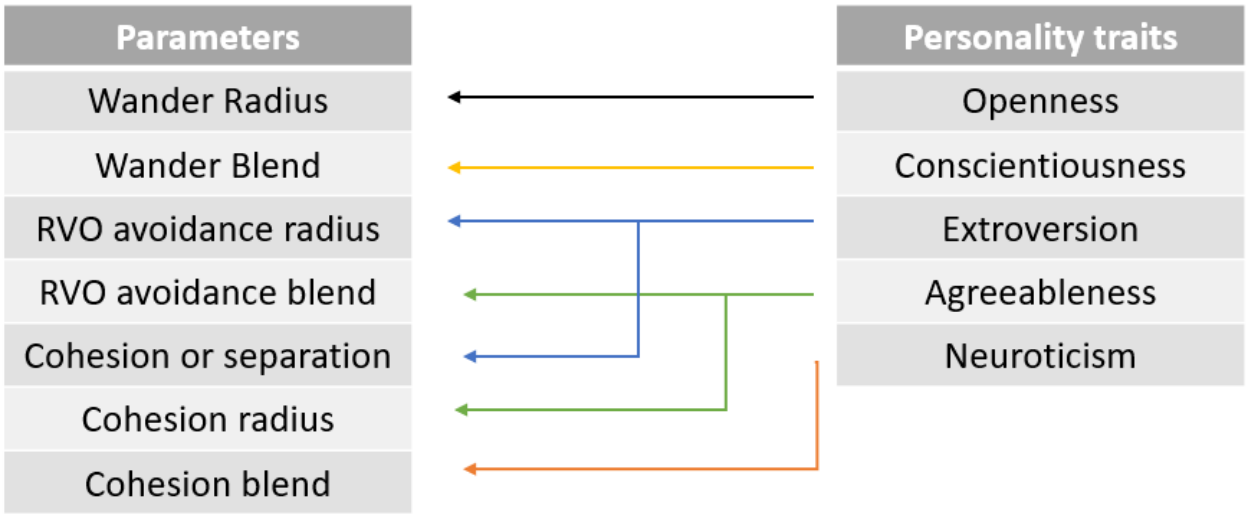

Figure 30: Demonstration of RVO avoidance

5.2. OPENNESS

The first trait of the OCEAN model is the openness trait. You can link this trait to the imagination/ideas/actions of the character. If a person would have a low openness score it would mean that they should prefer routine and have a more practical/conventional mindset. While having a high score for openness would mean that that person is curious and would have a wide range of interests.

Seeing that this trait relates to routine and curiosity the decision was made to link this trait to the wander behavior. More specifically this trait would define how big the wander range for the character would be. As mentioned before a low score would mean that the character prefers routine this would then correlate to a smaller wander radius meaning that the character sticks closer to the predefined path resulting in more routine behavior.

5.3. CONSCIENTIOUSNESS

The C in OCEAN model stands for Conscientiousness. This trait correlates to the amount of self-discipline or how goal driven that person would be. A low score for this specific trait would mean that this character is impulsive and disorganized while a high score for this trait would mean the opposite meaning that that person would be a more organized and dependable person.

This trait was again linked to the wander steering behavior. Unlike the openness trait this trait was linked to the blending of the different steering behaviors. For example, someone with a low conscientiousness score would be considered disorganized so therefore he would be more likely to wander aimlessly then someone with a high conscientiousness score. Therefore, when a character had a low score for this trait, he would put more weight on the wander behavior then if he had a high score for this.

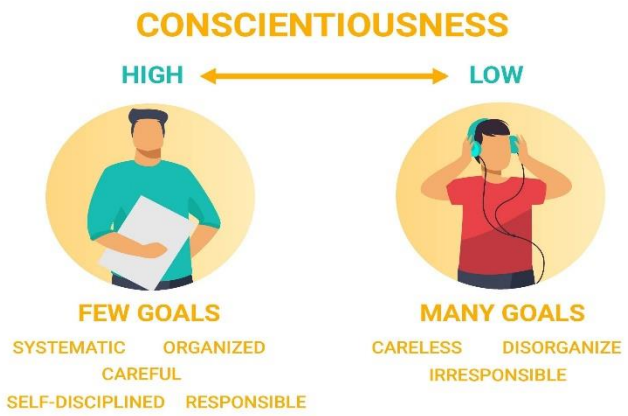

Figure 31: Spectrum of conscientiousness

5.4. EXTROVERSION

The next personality trait that was added into this project is the extroversion trait. Extroversion relates to how social a person is. Someone with a high extroversion score would be describable as outgoing and as having a warm personality meanwhile someone with a low score would be described as a reserved and quiet person.

The first connection that was made between extroversion and the underlying variables is the connection to the RVO radius. A low score would correlate to a bigger radius. This decision was made because someone that is reserved and withdrawn would typically take his environment into account to better avoid uncomfortable situations.

The other important connection between this trait and the behavior of the characters is in the cohesion/separation steering behavior. This trait will decide which one the character will use. Naturally someone with a high extroversion score would seek out social scenarios so there the steering behavior would be cohesion. On the other hand, a more introverted person would be more inclined to avoid other characters so in that case separation would be applied.

5.5. AGREEABLENESS

The second to last OCEAN model trait that was added to the characters is the agreeableness trait. This trait characterizes how cooperative and trustworthy a person is. A high scoring result for this trait would mean that that person is helpful and trusting of others while a person with a low score on this aspect would be less cooperative and be more suspicious of others.

Like the extroversion trait the agreeableness will influence the behavior of the character trough both the RVO avoidance and the cohesion/separation steering behavior.

A cooperative person would be more likely to go out of his way to avoid another person to not bother them. This would then result in the weight of the RVO avoidance being higher than if that character had a low agreeableness score.

Figure 32: Illustration of a person with high agreeableness letting someone else pass

The suspicious aspect would play a part in how big the radius for the cohesion/separation check would be. Someone that is suspicious of the people around him will pay more attention to the people than someone with a high agreeableness score. As a result, a low score would result in a bigger radius and vice versa.

5.6. NEUROTISCISM

To end the last personality trait that was implemented is the neuroticism trait. This trait can be defined as a person's tendency towards unstable emotions. This means that a person with a low score for this trait would be a calmer and even-tempered individual while a person with a high score on the other hand would be more anxious and unhappy.

This trait was the hardest to link to a particular behavior. After some further research and discussion, the decision was made to implement this trait into the cohesion/separation. Under the logic that an anxious and thus high scoring individual would be more prone to take his surroundings into account more weight would be put on either the cohesion or separation steering behavior.

6. ANTHROPOMETRIC TRAITS

6.1. CHOOSING THE ANTHROPOMETRIC TRAITS

The second type of traits are the anthropometric traits. However, unlike the personality traits that were discussed in the prior section the anthropometric traits do not have an established model of which traits are the most significant. This meant that an analysis had to be done to learn what traits would be the most influential in character behavior to make the decision on which traits to implement. This decision for the most part was based on the traits that they implemented in [2].

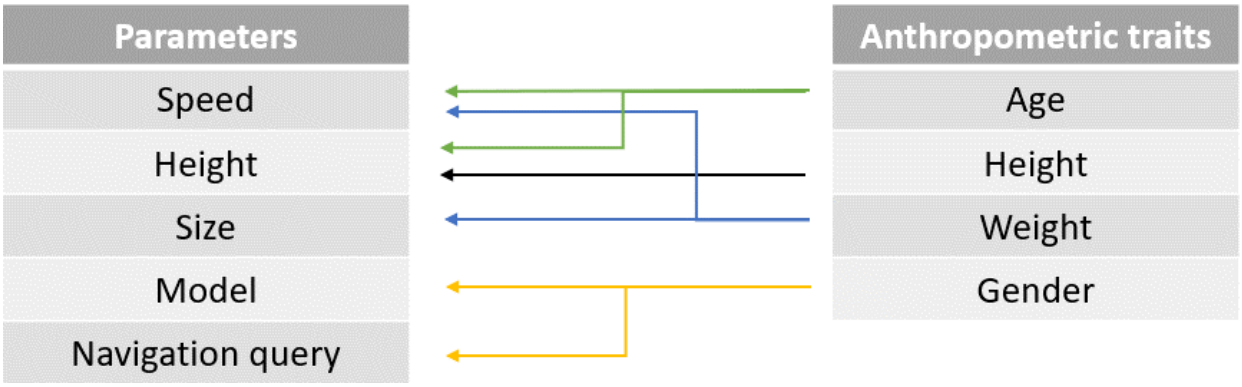

Figure 33: illustration of height over the years.

6.2. WEIGHT

The weight is another self-explanatory trait. Unlike the height trait however this trait influences 2 parameters. Namely the size of the character but also the speed since obese people tend to move a bit slower. The size of the character by scaling the pelvis bone depending on the value of the trait while the speed parameter gets an increase or decrease depending on the trait value.

Figure 34: illustration of the different type of body proportions

6.3. HEIGHT

The height trait is self-explanatory. While the age trait sets the base height for the character the height trait will take this base height and define a range depending on the value of the trait.

6.4. AGE

The first trait that was decided on was the age of the character. This trait would have influence on 2 parameters from the character. The first being the height. Logically children are much smaller than most adults and even the elderly. Typically, adults are taller than children and the elderly. In turn this would mean that the elderly population is somewhere in between the height of children and adults. So, as illustrated (see Figure 35) the average height over the years has a particular curve to it.

Figure 35: illustration of height over the years.

The other parameter that the age trait will influence will be the character speed parameter. Children and teenagers are often considered to be more energetic than adults and the elderly. This in turn often results to then walking around much faster than the older population. This resulted in the base speed for a character with the age trait being in the lower values will be faster than the others. With the age trait continuing to rise the base speed will decrease respectively.

6.5. GENDER

The final anthropometric that was implemented was the gender of the character. This trait will have a significant visual impact seeing it will change the model of the character (see Figure 36 and 37).

Figure 36: Example of some male models

Figure 37: Example of some female models

Another big part of what this gender trait will influence will be the default navigation query filters of each character. In [18] they discuss what differences gender makes in relation to pedestrian rules. In that paper they concluded that women are more compliant with traffic rules than men. As a result, we decided to make the female NPCs less prone to jaywalk or walk in the middle of the street in general. This effect has been created by making the street area volume cost higher for the female navigation query filter.

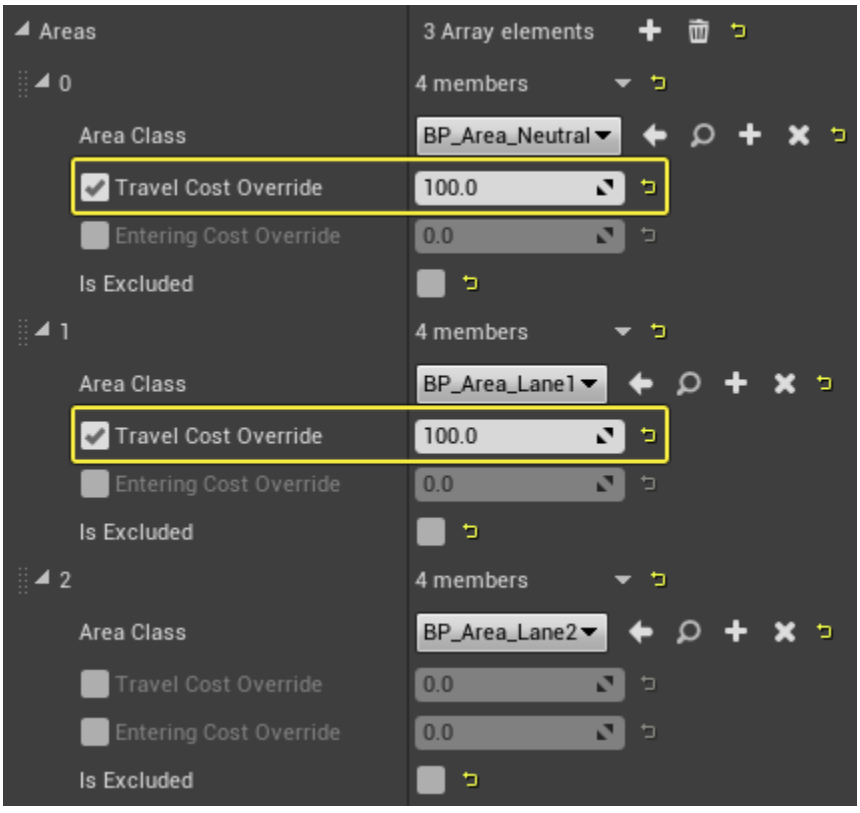

Figure 38: Example of a custom navigation query in [20]

CONCLUSION

When attempting to create a heterogenous crowd a viable approach could be to use both anthropometric and personality. These traits could then be linked to underlying parameters that could influence a plethora of aspects on the individual characters. This project focused on visual aspects such as height and model variety and behavioral aspects such as the blending and strength of a variety of steering behaviors.

When looking into which frameworks to use the Unreal Engine 4 came out to be the most suitable for this type of project. While the UCROWDMANAGER didn't work out as the base class for the crowd manager that was implemented it might be interesting to implement it into more complicated crowd simulations. If so, there would be a need to look at the source code of the interface class that is needed to add characters to this UCROWDMANAGER class otherwise with only the documentation it is not really made clear what this interface class entails.

When looking at the difference in variety that has been caused by the different type of traits it appears that the more visuals results that are mainly influenced by the anthropometric traits are the most helpful in diversifying the crowd. While the personality traits do also help to diversify the crowd, this is much more subtle and will likely go unnoticed unless close attention is paid to a smaller group of NPCs. Another issue with the personality traits is that the steering behaviors don't cause that much of a visual difference.

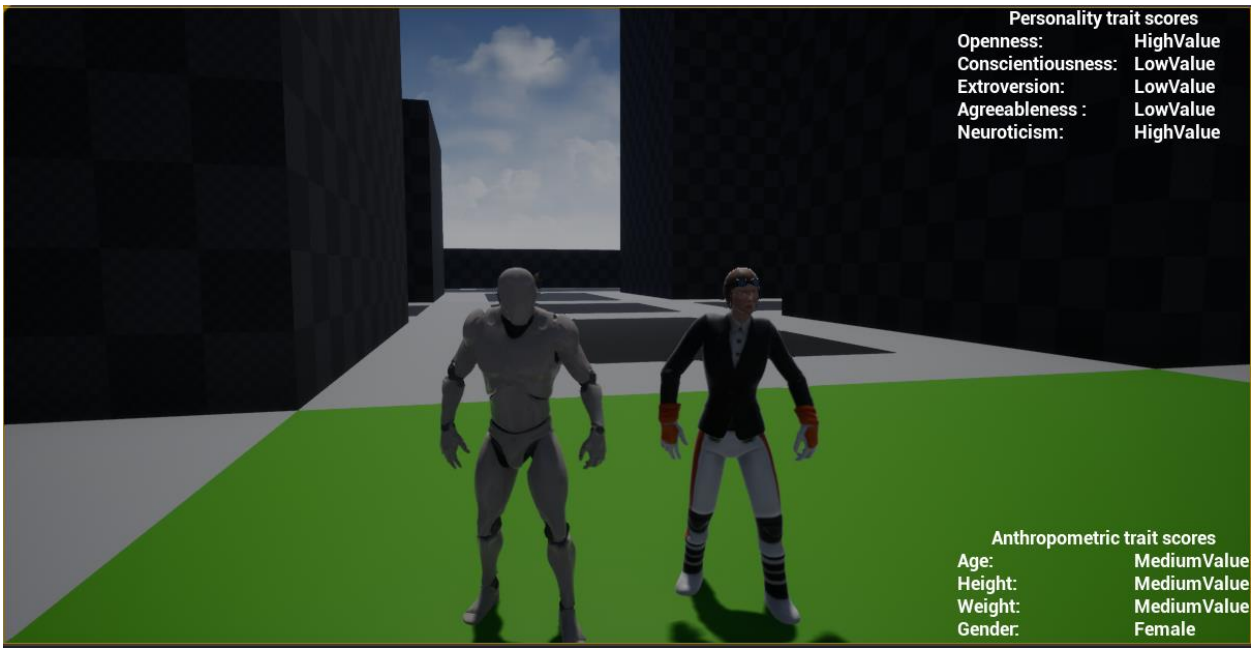

Figure 39: Example of a pedestrian that is selected so you can see its traits.

FUTURE WORK

There is still a lot of room for improvement/alterations in this model. For the anthropometric traits there are still a lot of traits that could be implemented to change the visuals of the model or some of the base parameters of the character.

For the personality traits there are different approaches that could be made to try to improve their effect on the crowd simulation. One of these approaches could be to try other personality models or add even more traits to see if they could make some other interesting connections to parameters that could help diversify the crowd. Another approach could be to link the traits to other more advanced behaviors or even to increase the strength of these behaviors seeing as the strength for a lot of the steering behaviors in this crowd simulation was limited for the sake of realism.

Another big aspect of this crowd simulation model that can be improved upon is the real time functionality. This meaning that as of now this crowd simulation model only supports the traits being randomly assigned to each individual character. The number of characters that spawn is also a predefined amount that can be changed for each individual spawner.

This means that it would adding functionality to spawn or destroy characters at runtime would be a very beneficial function for this model. You could even take this function further as to choose the traits for each character that you spawn or randomly generate them. This way you have a lot more control over the scale of the simulation as well as what specific types of traits you would like to simulate in a specific instance.

It could also be interesting to try to add functionality to change the traits of a character at runtime. This could work by for example when inspecting a certain character which allows you to see that character's traits (see Figure 39). While hovering over a specific trait a menu of some sorts would appear with the different values for that trait selectable. This additional feature would make the differences between the trait values easier to be tested and would again give more control over the entire crowd simulation.

REFERENCES

[1] D. Thalmann, H. Grillon, J. Maim, and B. Yersin, "Challenges in crowd simulation," in *2009 International Conference on CyberWorlds, CW '09*, 2009, pp. 1–12. doi: 10.1109/CW.2009.23.

[2] A. Ochoa *et al.*, "Humanitarian logistics and cultural diversity within crowd simulation," vol. 21, no. 1, p. 20, 2017, [Online]. Available: https://hal.archives-ouvertes.fr/hal-01517161

[3] Z. Xue, Q. Dong, X. Fan, Q. Jin, H. Jian, and J. Liu, "Fuzzy logic-based model that incorporates personality traits for heterogeneous pedestrians," *Symmetry*, vol. 9, no. 10, 2017, doi: 10.3390/sym9100239.

[4] S. J. Guy, S. Kim, M. C. Lin, and D. Manocha, "Simulating Heterogeneous Crowd Behaviors Using Personality Trait Theory." [Online]. Available: http://gamma.cs.unc.edu/personality/

[5] F. Durupinar, J. Allbeck, N. Pelechano, N. Badler, and F. Durupınar, "Virtual Reality View project Animating Facial Expressions View project Creating Crowd Variation with the OCEAN Personality Model (Short Paper)," 2008. [Online]. Available: https://www.researchgate.net/publication/237502245

[6] O. Szymanezyk, P. Dickinson, and T. Duckett, "From Individual Characters to Large Crowds: Augmenting the Believability of Open-World Games through Exploring Social Emotion in Pedestrian Groups," 2011. [Online]. Available: https://www.researchgate.net/publication/257821309

[7] "OpenSteer Documentation." http://opensteer.sourceforge.net/doc.html (accessed Dec. 16, 2021).

[8] R. Santos, G. Marreiros, C. Ramos, J. Neves, and J. Bulas-Cruz, "Personality, Emotion, and Mood in Agent-Based Group Decision Making," 2011. [Online]. Available: www.computer.org/intelligent

[9] "The Open Agent Architecture Documentation." http://www.ai.sri.com/~oaa/distrib.html (accessed Dec. 16, 2021).

[10] "UCrowdManager documentation." https://docs.unrealengine.com/4.27/en-US/API/Runtime/AIModule/Navigation/UCrowdManager/ (accessed Nov. 11, 2021).

[11] Ph. D. Christopher Dragert, "Census: The Systemic Backbone Behind Play As Anyone in 'Watch Dogs: Legion.'" Accessed: Oct. 14, 2021. [Online]. Available: https://www.gdcvault.com/play/1027018/Census-The-Systemic-Backbone-Behind

[12] S. Curtis, S. J. Guy, B. Zafar, and D. Manocha, "Virtual Tawaf: A Case Study in Simulating the Behavior of Dense, Heterogeneous Crowds." [Online]. Available: http://gamma.cs.unc.edu/LARGE

[13] Roxanne Blouin-Payer, "Helping It All Emerge Roxanne Blouin-Payer Game Designer, Ubisoft." Accessed: Oct. 14, 2021. [Online]. Available: https://www.gdcvault.com/play/1024426/Helping-It-All-Emerge-Managing

[14] "Kasper Fauerby Lead Programmer at IO Interactive Crowds in Hitman:Absolution."

[15] O. Szymanezyk, O. Szymanezyk, and G. Cielniak, "Group emotion modelling and the use of middleware for virtual crowds in video-games," 2010. [Online]. Available: https://www.researchgate.net/publication/257821531

[16] "Crowd Manager Avoidance config Forum." https://answers.unrealengine.com/questions/212408/crowd-manager-avoidance-config.html (accessed Jan. 19, 2022).

[17] CodeLikeMe, “Unreal pedestrian system.” https://www.youtube.com/watch?v=E6pd7s72_Zk&list=PLNTm9yU0zou7NlLkMVAMFZWD_tXIPkb0n&index (accessed Jan. 12, 2022).

[18] A. Tom and M.-A. Granié, “Gender Differences in Pedestrian Rule Compliance and Visual Search at Signalized and Unsignalized Crossroads”, doi: 10.1016/j.aap.2011.04.012ï.

[19] “Using Avoidance with the Navigation System.” https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/ArtificialIntelligence/NavigationSystem/Avoidance/ (accessed Jan. 24, 2022).

[20] “Custom Navigation Areas and Query Filters.” https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/ArtificialIntelligence/NavigationSystem/CustomNavigationAreasAndQueryFiltersLanding/CustomNavigationAreasAndQueryFilters/ (accessed Jan. 24, 2022).

[21] “Big Five personality traits.” https://en.wikipedia.org/wiki/Big_Five_personality_traits (accessed Jan. 24, 2022).

APPENDICES

Figure 19 behavior without avoidance:
https://docs.unrealengine.com/4.27/Images/InteractiveExperiences/ArtificialIntelligence/NavigationSystem/Avoidance/Avoid-NPC-Walk-NoAvoidance.gif

Figure 20 behavior with DetourController:
https://docs.unrealengine.com/4.27/Images/InteractiveExperiences/ArtificialIntelligence/NavigationSystem/Avoidance/Avoid-NPC-Walk-Detour.gif

Figure 25 Navigation query filter example 1:
https://docs.unrealengine.com/4.27/Images/InteractiveExperiences/ArtificialIntelligence/NavigationSystem/CustomNavigationAreasAndQueryFiltersLanding/CustomNavigationAreasAndQueryFilters/Custom-NPC-Walk2.gif

Figure 26 Navigation query filter example 2:
https://docs.unrealengine.com/4.27/Images/InteractiveExperiences/ArtificialIntelligence/NavigationSystem/CustomNavigationAreasAndQueryFiltersLanding/CustomNavigationAreasAndQueryFilters/Custom-NPC-Walk3.gif

Figure 29 behavior with RVO avoidance:
https://docs.unrealengine.com/4.27/Images/InteractiveExperiences/ArtificialIntelligence/NavigationSystem/Avoidance/Avoid-NPC-Walk-RVO.gif