Karelia-University of Applied Science
Bachelor of Business Information Technology (UAS)

Sensory Simulation in Video-game Characters Using AI Perception

Nanuli Mäntylä

Bachelor's thesis, November 2024
Business Information Technology
www.karelia.fi

Karelia AMMATTIKORKEAKOULU

BACHELOR'S THESIS
November 2024
Bachelor of Business Information Technology

Tikkarinne 9
80200 JOENSUU
+358 13 260 600

Author
Nanuli Mäntylä

Title
Sensory Simulation in Videogame Characters Using AI Perception

Abstract

This thesis examines AI perception in modern video games and functionality provided by perception components of Unreal Engine 5. AI Perception systems enhance immersion by enabling non playable characters to detect players or environmental events through sensory awareness, such as sight, hearing and touch.

The thesis involved building a game demo in Unreal Engine 5 to test and implement the perception system in practice. In the demo, AI Sight, Hearing and Touch senses were tested with an AI character. The senses were chosen due to their wide applicability in the gaming industry and accessibility through visual scripting system of Unreal Engine 5. Each sense was configured and tested with various parameters and their effectiveness was evaluated.

The results suggest that Unreal Engine 5 offers a user-friendly and adaptable AI perception system suitable for developers at various experience levels. Even beginners can build a rudimentary perception system with the visual scripting tools. However, achieving a more nuanced perception system requires further refinement beyond visual scripting, such as the use of C++ coding.

Language of publication:	Pages 40
English	Appendices 2
	Pages of Appendices 4

Keywords
Unreal Engine, game development, artificial intelligence

Karelia
UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ
Marraskuu 2024
Tradenomi, Tietojenkäsittely

Tikkarinne 9
80200 JOENSUU
FINLAND
+ 358 13 260 600

Tekijä
Nanuli Mäntylä

Nimike
Aistien simuloiminen tekoälyn avulla videopeleissä

Tiivistelmä

Opinnäytetyö käsittelee videopelien tekoälyhavainnointijärjestelmiä keskittyen hahmojen havainnointiin aistijärjestelmien, kuten näön, kuulon ja tuntoaistin avulla. Työssä tarkasteltiin havainnointijärjestelmien kehitystä videopeleissä, Unreal Engine 5:n tarjoamaa tekoälyhavainnointijärjestelmää sekä havainnointijärjestelmän käytännön toteutusta.

Opinnäytetyön tueksi toteutettiin Unreal Engine 5:lla yksinkertainen peliprojekti, jossa luotiin aistihavainnointijärjestelmää hyödyntävä tekoälyhahmo. Peliprojekti osoittaa, että käyttäjällä ei tarvitse olla vuosien kokemusta Unreal Enginen käytöstä, vaan yksinkertaisen aistihavainnointijärjestelmän saa toteutettua Unreal Engine 5:n tarjoamilla valmiilla komponenteilla.

Kieli
englanti

Sivuja 40
Liitteitä 2
Liitesivumäärä 4

Asiasanat
pelikehitys, Unreal Engine, tekoäly

Contents

1 Glossary

Term	Definition
Academic AI	See “AI”
Actor	An Actor is a base class for an object that can be based in a level in Unreal Engine. For example, the player start location, a static mesh or a NPC. (Epic Games 2024)
AI	Abbreviation for artificial intelligence: computer systems that have some of the qualities that the human brain has, such as the ability to understand and produce language, recognize, or create pictures, solve problems, and learn; (Cambridge University Press & Assessment 2023)
Blueprint coding	Visual scripting system in Unreal Engine, that allows the user to create gameplay elements using node-based interface with Unreal Editor. (Epic Games 2024)
Game AI	System that tries to give appearance of intelligence, creating particular experience to viewer (Dill 2014)
Neutral NPC	Supportive non-playable character, for example characters that can give missions, sell things or “fill the area”. (See also “NPC”).
NPC	Abbreviation for non-playable character: a character in a computer game that is not controlled by someone playing the game; (Cambridge University Press & Assessment 2023)
Pawn	Child Class of Actor, that can be controlled by player or AI. (See also “Actor”) (Epic Games 2024)

2 Introduction

This thesis explores the most commonly used sensory systems in video games, how different AI perception senses can be implemented and utilized in Unreal Engine 5 and practical application of these systems in a game demo, created as a part of the thesis.

The thesis consists of four main sections. In the first section, AI Perception is explained in more detail, with a short introduction to history and modern uses and how AI Perception can be used in Unreal Engine 5. In the following sections, three different AI Perception Senses will be analysed in more detail, with examples on how they are used in games, what kind of components Unreal Engine offers and finally, how the sense was implemented in the practical part.

2.1 Unreal Engine 5

Unreal Engine 5, developed by Epic Games and released in 2022, is the latest version of the widely used game engine, with its most recent stable release, version 5.4, in April 2024. Unreal Engine 5 is popular among AAA game developers, such as Konami, Ubisoft, and Naughty Dog, and is also used by indie game developers and the movie industry. (Epic Games, 2024).

Unreal Engine offers a wide variety of AI systems to give characters artificial intelligence, including behaviour trees, a navigation system, neural networking, and an environment query system. This thesis, however, focuses specifically on AI Perception, which enables sensory awareness for AI characters (Epic Games, 2024).

Aside from Unreal Engine, other widely used game engines include Unity and Godot. However, neither Unity nor Godot provides built-in perception tools. Additionally, Godot removed visual scripting from its core engine in version 4.3, requiring users to have more extensive programming knowledge, particularly in languages such as C++ or C#.

2.2 Game demo

The Practical part of this thesis is a game demo, where various AI Perception Components were tested and demonstrated. The game demo was built using

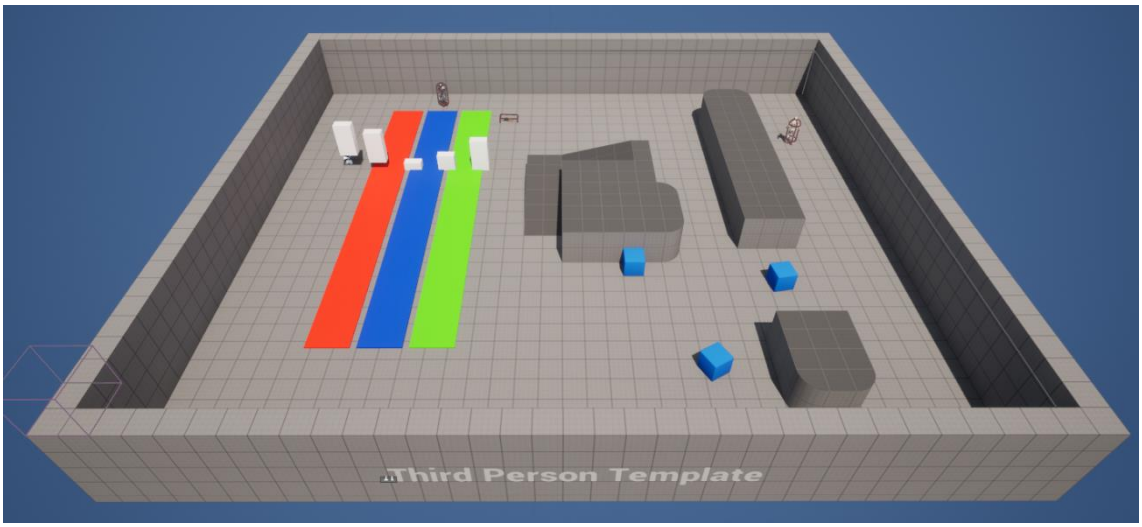

Figure 1. Screenshot of the demo level. (Screenshot by Nanuli Mäntylä)

the “Third Person Project”- template provided by Unreal Engine 5. The template includes a small level (see Figure 1) and a third-person player character setup, allowing the focus to be on implementing the AI Characters.

The Third Person Project template already includes platform meshes in the level, which were used to test the sight perception. Hearing perception was tested using additional meshes with new physical materials, which will be explained in more detail later in this thesis (see chapter, 5.3.2: Setting up the player character for more details).

In addition to the AI character with AI Perception, another AI character was added to test the Sight perception on different characters. This second AI character does not have an AI Perception component; it only has a simple behaviour tree, with a roaming sequence, enabling it to move around the level.

To make this thesis more accessible to beginners with Unreal Engine, only visual scripting or “Blueprint coding” was used in the project. AI Perception

components can also be accessed using C++-coding, explained in more detail in the engine's C++ API references and documentation.

3 AI Perception

This chapter gives a short overview of AI Perception, what the term actually means, short history and some examples of how it is used in modern video games. Furthermore, there is a brief introduction to all the AI Perception components in Unreal Engine 5.

3.1 Overview and short history of perception in videogames

In essence, game AI perception refers to how AI characters detect players and each other. Before advanced AI was introduced in video games, enemies "detected" players primarily by their proximity, when they entered certain radius or line of sight. In games like Mario Bros. (1983) or Castlevania (1986), enemies followed pre-determined routes and if the player touched them, they would simply die or faint.

Video game AI differs significantly from conventional AI (referred to hereafter as "academic ai"), as it does not aim to mimic a real intelligence. Instead, the main goal of video game AI is to make the game engaging by making the characters appear intelligent and realistic, providing players with an immersive and enjoyable experience. (Dill, 2014).

The earliest example of a simple AI in video games is Pac-Man (1980), where the enemies moved based on the player's location, either attempting to "kill" or block the player's path (Pittman 2009). A year later the first game to incorporate sight detection, Castle Wolfenstein (1981), was released. The player played as an allied soldier, who had to escape from enemy soldiers, by either eliminating them or by stealth. (Alkaisy, 2011).

Although neither of these games used what is now considered modern game AI or AI perception, they were the pioneering in featuring enemies that moved in response to players movement and actions rather than following a fixed route.

In the late 1990's, several games with sense simulation systems were released. GoldenEye (1997) was one of the first to incorporate such system: sounds of gunfire alerted distant enemies and NPCs could also "see" their colleagues and react if they were killed. Similar systems also appeared in games like Metal Gear Solid (1998), Thief: The Dark Project (1998) and Half-Life (1998), where players had to use stealth to avoid detection and NPCs displayed awareness of each other's presence.

As the game AI improved, NPCs' reactions to the player's actions became more varied. For example, if a player is seen stealing, previously friendly or neutral NPCs might become hostile, especially if they are the ones being stolen from. Similarly, if the player suddenly starts running and bumping into characters, NPCs might respond angrily or appear offended.

While sense simulation or perception is only one aspect creating believable AI characters, a well-functioning system can greatly enhance the variety and immersion of the gaming experience. These perception systems not only add depth to NPC behaviour but also create a more dynamic and unpredictable environment, making each player's experience unique.

3.2 AI Perception in Unreal Engine

AI Perception is one of the systems in Unreal Engine that enables AI characters to gather sensory date about the level and the actors within it. For example, it can help AI characters detect the direction of noises or see objects or other actors. (Epic Games, 2024)

Unreal Engine provides six distinct AI Senses that can be added to AI Characters:

- AI Damage allows the AI character to react to damage events, such as “Event Any Damage” and “Event Point Damage”.
- AI Hearing enables the AI character to detect sounds generated by the “Report Noise Event”. AI Hearing will be explored in greater detail in this thesis and applied in the practical project. (See chapter 5, Hearing sense, for more details).
- AI Prediction allows the AI character to calculate a predicted location for the stimuli source, such as where the previously detected character will likely walk.
- AI Sight enables the AI to see other Actors in the level. AI Sight will also be covered in greater detail and used in the project. (See chapter 4, Sight sense, for more details).
- AI Team notifies the AI character when another character of the same team is nearby.
- AI Touch allows the AI character to detect when something bumps into it. AI Touch will also be discussed in more detail in this thesis. (See chapter 6, Touch sense, for more details). (Epic Games, 2024).

This thesis only explores sight, hearing and touch perception in greater depth. These senses were chosen, as they are most commonly used in video games, and can be implemented using only visual scripting.

AI Damage is similar to AI touch. The main difference is that it uses Damage event instead of Touch event in the code. The Damage event also includes an output for damage type, which differentiates between damage types such as blunt or piercing hits. This information is useful for scenarios where different types of damage affect the character differently.

AI Prediction can be useful when an AI character needs to follow a player or predict their movements. However, since the AI character in this project is stationary and does not require complex following behaviours, AI Prediction was not further explored in this thesis.

AI Team Sense relies on Affiliations, which are currently only available in C++. Since one of the project goals was to use visual scripting exclusively, AI Team Sense was not included in the scope of this thesis.

The AI perception can be added to any AI Character by adding an AIPerception Component to the character's AI controller (Appendix 1). This component can be used to enable and disable what senses to use, configures the data to be collected and defines which Actors will be detected. The AIPerception Component acts as a stimuli listener, gathering data from registered Stimuli Sources. (Epic Games, 2024).

In the Details panel of the AIPerception Component, all required Senses (for example sight or hearing) can be added to the Character (Appendix 1). These sense configurations are explained in more detail later in this thesis. One sense can also be designated as the dominant sense, which will be used by the AI Character to determine the location or detected Actors. (Epic Games 2024).

The Perception Events section allows defining what occurs when the AI Perception system is updated, or AI Perception Component is activated or deactivated. The system provides the following five events:

- "On Perception Updated" event fires when the Perception System receives an update and returns an array of Actors that caused the update.
- "On Target Perception Info Updated" event notifies all bound Actors when perception info is updated for a specific target. This notification broadcasts for any received stimulus or state change, depending on the stimulus configuration.
- "On Target Perception Updated" event fires when the perception system receives an update, returning the Actor that triggered the update and the stimulus responsible for the update. The stimulus structure can be broken to get additional information of the stimulus, such as age, location and strength. This event is particularly useful when the AI character has multiple senses activate, as it enables differentiated responses, based on the updated stimulus. For example, if the character sees the player, it

may start following them; if it only hears the player, it might begin searching the area for the sound instigator.

- “On Component Activated” event is fired when AI Perception Component is activated.
- “On Component Deactivated” event is fired when AI Perception Component is deactivated.

To register an Actor as a Stimuli Source, the Actor must have an AIPerception Stimuli Source component added to its Blueprint (Appendix 2). In addition, all necessary senses should be added in the AIPerception Stimuli Source Details panel. (Epic Games, 2024)

The AI Debugging tool is helpful for diagnosing and troubleshooting the AIPerception system. During runtime, it can be accessed by pressing the apostrophe key (‘). To toggle the AIPerception debugging feature, the user must press number “4” key on the NumPad. The debugging interface displays all Senses integrated into the AI Perception Component. Depending on the selected senses, the debugger provides detailed information such as the last known location of stimuli, age of stimuli and the range of each sense. (Epic Games, 2024).

4 Sight sense

This chapter focuses on the sight perception. The first section provides a brief overview of the history of sight sensing in video games and offers examples of its use in modern video games. The second section introduces Unreal Engine’s Sight perception system and details the variables that control the sight range and other parameters. The Third section explains how sight perception was applied in the demo project, including observed limitations.

4.1 Sight in video games

This chapter explores sight perception, one of the most common senses used in game AI. To understand its development and significance, sight perceptions evolving was studied, from early implementations to its modern-day applications in some popular video game titles.

Sight sensing is one of the first sensing mechanisms used in video games. The first known example appears in Dungeon, a game created by Don Daglow for PDP-10 in the mid-1970's. (Daglow 1988, p.18).

In its simplest form, sight detection occurs when a player enters a specific range in front of an NPC. Typically, sight detection happens in a conical shape, sometimes divided into two or more fields of accuracy, trying to mimic humans' field of vision, as illustrated in the top right of Figure 2. Depending on the game system, this field of vision can be obstructed by objects, or a certain percentage of the players' body may need to be visible before they are detected.

Figure 2. Screenshot from Metal Gear Solid, enemies vision shown with cones. (Konami 1998)

In Assasin's Creed (2007), for instance, players are often required to use stealth to tackle their enemies. However, if a player continuously keeps crouching, even neutral NPCs may become wary, potentially attacking the player or alerting the guards. If the guards are alerted, the player can attempt to hide again in places such as piles of hay or by blending in with crowds. However, if the enemies see the player entering a hiding spot, the enemies will continue their attack.

Some games encourage the player to use the environmental factors to avoid detection. For example, in Thief: The Dark Project (2018), players can try to move only in the shadows; in Hitman 3 (2021), players can disguise themselves to blend in with NPCs; and in Metal Gear Solid V: Phantom Pain (2015), player can enter certain areas undetected by moving only at night.

Sight perception can also alert NPCs to environmental changes. For example, in Assasin's Creed (2007), enemies become suspicious if they discover bodies left by the player, prompting them to search for the culprit. In Metal Gear Solid 2: Sons of Liberty (2001), NPCs can even follow player's tracks in snow or blood splatters.

A well-developed sight perception system can add significant realism, making the NPC appear more human and subtly guiding players' choices in gameplay. However, sight perception has its limitations and can sometimes be easily bypassed. For instance, in The Elder Scrolls V: Skyrim (2011), players can avoid

being seen while stealing, by placing a bucket over an AI character's head (Figure 3).

Figure 3. Screenshot from Elder Scrolls V: Skyrim, showing NPC with a bucket on their head. (Bethesda 2011)

In real life, a person would likely remove the bucket and react to the thief, but unless this behaviour is explicitly scripted, the NPC will not respond to their obscured vision.

4.2 Sight in Unreal Engine

While sight perception in early video games was relatively simple, modern game engines, such as Unreal Engine 5, offer more complex systems. In this chapter sight perception of Unreal Engine 5 is explained in more detail.

AISight configuration allows AI characters to "see" Actors within the game environment. When an Actor enters the Sight Radius, the AI perception system updates, prompting the NPC to respond — for example, by attacking the actor. (Epic Games 2024).

To use AISight for a Character, the AIPerception Component must be added to the Characters' AIController. Then, on the Details panel of the AIPerception

component, AI Sight config should be added to the Senses Config array (See Appendix 1).

In the AISight Config details panel, the following variables can be adjusted (all distance/length measurements are in centimetres by default, though this can be changed in project settings):

- “Implementation” is an Enum, that specifies which sense class to use (default is AISense_Sight). This can be customized if the user wants to use a custom class for sight detection.
- “Sight radius” is a float, that defines the maximum distance within which the Actors with AISense_Sight stimuli source can be detected.
- “Lose Sight Radius” is a float, that sets the distance at which the AI Character loses sight of the Actor. This value is typically larger than the “Sight radius”, meaning the AI Character can continue to see the target over a greater distance if it was already detected.
- “Peripheral Vision Half Angle Degrees” is a float ranging from 0.0 to 180.0, that specifies the angle of vision (in degrees). This angle is calculated from the AI character’s forward vector.
- “Detection by Affiliation” is an Enum, that determines if enemies, neutrals or friendlies are detected. This setting is configurable in C++ only. For Blueprints, tags can be used to filter out Actor types.
- “Auto Success Range” is a float, that defines an automatic detection range based on the Last Seen Location of a previously detected target. If the variable is set above zero, the AI character will always detect the target withing this range.
- “Point of View Backward Offset” is a float, that sets how far back the AI characters “eyes” are from its actual position, helping improve situational awareness behind the character or withing its periphery.
- “Near Clipping Radius” is a float, that defines the minimum distance in front of the AI where the objects are detected. When combined with the Point of View, AI Character’s sight sense can be fine-tuned, especially enhancing close-up awareness, like in real humans and animals.
- “Debug Color” specifies the colour of debug lines in the AI Debugger tool.

- “Max Age” is a float, that defines the duration (in seconds) for which stimuli detected by the Sight Sense are remembered. A value of 0 means the stimuli are never forgotten.
- “Starts Enabled” is a Boolean, that determines whether the sense is enabled automatically or requires manual enabling/disabling. (Epic Games 2024)

For an Actor to register as AISight stimulus source, an AIPerception Stimuli Source Component must be added to its Blueprint. In the component’s Details panel, AISense_Sight should be included in the “Register as Source for Senses” array. (Epic Games 2024).

If different reactions are needed for various Actors, Tags can be used to categorize characters. For instance, characters from the same faction can be labelled with a “friend” tag, while enemies may be tagged as “enemy”. These tags can then be used in the AI Characters Blackboard and Behaviour Tree for customized responses.

4.3 Sight in the project

In the project, two of the most common applications of sight perception were tested: detecting the player character and detecting another type of NPC. Additionally, specific limitations of the Unreal Engine 5’s sight perception system were also tested, such as the minimum visible portion of the player required for detection and whether certain body parts (like head and feet) affect detection.

4.3.1 Setting up the AI character

To set up sight for an AI Character, AIPerception component was added in the characters AIController, and the AI Sight config added to the Senses Config array. Given the compact test level, the sight radius was set to 2000cm and the lose sight radius to 2500cm, with the peripheral vision half-angle at 50 degrees. Additionally, to improve the peripheral vision, the backward offset was set to 200cm and clipping radius to 200cm. This configuration creates a conical field

of vision with a flatter top, as illustrated by green lines in Figure 4.

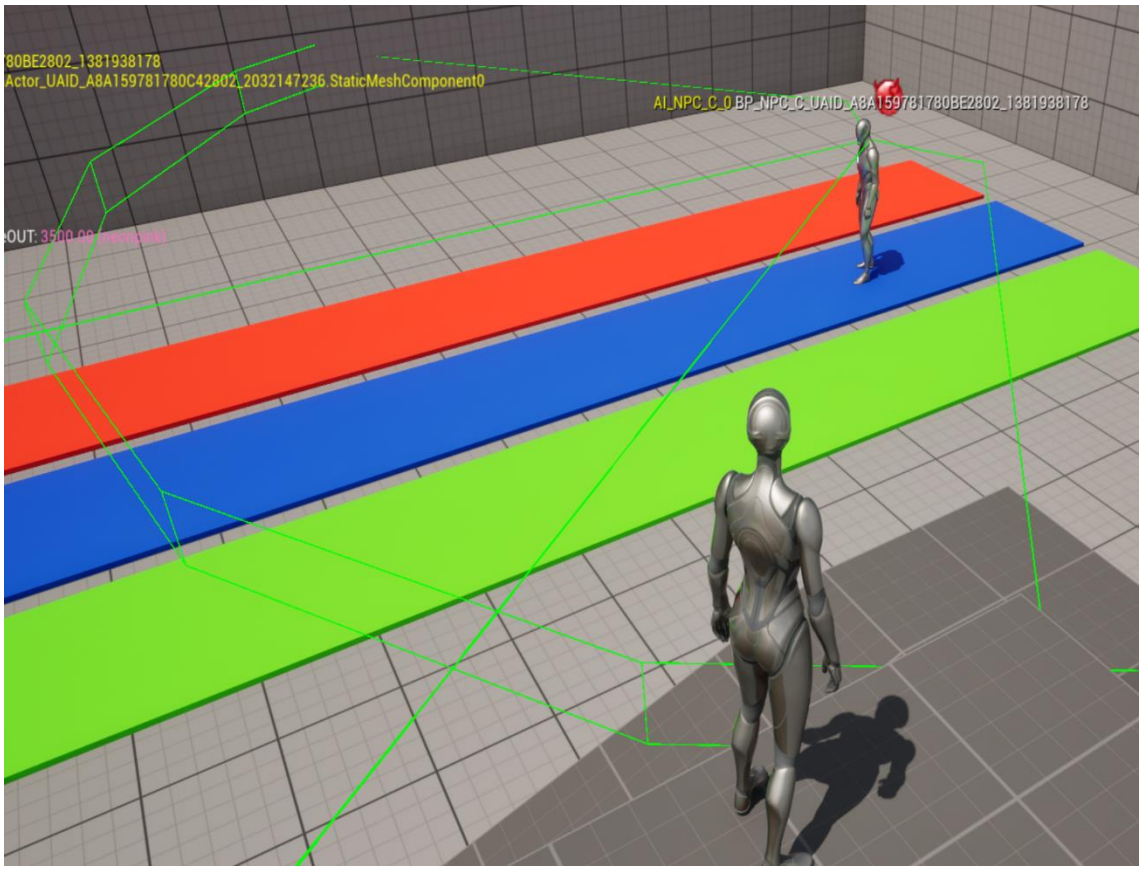

Figure 4. Image of the sight perception radius. (Screenshot by Nanuli Mäntylä)

Once the sight perception system was configured, the next step was to set up the character's reactions. In the AI character's event graph, the "On Target Perception Updated" event was used to identify which actor was perceived and what type of stimulus triggered the event. Since this same event activates for every type of stimulus used in the project, a custom "Handle Sight Sense" function was created to distinguish sight perceptions for other senses.

In this function, the stimulus and actor are passed on as inputs. First, the function checks if the stimulus is related to sight perception; if not, the function

terminates and returns to the Event Graph to process other senses (Figure 5).

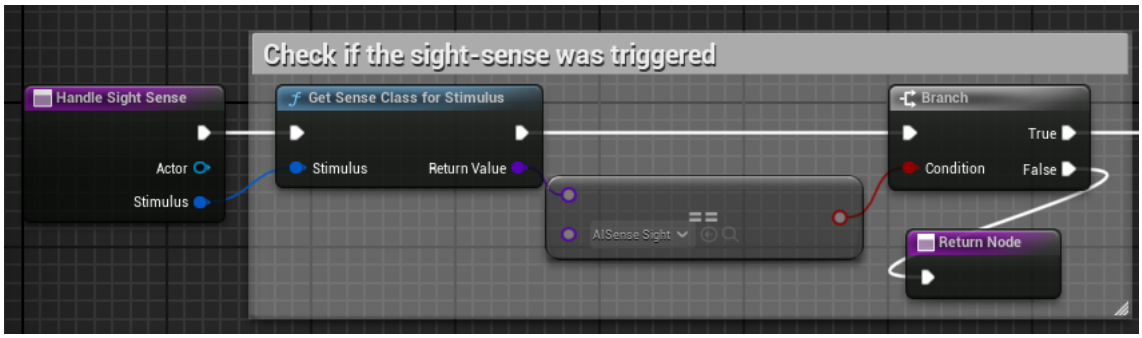

Figure 5. Handle Sight Sense-function, checking the received stimulus. (Screenshot by Nanuli Mäntylä)

If the stimulus is related to sight, the function then determines whether the detected actor is the player or another NPC. If it is the player, “Player seen” is printed in the play mode and if it is another NPC, “Friend seen” is printed on the screen (Figure 6).

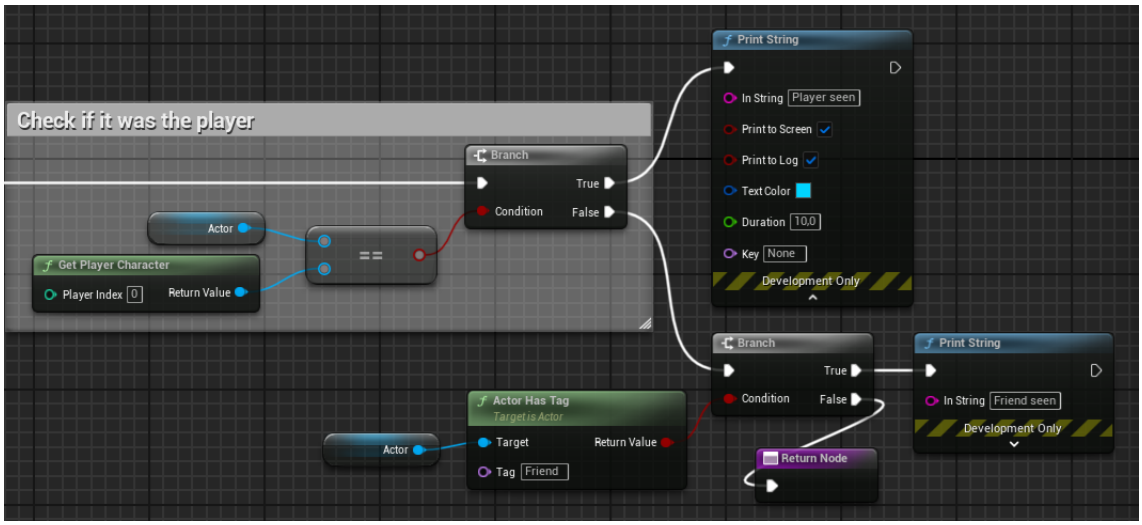

Figure 6. Handle Sight Sense function, checking if stimulus was activated by the player or another NPC. (Screenshot by Nanuli Mäntylä.)

By using the Actor variable with branches, the AI character’s behaviour can be customized based on whom it sees. For instance, if the player is detected, the AI character could initiate a chase; alternatively, if the detected NPC belongs to the same faction, the character might greet them or deliver a voice line.

4.3.2 Setting up the sight perceptible characters

For a character to register as a sight stimulus, the AIPerception Stimuli Source component must be added to the character's Blueprint, with AISense_Sight included in the "Register as Source for Senses" array. (See Appendix 1.)

Additionally, to implement different reactions for various characters, a "Friend" tag was added in the second AI character's Blueprint. Each Blueprint can contain multiple tags, which allow for varied reactions based on the character. The player character can easily be referenced in the code by comparing the stimuli source Actor reference to the output of the "Get Player Character"-function.

4.3.3 Found limitations

While configuring sight perception for the AI character is relatively straightforward, requiring only the addition of the necessary component to the character's AI controller and stimulus component to the detectable Actors. However, there are limitations to the scope of this perception. By default, sight perception

targets only the Character’s torso (as illustrated in the Figure 7), rather than the entire character mesh or capsule component.

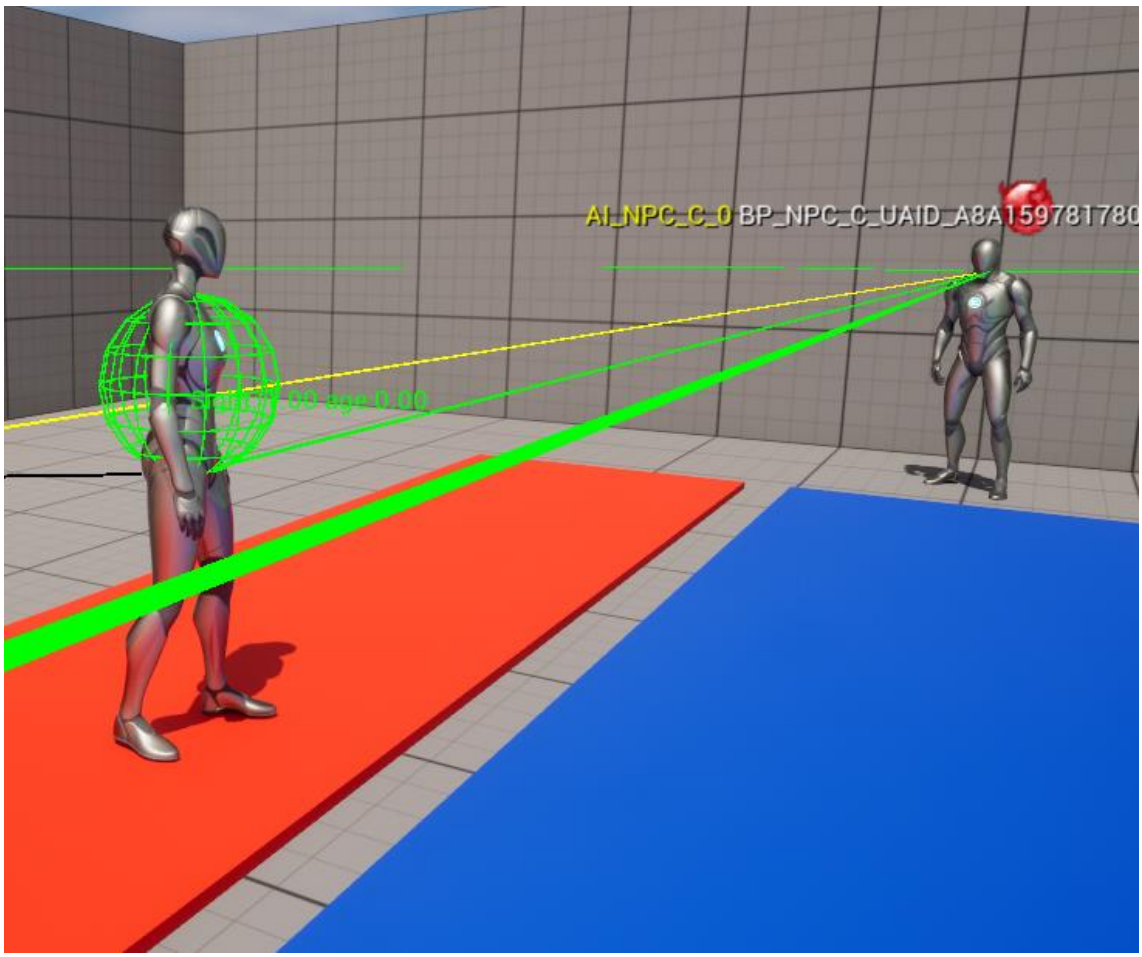

Figure 7. AI Character's sight perception in debugger. (Screenshot by Nanuli Mäntylä).

While this limitation does not always cause issues, it can reduce the immersion. For instance, the AI character may fail to detect the player’s legs when they should be visible, as seen in the Figure 8. Also, about half of the torso is

required to be visible, as seen in the Figure 9.

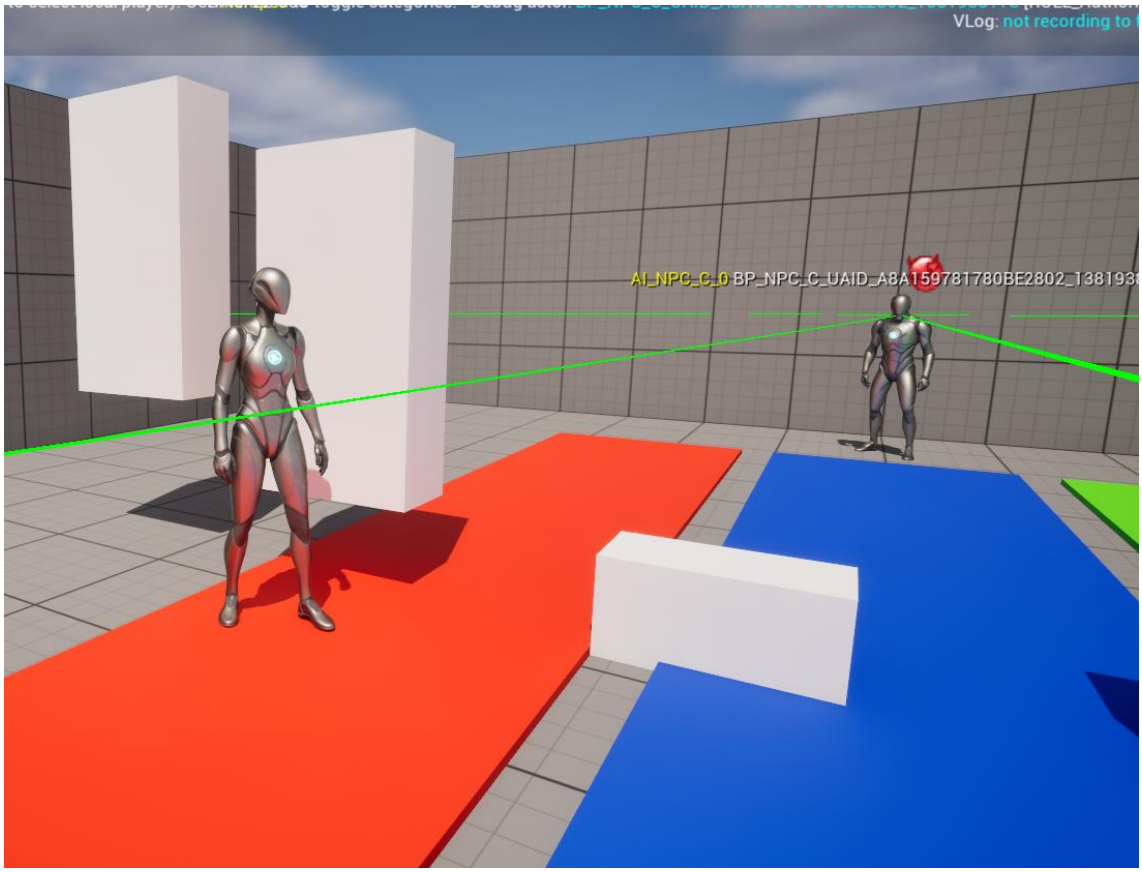

Figure 8. Player not detected by the AI Character, even though their legs should be visible.

Figure 9. Half of the player is required to be visible before sight perception activates.

On the other hand, this limitation can also enhance the player experience, particularly in first-person games, by preventing immediate detection when the player peers over an object or around a corner.

5 Hearing sense

This chapter focuses on the hearing perception, another commonly used sense perception in video games. The first section examines the history of sound detection in video games and how hearing perception is implemented in modern video games. The second section introduces Unreal Engines Hearing perception and variables that can be used to control the hearing. Finally, the third section explores how hearing perception was utilized in the demo project, along with some limitations encountered during implementation.

5.1 Hearing in video games

Hearing is another sense commonly utilized in video games to enhance AI perception. One of the earliest examples of sound detection is in GoldenEye 007 (1997), where enemies could “hear” gunshots and react by moving toward the source. This concept was further refined in Thief: The Dark project (1998) and Metal Gear Solid (1998), where enemies could detect more subtle sounds, like player’s footsteps, especially when the player was running.

In many games, the player’s movement speed directly affects the proximity at which they can be detected by sound. Like in real life, sneaking or crouching reduces the noise generated, making it harder for the enemies to hear the player or requiring them to be closer for detection. Conversely, running increases the noise, making it easier for enemies detect the player from greater distances compared to walking. (Sapio 2019, 173).

Additionally, the type of surface the player moves on can also affect the sound perception. For instance, walking on soft surfaces like grass may make the

player nearly silent, significantly reducing the chance of detection. Conversely, moving over noisier surfaces, such as gravel or dry leaves, the sound generated increases, making it easier for nearby enemies to hear the player. In some games, like Elder Scrolls V: Skyrim (2011), even the player's armour can influence the amount of noise player generates while moving.

Environmental elements can also contribute to sound detection. For example, objects like old branches or pieces of glass can make loud, detectable noises if the player accidentally steps on them or physics-based objects can create sounds when knocked over. These types of environmental sounds can also be used strategically by the player. For example, in The Last of Us (2013), the player can throw glass bottles or bricks to distract enemies, allowing them to safely sneak past.

5.2 Hearing in Unreal Engine

In Unreal Engine 5, AI Hearing can be used to detect sounds created by "Report Noise Event", such as player's footsteps or a projectile hitting an object. Similar to Sight sensing, the AIPerception component must be added to the Actor's AIController and AI Hearing config added to the Senses Config-array (Appendix 1). (Epic Games 2024).

In the AI Hearing details panel, the following variables can be adjusted, with all distance and length measurements set in centimetres by default (the unit of measure can be changed in project settings):

- "Implementation" is an Enum that specifies which sense class will be used for hearing. By default, AISense_Hearing class will be used.
- "Hearing range" is a float that defines the distance at which sound events can be detected by the AI Perception system.
- "Lo SHearing Range" is a float that is used to display different radius for the hearing range in the debugger.
- "Detection by Affiliation" is an Enum, that specifies whether enemies, neutral or friendlies can trigger hearing.
- "Debug color" sets the colour of debug lines in the AI Debugger tool.

- “Max Age” is a float that determines the duration in which the stimuli generated by hearing will be remembered. A value of 0 means the stimulus will always be remembered.
- “Starts Enabled” is a Boolean that specifies if the hearing sense starts enabled or if it requires manual enabling/disabling. (Epic Games 2024)

For an Actor to register as an AIHearing stimuli source, the Actor’s Blueprint must include the AIPerception Stimuli Source Component, with AISense_Hearing added to the “Register as Source for Senses” array (Appendix 2). Additionally, all the actions that would cause sound (for example footsteps and shooting), must have a “Report Noise Event” node added in the Blueprint. (Epic Games 2024).

5.3 Hearing in the project

In the project, three most common ways to implement hearing perception were demonstrated: the player’s footsteps on different surfaces, the player walking on a specific object and the player throwing objects. Additionally, to enhance the sound detection system, physical materials were used to enable dynamic sound perception.

5.3.1 Setting up the AI character

To set up hearing for an AI Character in Unreal Engine 5, the AIPerception component must be added to the AIController, and add the AI Hearing config should be included in the Senses Config-array (See Appendix 1 for more details). For this project, the hearing range was set to 3000cm, given the compact size of the test level.

In the AIController's event graph, the "On Target Perception Updated" function is utilized to receive notifications whenever the perception component detects a stimulus. Similarly to sight sensing, “Handle Hearing Sense” function was created. The function first checks if the perception component was activated by hearing and then prints a “Hearing sense activated”-string to screen in play

mode (Figure 10).

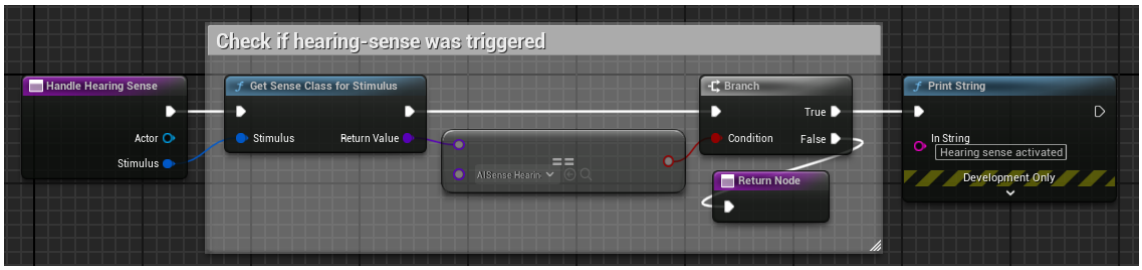

Figure 10. Handle hearing sense function. (Screenshot by Nanuli Mäntylä.)

5.3.2 Setting up the player character

As with the sight, the player character must have the AIPerception Stimuli Source set up (see Appendix 2), with AISense_Hearing added to the “Register as Source for Senses”-array.

To create a dynamic sound system for the player’s footsteps, a system was implemented to detect the material the player is walking on. Depending on the material, the player’s footsteps would be either louder or quieter than normal. The loudness of the sound is further influenced if the player is crouching.

An AnimNotify event was added to all the player character’s animation sequences, triggering every time the characters foot contacts the ground. When the animation updates, these notifiers call a line trace from the player’s current

location in the upward vector direction (see Figure 11). This line trace checks the type of surface the player is on.

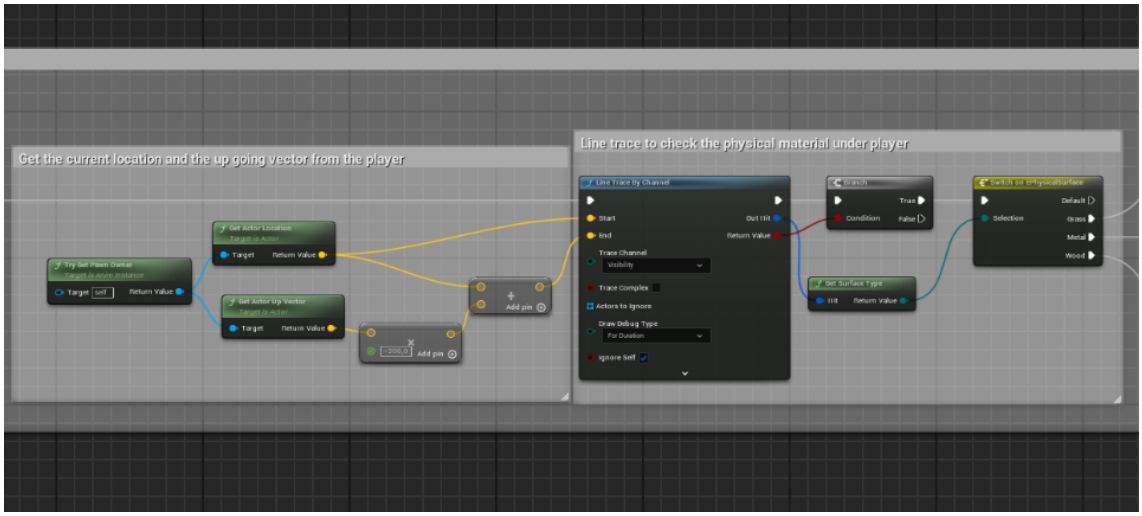

Figure 11. Line tracing to check the physical material. (Screenshot by Nanuli Mäntylä).

Unreal Engine 5 allows the use of 62 different surface types, but for this project only three types were created: grass, metal and wood. These surface types can be added in the Project Settings – Physics Settings (see Figure 12).

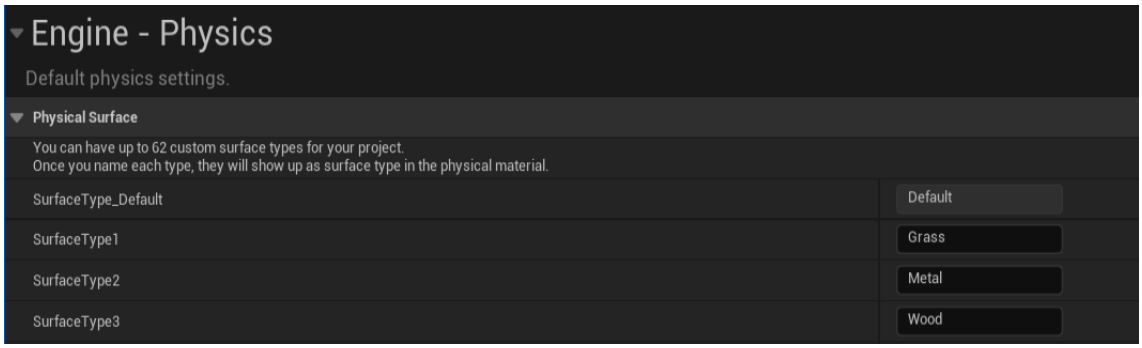

Figure 12. Adding physical surfaces in the Engine Settings. (Screenshot by Nanuli Mäntylä).

For each surface type, a physical material and regular material are required, with the correct physical material added to the corresponding material. The physical material can be assigned to the material in the materials Details-panel.

The loudness of the sound event is determined by the type of surface and the physical material, as seen in Figure 13.

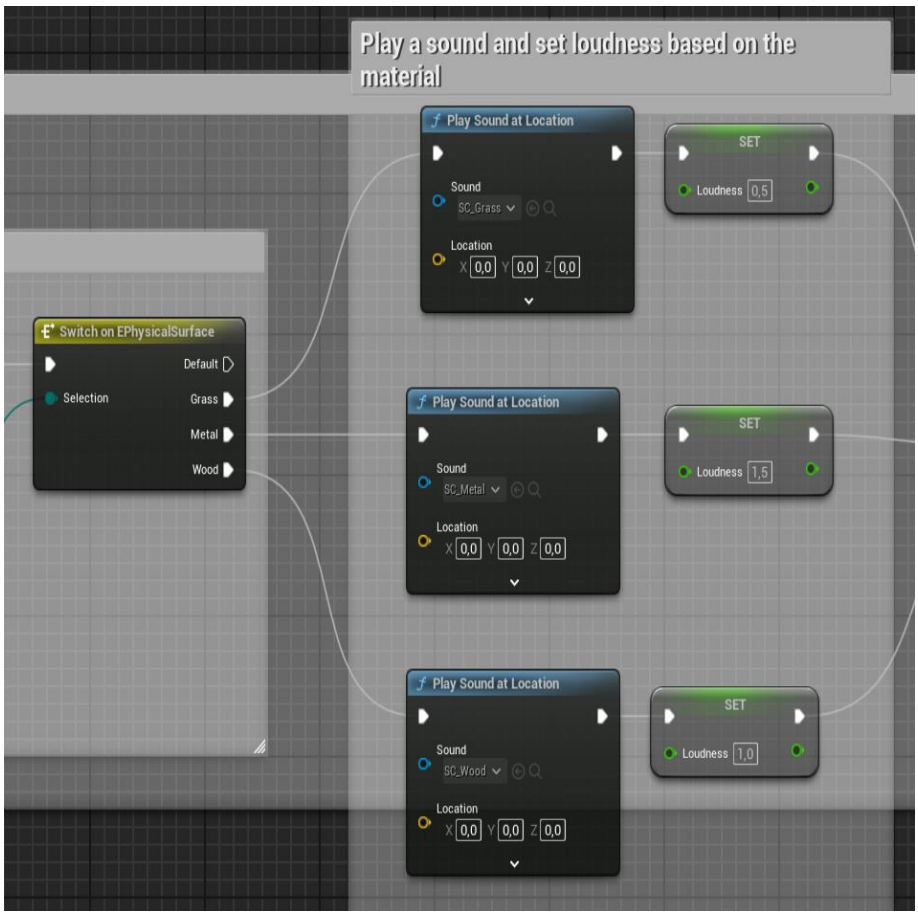

Figure 13. Setting the Loudness-variable based on the Physical Surface. (Screenshot by Nanuli Mäntylä.)

The loudness variable is a multiplier to the max range, which determines the maximum range for the sound perception for that particular noise event. For example, if the max range for the noise event is set to 5000, and the loudness for grass is 0,5, the max range is halved to 2500. However, if the AI character's max range is set to 3000, they still cannot detect sounds beyond that distance. The combination of the loudness multiplier and the AI's max hearing range can also be used to differentiate how enemies perceive sounds at various distances.

Another factor influencing the loudness of the sound events is whether the player is crouching. Using an "isCrouching" Boolean (see Figure 14), the sound loudness of the event is halved when the player is crouching, allowing the player to move closer to the AI character without being detected. Finally, the adjusted loudness, player's current location and the sound instigator (the player)

are sent to the “Report Noise Event”.

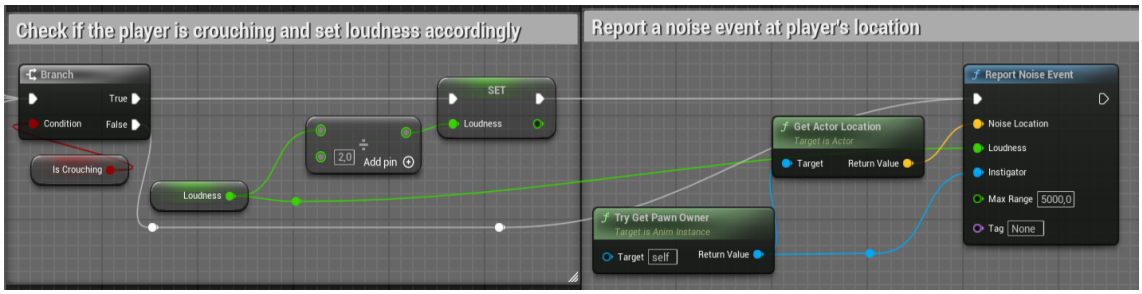

Figure 14. Setting the loudness based on the isCrouching Boolean and reporting the noise event with correct loudness. (Screenshot by Nanuli Mäntylä).

Since the default mannequin was used for both the player character and the AI character, in the beginning of the notification code, it is important to check the notification comes from the player character, not the AI character. Otherwise, the AI character would be constantly detecting itself, causing an infinite loop. If different animations are used for different characters, this check is not required.

5.3.3 Sound creating objects

Two different types of sound creating objects were implemented in the project to create a noise event detectable by the AI character. One is a “branch” that the player can walk on, the other is a throwable object.

The “branch” is a static mesh, with a collision box and AI Perception Stimuli Source component (Figure 15). When the player walks over the branch, it generates a sound event triggered by the collision box. The location of the “branch” itself is used as the noise location, and the branch object, not the player is designated as the instigator. (Figure 15). This ensures that the AI character moves toward the branch location, not the player. The Loudness and Max Range variables for the noise event are instance editable, which allows for customization in

the level viewport. Instance editability can be useful if there are multiple objects that the player can walk on, each requiring different noise levels for detection.

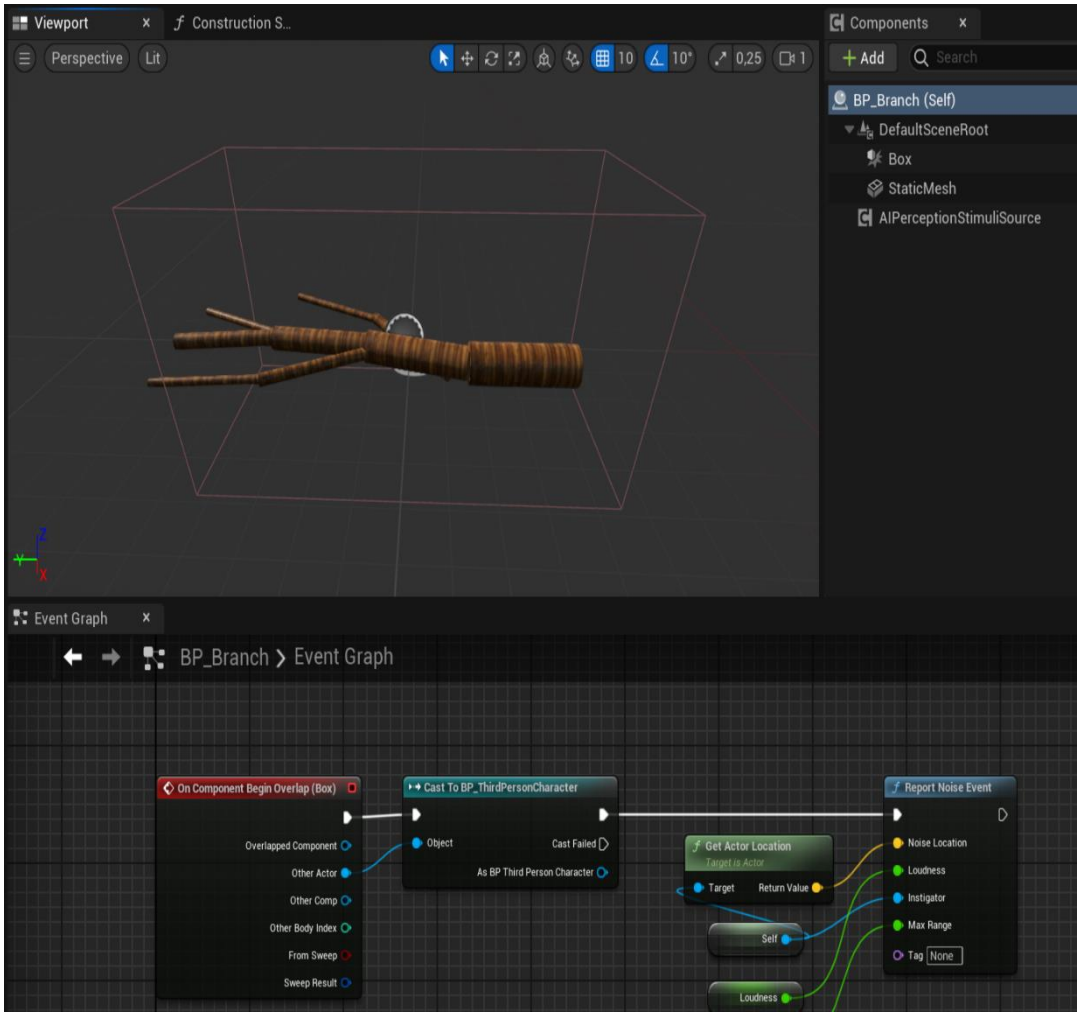

Figure 15. Image of the Branch's static mesh and On Component Begin Overlap-function. (Screenshot by Nanuli Mäntylä.)

The throwable object is another static mesh, with AIPerception Stimuli Source-component. It also has an ProjectileMovement-component that automatically counts the trajectory for the Actor (Figure 15). When player presses “P” key, the throwable object spawns into player’s hand and can be thrown.

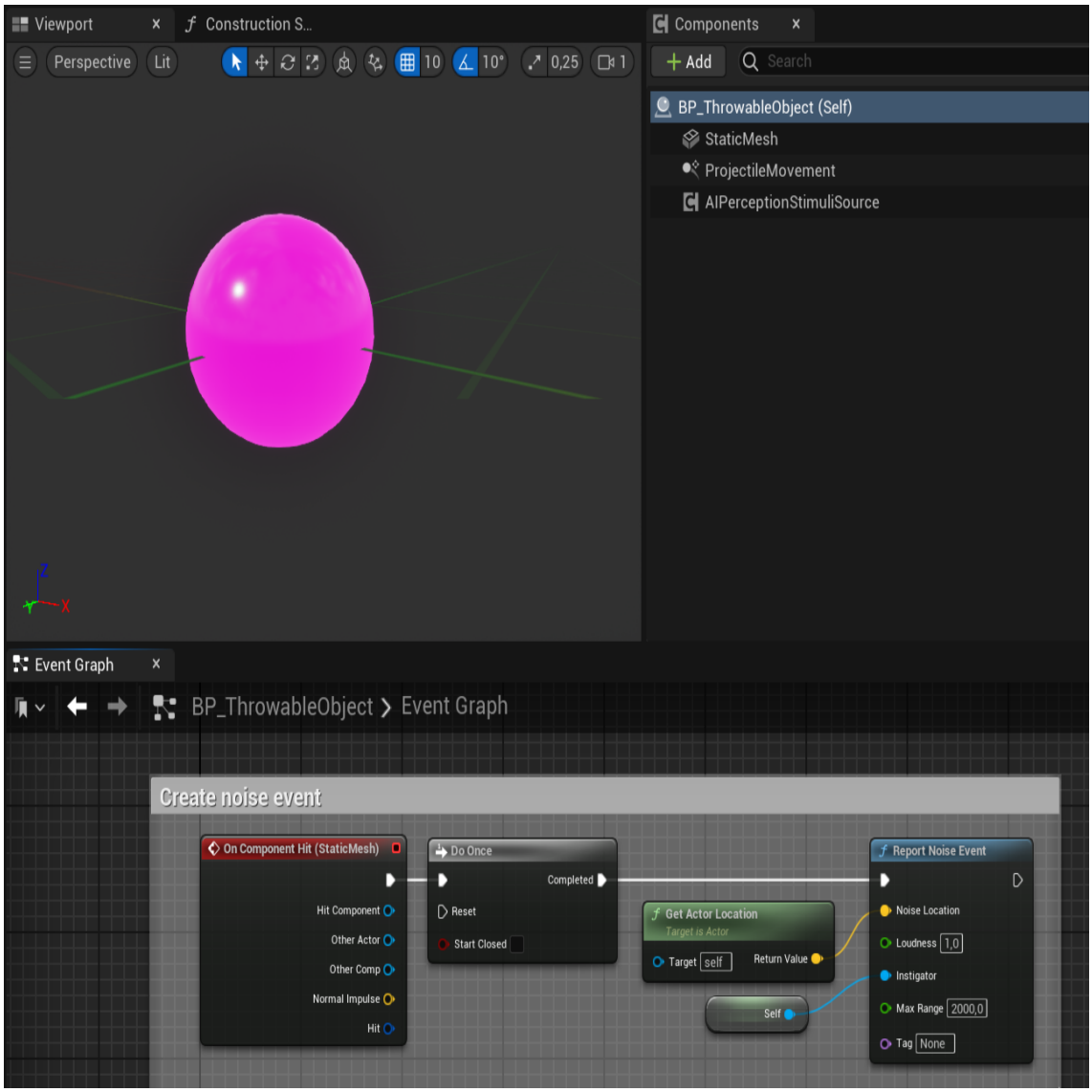

Figure 16. Throwable Objects Actor Components and On Component Hit function. (Screenshot by Nanuli Mäntylä).

Contrary to the “branch”, the throwable object generates a noise event only when it collides with another surface, which is detected using the “On Component Hit” event (Figure 16). Similarly to the branch, the throwable object is the instigator and its location the sound location. This allows the AI character to react to the location where the object lands, for example moving to the sound location, enabling the player to move forward without being detected.

5.3.4 Found limitations

While the hearing perception system in Unreal Engine is generally effective and relatively simple to configure for AI characters, there are a few limitations. The

most significant issue is the more time-consuming nature of setting up the sound perception compared to sight perception.

One key issue is the requirement to add animation notifications for every animation sequence used by the player character. For example, enabling footprint detection necessitates the addition of an AnimNotify event to each animation where the player character's foot hits the ground, like walking, running and jumping. This becomes especially cumbersome when there are multiple characters, each with different movement sequences, which significantly increases the workload.

Additionally, determining the optimal range for sound detection is not straightforward. Extensive testing is required, especially when dealing with a variety of surface types, as different materials affect the volume and range of sound events. This variability requires fine-tuning to ensure the detection range is appropriate for each scenario, making it a more iterative and time-intensive process compared to configuring the sight detection.

6 Touch

This chapter of the thesis will focus on touch perception. As with the previous chapters, first section will cover a short history of touch detection in video games and how touch perception is used in modern video games. The second section goes through touch perception components in Unreal Engine and the third section explains in detail how touch perception was used in the project.

Unlike sight or hearing, which are often used for passive detection (i.e., perceiving stimuli from a distance), touch perception requires direct and immediate interaction between the player and the AI. This makes it less common in most games, particularly in genres that prioritize long-range engagement, like first-person shooters. Touch sensing is more localized, relying heavily on precise collision detection and immediate player interactions. As a result, touch-based

detection introduces a higher level of realism, where AI behaviours can be driven by physical contact rather than distant detection. However, this also presents significant technical challenges, such as ensuring accurate detection for different appendages, requiring more complex coding than sight or hearing detection. While it can enhance gameplay in specific genres, especially in survival or horror games, touch perception often disrupts the flow of gameplay in more fast-paced genres.

6.1 Touch in videogames

Touch perception in video games has evolved over time, with some of the earliest examples seen in games like Space Invaders (1978) or Pac-Man (1980), where “touching” the enemy or projectile would automatically result in death for the player or the NPC. These examples, while primitive, laid the groundwork for collision-based mechanics, where the act of touching or colliding with something had consequences for the player.

As video games progressed, the concept of touch became more nuanced. Katamari Damacy (2004) provides a unique example, where the player controls a ball that collects objects by rolling into them. The ball grows as it collects smaller items, but if it touches something too large, the player loses balance or may lose the previously collected objects.

One of the most prominent examples of true touch perception in modern gaming is Alien: Isolation (2014). In the game the player navigates through a space station, while trying to avoid an alien. The enemy’s sensory system includes sight and hearing, but the player must also be cautious of physically touching

the enemy. This can prove especially hard, since the alien's tail is thin and long, making it occasionally hard to see in the dark environment. (Figure 17).

Figure 17. Screenshot from Alien: Isolation, showing the alien's tail. (SEGA 2014).

Another, quite a different example of touch perception is In Amnesia: The Dark Descent (2010) where a water-based enemy reacts to water's movement. Like with the sound detection, the enemy cannot differentiate the instigator, but only moves towards the movement, be it the player or an object player dropped into the water. The only way for the player to avoid encountering the enemy, is by either distracting it with food or other objects, or to jump on higher ground.

In addition to these examples of enemies reacting to touch, many modern games use touch perception to enhance NPC behaviour and realism. For instance, in The Elder Scrolls V: Skyrim (2011), when the player collides with NPCs, they often react with comments like "Watch it!" or "Do you need something?". If the player continues to push and bump into them, the NPC's may become increasingly annoyed, even attacking the player. Red Dead Redemption 2 (2018) further refines this system by using dynamic reactions based on NPC personalities, relationship with the player and the intensity of the physical

interaction. These subtle interactions help create a more realistic world, where a simple touch-based action can change how the NPCs around the player interact with them.

6.2 Touch in Unreal Engine

In Unreal Engine 5, the touch sense can be integrated into AI characters by adding the AI Touch configuration to the AI Perception component (see Appendix 1), similarly to other sense configurations. This enables the AI character to detect touch-based stimuli, whether the character bumps into another Actor, or another Actor bumps into it.

In the AI Touch details, the following variables can be modified:

- “Debug color” sets the colour for the debugging tool.
- “Max Age” is a float, determining the duration (in seconds) after which the stimulus will be forgotten. A value of 0 means it is never forgotten.
- “Starts enabled” is a Boolean, determining whether the touch sense is enabled by default or must be manually enabled and disabled.

To register a touch event with the AI character, a “Report Touch Event” must be added to the desired Actor’s Blueprint. This node can be connected, for instance, to the “Event Hit” function, which automatically detects when the Actor is touched by another Actor. This way, any collision or physical contact triggers a touch event, which can be used to prompt AI responses, such as reacting to player’s actions or starting a new behaviour sequence.

6.3 Touch in the project

In this project, a simple touch sensing system was implemented. When the player collides with the AI character, a string is printed on the screen and a 3-second cooldown is activated. During this cooldown period, subsequent touches are ignored. This cooldown mechanism can be used to trigger specific AI behaviours, such as playing animations or voice lines, enhancing the interactivity and realism of the character’s response to the player.

6.3.1 Setting up the AI Character

To integrate touch perception into the AI character, the AIPerception component must be added to the character's AIController, with AI Touch Configuration included in the "Senses Config" array (See Appendix 1). If different reactions based on Affiliation are required, these can be set in the details panel. Note that currently Affiliation can only be set in C++-code, but various Affiliations can be added to Actors using tags (as demonstrated with AISight; see section 4.3.2 for more details.)

For handling touch perception, a "Handle Touch Sense" function was created and connected to the "AI Perception Updated" event. This function takes the Actor and Stimuli as inputs. Inside the function the received stimulus is compared to the AISense_Touch stimulus. If applicable, the "cooldownActive"-Boolean is checked (Figure 18). If the cooldown is still active, the touch event is ignored. If the cooldown is inactive, "Touch Sense Activated" string is printed on the screen in play mode and "cooldownActive" Boolean is set to true.

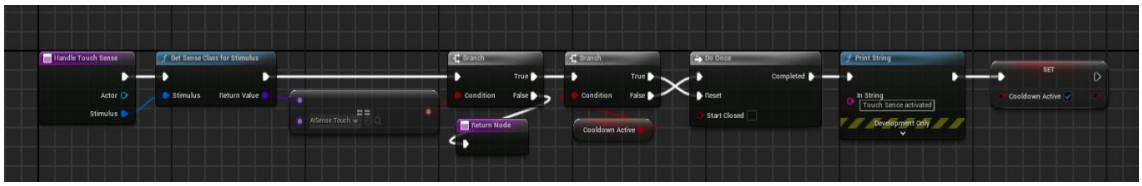

Figure 18. Handle Touch Sense-function. (Screenshot by Nanuli Mäntylä).

In the AIController's event graph, the cooldown is monitored during the event tick. If activated, a cooldown delay of 3 seconds is triggered. After this cooldown period, the "cooldownActive" Boolean is reset to false, enabling the AI character to register future touch stimulus (Figure 19).

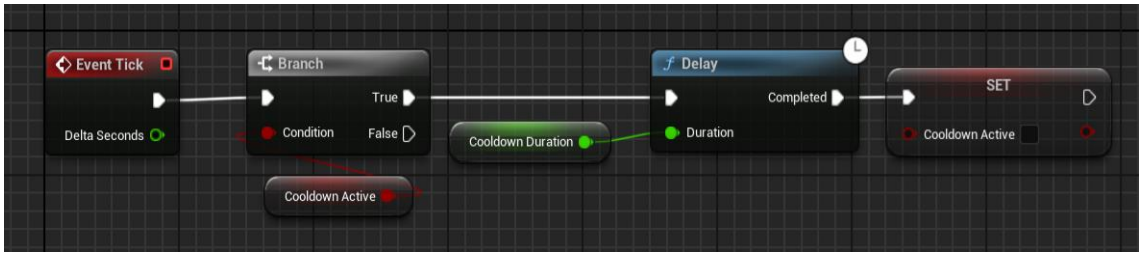

Figure 19. Cooldown for the touch sensing. (Screenshot by Nanuli Mäntylä).

Additionally, an “Event Hit”-function was added to the AI character’s Blueprint to detect touch events. This function, provided by Unreal Engine 5, is triggered whenever something collides with the character’s capsule component. The function also outputs various information, such as the hitting actor and the hit location. These variables can be used within the AI Controller or behaviour tree to implement different reactions depending on the actor triggering the touch event.

6.3.2 Setting up the player character

To ensure player character registers as a stimulus for AI touch perception, the AIPerception Stimuli Source-component must be added to the character Blueprint and AISense_Touch must be included in the “Register as Source for Senses” array. (See Appendix 2.)

If the sense is not added as a stimuli source, the character may still be detected as a stimulus, but the touch sense activation will not be displayed in the debugger. This behaviour is intended to avoid registering irrelevant stimuli, such as the ground and walls, which could lead to unnecessary detections.

6.3.3 Found limitations

Setting up touch perception in Unreal Engine 5 is relatively simple. It requires adding the perception component in the characters AI Controller, the “Event Hit”-function in the characters Blueprint and the AIPerception Stimuli Source component to the detectable characters Blueprint. However, this approach was only tested with only a single character and one reaction. As with hearing perception, the complexity of the code could increase significantly if multiple reactions or behaviours are needed from the AI character.

Another potential limitation involves non-humanoid characters. For example, if the AI Character has a tail or other appendages, additional collision components would be required to properly detect touch events, complicating the setup.

7 Discussion

The goal of this thesis was to study how AI perception is used in modern video games and what kind of components a modern video game engine, such as Unreal Engine 5, offers for implementing AI perception in a project. These AI perception components were also tested in practice by creating a small game demo in Unreal Engine 5.

AI Perception has become a standard tool in many modern games, significantly enhancing both player immersion and NPC realism. Games like Metal Gear Solid (1998), The Last of Us (2013) and Red Dead Redemption 2 (2018) rely heavily on AI Perception systems to create reactive and immersive worlds where NPCs respond to the player’s actions based on their senses. Such systems have raised players expectations for intelligent NPC behaviour, setting new standards for the industry.

However, while AI Perception systems can simulate awareness, they often lack the adaptability needed to respond to the diverse and unpredictable actions of players. In open-world games or player-driven narratives, scripted AI behaviours may fail to react to actions outside the design parameters, leading to immersion-breaking moments. For example, an AI that only reacts to noise events might not effectively account for more subtle interactions, like intentionally luring an enemy away with a quiet action.

On the other hand, creating 'too humanlike' AI can lead to negative gameplay experiences, particularly in terms of challenge and player enjoyment. For example, overly intelligent AI might become frustrating or even unnerving if it anticipates the player's every move, making the game feel unfair or too difficult. Striking the right balance between unpredictable, believable AI behaviour and maintainable challenge remains one of the key hurdles in AI design.

Unreal Engine 5 offers a wide array of AI perception tools, making implementation relatively accessible for beginners. All of the coding in the game demo was completed using visual scripting (Blueprints), demonstrating that even with limited game development experience, it is possible to create a basic sensory perception system in Unreal Engine. Epic Games provides a wealth of documentation and tutorials, as well as template projects, which helped me navigate the system effectively.

Through hands-on testing, I observed both the benefits and limitations of AI perception components. AI Sight was straightforward to implement, though it required adjustments to align with the genre and project's vision. For instance, creating a stealth game would likely need further customization, such as adjusting perception to specific body parts like a character's head, which could be achieved with C++ coding.

AI Hearing took longer to implement but allowed for a rich variety of gameplay experiences. Having no prior experience with creating a dynamic sound system, I was surprised by how intuitive Unreal Engine's system was, allowing me to create sound-producing objects that contribute to a more immersive environment.

AI Touch was also simple to set up and offered potential for more interactive NPC reactions. Adding touch perception added another layer of realism to the AI characters, allowing them to recognize and respond to interactions like physical contact.

Overall, this project taught me a great deal about working with the three core AI perception senses in Unreal Engine, along with an awareness of some of the system's limitations. Future work could involve enhancing these systems with behavior trees to provide more complex responses. Another area of exploration could be customizing AI perception with C++ for features like sound propagation or echolocation, which would enable more advanced perception behaviors.

References

Alien: Isolation. 2014. SEGA.

Alkaisy, C. 2011. The history and meaning behind the “Stealth genre”. Gamasutra. https://web.archive.org/web/20170702224925/http://www.gamasutra.com/blogs/MuhammadAlkaisy/20110610/7764/The_history_and_meaning_behind_the_Stealth_genre.php. 21.10.2024.

Amnesia: The Dark Descent. 2010. Frictional Games.

Cambridge University Press & Assessment. 2023. Dictionary. https://dictionary.cambridge.org/dictionary/. 26.9.2023.

Castlevania. 1986. Konami.

Daglow, D. 1988. The Changing Role of Computer Game Designers. Computer Gaming World no. 50. 18, 42 29.12.2023

Dill, K. 2014. What is Game AI?. In Game AI Pro. Rabin, S. Taylor & Francis Group. 21.10.2024

Epic Games. 2023. Unreal Engine 5.5 Documentation. https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation. 26.9.2023

Epic Online Learning. 2023. Introduction to AI with Blueprints. https://urly.fi/3ejz. Epic Games.

GoldenEye 007.1997. Nintendo.

Half-Life. 1998. Half Life.

Hitman 3. 2021. IO Interactive.

Katamari Damacy. 2004. Namco.

Mario Bros. 1983. Nintendo.

Metal Gear Solid. 1998. Konami.

Metal Gear Solid 2: Sons of Liberty. 2001. Konami.

Metal Gear Solid V: The Phantom Pain. 2015. Konami.

Pac-Man. 1980. Namco.

Red Dead Redemption 2. 2018. Rockstar Games.

Sapio, F. 2019. Hands-On Artificial Intelligence with Unreal Engine. Packt Publishing. 26.9.2023

Space Invaders. 1978. Taito.

The Elder Scrolls V: Skyrim. 2011. Bethesda Softworks.

The Last of Us. 2013. Naughty Dog.

Thief: The Dark Project. 1998. Looking Glass Studios.

AIPerception Component

AIPerception Component is a component that needs to be added to all AI character's AI Controllers, that require AI perception (See Figure 1). AI Controller needs to be also added to the Characters Blueprint to successfully control the character.

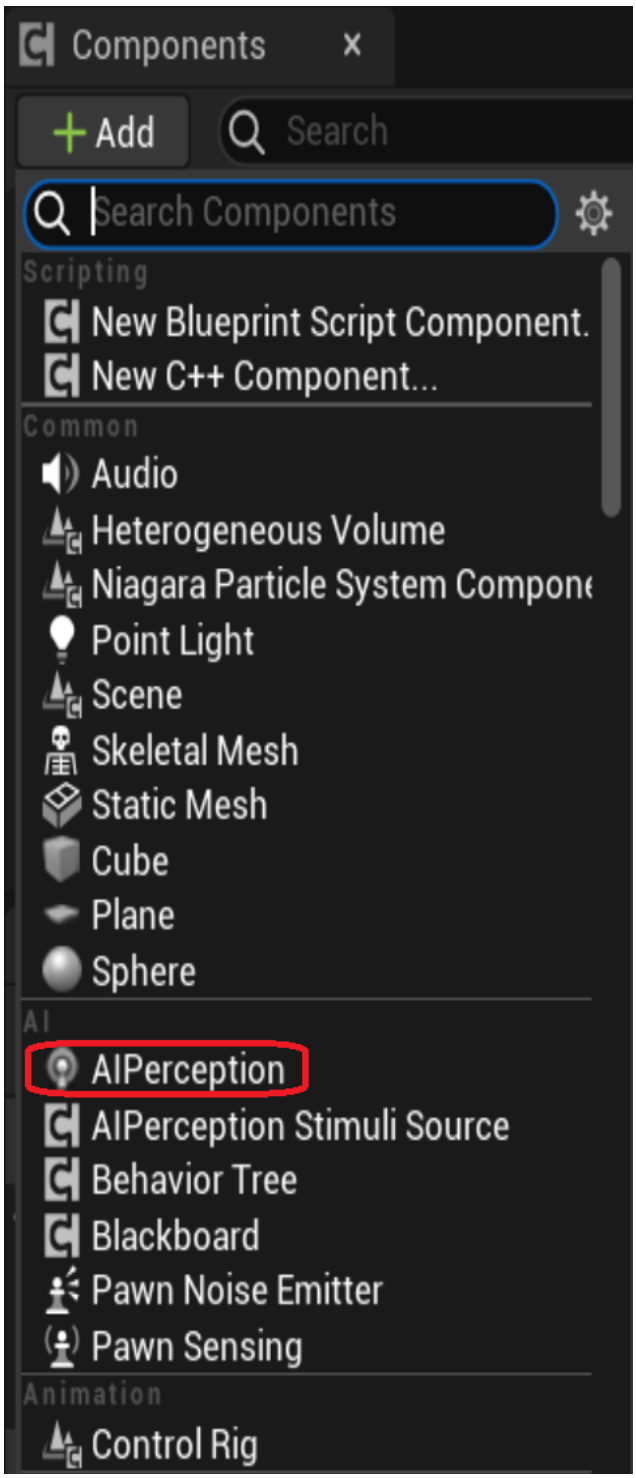

Figure 1. Adding the AIPerception component in AIController.

In the AI Perception component's Details-panel, user can add the required senses to the AI Component (Figure 2). In this thesis, only sight, hearing and touch were used.

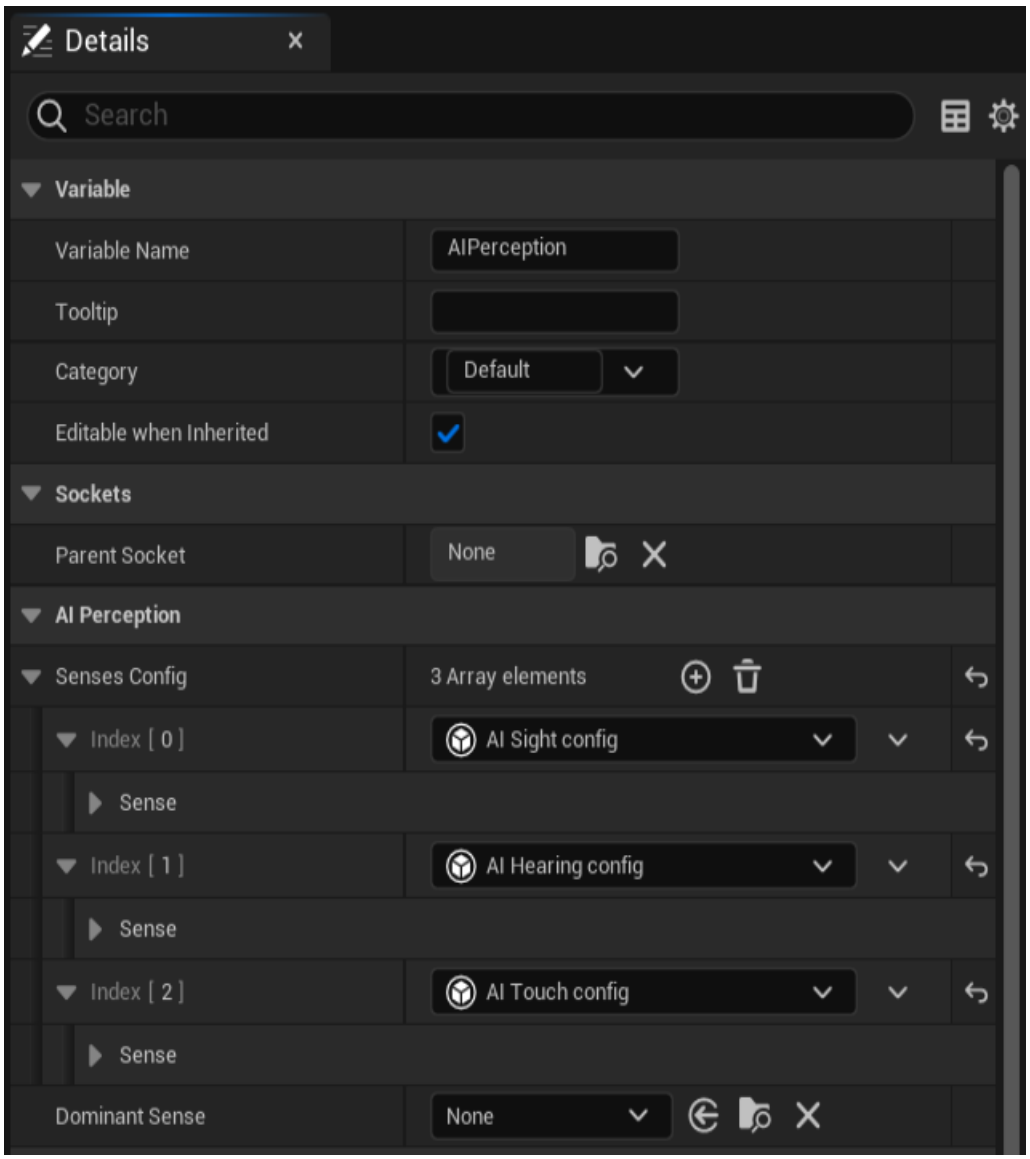

Figure 2. Adding AI perception senses to Senses Config array.

AIPerception Stimuli Source

AIPerception StimuliSource is a component that needs to be added to all actors (See Figure 3), which are meant to register as a stimuli source for the designated senses. Without this component, the NPC-characters will not register the character, even if they are perceptible in the scene.

The AIPerception Stimuli Source can be added to all blueprints from the actor class, including interactable objects in the game levels (for example objects that can be knocked over or thrown.)

When the component is added to the blueprint, several options are available from the Details-menu (Figure 4). All the different senses must be added manually, to be registered as the source for that sense.

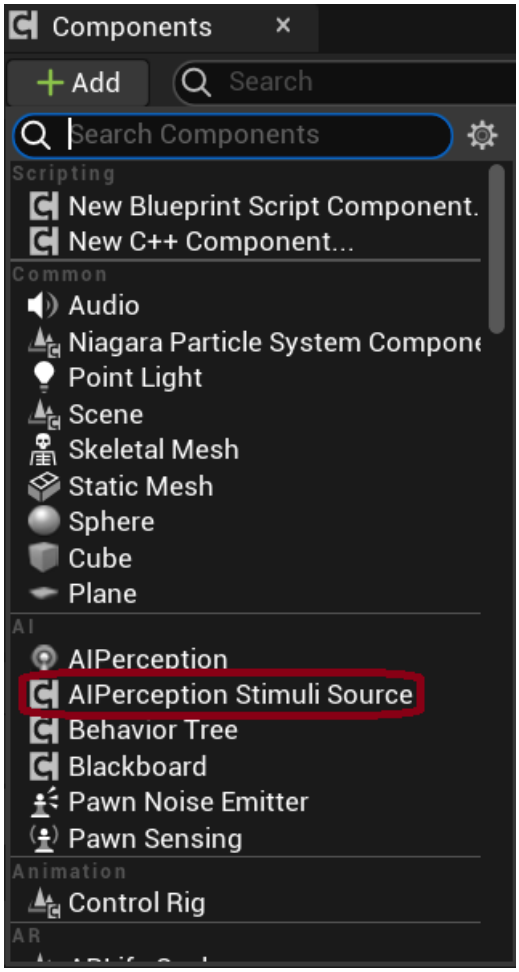

Figure 3. Adding AIPerception Stimuli Source to Actor Blueprint. (Screenshot by Nanuli Mäntylä).

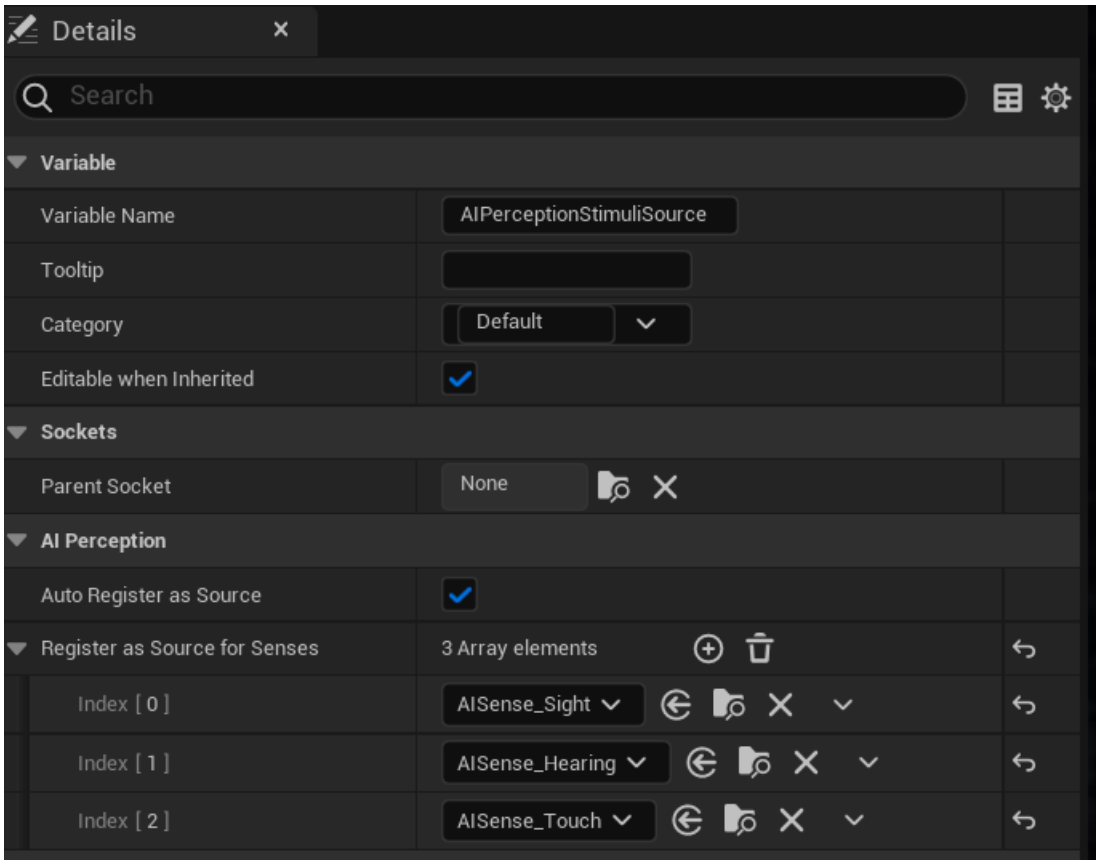

Figure 4. Adding senses for "Register as Source for Senses" array. (Screenshot by Nanuli Mäntylä).