University Institute of Lisbon

Department of Information Science and Technology

A Unity-Based Framework for Sound Transmission and Perception in Video Games

Jorge Miguel Pereira de Sousa Calado

A Dissertation presented in partial fulliment of the Requirements for the Degree of

Master in Computer Science

Supervisor

Pedro Figueiredo Santana, Assistant Professor

ISCTE-IUL

Co-Supervisor

Rui Jorge Henriques Calado Lopes, Assistant Professor

ISCTE-IUL

October, 2017

Resumo

A capacidade humana de ouvir, percepcionar e compreender sons é afetada, no mínimo, por fatores ambientais, físicos e cognitivos. Implementações atuais de audição em agentes virtuais apenas consideram a distância entre o emissor do som e o seu receptor como fator influenciador da sua receção. De forma a superar essa limitação, esta dissertação apresenta uma framework direcionada a programadores de jogos de vídeo interessados em dotar os seus personagens virtuais de audição reactiva a ruído, às características do agente e dependente do contexto.

A framework é direcionada a jogos desenvolvidos para o motor de jogo Unity, através da disponibilização de ferramentas que facilitam a integração em jogos do motor de jogo Unity. Um jogo do tipo First Person Shooter , Fortress, foi desenvolvido com o objectivo de integrar e testar as funcionalidades de transmissão e percepção de som disponibilizadas pela framework.

Os programadores de jogos de vídeo passam a poder dotar as suas personagens virtuais de perceção de sons provenientes do ambiente virtual, relacionando a capacidade de perceção com o contexto do personagem virtual. Os programadores podem costumizar os atributos físicos e psicológicos que afetam a capacidade de perceção do som por parte dos personagens virtuais. Em consequência da melhor aproximação da capacidade auditiva dos personagens virtuais com a capacidade auditiva humana, os personagens apresentaram um comportamento mais el à realidade, aumentando a imersão do jogador no mundo do jogo.

atenção auditiva, som, perceção de som, comportamento, personagem virtual, npc, jogos de computador

Abstract

The ability for humans to hear, attend to, and understand incoming sounds is affected, at least, by environmental, morphological, and cognitive factors. Conversely, current implementations of audition in virtual characters often only consider as constraint to the auditory process the distance between the sound emitter and the sound receiver. To cope with this limitation, this dissertation presents a novel framework, directed to game developers interested in implementing non-player characters with noisy, character-specific, and context-dependent auditory perception.

The framework is prepared to be integrated with games developed in Unity's game engine, providing custom made Unity scripts that simplify the integration into Unity games. A First Person Shooter game named Fortress was developed, which was integrated with the framework in order to test the mechanisms for sound transmission and perception provided by the framework.

The framework allows developers to enable the ability for virtual characters to perceive sound from their surroundings, while keeping the ability of perception closely binded to the current context the virtual character is in. The developer may then customize each virtual character's physical and psychological traits that react the way the virtual character perceives sound from its surroundings. Consequently, the level of immersion for players is increased, as the characters from the virtual world react to sound from their surroundings in a more human way.

Keywords: auditory perception, sound, sound perception, behavior, virtual character, npc, videogames

Chapter 1

Introduction

1.1 Context

Alongside the evolution of videogames graphics, non playable characters(NPC) have an increasingly important role in the production of realistic gaming experience. Besides making a NPC appear real, it is also important that it behaves as a real living being would. As stated in [7], NPCs can look real and smart when it comes to many basic behaviors, such as walking towards a certain point and avoiding unwalkable terrain, but that does not mean that they are intelligent. For them to be intelligent, they need to have the ability to shape their behaviour by performing relevant and timely actions that have the surrounding context into consideration [12].

The evolution of AI has greatly improved the believability of NPCs by introducing a more complex and organic behaviour. To feed this complexity in behavior, there must be a way for the NPC to acquire information regarding its context, this is, the setting it is placed in, what is happening and which is its status. The NPC must be able to survey its surroundings by using its sight, hearing and every other senses it has simulated and, then, be able to act based on the newly acquired information. It is important that it is sensible to other NPCs and to the player's presence as well, in order to deliver an enveloping experience to the player.

As with all human sensory capabilities, audition plays an important role in the perception of the environment. The act of perceiving a sound, interpreting its content and origin provides a rich amount of data regarding the surroundings. The implementation of sound perception is crucial for the proper simulation of human behavior in videogames, therefore sound plays an increasingly important role in videogames [2]. Most recent developments in environment perception by NPCs tend to focus on sight and often provide simplified hearing capabilities, not taking into account the individual capabilities and limitations of each NPC. In CRYENGINE [6], Crytek's game development engine, game developers can con-trol the visual and aural acuity of a character through a single scalling variable [9], which is then used to compute if an entity is seen or heard, disregarding the character's current psychological and physiological state. This method also discards any influence that other sounds or solid obstacles may have in the perception of a given sound, which are considered when it comes to computing line of sight.

1.2 Motivation

In older videogames like Pong, the simplicity of the context did not provide much ground to come up with a believable behavior for the enemy paddle, as the only the only variables that mattered were the position of the ball and the position of the enemy paddle. In recent videogames the realism of the environment has increased tremendously, making the amount of variables to be surveyed by the NPCs immense. In the Farcry series [16], which use a modified version of the CRYENGINE, Dunia, NPCs receive feedback from the environment by scanning all the relevant characters and objects with their sight, identifying all the relevant sounds they hear and through an auxiliary system called bullet rain, which takes notice of bullets passing close to them [9].

The evolution of videogames has lead to a demand for greater believability in NPCs behaviors and capabilities. This evolution lead to the creation of mechanisms to enhance the NPCs perception of the environment and communication with other

NPCs. In the current generation of videogames, sound perception capabilities of NPCs are often simplied. This leads to a less faithful representation of reality, which impacts on the credibility of the NPCs and, therefore, the virtual world. In many videogames, like games running in the Source engine[19], NPCs have perfect hearing and attention. In the source engine, sound is emitted through an in-game entity called ai_sound [18], which enables all NPCs to hear a sound if they are within the acoustic range of its source, with the only other decisive factor being a ag that indicates if the NPC is from a particular enemy class, which might not be able to hear sounds. This method completely disregards physical and psychological factors, such as the NPC being distracted performing a given task or the existence of a loud sound masking a low intensity one, both of which could cause the NPCs to not be able to perceive a given sound. Accounting for imperfection is crucial for the correct simulation of organic hearing capabilities. Due to processing power restrictions, but not solely, imperfections in the hearing capabilities of NPCs were cast aside in favor of other capabilities that are more noticeable by the player, like the path planning in a crowded environment or the selection of the best route of actions to complete a given puzzle.

There is a need to create a way to better simulate human-like sound perception for the current generation of gaming engines. The implementation of such behavior should be exible to allow the expansion on the offered functionalities and should be of simple integration on a currently existing game. One of the main challenges is the efficient use of computational power in a way not to harm the performance of a game with the computation of sound perception. This computational burden will be even bigger when dealing with multiple NPCs at the same time, requiring that the delivery and analysis of each sound should be done in an efficient way.

1.3 Research Questions

During the initial steps of this work, the research regarding sound transmission in videogames raised many questions. These questions provided the core motivation of the whole research and development process, contributing to the resulting framework. The most relevant questions were the following:

Is there a way to survey an artificial videogame environment and emulate conveniently the way the human brain discards, perceives and processes sound messages?

If so, can a virtual sound be associated with artificial sound properties to be processed by the NPCs and still deliver a realistic representation of the way sound is perceived by a human being?

Can such behavior be delivered to developers in the form of a framework that provides easy integration with an existing game and allow the developers to build on the existing functionalities, by adding their own?

1.4 Objectives

The main objective of this research is to provide game developers with a framework that provides mechanisms for the transmission and perception of sound messages for NPCs. The NPCs and other objects in the game environment should be able to, besides playing a sound le for the player to hear, emit a sound's description, with all the properties of the emitted sound. NPCs should be able to receive the sounds that are emitted and, through the analysis of the sound description and according to their psychological and physical state, declare the sound as perceived or discard it. Upon perceiving the sound, the developer must be able to associate an action according to the sound's description.

The framework should provide an easy integration with existing videogames. As to reach a bigger audience, this work is aimed at games developed in Unity's game

engine, which is the gaming engine with the biggest marketshare and developer community. [17].

Alongside the integration process, it should be kept in mind the extensibility of its mechanisms, allowing developers to make further modifications to existing features and the addition of new ones, such as implementing different mechanisms for the transmission of sound messages or adding the ability for the framework to handle the playback of sound messages..

1.5 Research Method and Document Structure

The Design Science Research methodology was applied during the whole research, development and evaluation of the framework. The methodology can be described by the following steps:

1. Problem identification and motivation
2. Definition of the objectives of a solution
3. Design and development
4. Demonstration
5. Evaluation
6. Communication

The primary step, regarding problem identification and motivation is present in the Sections 1.1 and 1.2. The definition of objectives is made in the Section 1.4, based on answers to the questions posed in Section 1.3. Chapter 3 reflects the design and the development that is part of the third step. The fourth step is reflected in the game developed in Chapter 4. The step regarding the evaluation of the proposed framework is provided in Chapter 5. The communication step is reflected in Chapter 6.

Chapter 2

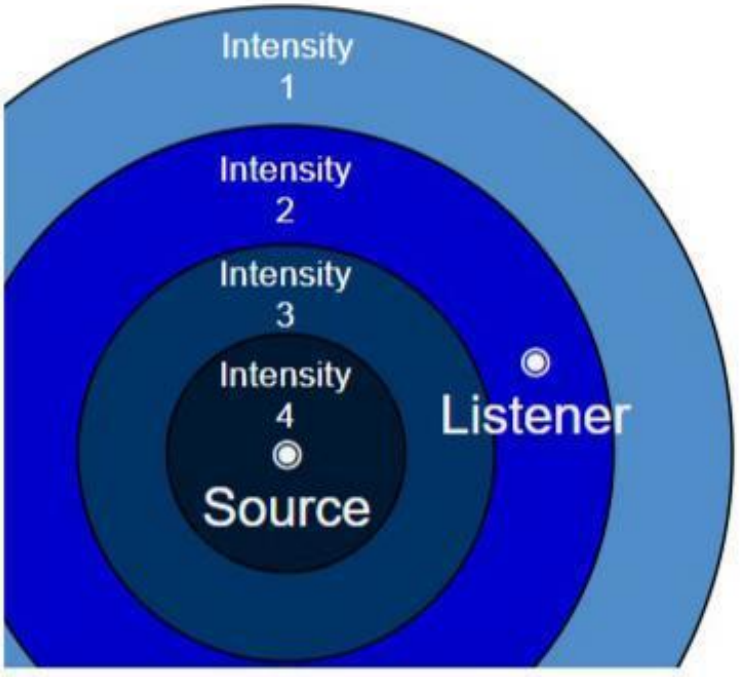
Related Work

Currently, there are two predominant implementations of sound transmission and perception in videogames: Region Sense Manager (RSM) and Finite Element Model Sense Manager(FEMSM) [15]. Both implementations provide different mechanisms for the transmission of sound, and allow the developer to simulate imperfect auditory perception. Besides the transmission of sound, the representation of sound is also explored in concepts like Sonic Virtuality (SV), in which the act of perceiving a sound is translated into a representation of the perceived sound and the Listener's psychological and physical state.

2.1 Region Sense Manager

RSM provides a simplified implementation of sensory perception using spherical regions. The transmission of a signal (which, with this method, can be a sound or a smell), occurs in three distinct phases.

In the aggregation phase, the signal's maximum reach is calculated using the initial signal's intensity and then applying an attenuation value, which determines how much a sound's intensity decreases or how the intensity of a smell decays with the distance. All the characters within that maximum range might be able to perceive the signal.



A two-dimensional view of the RSM spheres, showing a NPC and different intensity regions defined during the testing phase. each representing an area where the sound has a different intensity, and a target NPC.

In the testing phase, every potential Listener is tested wether or not it can perceive the signal. The test attenuates the signal's intensity gradually, at a xed distance interval, defining different regions around the source. Figure 2.1 displays a testing phase in which a target NPC, with a given threshold, is within one of the regions in which it may perceive the signal. The regions with a darker color are regions where the signal's intensity is higher. The test is based on wether or not the signal's intensity, for the region in which the NPC is located, is higher than a threshold value defined by the NPC. If the intensity is higher than the threshold, then the NPC is able the perceive the signal. Other tests may be made during this phase, which could take in consideration possible obstacles between the origin of the signal and a NPC or other environment factors that would affect the signal transmission. If the testing phase is successful for a given signal/NPC pair, information related to this pair is stored in a temporary record queue. The information describes the signal, the target NPC and when the signal should be delivered. The time for delivery is calculated based on the travel time from the signal source to the target,

while keeping in mind the type of signal.

Signals that are stored in the record queue are sent during the notification phase. In this phase, signals are sent to the destination NPCs based on their time for delivery.

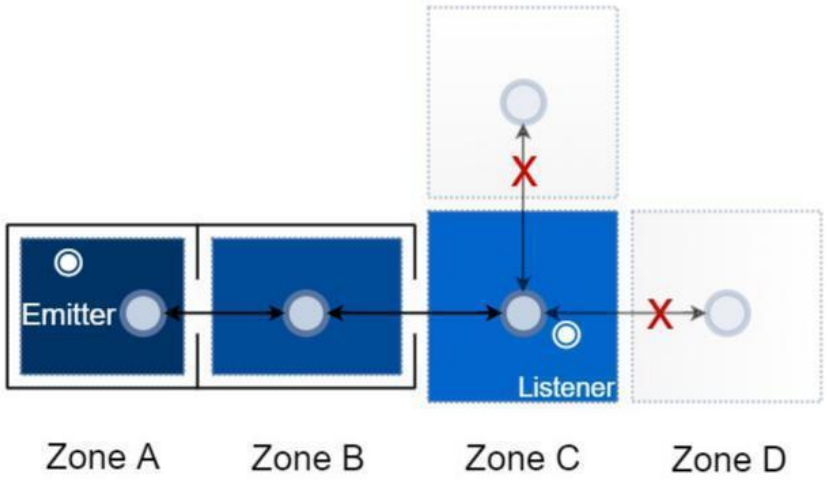
2.2 Finite Element Model Sense Manager

The FEMSM method divides the virtual environment into discrete spaces. The environment spaces are described in a graph, with each node describing the trans-mission properties of that given space, and which are the adjacent spaces. This allows the simulation of propagation of sound between each discrete space to be made by transversing the graph.

When a sound is emitted, it starts at the node representing the discrete space where the emitter is located. The sound is then replicated to the neighbour nodes, and its intensity attenuated accordingly. The sound stops propagating when the intensity of the sound drops below a determined minimum value. If a NPC is at a space where the sound has an audible intensity, the sound is perceived and delivered to the NPC. Figure2.2 represents a graph with ve discrete spaces and a sound being emitted in zone A. The sound travels to zone B, where it is attenuated according to that space's attenuation value. The sound travels to the neighbor space, zone C, where it is attenuated even more. It stops propagating beyond zone C because, upon reaching the neighbour zones, the attenuation renders the sound unperceivable. A NPC is present in zone C and if the sound's intensity is superior to the NPC's defined threshold, the sound is perceived.

This method gives the developer the advantage of replacing complex path planning for sound propagation in favor of a simpler algorithm that transverses the graph.

Zambetta [22] describes a model which resembles FEMSM but revolves around creating a graph each time a NPC needs to perceive the environment. The graph is created based upon the global environment graph, with the NPC's location being



A
FEMS

M graph of map with ve distinct discrete spaces, with

each node representing discreve space's information of the sound transmission attenuation. Sound intensity is represented by the space's background color, with the darkest color representing higher intensity.

the first node and with the graph's reach being based on the maximum distance at which the NPC can perceive.

When there is the need to perceive the environment, a search is made for any audible sounds being emitted within the NPC's perception graph. Each character has a set of predicates in order to simulate different hearing capabilities. This oers some degree of limitation to the hearing capabilities of NPCs, providing a less perfect, but still rather artificial sound perception method.

2.3 Sonic Virtuality

Grimshaw et al [13], and later Garner et al [12], describe a representation of sound for virtual environments that moves away from the classic acoustic representation. Instead of describing sound as a wave, SV describes sound as both an object and an event. In everyday hearing, humans regard perceived sounds as being at the source of their emission, direcly relating them with the object that emits them, negleting the physical properties of sound propagation. This definition is the base for SV. In SV, sound exists within the Listener's brain and body, with the

perception emerging from a interaction between external stimuli and the Listener's psychological state and capabilities.

Sound description, in SV, takes into account both internal and external factors, placing the sound's properties into two distinct groups. The rst, represented by physical factors such as the environment, the Listener's body and brain, is called exosonus. The seconds group, called endosonus, relates to immaterial properties of the NPC's phychological state, composed by its emotions and memories. This description aims towards the categorization of sound for interpretation, taking all the endosonus and exosonus properties into consideration. By determining in which category the sound is inserted, the Listener may select the course of action it should take, for the given set of properties.

2.4 What is missing

Both RSM and FEMSM fail as they deliver a sound in a single frame, in a similar manner a fast moving character might pass past the area of the sound when sound should linger in the air. This problem could be solved in FEMSM as the sound could stay in a node and rapidly decay, but this would consume a great amount of memory and processing power.

RSM is less demanding when it comes to processing power, but it fails to provide a way to calculate the interaction of sound with obstacles.In RSM, sound is perceptible through walls and other objects has they would oer no resistance to the propagation of sound. This could be solved with the implementation of additional heuristics, such as the usage of a raytrace to detect obstacles between source and the destination, and then apply an attenuation according to the number of obstacles and their acoustic permeability.

In addition, RSM, FEMSM and Zambetta's [22] approaches do not consider the psychological filtering performed by the human auditory cortex, nor the physical limitations of the hearing system. In these methods, to determine whether a

given sound has been heard by the NPC, its intensity is checked against a given threshold. However, sound can be characterized by many properties (intensity, frequency, etc...), rendering an intensity-based threshold rather simplistic and, thus, an unrealistic simulation of the human auditory perception.

SV, in comparison to RSM and FEMSM, albeit having a dierent goal, does map sound towards a more complete interpretation, taking into account the psychological state of the Listener. Unfortunately, the way SV maps sound is a step beyond the objective of this work, as it aims for the interpretation of sound rather than its perception, while also displaying an excessive demand for processing power. Taking all this into account, SV's description of sound, as is, ties too closely the environmental, physical and psychological aspects. A simplification of this description can serve as an inspiration for our description of sound and, for future work, could help understand how to better prepare our framework for a more realistic cognitive module.

Chapter 3

Proposed System

3.1 Requirements and specifications

The framework's target users are game developers in need of a way to simulate the transmission and reception of sound by NPCs in the game environment. It should be easy to integrate into currently existing Unity games while providing an adequate set of customization parameters to allow tweaking of how sound propagates and is perceived by NPCs. The customization parameters should relate to the physical and psychological aspects of the perception of sound, as to provide a way for NPCs to perceive sound as humans do.

3.2 Human auditory perception

The transmission of sound to a NPC is just the beginning of the processes that must be modeled. The perception of sound should be affected by the NPC's current state, personality, the action under execution and the current context in which the NPC is present.

Soon after a sound reaches the human ear, the auditory cortex must determine whether the sound is relevant, given the current task [11]. Sounds which are not

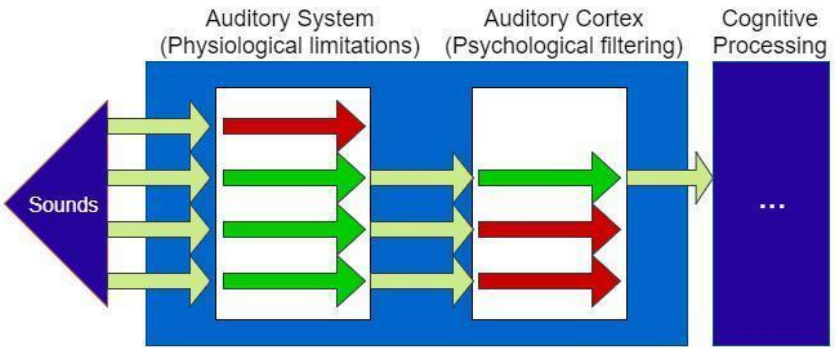


Figure 3.1: Four sounds are emitted and reach the Listener's ear, but only one is perceived.

relevant are considered noise and, therefore, discarded from further processing. Therefore, the Listener's capability of perceiving a given sound is constrained by the state of its physical (Auditory System) and psychological (Auditory Cortex) aspects. Only by analyzing the sound, while taking both aspects into consideration, can a sound be properly perceived or discarded.

The Listener may also identify the source of the sound, which is a significant feature to consider in video games where NPCs and player share a relationship tree [1]. Such relationship tree would describe the strength of the bond between each character in the game. The use of a relationship tree would enable each NPC to possibly identify the emitter of a given sound, and if the Listener has a strong bond with the emitter, the Listener could give greater relevance to that emitter's sounds. We all often observe this phenomenon in our daily lives, such as when we feel higher hearing acuity towards familiar voices or cry of our children.

Not all humans have the same hearing acuity. With age, the range of frequencies a human can perceive gets narrower. This narrowing may be aggravated by hearing trauma caused by exposure to loud or continuous noises [5]. In these cases, the sound reaches the ear but the auditory cortex is unable to process it. Genetics also take part when determining the human listening acuity.

The identity of the source of the sound, the frequency in which the sound is emitted and other relevant sound properties of the emitted sound should be described for the Listener, as the framework does not analyze sound files for the purpose of

sound perception. Figure 3.1 shows an example in which 4 sounds are emitted in the environment. All these sounds reach the Listener's auditory system, but due to physical limitations such as hearing loss, one of these sounds is not perceived. After getting through the auditory system, the auditory cortex, which is affected by the Listener's emotional state, cognitive load and other psychological traits, filters out two of the sounds, thus rendering the remaining sound as perceived. The Listener may then react accordingly to the perceived sound. A Listener may perceive multiple sounds at once, but the auditory cortex rules out sounds that are not relevant to the task at hand. The auditory cortex does so by attributing higher relevance to the sound that is the most relevant for the current context and, thus, processes it and discards the remaining [11].

In the proposed framework, each of the physical and psychological processes are simulated through an array of sound filters, with each individual filter analyzing different aspects of the description of a sound. Each of the filters attributes a value of salience, which describes, for a given sound, how perceptible the sound is for that specific simulated process. All the filters contribute towards a nal value of salience, which will then determine how perceptible the sound is to the NPC. The greater the salience, the more perceptible it is. A minimum threshold may be dened by the developer, which determines the value below which sounds are not perceived by the NPC. If a sound has its nal salience below the defined threshold, it is discarded.

3.3 Framework Overview

The framework is divided intro three layers, each of them represents a different module that contributes to the transmission and processing of human like auditory perception, in games developed in Unity(see Fig. 3.2):

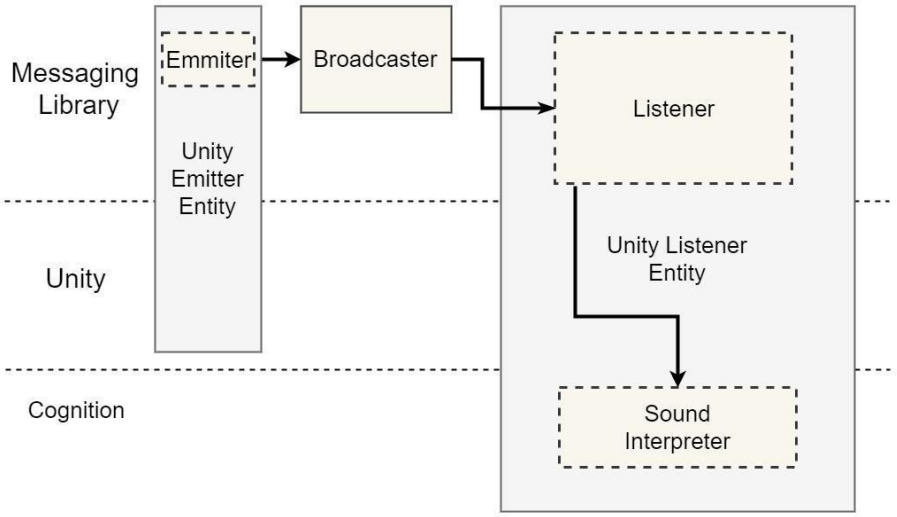


Figure 3.2: An overview of the interactions of the components of each layer.

1. Messaging Library is responsible for the transmission of messages from Emitters to Listeners. The interface of the communication between the Emitters and Listeners is dened in this module, which also provides the structure that describes a sound message.

2. Unity Integration Module is the game engine which ties together the Messaging Module and the Cognition Module into the virtual world.

3. Cognition Module defines the behavior of the NPCs, which receives the auditory stimuli information and, then, provides the set of actions to attain a given behavioral objective.

3.4 Messaging Library

All the operations regarding sound messages (emission, transmission, perception and delivery) happen in this layer. The following sections will describe the main classes that participate in the aforementioned operations. The interaction between each class is represented in Figure 3.3.

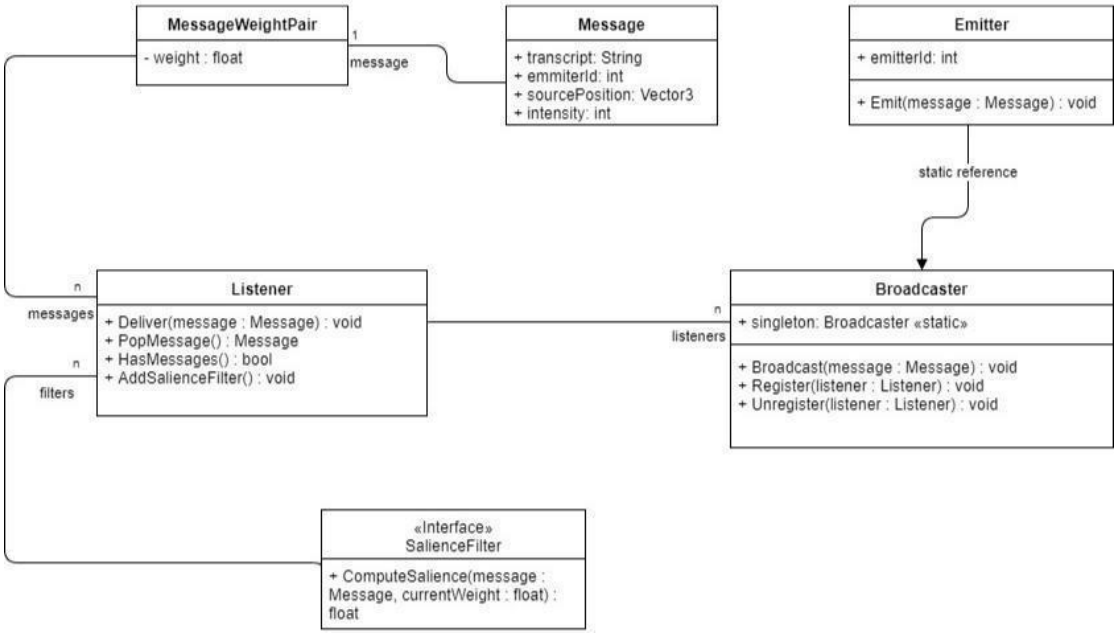


Figure 3.3: Classes interaction diagram for the Messaging Library Layer

3.4.1 Sound Description

Whenever there is the intention of emitting a sound, a correspondent sound message description should be created. The sound description describes the emitted sound and provides a way to announce the emission of a sound to potential Listeners. It contains some properties that oer a description of the sound and its properties:

Intensity of the sound, in dB, represented by an integer;

Average frequency, in Hz, represented by an integer;

Text describing the content or the type of sound (e.g. dialog transcript; footstep on metal oor; shout), represented by a String;

Flag identifying that the sound message represents a continuous sound, a noise sound message;

The identification of the Emitter, with a unique integer for each Emitter; Location in which the sound was emitted, represented by 3D coordinates;

A sound description may also represent a noise sound, which, unlike a sound description of a regular sound, is not delivered to the Listener to be perceived and processed, but to affect the perception of other sounds through the usage of filters (see Section 3.5.3).

3.4.2 Emitter

An Emitter is able of emitting sound messages, being the one responsible for creating and emitting sound messages. The Emitter should describe the content and properties of the sound message. It should identify itself and its position when the sound message is created.

When an Emitter is created it is associated with a Broadcaster. It is through the Broadcaster that the Emitter spreads its sound message to potential Listeners.

3.4.3 Broadcaster

The Broadcaster's role is to receive a sound message and deliver it to registered Listeners. The Broadcaster is not responsible for the description of the message and, therefore, should not modify the content of the sound message description.

The default Broadcaster's behavior is to deliver the sound message to all the registered Listener. This behavior could be easily replaced with the integration of RSM (see section 2.1) or FEMSM (see section 2.2), thus delivering the message only to Listener's which meet a certain criteria. These criteria can be based on the distance between the Emitter and the Listeners, while taking into consideration the attenuation of sound caused by obstacles between them. A method that simulates the propagation of sound using a navigation mesh could also be used, this way the travel distance taken into consideration would not be a straight line. In both examples, if the distance would attenuate the sound so much as to make it imperceptible, the Broadcaster has no need to deliver the sound message to the target Listener.

3.4.4 Listener

A Listener has a queue of received messages and a queue of perceived messages. When it receives a sound message from a Broadcaster, it places it in the received messages queue. In the game engine's update cycle all the messages in the received messages queue go through the sound salience filters owned by the Listener (see subsection 3.5). If the message is marked as perceived by the filters, it is placed in the perceived sound messages queue.

After a sound is marked as perceived, it is a candidate to be processed with the purpose to affect the behavior of the Listener, by being delivered to the Cognition module.

3.5 Messaging Library Filters

Filters are associated to a Listener. Each filter describes a way of calculating the salience of the sound message by analyzing its parameters. If after the application of all the filters the message has a non-positive salience, it is regarded as not perceived by the Listener and, thus, discarded. If a sound has a positive salience, it is marked as perceived.

3.5.1 Hearing loss Filter

The purpose of this filter is to simulate the decay in the hearing range cause by aging. This condition is called Presbyacusis and, according to Robinson et al [8], the decay of hearing capabilities can be described through an equation, which is also used in the standard ISO 7029:2017 [14]. The threshold for hearing loss can be modeled by the equation:

$$H = a(N \quad 18)^2$$

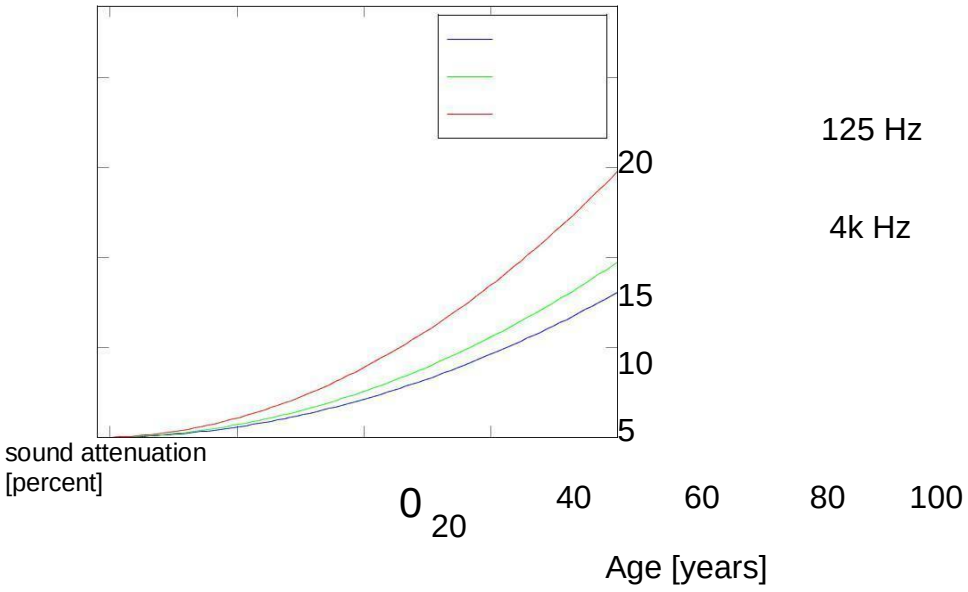


Figure 3.4: Hearing Loss attenuation percentage for the frequencies of 125 Hz, 2k Hz and 4k Hz.

Frequency (kHz)	Coecient
0.125	0.00120
0.25	0.00120
0.5	0.00124
1	0.00128
1.5	0.00132
2	0.00145
3	0.00160
4	0.00220
6	0.00240
8	0.00300
10	0.00380
12	0.00490

Table 3.1: Hearing loss coefficients for each evaluated frequency.

Where N refers to the age of the Listener and a refers to a coefficient which is

directly dependant on the frequency in which the sound is emitted. The values for the coecient may be consulted in table 3.1 and provides coecient values adapted from the table of coefficients provided by Robinson et al [8].

The plot shown in Figure 3.4 shows evolution of the coecient of hearing loss for the frequencies of 125 Hz, 2k Hz and 4k Hz.

20

3.5.2 Relationship Filter

The input for this filter is a dictionary that represents the level of importance of specific entities. For a given Listener, all the relevant Emitters have a key pair entry of the Emitter's id and its coecient of importance. The level of importance is represented as Not Important, Neutral or Important, with the sound message having its salience attenuated by, respectively, 20%, 10% or 0%.

As an example, a given Listener which would hear the voice of its brother at the same time as the voice of a stranger, would nd more important and relevant the voice of its brother, therefore attributing its relationship with its brother the level of importance "Important". The voice of the stranger would have the suitable value of Neutral, as the stranger shares no relationship with the Listener. These values would make the sound messages of the brother more salient than the sound messages of the stranger. A default value of importance can be attributed in the Filter's parameters, for Emitters that do not have an entry in the dictionary of the Listener, like the one attributed to the stranger from the previous example.

3.5.3 Noise Filter

Every Listener entity has a Noise Filter attached to it by default. This Filter's responsibility is to compute the salience of a given sound message according to surrounding sound messages that were emitted as noise. To compute the contribution of noise, each noise message attached to the Listener has its intensity calculated according to the distance to the source, approximating a Gaussian curve. The interference of obstacles is not contemplated. When a sound message has an intensity inferior to the noise's intensity, it has its salience reduced.

This filter can also take into account sound messages which were sent but not agged as noise. This way, every sound may be regarded as possible noise, as happens in real life but, for the current implementation, only sound agged as noise are considered for salience computation.

Table 3.2:

Proposed System

Focus

Level	Attenuation (%)
1	0
2	25
3	50
4	75
5	100

The Focus Filter attenuates the salience of the sound message depending on the focus the Listener has in the current task it is performing.

The salience of a perceived sound is attenuated differently whether the noise intensity is superior or inferior to the salience of the received sound. If the noise as a superior intensity, the received sound message has its intensity reduced by the difference between the noise's intensity and the received sound salience. If the noise as an inferior intensity, the difference between the received sound salience and the intensity of the noise is divided by 5, value which is then subtracted to the salience of the received sound.

Due to the filter's dependency on the management of the noise messages received by the Listener, the actual implementation of the filter can be found inside the Listener's class.

3.5.4 Focus Filter

As performing a given task demands a varying degree of concentration, this filter can reflect the amount of focus put into a task. The coecient of focus, ranging between 5 levels, with 1 reflecting a deeply focused Listener and 5 representing a Listener that is very alert to the surroundings. The salience value is affected in accordance to the values presented in Table 3.2. Besides these coefficients, the filter may also be parameterized with a coecient that denes how much does these filter affects salience, by directly multiplying the attenuation values by the coecient, before computing the salience.

3.6 Sound Message Transmission and Perception

3.6.1 Sound Message Emission

The process of transmission and perception of a sound message to potential Listeners starts at the Emitter. The Emitter is responsible for creating the description of the sound it wishes to emit. It should ll out the details of the properties of the sound. After creating the sound message, the Emitter should deliver the sound to the Broadcaster. The broadcaster then transmits the sound message to registered Listeners. The message may be delivered to all the registered Listeners or to only some, depending on the Broadcaster implementation being used (more details at section 3.4.3). Figure 3.5 shows the Broadcaster spreading the same sound message, delivered by the Emitter, to all the registered Listeners.

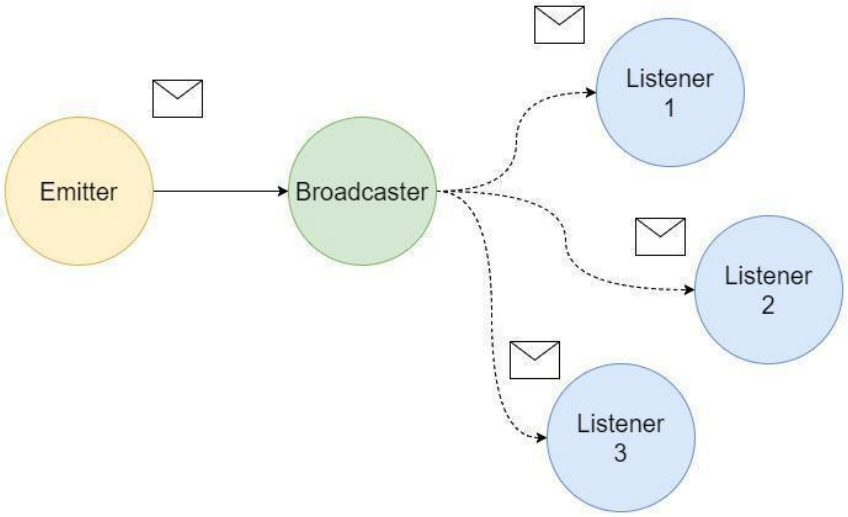


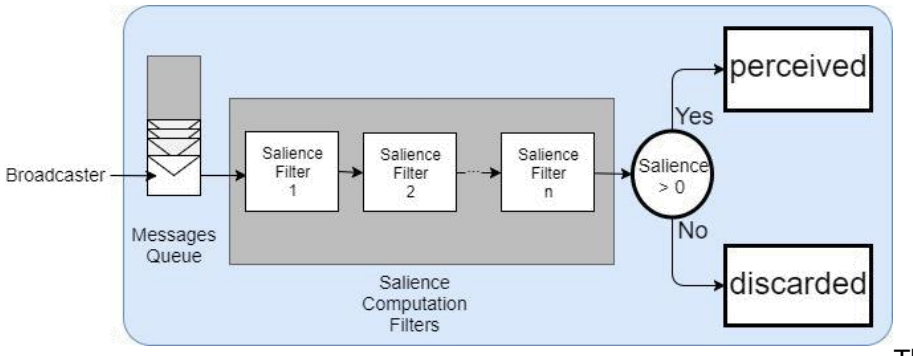
Figure 3.5: Emission and broadcast to the Listeners of a sound message.

3.6.2 Sound Message Reception

Upon receiving the sound message, the Listener stores the message in the received messages queue. These messages are stored in this queue until they have their salience calculated.

Figure 3.6:

Proposed System



The Listener dequeues the received sound messages and computes their salience with the filters. If the salience is positive, the sound message is marked perceived.

3.6.3 Noise Sound Reception

If the message is marked by the Emitter as being a noise sound message, it is also stored in a noise queue, making it a representation of a continuous source of sound. Being a continuous source of sound, this means the sound affects the ability of the Listener to perceive other sound messages, as they compete for the Listeners attention. The noise messages present in the queue can then be used by a filter which would compute how much the noise messages affect the Listeners (see Section 3.5).

When a noise message from a specific Emitter already exists, it is replaced by the new one. If the noise message has an intensity of 0, it represents that the Emitter has stopped emitting the noise, therefore causing the noise message to be removed from the noise messages queue.

3.6.4 Salience Computation

The salience computation happens in every update cycle of the game engine. Each filter affects the salience of the sound message, taking into account the sound message's description, the Listener's state and the surroundings. Figure 3.6 shows a sound message being dequeued from the received messages queue, which then passes through all the filters assigned to the Listener. If the nal salience is greater than the minimum threshold defined in the creation of the Listener, the message

is regarded as perceived, if not, it is discarded. When a message is marked as perceived, it is placed in the perceived sound messages queue.

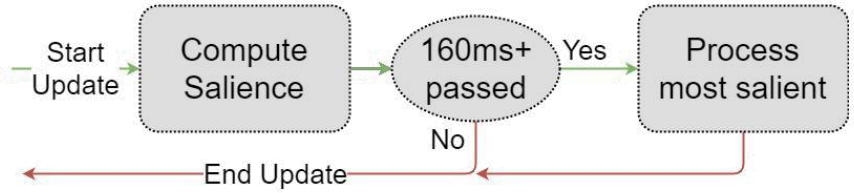
3.6.5 Behaviour Selection

Figure 3.7 shows the ow of the update cycle that is composed by the task that computes salience (salience cycle) and task that processes the perceived messages (processing cycle). The processing cycle, in which sound messages are removed from the perceived queue, is only executed every 160ms. This interval resembles the average reaction time of young adults [20]. This time interval allows multiple messages to be accumulated in the perceived queue. In each of the processing cycles, the sound message with the greatest salience is considered to be processed to affect the behavior of the Listener. By the end of the processing cycle, the perceived queue is cleared, as the relevant sound message was processed and all the others are considered noise because of their lower salience. This way we take into account the filters that may dene that, in spite of the perception of multiple sound messages, the ones with the least salience are discarded as they are not relevant for the presented context. This behavior may be modified by extending the Listener and redefining the processing cycle.

The message with the greatest salience is then passed to the behavior selector, which interprets the information contained by the sound descriptor. The selection of a behavior is based on the current action being done, the NPC's state and its goal. The message description may prompt an action or goal different from the current ones. In a game in which the NPC is defending a building (the goal), it might be walking around, patrolling (the action). If the NPC perceives footsteps, the action might modify so that the NPC goes check for the sound source. If the NPC is injured and perceives gunshots, he might modify its goal to protect its well-being and ee to a safer place, discarding the goal of protecting the building.

Figure 3.7:

Proposed System



In each update, the received messages get their salience computed.

If in a given update, at least 160ms passed since when the last message was processed, the most salient perceived message is sent to the behavior selector

3.7 Unity Layer

This layer's purpose is to connect the Messaging Library with Unity's game engine, by using the game engine's lifecycle to manage the Messaging Library's instances and perform their actions. Such actions might be the creation of a sound message when a sound le is played or ask a Listener to provide a perceived sound message. The interaction between the different created classes might be observed in Figure 3.8.

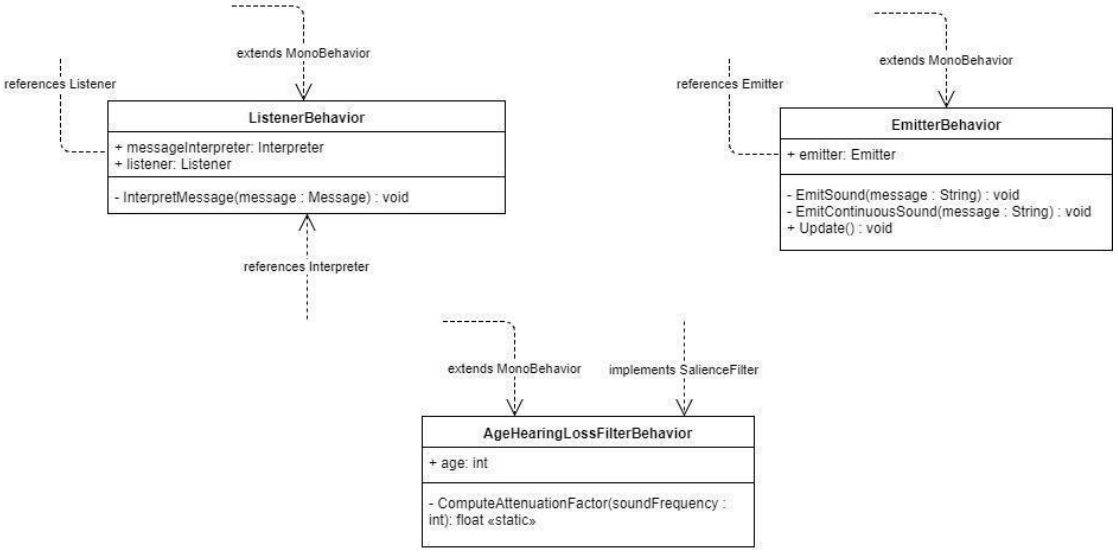


Figure 3.8: Classes interaction diagram for the Unity Layer

3.7.1 Unity Engine Scene

In the Unity Engine, the game environment can be described in an object called Scene. A scene is composed by static GameObjects, which are usually terrain, buildings, ora and other decorative objects. There are also dynamic GameObjects which can interact with other GameObjects, thus having a more active role in the game environment.

GameObjects visual aspect and capabilities are described by Components. Components can range from text or 3D object renderers to light and sound Emitters. Every GameObject is associated with a Transform Component, which denes the GameObject's current position, rotation and scale.

When it comes to NPCs, these are implemented as dynamic entities and they have components called scripts associated to them. These scripts, also known as MonoBehaviors, are pieces of code which can interact with all of the GameObject's components. MoonoBehaviors may also communicate with other GameObjects and their components, allowing NPCs to scan the environment and to interact with it.

MonoBehaviors most important callback methods are Start and Update. When the entity is created, all of its MonoBehaviors have their Start methods called. In these methods, the MonoBehavior can setup all the needed references to existing GameObjects and Components and initialize 3rd party libraries. The Update method is called at every frame. Custom behaviors for the GameObjects are implemented here.

3.7.2 EmitterBehavior

To give a GameObject the ability to emit sound messages, a MonoBehavior called EmitterBehavior was created. When the EmitterBehavior is created, it creates an instance of Emitter from the Messaging Library. The Emitter is automatically associated to the default Broadcaster, but a custom one may be provided.

The GameObject that has an EmitterBehavior must have an Audio Source Component associated to it, so that it can play sound les ingame. The Unity IDE will issue a warning message if no Audio Source Component is provided with the EmitterBehavior, because that way the player would not be able to hear the sounds.

During the Update cycle, the NPC may emit a sound, which is composed of the following steps:

1. The sound description is created (see Subsection 3.4.1);
2. The sound is played through the Audio Source, so that the player can hear the sound;
3. The sound description is sent to the Broadcaster associated to the Emitter;

3.7.3 ListenerBehavior

The ListenerBehavior allows a NPC to perceive sound messages. In the Start callback, the ListenerBehavior creates a Listener instance from the Messaging Library. The Listener is then associated to the default Broadcast instance, but as for the Emitter, a custom Broadcaster may be provided as well. The ListenerBehavior also searches the NPC's components for FilterBehaviors (see Subsection 3.7.5, which describe salience filters from the Messaging Library, and saves a reference to them.

During the Update cycle, the ListenerBehavior surveys the Messaging Library's Listener instance for available messages. When the survey is done, the Listener dequeues the sound messages and calculates their salience. If any of the sound messages are perceived, they are returned to the ListenerBehavior. The Listener-Behavior then passes the message to the Cognition layer, which is responsible for checking the sound message description and select a suitable action for the Listener to perform.

3.7.4 ControllerBehavior

The ControllerBehavior has access to all the NPC's components and is allowed to manipulate the NPC's model current animation, shader, texture and position. The ControllerBehavior has access to the component which allows the NPC to navigate

 28

through the virtual world, with automatic path planning. Other MonoBehaviors communicate with this component to request modifications regarding the NPC's location and look.

During the Update callback this behavior modifies the location of the NPC if a target destination is set. The Update stops modifying the location when the NPC reaches the target destination.

3.7.5 FilterBehavior

A filter should be associated with a Listening GameObject through a FilterBehavior. The FilterBehavior may have public properties, which can be modified inside Unity's IDE, thus facilitating the parameterization of filters.

Messaging library Filters are created and parameterized in the Start method, according to the public properties' values.

The Update callback might be used to update the Filter's properties. Such behavior might be suitable for filters that depend on other Components values, such as a filter that depends on the distance of the Listener from the source position of the emitted sound.

3.8 Cognition

The Cognition layer receives sound messages that are perceived by the listening NPC. It is implemented in Unity through the CognitionBehavior, which is a MonoBehavior component that receives a sound message description and prompts modifications to the NPC's behavior. This layer's classes interactions may be observed in Figure 3.9

When the CognitionBehavior is created, it obtains an instance of the Controller-Behavior, which is responsible for manipulating the NPC's model, position and sounds. The provided cognition behavior will then create a Finite

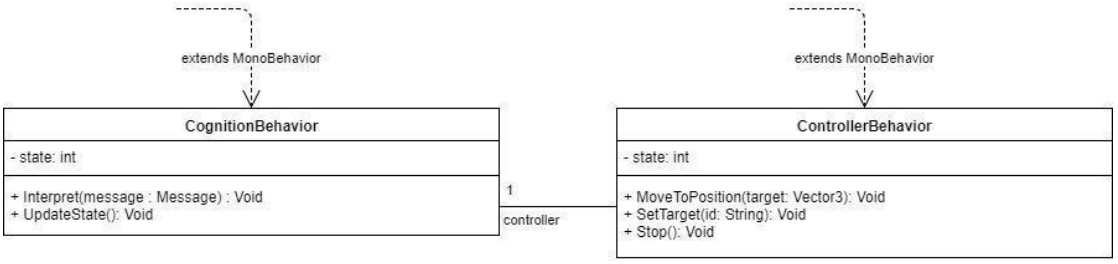


Figure 3.9: Classes interaction diagram for the Cognition Layer

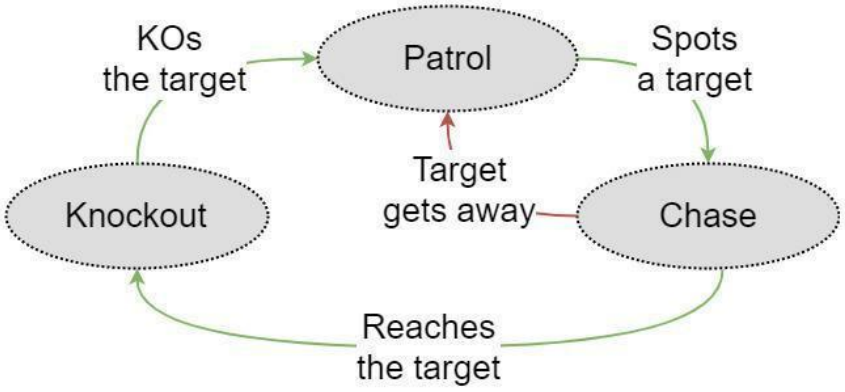


Figure 3.10: Example of a Finite State Machine in which the NPC switches between three states: patrol, chase and knockout

and start the default behavior. A Finite State Machine denes a set of states, in which the NPC performs a certain action. During a certain state, the NPC may move to another if the current context and the NPC's current state fulll a certain criteria.

Figure 3.10 shows a Finite State Machine with three different states. In the example, the NPC starts in the patrolling state. If it perceives a target, it changes to the chase state. When it reaches the target, it proceeds to the knockout target state and, after knocking out the target, resumes the patrolling state. If the NPC fails to reach the target because the target is too far, it may give up and return to the patrolling state.

In the Update cycle, and according to the current state or new state, the CognitionBehavior makes the NPC continue with the current task or modifies it to a new one, through the ControllerBehavior.

30

When the CognitionBehavior receives a new sound message, and depending on the sound message's description, it may queue a new state or update the NPC's information of the Scene. The modification of the state occurs in the Update cycle, not when the message is processed.

Figure 4.1:

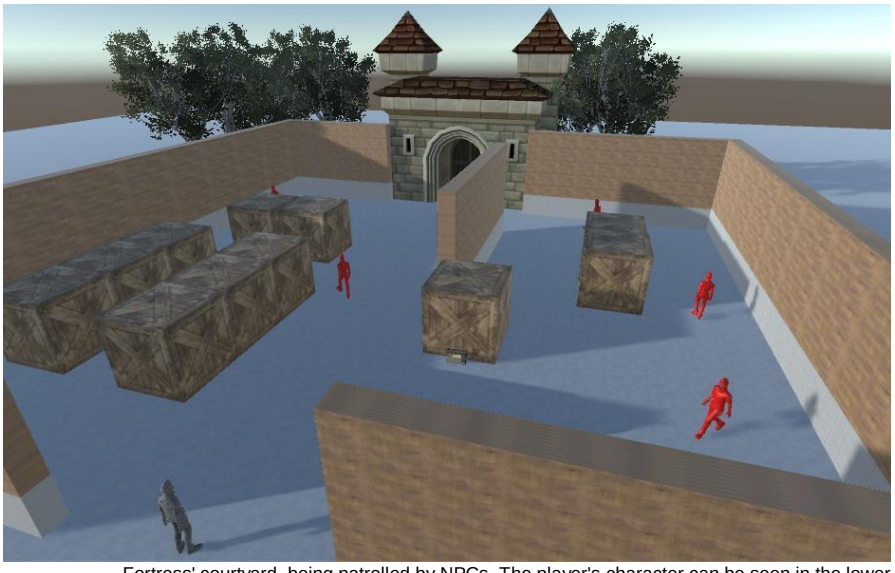
Chapter 4

Case Study

4.1 Sample Game - Fortress

In order to test out the integration of the framework, a simple FPS (First Person Shooter) game, Fortress, was created. The game takes place in a fortress, which is patrolled by NPCs (see Fig. 4.1).

The game was developed in Unity's IDE, version 2017.2.0f3, and the scripts were written in MonoDevelop. The game was deployed as a Windows Standalone game,



Fortress' courtyard, being patrolled by NPCs. The player's character can be seen in the lower left corner, at the starting point.

Figure 4.2: An exclamation mark is displayed above the NPC whenever the NPC detects the presence of the player.

but can be easily exported to other platforms, with some modifications regarding the player's controls.

The game was developed independently from the framework, in order to test how the framework integrates on an already developed game. Each NPC and the Player were adapted in order to become, respectively, Listener and Emitter, to test the emission, transmission and perception of sound. A few elements were introduced in the scene in order to complement the game's features, so that the framework can be used to its full potential. On of this examples is a radio, which was introduced so that the noise filter can be tested.

4.2 Game Logic

The objective of the player is to cross the courtyard of the fortress and reach the fortress's gate without getting caught by the patrolling NPCs. The NPCs patrol the courtyard while being alert for the present of intruders. The main role of NPCs is to prevent the player from reaching the fortress' gate, by detecting and catching it. When the NPC detects the presence of the player, it displays an exclamation

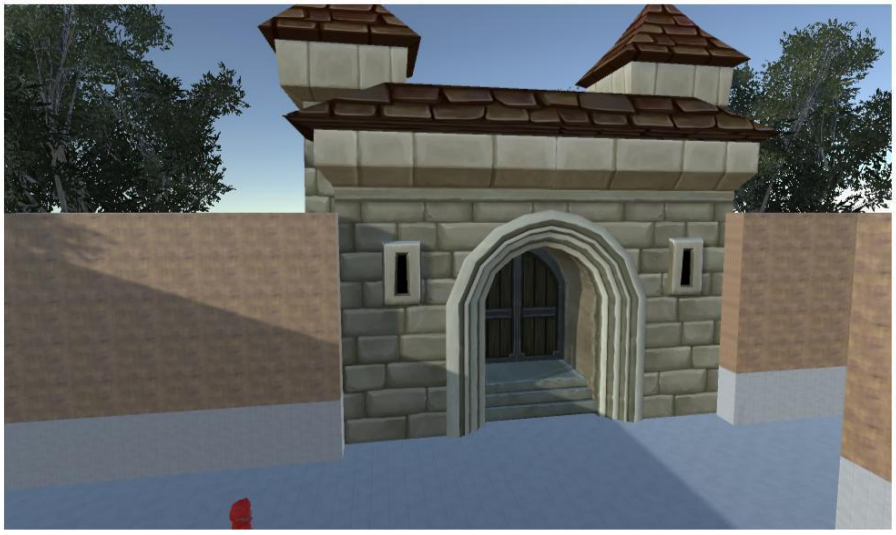


Figure 4.3: The used Gate and trees models. The player must reach the game in order to beat the NPCs.

mark on the top of its head (see Figure 4.2). If the NPC touches the player, the game is over.

4.3 Game Art

To simplify the development of the game Fortress, some dependencies from Unity's standard assets were added. The project depends on Unity's characters package, which exposes scripts that facilitate the control of the player and enemy characters. This dependency also provides a model for the enemy character, with all the available animations, and sounds for the player's footsteps.

The game also uses the Tree(Mediterranean) package [10] for the tree model and the Fortress Gate for the Fortress' gate model [21]. Both can be seen in gure 4.3.

4.4 Player

The player emits sound by walking around the map and jumping. When the player jumps, a sound is played when the player leaves the ground and another one whe

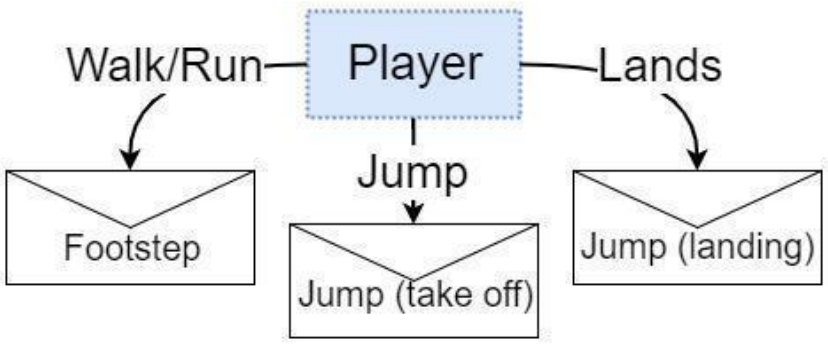


Figure 4.4: The player emits sound by moving around and jumping.

he lands (see Fig. 4.4). The player can also run, emitting louder footsteps, repre-sented by emitting the sound messages of those footsteps with a higher intensity value.

The main Player's component is CustomFirstPersonController, which handles input to move the player around by using the Character Controler. In addition, the CustomFirstPersonController communicates with the PlayerEmitterBehavior in order to play sound les for the footsteps and jumps through the Audio Source.

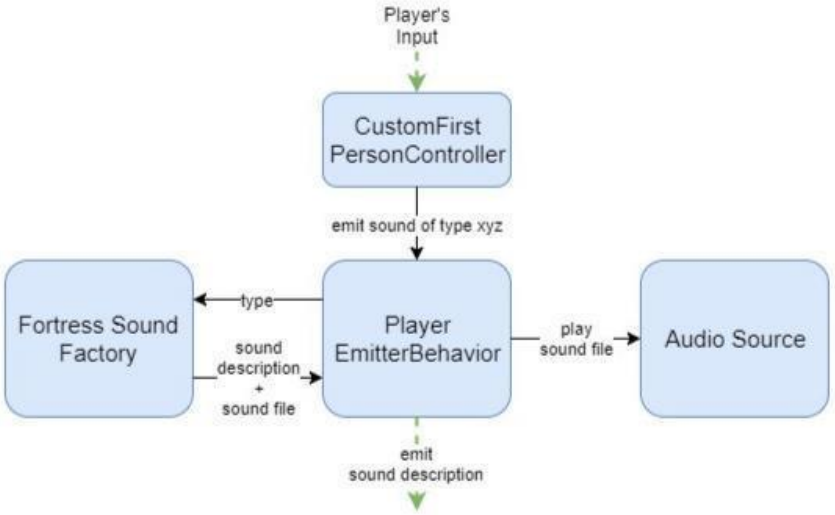
4.4.1 Emission of Sound Messages

To integrate the framework, every sound emitted by the player, through the sound engine, should be led by the creation and emission of its representation. The representation should be made via the emission of a Sound Description (see Section 3.4.1 for a list of the properties).

To emit sound messages, the player controller must be associated with the EmitterBehavior, which communicates with our Framework through an Emitter object. The EmitterBehavior exposes an Emit method to send the sound message to the broadcaster, but its responsability could be extended so that it aids in the creation of the sound message's related to the footsteps and jump sounds.

An utility class, FortressSoundFactory, was created to create a sound message for a given sound, to reduce boilerplate code related to the creation and preparation of a

Figure 4.5:



..

e FirstPersonController hands the sound emittion responsibility to the PlayerSoundEmitter, which uses the game's utility class to obtain the sound les and sound descriptions objects.

sound message. This class also exposes an enum which associates a sound le with a message description, allowing the EmitterBehavior to not only emit sound mes-sages, but also to play sound les. This way, the FirstPersonController hands all the process of sound and sound description emission to a custom implementation of the EmitterBehavior, the PlayerEmitterBehavior (see Fig. 4.5).

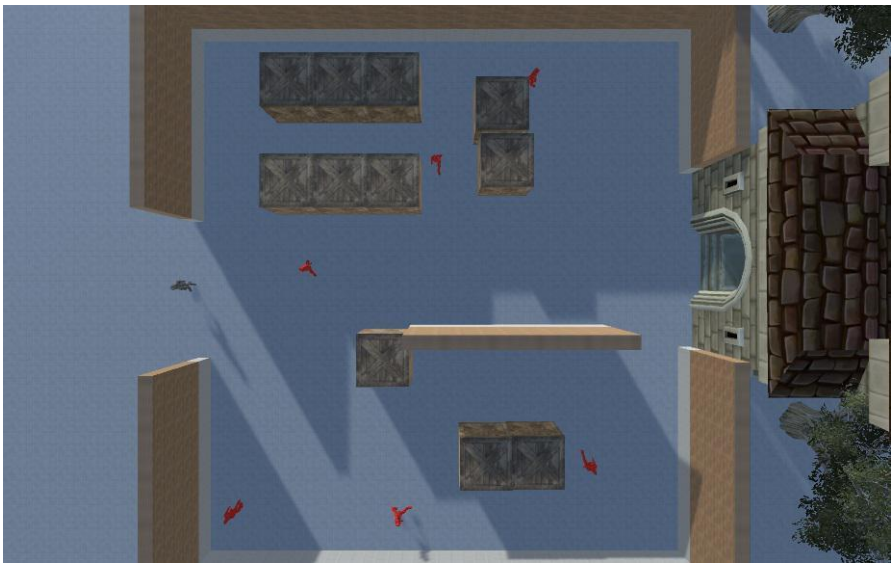
4.5 Non Player Characters

Scattered around the courtyard of the fortress are six NPCs (see Figure 4.6. Each NPC can be either standing still, patrolling along a certain path, walking to the source of a sound or chasing the player.

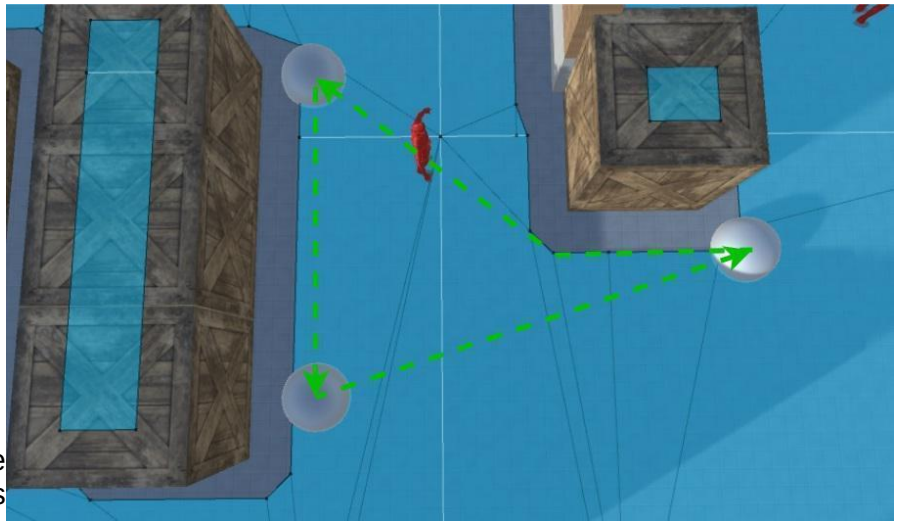
To handle the behavior and the transitions between each state of the NPC, a Finite State Machine was implemented (see Fig. 4.8). The NPC begins the game in the patrolling state, and moves between a set of waypoints it has dened. Figure 4.7 shows the Scene as seen in Unity's IDE, where we can see the waypoints represented as white spheres, which are not visible when in-game, with the green lines display the path the NPC does when patrolling between these points. The

Figure 4.7:

Figure 4.6:



The NPCs are scattered across the courtyard, near their dened patrol routes. The player can be seen near its starting position.



The NPC's patrolling path, with is dened by the placement of three white spheres. The blue oor is displayed the navigation mesh, which denes the area where the NPC may freely walk through. The green arrows indicate the path the NPC makes while patrolling across these waypoints.

38

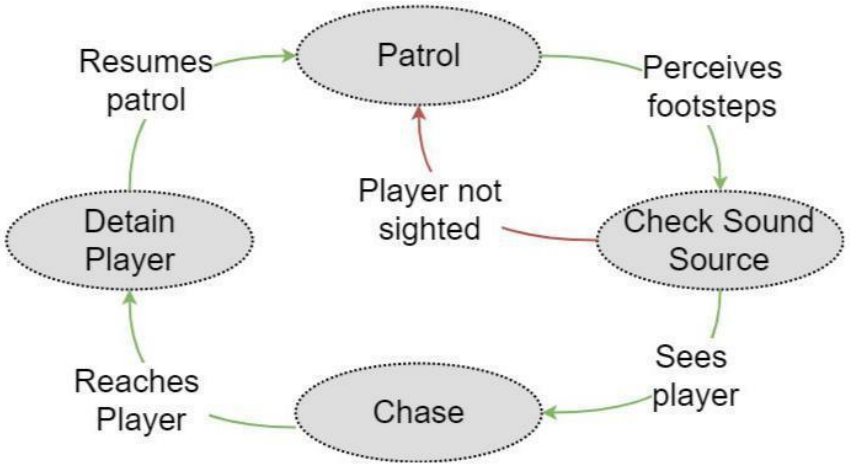
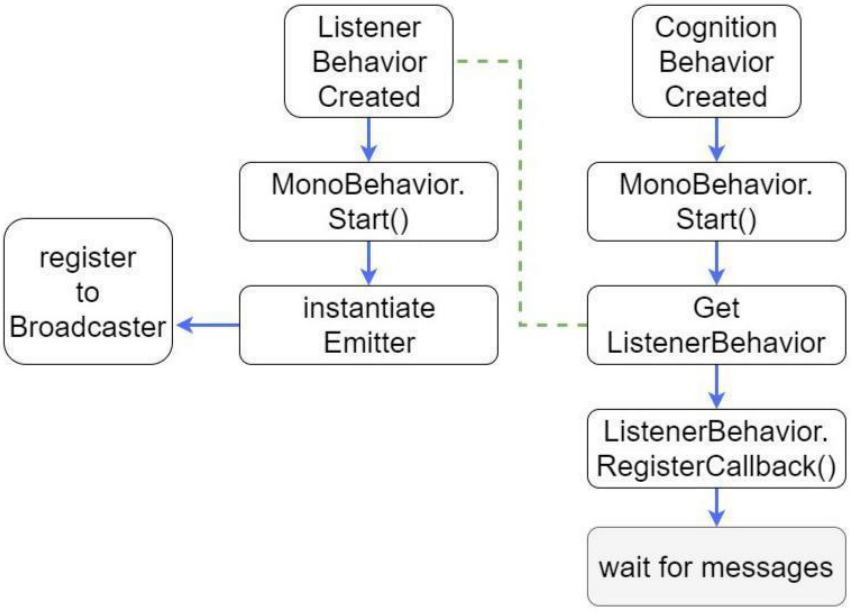


Figure 4.8: Patrolling NPCs' Finite State Machine

blue floor shows the navigation mesh, which determines the places where the NPC can walk on, by using Unity's navigation agent for automatic path planning. If the NPC perceives a sound coming from the player during the patrolling state, it transitions to a state where the NPC checks the source of the sound, heading to the location at which the sound was emitted. If the player is not seen during this state, the NPC resumes the patrolling state, otherwise the NPC starts chasing the player and tries to catch it. If the NPC succeeds in this, the player loses and must start over.

4.5.1 CognitionBehavior

The CognitionBehavior implements and manages the Finite State Machine, making this MonoBehavior responsible for controlling the NPC's actions. Through the ControlBehavior, it has control over the NPC's model and the NavigationMeshAgent, which allows the NPC to navigate the map with automatic path planning, with base on the navigation mesh seen in Figure 4.7. This behavior is also responsible for surveying the environment for the events defined in Figure 4.8, which affect the NPC's current performed task.



The ListenerBehavior creates the Emitter and register it for broadcasts. Afterwards, the CognitionBehavior obtains the ListenerBehavior instance and registers itself for the reception of sound messages.

4.5.2 Listening NPC

The NPC's sight is disabled during the Patrol state, making auditory perception the only source of information regarding the player's position. This ensures that the only way the NPCs can perceive the player is by using our framework. With this in mind, and to allow NPC to perceive emitted sound messages, a Listener-Behavior must be attached to the NPC's GameObject.

As show in Figure 4.9, the ListenerBehavior of a given NPC automatically creates and registers a Listener to the Broadcaster during Unity's Start method, which runs before the game's update cycles begin. Afterwards, the CognitionBehavior runs its Start method, in which it obtains the NPC's ListenerBehavior and then registers itself as a callback for any perceived sound messages. The order upon which these MonoBehaviors Start() methods are called is ensured by Unity's Script Execution Order Mechanism, in which we defined that ListenerBehaviors scripts must be created before CognitionBehavior scripts. After the registration as a callback, whenever a sound message is perceived, the CognitionBehavior gets the message through the callback's interface method.

40

Figure 4.10: The HearingLossFilterBehavior allows the developer to manually input the age of the Listener.

By default, the NPC will be able to perceive every sound emitted, as it has no attached Salience Filters. In order to add Salience Filters to the Listener, MonoBe-haviors that extend FilterBehavior that are attached to the NPC's GameObject are registered as filters for that NPC's ListenerBehavior. By attaching the filters as components of the NPC, the ListenerBehavior is able get them all during its initialization method and then passes them to the Listener. The Salience Filters arguments can be customized in the IDE, as seen in Figure 4.10, where the developer can choose the age of the Listener through a simple input eld (see Section 3.7.5).

4.5.3 Sound Processing

The CognitionBehavior can now perceive sound messages emitted by other entities, but must provide a way to handle the sound messages and act accordingly. Taking into consideration the objective of the enemy NPCs, the only sounds that matter are sounds emitted by the player. A sound message that belongs to the player contains the player's id to identify it as the Emitter. This solution provides a simple and effective way to perceive the player's emitted sounds.

When a message is received by the CognitionBehavior, it takes into consideration the current state the NPC is in. If the NPC is already chasing the player or checking the source of a sound, it ignores sound messages, as it is cognitively involved in one of those tasks, and therefore cannot spare its attention. If, on the other hand, as seen in Figure 4.8, the NPC is patrolling and perceives the sound of footsteps, it heads to the place where they were perceived.

Figure 4.11: The radio is a constant source of noise, affecting the NPCs ability to perceive sound.

Given the need to test how the existence of noise affects the perception of the NPCs, a radio (see Figure 4.11) was added into the game, which emits sound messages considered noise. NPCs that are closer to the radio will have their ability to perceive sound reduced. The closer the NPCs are to the radio, the more likely they will miss the sounds emitted by the player. This is handled automatically by the Noise Salience Filter, which takes into account all the sounds marked as noise (see Section 3.5.3).

4.5.4 Other sounds and communication between NPCs

In another setting, different sounds could be handled without looking at the Emitter's id, but instead at the type of sound. If there were guns in the game, and if a given NPC did not own a gun, the sound of a gunshot could be handled as a danger sign, and thus trigger the NPC to ee from its source.

Besides being Listeners, NPCs could be Emitters as well. This would give them the ability to perceive sound messages and to emit. In the context of the game Fortress, the NPCs could shout out to the others upon having visual confirmation of the player, making them rally together to catch the player, or could shout a warning to a particular NPC, to have it move to a safe place.

42

4.6 Custom Broadcaster

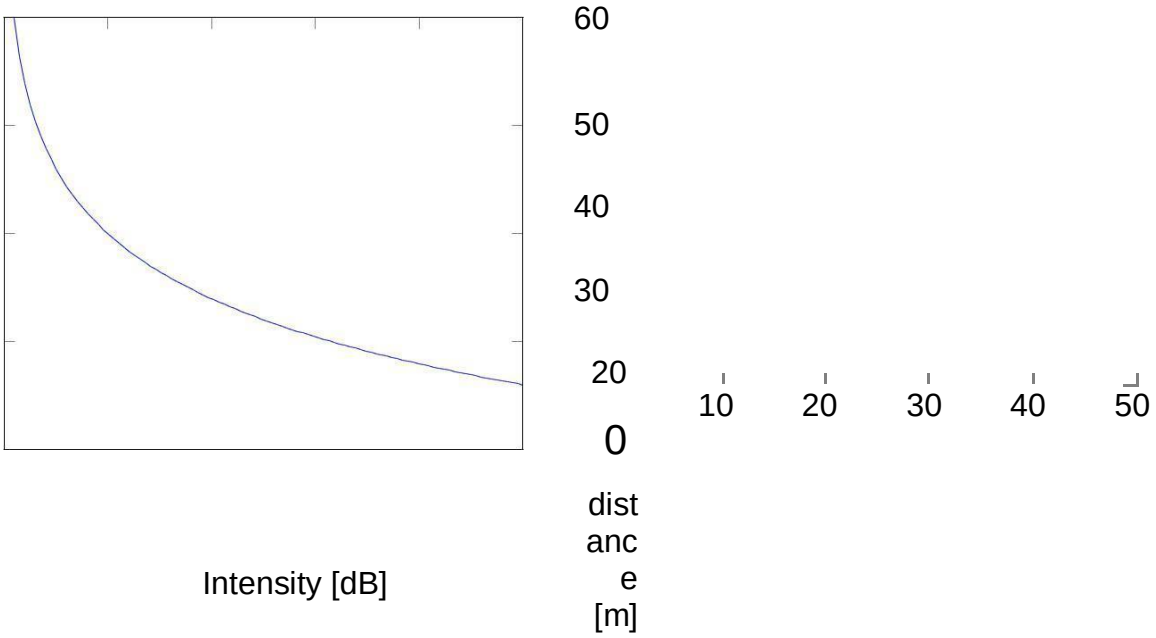
With the default Broadcaster, every Listener gets every emitted message. This happens as the default Broadcaster delivers the sound messages to all the registered Listeners. In order to have only the Listeners that are close enough to the source, perceive the sound message, a custom implementation of the Broadcaster was made, called FortressBroadcaster.

When FortressBroadcaster receives a sound message, it determines the sound intensity at the location of the Listener of each of its registered Listeners. If the sound's intensity upon reaching the Listener is above a customizable threshold, the sound is delivered to the Listener. To determine the sound intensity at the location of the Listener, CustomUnityBroadcaster uses an equation based on the inverse-square equation:

$$Lp2 = Lp1 \quad 20\log(R2=R1)$$

Where Lp1 and R1 refer to the sound level pressure and distance from the source, respectively, of a Listener at position 1 (Listener 1). Lp2 and R2 refer to the sound level pressure and distance from the source, for a Listener at position 2 (Listener 2).

The value used to represent sound intensity in this equation (referred as Sound Pressure Level) cannot be interpreted in the virtual world as it is in real world physics and human perception. Humans can perceive sounds with a pressure of approximately 0dB when in absolute silence, which means this equation would always return a perceptible level. In order to avoid this situation, which would cause all the Listeners to perceive a sound message, the threshold value for minimum sound intensity is set to 30, which is the value that allows NPCs to hear footsteps when they are around 6 meters from the player. In an eort to further simplify the usage of the equation, the position of Listener 1 will refer to an area of 1 meter around the Emitter, where, inside the area, the sound will have maximum



Attenuation in the intensity of a sound with an initial intensity of 60. The further the sound with travels, the greater is the decay in its intensity.

intensity. This way, we do not need to use the original equation and input values for two listeners. Therefore, the distance of 1 meter, the following simplication of the previous equation will determine the sound intensity at the Listener's location:

$$I = O \ 20\log(D)$$

Where I represents the intensity of the sound at the Listener's location, O refers to the sound intensity at the origin and D the distance of the Listener from the Emitter's position. The plot shown in Figure 4.12 shows the evolution of the decay in sound intensity with distance, for a sound of intensity 60:

For FortressBroadcaster to be able to retrieve the location of the Listener, the Listener must provide the broadcaster with the instance of its Transform component, which holds the information regarding the Listener's GameObject location. The Listener provides the Transform component to the broadcaster when it registers itself for broadcasts, in the Listener's Init() method. The Broadcaster stores the Transform in a Dictionary, which links the registered Listeners with their Transforms.

Chapter 5

Evaluation

After the implementation and integration of the framework and of the game Fortress (see section 4), a set of tests were performed, in order to evaluate the exchange of sound messages and demonstrate its value for the game logic. The tests were performed by the application and tunning of different salience computing filters, as well as some tweaks regarding the Broadcaster, through multiple runs of the Fortress game. The tests were expected to reflect how faithfully the proposed framework simulates human sound perception and, so, the primary focus was on how human-like is the NPC perception.

Similarly to what happens in the real world, it is expected that all the created filters are used simultaneously, on each NPC, as to create a credible, human-like sound perception. However, in order to perform tests on how each filter impacts the way a NPC perceives sound messages, each filter had to be tested individually by disabling the remaining.

5.1 Evaluation Method

In order to test how the proposed framework performs in the simulation of human sound perception on NPCs, a consistent test environment, based on the game Fortress, was created.

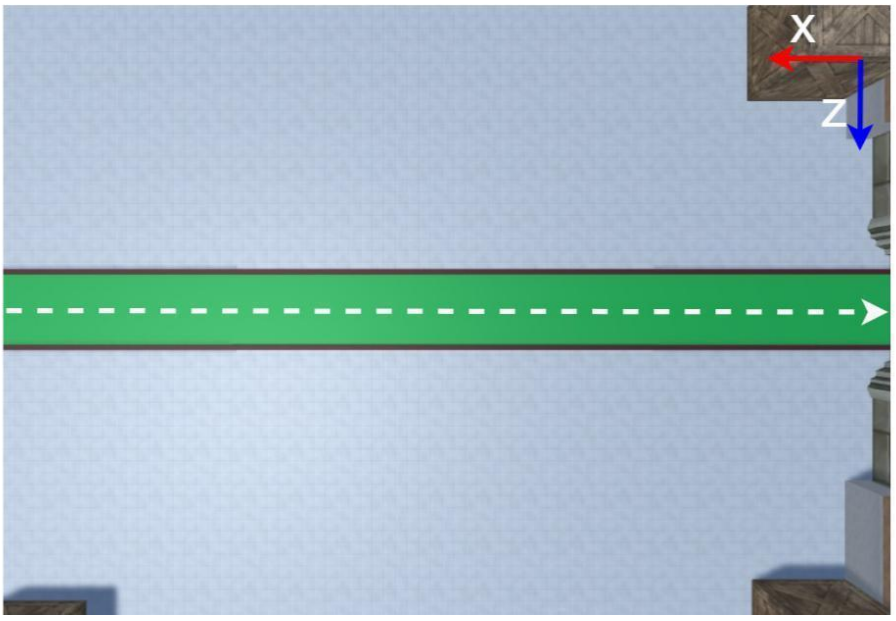
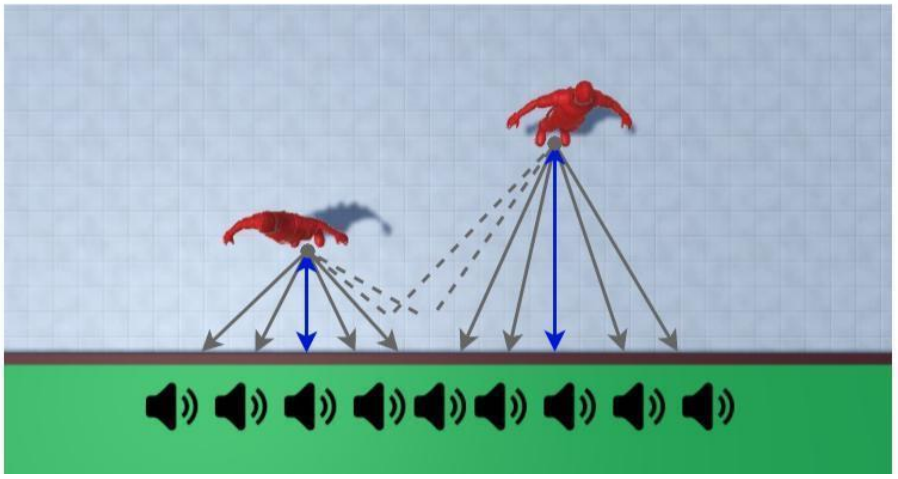


Figure 5.1: The Player character can only move along the x axis, int the path described by the white arrow.

It is mandatory to test how each individual filter contributes and how realistic is the impact of the filter when customizing its properties. Each filter must be tested over the same controlled circumstances, so that, for the same filters and parameters, and for the same Emitter's performed actions, the results are the same. The game's feedback on how well the proposed framework is working is based on the ability of each NPC to perceive, or not, a given sound. As stated in the section regarding the transmission of sound messages (see Section 3.6), the perception of a sound, for a given set of NPC, depends on the NPC's set of salience filters and on the sound's parameters. Sound parameters, like the description, the Emitter's identification and the noise ag will have no impact on the performed tests and, thus, these parameters remained xed during all the performed tests. For the sake of the tests, the player is always the sound emitter. Frequency, sound's intensity and the Emitter's location aect some of the NPCs filters.

For most of the performed tests, the emitted sounds to be perceived by the NPCs were only footsteps sounds. Footsteps are assumed to have a xed intensity of 60 db and a frequency of 250 Hz. These values may vary, depending on the salience filter being tested. These values were only for testing purposes and were merely an approximation and, so, do not reect exactly the physical properties of real

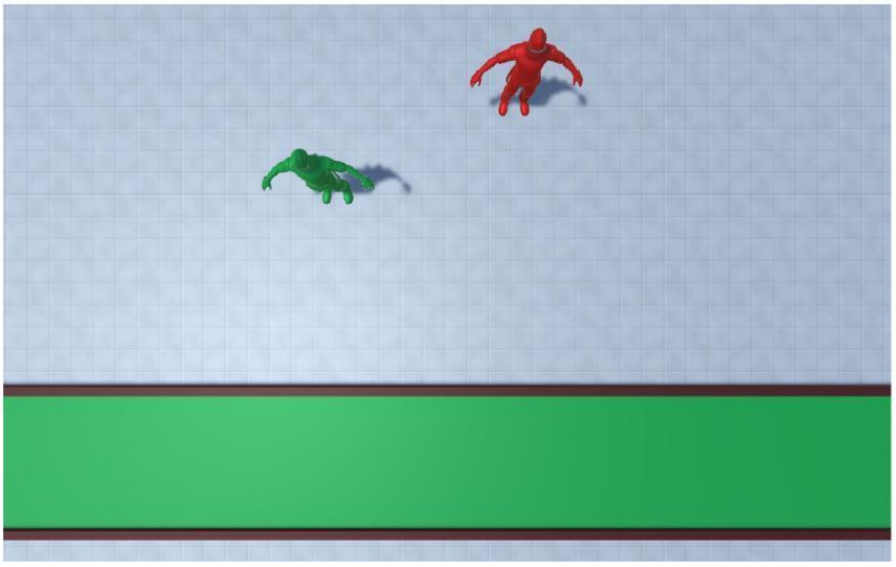
 46



The player moves along the green path. Every 0.1 meters, the player emits the sound of footsteps, represented by the speakers icons. The arrows represent the instants were footsteps that are perceived by each NPC, with the blue arrows being the ones that represent sound perceived with greatest salience, directly related to the shortest distance between the Emitter and the Listener.

world footsteps. In some of the tests, the values which describe the sound were modified in order to t the context of the test that was being performed.

Regarding the Emitter's location, which impacts the distance through which sound travels, it remained xed in the z axis, with only the x axis controller input being taken into consideration. Figure 5.1 shows a player's possible path, represented by the white arrow, along which the test player travels. This way, the shortest distance between each of the NPCs in the scene, and the player, will be met in the moment the player crosses in front of the NPC. The Figure 5.2 represents with a blue arrow the instant where the NPC and the player emitted footstep are the closest to each other. The remaining grey arrows show other footstep sounds that are transmitted to the NPC, but which are perceived with a reduced intensity, due to the greater distance between the Emitter and the Listener. During each of the tests, the player walked along the full path and, as represented in Figure 5.2, footsteps' sound is emitted at a regular interval, which is every 0.1 meters. This provides a more granular array of intensities being transmitted to every NPC, as in the Fortress implementation, steps sounds are emitted every 5 meters. This reduced value between steps makes more likely that the minimum distance between the Listener and the Emitter is met.



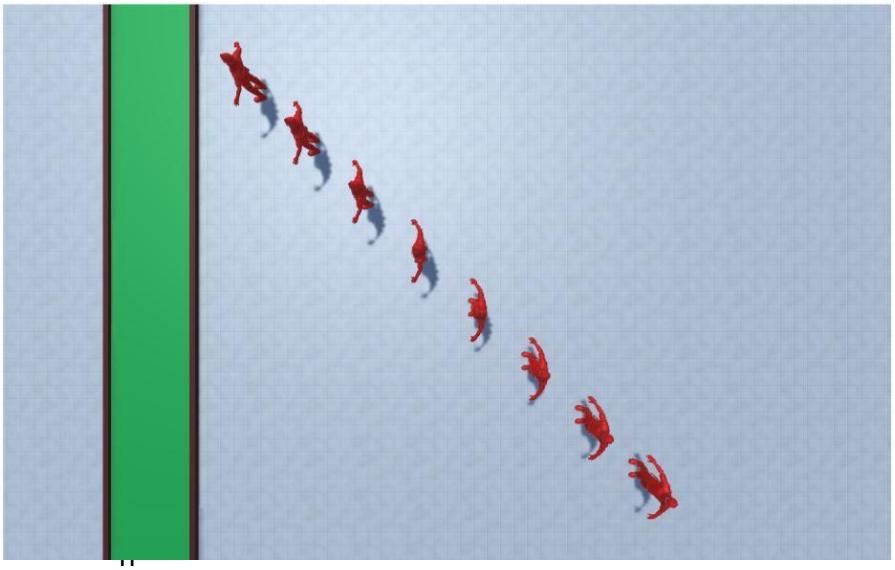
After the Emitter crossed the path, as the NPC on the left perceived sound, it turned green, whereas the NPC on the right did not, remaining red.

When an NPC perceives a sound message, the graphical representation of the NPC turns its color from red into green, as shown in Figure 5.3. This gure portraits a test in which, after the player walked along the path, the NPC on the right did not perceive a sound message, but the one on the left, which is now green, did perceive. By the end of each test, every NPC prints in Unity's log console the greatest salience value of all the messages they perceived, which is the salience value of the footstep sound that was emitted the closest to the NPC.

5.2 Results

The following sections describe the setup and results of the tests made to each individual filter. Towards the end of the section, all the filters are tested simultaneously, in order to test how the filters impact sound perception when interacting with each other.

48



n
en there are no filters, the only variable that affects the perception of a message is distance. In this gure, the NPCs were placed increasingly more distant from the path, in steps of 1 meter.

5.2.1 Without filters

Initially, as part of the gradual increase in complexity of the perception system each NPC has, the framework was tested in a set of eight listening NPCs, each of them with no attributed filter. Having no filters means that the listening ability of each of these NPCs is perfect, as the salience value of received sound messages will exhibit no attenuation.

As there were no filters to customize, and as the emitted sounds always had the same properties (except for the Emitters location), the sole variable was the minimum distance the NPC was from the player. Figure 5.4 shows the location of each NPC, with the closest NPC being 1 meters away from the path, and the remaining increasingly further, in steps of 1 meters.

As we can observe in the plot depicted in Fig. 5.5, the evolution of the intensity in relation to the distance follows the shape of a inverse-square function. This happens as part of the broadcast logic implemented in the Custom Broadcaster from the Section ??, which rules that the intensity of delivered messages is reduced according to a inverse-square function. This way, the Broadcaster portraits the

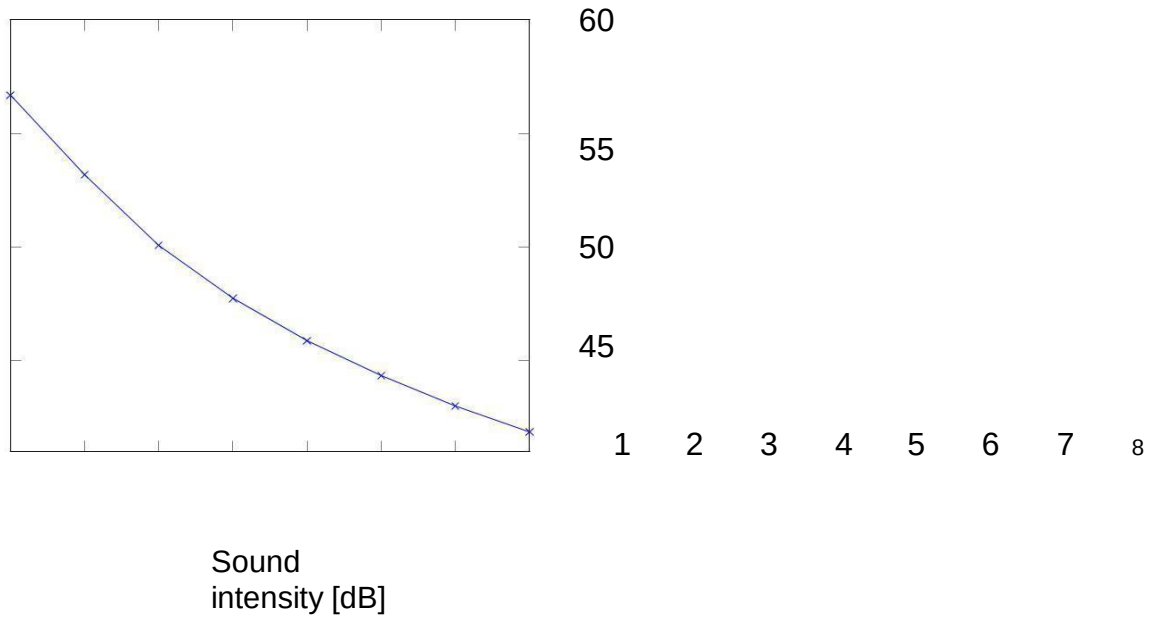
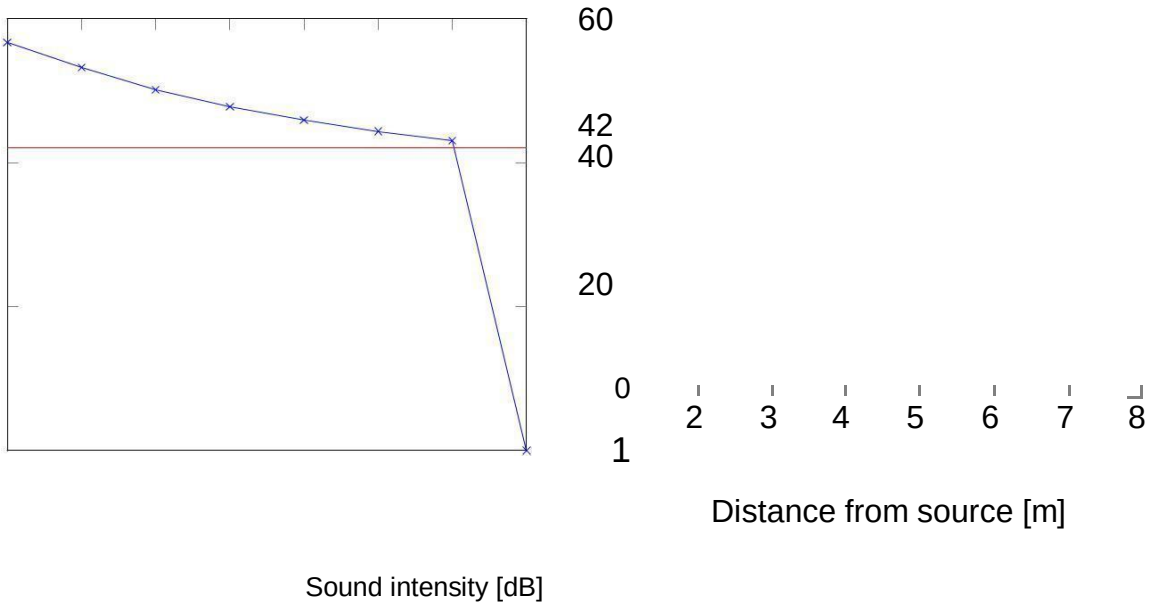


Figure 5.5: Attenuation of the soundsteps intensity in relation to the distance from the Emitter and the Listener.

realistic attenuation that the intensity of sound suffers when it is being transmitted through air. Although the attenuation was expected, the tests showed that all NPCs were able to perceive the soundsteps, even when they were emitted from 8 meters away. The sound is perceived by all the Listeners because the delivery system and environment are too perfect. In real world, the human brain adapts to the constant level of noise of the surroundings, thus affecting our sensibility to emitted sounds, in an eect called perceived loudness. The FortressBroadcaster employs the perceived loudness of the environment as a threshold value that represents the minimum intensity required for a sound message to be delivered to the Listener. For the remainder tests, this value was empirically set to 42 dB, which is an intensity below the intensity of the NPC that was 7 meters away, and above the intensity of the NPC that was 8 meters away. This way, we simulated real life's attenuation of sound that leads to a sound not being perceived when the Listener is too far away from the source of emission.

By re-running this test, the NPC that is 8 meters away no longer turns green, which means it is not able to perceive the message, despite having perfect hearing (no filters). All the other seven NPCs are able to perceive the sound of the footsteps, with the same intensity values as previously (see Figure 5.6), validating

50



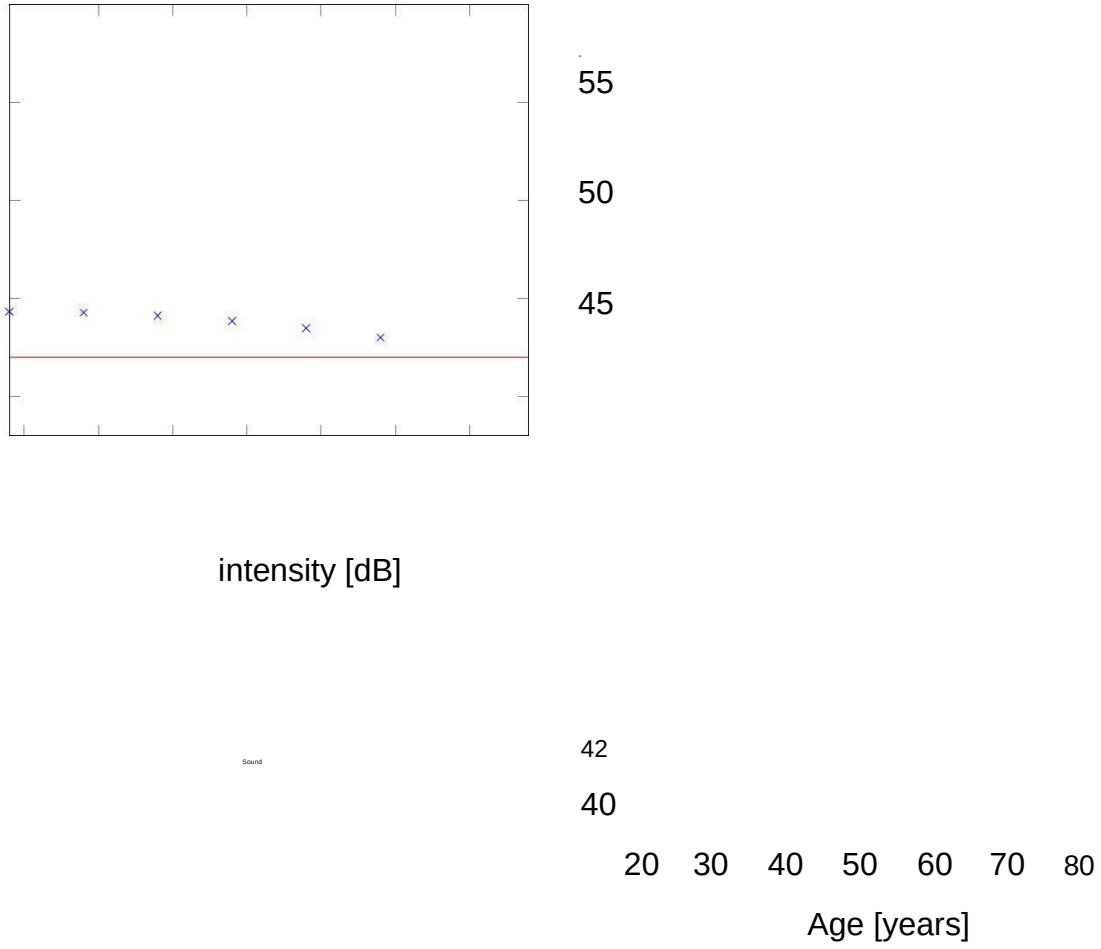
Listener's maximum perceived intensities of sounds is represented by the blue line, with the broadcaster not delivering sound messages when their computed attenuated intensity is below the minimum intensity threshold of 42 (represented by the red line).

therefore the most basic version of the framework with the implementation of the Broadcaster that computes the attenuation of sound according to distance.

5.2.2 Hearing loss filter

The hearing loss filter described in Section 3.5.1 simulates the condition that leads to hearing capabilities being lost due to age. As this filter is affected primarily by the sound's frequency and intensity, and not by the distance, it was assessed in a setup in which eight listening NPCs are equidistant from the path the player walks, 6 meters. Each of the NPCs has been set to be increasingly older, with the rst having 18 years and the remaining having their age increase by 10 years steps.

For the rst test, the emitted sound's frequency was set to a value corresponding 250 Hz. During the test, all NPCs, except for the one with 88 years old, were able to perceive sound, with the nal salience, after the application of the hearing loss filter, presented in Plot 5.7, as its computed salience is below the minimum threshold of 42. This values were the ones expected, as the NPCs is the oldest one.



Resulting Listener's perceived intensities, with each Listener having the hearing loss filter. The emitted sound's frequency is set to 250 Hz.

We also noted that the Listener that is 18 years has perfect hearing, suffering no attenuation in the salience value of the sound, in relation to the value of salience presented in the previous test (without filters 5.2.1, for the NPC located 6 meters away from the source (44.3 of salience). This was intended as the attenuation calculation starts at the age of 18, which we considered as being the age at which the hearing capabilities are at their prime.

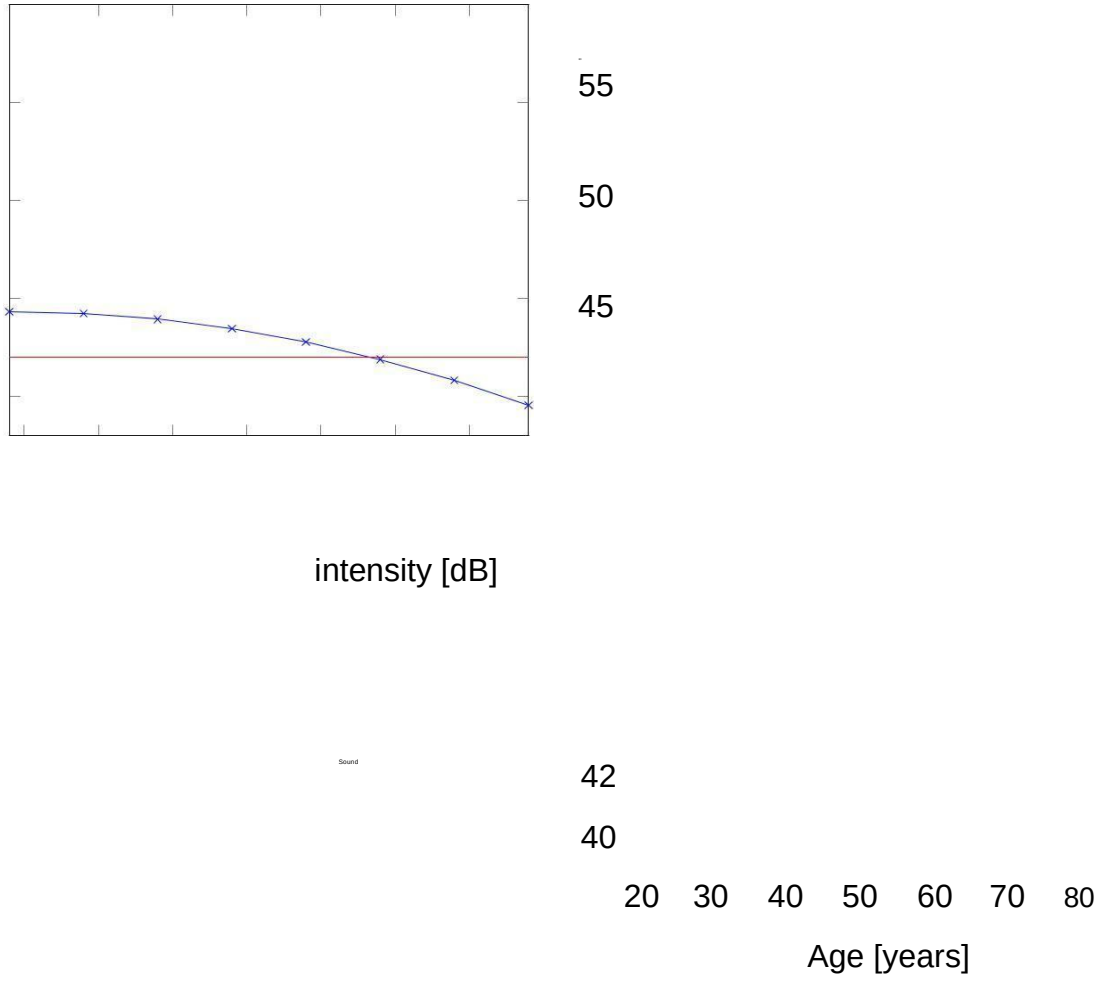
As hearing loss is also related to the frequency in which a sound is emitted, a second test for this filter was performed, in which the frequency of the footsteps was set to 8k Hz. In the resulting plot in Figure 5.8, the Listeners with 68 and 78 are no longer able to perceive the higher frequency sound. This was an expected evolution of hearing loss, as the higher is the frequency of sound, the higher is the decay in the ability of perceiving it.

5.2.3 Focus Filter

In order to test the focus filter, five different NPCs were placed in the scene, each performing a different task. Each task is intended to simulate a situation in which the NPC has a certain degree of focus in the task it is performing, which affects its ability to share attention between the task and incoming sounds. The sample

52

Figure 5.8:



Resulting Listener's perceived intensities, with each Listener having the hearing loss filter. The emitted sound's frequency is set to 8000 Hz.

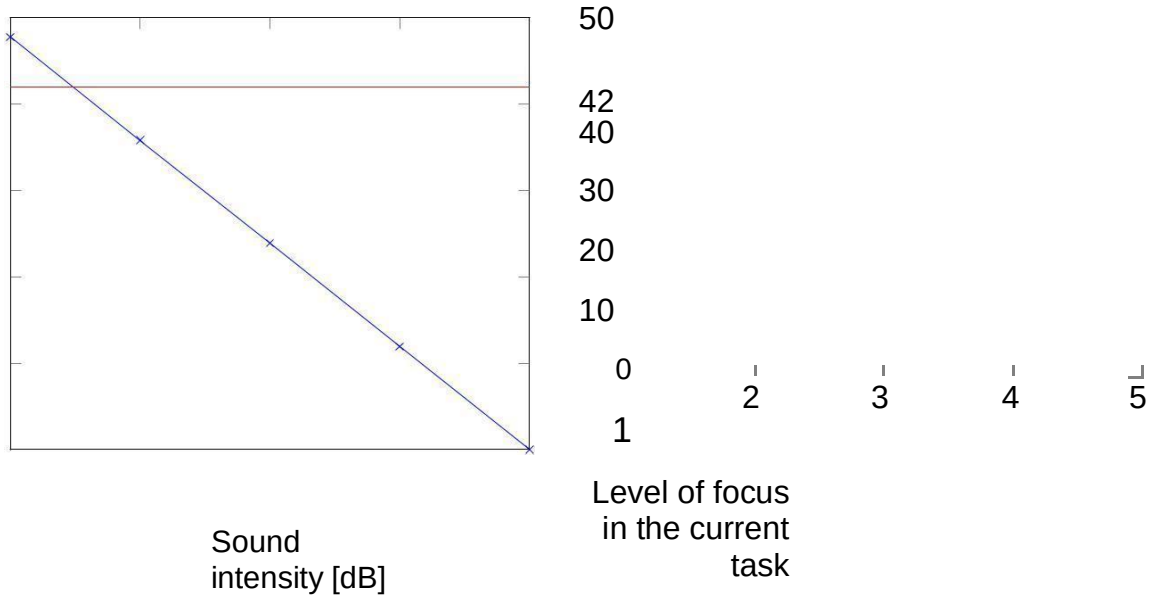
tasks performed do not reflect a concrete task as they are only a way of testing the filter, so the NPCs stand still but have a different level of focus according to a set value that range from 1 to 5, with each NPC having a different level of focus in the current task than the others, each affecting differently the NPC's ability to perceive incoming sound (see table 3.2 from section 3.5.4).

For the rst test of the focus filter, the NPCs were placed at a distance of 4 meters away from the player path. In the plot depicted in Fig. 5.9 we can see how the decay in salience is linear, with the NPCs having the computed salience more reduced the bigger the focus they have on their current task, with only the NPC with a focus level of 1 being able to perceive the sound message. All the remaining NPCs are not able to perceive the sound messages, given their focus on the task they are performing. These results may not be satisfying for a developer using our framework, as the computed salience drops too abruptly.

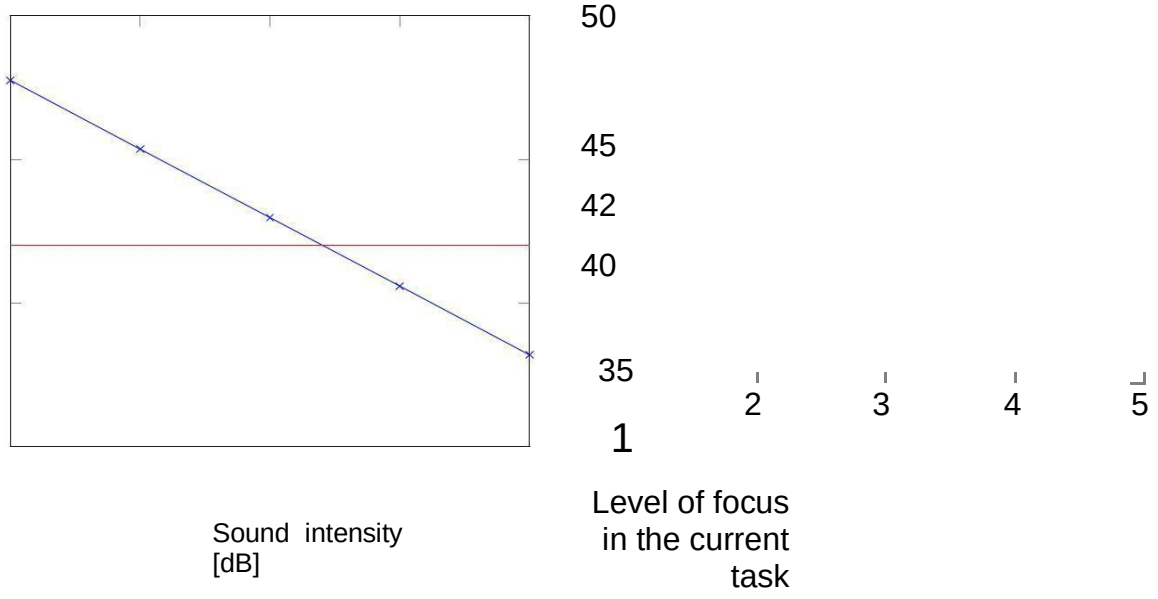
In order to x the abrupt drop in salience for each of the focus levels, a new variable associated to the Focus Filter was added, which determines how much does the focus level filter contributes to the decrease of salience. This value is a oat and ranges from 0 to 1, with 0 meaning it does not contribute and 1 meaning it contributes fully, as happened in the previous test. The plot depicted in Fig.

Figure 5.10:

Figure 5.9:



Only the NPC with the lowest level of focus is able to perceive the sound message, as the other's perceived salience drops below the 42 intensity threshold.



With a contribution of 0.2 in the computation of salience for the

Focus Filter, the NPCs with a level of focus of 4 or 5 are not able to perceive the footsteps emitted 4 meters away from them.

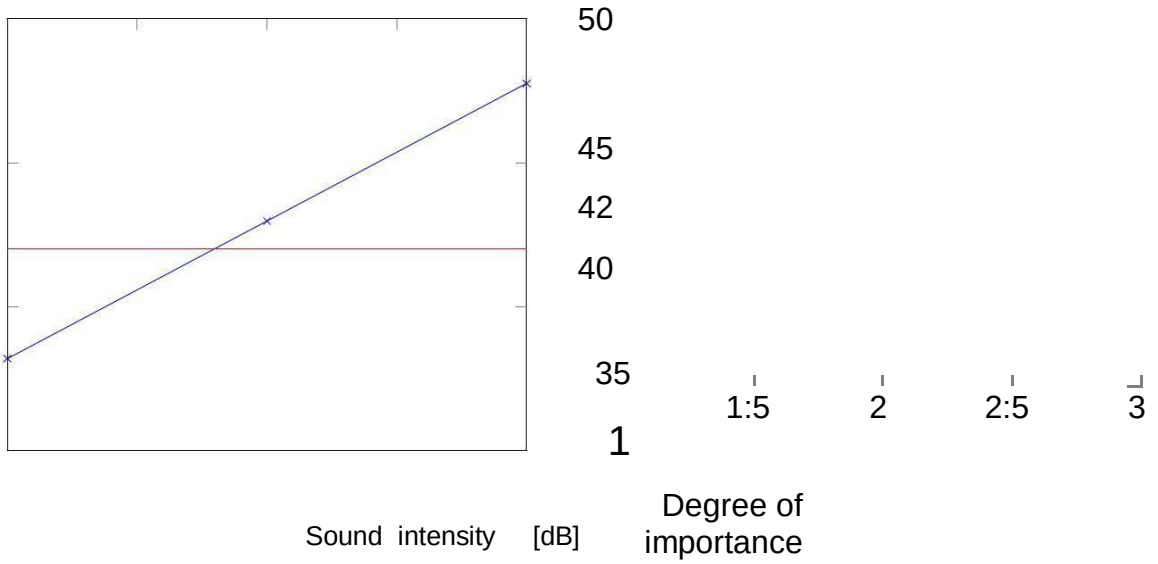
5.10 show how now all the NPCs with a focus level ranging from 1 to 3 are able to perceive the footsteps sounds.

5.2.4 RelationShip filter

As for the relationship filter, the salience is affected by how the Listener relates to the Emitter, this is, the degree of importance the Listener places in sound coming

54

Figure 5.11:



The NPC that places no importance in its relationship with the player is not able to perceive it. The NPCs that place higher level of importance in their relationship with the player are able to perceive the player.

from the Emitter. The test was performed with each of three NPCs having the different possible filters value for the description of their relationship with the player, which are "Not Important", "Neutral" and "Important". These values tell how important it is for the NPC to perceive the sounds emitted by the player. All the NPCs where placed 4 meters away from the player's path, in order to have the computed salience only being affected by the relationship filter.

Plot 5.11 shows how the NPCs that place the lowest interest in their relationship with the player, with 1 representing Not important, do not perceive its sounds. The NPCs that are more interested in perceiving the player, with the values 2 and 3 representing, respectively, Neutral and Important, are able to perceive the sounds it emitted and, thus, turn green. The NPC that places the highest level of importance to its relationship with the player, Important, displays no decrease in the salience of the players sounds.

5.2.5 Noise filter

The noise filter affects the salience of sound messages as a function of the presence of background noise in the surroundings of the Listener. To test this functionality,

Figure 5.12: The radio emits a continuous source of sound, which is considered by the NPCs as noise.

a radio emitting background noise was placed 1 meter away from the right most NPC, as depicted in Figure 5.12. Every other NPC is spaced 1 meter from each other.

In the beginning of the test, the radio starts emitting a sound with the noise ag set to true, with an intensity equal to the intensity of the player's footsteps. The player then passes 3 meters away from the NPCs. As seen in the plot depicted in Fig. 5.13, the NPC that is closer to the noise source is unable to hear the player, as the salience of the player's footsteps are greatly reduced by the background noise. The remaining NPCs have their salience reduced by a fth of the difference between the noise's intensity and the footsteps intensity, when the footsteps intensity is superior to the intensity of the noise. Whenever the noise has a greater intensity than the footsteps, the footsteps salience is reduced by the difference between the intensity of the noise with the intensity of the footsteps. The further away the NPC is from the noise source, the smaller is the reduction in salience.

5.2.6 All filters

As a nal test, all the filters were activated to see how able would the NPCs be to perceive sound messages, with a more complex and realistic set of salience filters.

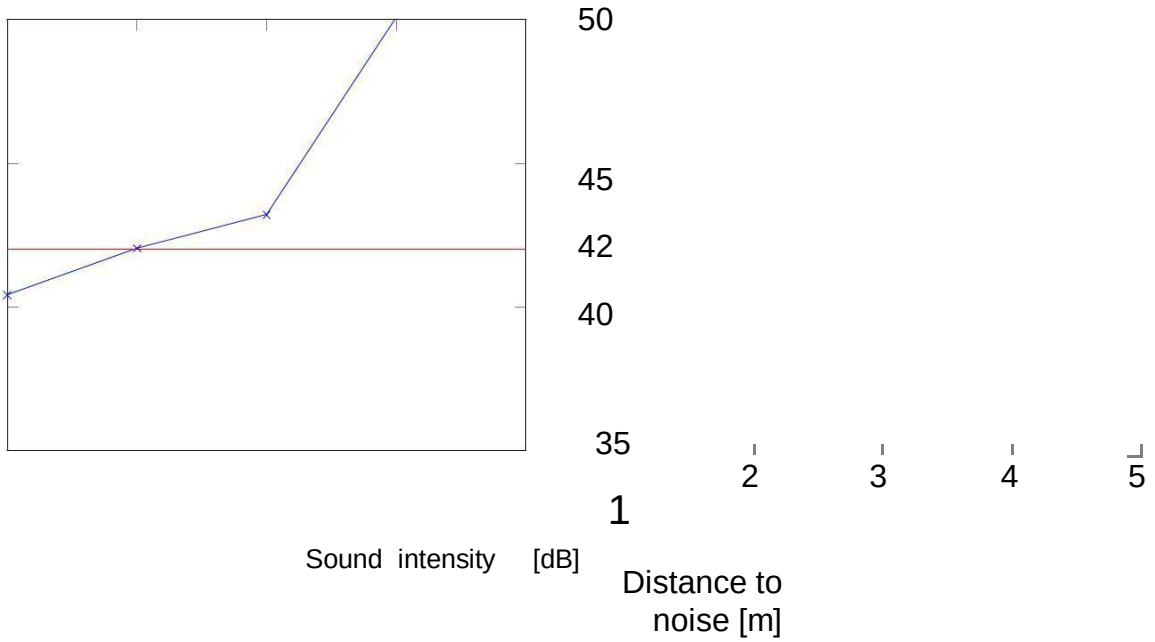


Figure 5.13: The closer the NPCs are to the source of noise, the bigger is the attenuation of the salience of the footsteps.

NPC Number	Age	Relationship w/Player	Focus Level	Distance	Distance from Noise
1	18	Important	1	3m	8m
2	18	Neutral	5	3m	7m
3	28	Not Important	3	3m	6m
4	80	Important	2	3m	5m
5	46	Neutral	2	3m	4m
6	40	Neutral	3	3m	3m
7	20	Important	5	3m	2m
8	25	Important	1	3m	1m

Table 5.1: Hearing loss coefficients for each evaluated frequency, according to Robinson et al article.

All the NPCs placed in the scene had all their filters activated and set with the values present on table 5.1. The noise emitting radio, from the previous test, was place in the scene as well, with the same parameters for its emitted noise.

The plot depicted in Fig. 5.14 shows the salience values of perceived sounds for each of the NPCs. We can observe that all the filters together provide too much salience decrease, making NPC number 1 the only

one that is able to perceive the sound message emitted by the player, with NPCs 4 and 5 close to perceive the player's footsteps. Despite this being the way the filters were design to attenuate salience, the result could be more realistic, as too few NPCs perceive the player. In a real setting, despite some of the characteristics of the NPCs, more of them should be able to perceive the player given the context of the scene. By lowering

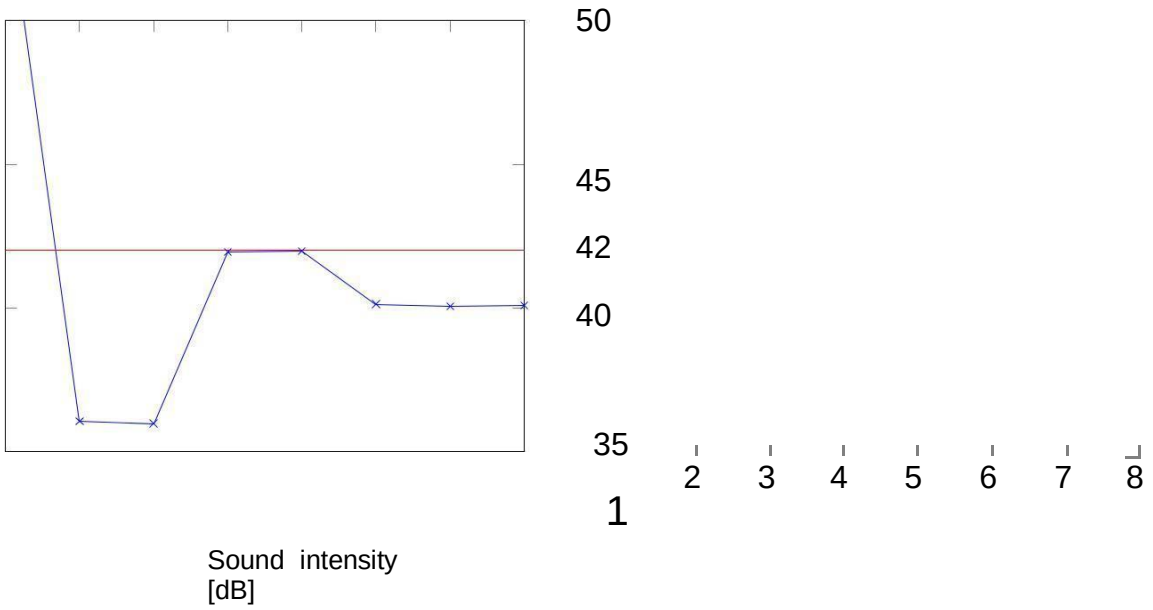


Figure 5.14: Final salience values when the parameters from table 5.1 are applied to the NPCs.

the minimum salience threshold from 42 dB to 41 dB, the number of NPCs that are able to hear the player is increased, displaying an acceptable number of NPCs that can perceive the sounds.

There are three ways to make more NPCs able to perceive the footsteps. The rst one is done by setting different values for the NPCs filters parameters, which can lead to a difficult use of the framework, as the filters should dene an easy way to setup the traits that compose the physical and psychological state of an NPC. The developer should not have to tweak each and every NPC's parameter in order to obtain the wanted traits. The second solution involves creating a weight variable for each filter that determines how each contribute to the computation of salience, similarly to what was done during the test done for the Focus Filter (see Section 5.2.3). A third solution which would involve less tweaking of the scene, involves modifying the minimum salience threshold to a value that allows more NPCs to perceive the emitted sounds.

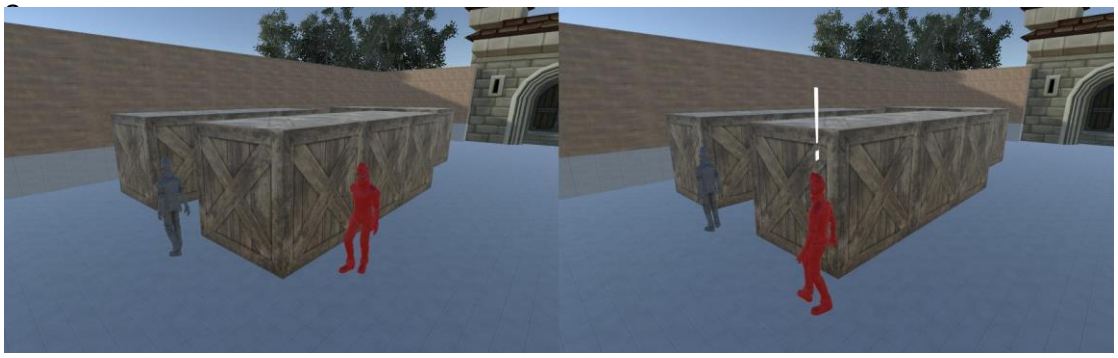
We nd the second solution to be the best, as it allows the developer to soft tune and adapt the relevance that each filter has in its game, without having to modify each filter's parameters in order to obtain the desired perception capabilities.

58

Figure 5.15:

E

v



Left: The NPC heads to the source of the sound, to see if the player is there. Right: The NPC sees the player, and proceeds to chase it.

5.3 Fortress

The proposed framework was fully integrated in the game Fortress 4 in order to provide the NPCs with the ability to perceive sounds emitted by the player, which helps in their objective.

With the library fully integrated, any NPC will the ListenerBehavior is able to perceive sound coming from the emitters. In the concrete case of Fortress, each NPC may be able to perceive sounds emitted by the player and, if a sound is perceived, they go to the source of the sound and try to nd the player.

In Figure 5.15, on the left side, we can see a NPC that perceived the footsteps of the player, which is now checking the location of the emitted sound. On the right side, the NPC sees the player, which is expressed by the exclamation mark over its head.

In a similar fashion, another NPC, which is listening to music, has the sound salience of the players sounds salience reduced by the music's contribution as noise, and by the fact that the NPC is distracted listening to music, with a focus level of 3. This situation was tested with the 3 different configurations that resulted in Figure 5.17, which shows the NPC that listens to music surrounded by three circles, which approximate three different areas where the NPC, given the current circumstances, has the salience of incoming sound messages reduced differently. The red circle shows the area that, whenever the player emits a sound inside it, the NPC is able to perceive the sound. The orange circle shows the area in which 59

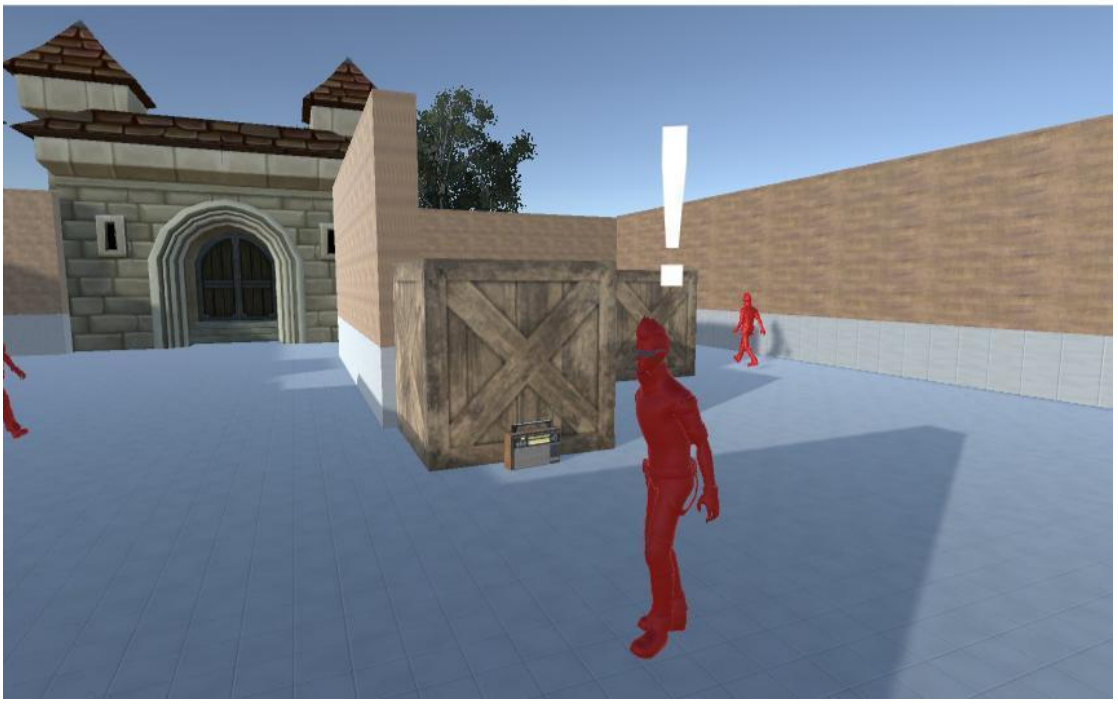


Figure 5.16: When the NPC perceives the player, it displays an exclamation mark.

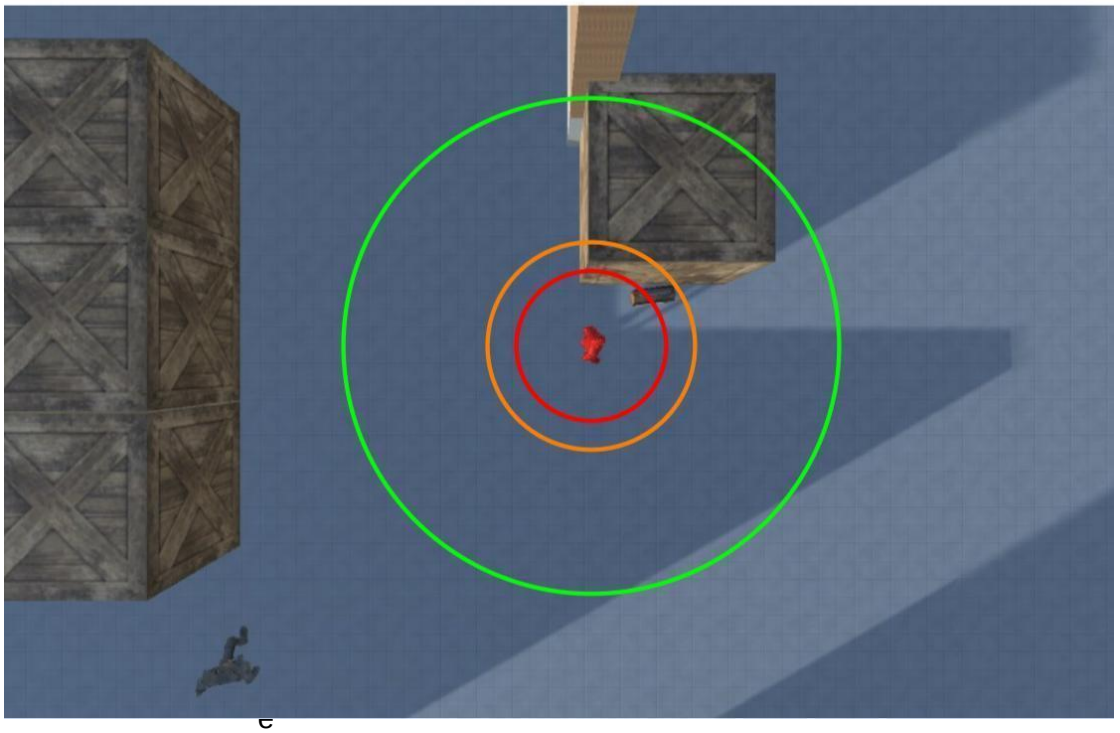
the NPC detects the player if it stops focusing on the music, by reducing the focus level in the Focus Filter to the value of 1. The green circle shows the area in which the NPC detects the player if the NPC is neither focuses on the task at hand, with a focus level of 1, nor the music is turned on, thus not emitting noise. The difference between the red and the orange circle is not big, since the Noise filter is the one that is contributing the most to the attenuation of salience. By moving the NPC away from the noise source, we are able to see both the red and orange circle have their radius increase, while the green keeps its current size, as seen in Figure 5.18. The radius for the circles were discovered by having the player slowly approximate the NPC in the given situations and, by using the top-view camera, determine the location in which to start drawing the circle. The location was the one in which the NPC detected the player, as shown in Figure 5.16.

The need of providing the NPCs with sound perception capabilities also leads the player to become more aware of the sound it is emitting and the sounds being emitted, which makes the player's gameplay experience more immersive. Running the Fortress game with the framework fully operational, leads to an experience in

60

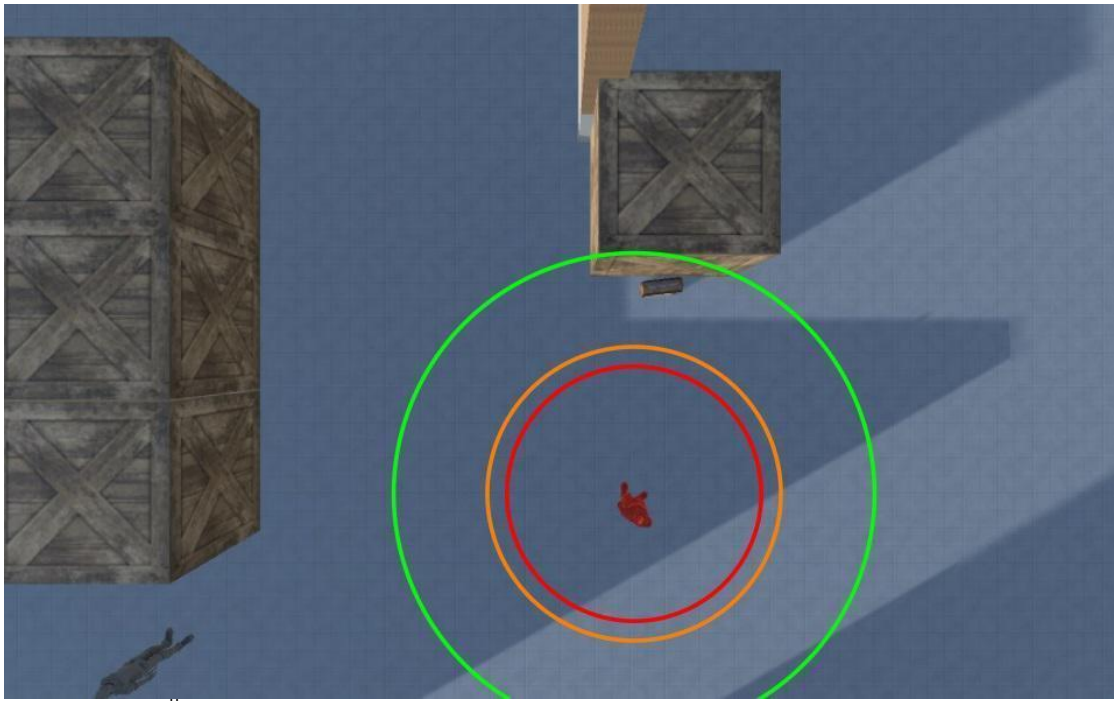
Figure 5.18:

Figure 5.17:



e

NPC is listening to music, having its noise and focus filters affecting is perception. The red circles shows the area within which sound salience as sucient salience for the player to perceive it. The orange circle shows the area within with sound is perceived it the focus is reduce to the minimum. The green circle shows the area in which the sound is perceived when the focus is at its minimum and the music is turned on.



e NPC moves away from the radio, which decreases the attenuation in salience caused by both the Noise and Focus filters.

which the NPCs capacity to detect the player is not as linear, leading the player to be more careful, as the NPCs have less predictable perception capabilities.

During the earlier stages of the integration, some modifications to the Messaging Library layer were made in order to make the integration work. As is now, the proposed framework provides auxiliary Unity classes that facilitate even further the integration of the framework into existing games. Other aspect of the integration is the need to, during the initial stage of the integration of the framework, tune the minimum threshold for salience, the sound messages properties and the filters parameters in order provide a gaming experience that more closely resembles reality. This must be done until the game developers and designer achieve the desired results in terms of the NPCs sound perception capabilities.

62

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The presented framework allows developers to integrate mechanisms for sound transmission and perception into their Unity powered videogames. These mechanisms provide sound perception capabilities to NPCs by simulating the physiological and psychological aspects that affect human auditory perception. The developer may select how sound perception is computed for each individual NPC, allowing each of the NPCs to perceive and react in an unique way to sound stimuli, further increasing how believable is the interaction of NPCs with sound.

Current available implementations of sound transmission in videogames proved to lack control over how each NPC trait affects its perception of the gaming world. The current implementations favor simplicity when it comes to configure the NPC's sensibility to sound, only providing a threshold value that is matched against the sound's intensity. The present work allows the developer to provide NPCs that are context aware, while also allowing the developer to increase or decrease the complexity of the perception system as it pleases.

After the initial implementation of the framework, a sample game called Fortress was created, which uses all the capabilities of the framework by placing NPCs and the player inside the courtyard of a Fortress, and has the NPCs interact with the 63

player according to their ability to perceive the player's emitted sounds. Here, the degree of eort in the integration of the framework in an existing game was also tested. In the beginning, some modifications were made to further decrease the degree of dependence that each Unity component had with the framework, but this posed a challenge as some of the filters needed context that only Unity can provide, such as the location of a given NPC or the source of a sound. Each of the framework's features were validated by carefully determining how each salience filter takes part in the computation of salience of received sound messages. Each filter was test against all the variable properties that play a role in the computation of salience for a sound, given the sound's properties. During some of the tests, some adjustments were made in the way the filters work, so that they provide the behavior that was proposed earlier in this document.

The framework proved to be able to provide believable perception of sound, while keeping its integration simple for game developer. By allowing the developer to adapt which salience filters are used, and by allowing the inclusion of the ones created by the developer, the transmission and perception of sound can easily adapt to the current software architecture the developer may be employing, with-out requiring heavy modifications of the game's code. Most of the games display characters with different looks, age and with different behaviors. This framework allows developers to shape how a character reacts in accordance with their aspect and background. The ability to individually tweak each of the characters pe-ception capabilities, oers players a greater degree of immersion when it comes to how the NPCs react to sounds in their surroundings. As a side-eect, players themselves will become more cautions on how the sound they emit might impact the NPCs, thus further increasing the immersion in the game world.

A balance between complexity and simplicity is crucial when developing a videogame, specially when the complexity has an impact on the time of development. The most challenging component to integrate was the component of the Emitter. The Emitter must build a sound message in order to communicate with the remaining world, and, depending on how realistic the game designers want the game to be, all the Emitters must emit sound messages that respect the same degree

 64

of realism. This is the only way for Listeners to perceive and interpret received sounds in a consistent fashion. For the game Fortress, a sound factory was created in an eort to normalize the creation of sound messages, through which all the Emitters must create their sound messages. This still posed to be a tedious and problematic process, as the developer must nd a way to determine which should be the intensity of a given sound while taking into account the intensity of others, while pairing that intensity with the other properties of the sound message, which should be consistent as well. This leads developers to guess which values are right and which are wrong, which is not the ideal way to integrate our framework, as the description of the sound plays a major role on how our framework simulates sound perception.

Another impact of the complexity is on game performance. In order to provide a realistic set of traits that shape the perception of a NPC, a high number of salience filters may be attached to the NPC, which impacts the performance of the game by having each sound message delivered to Listener, which then computes the sounds salience by running it through all the filters. This effect is even more aggravating in situations where the game displays a crowd of NPCs, where multiple NPCs receive sound messages all the time, exponentially increasing the demand in processing power.

Also, as part of the eort to provide a set of multiple physical and psychological traits, the interaction between filters was neglected, leading to NPCs that had all the available filters active, not being able to detect sounds. This could be corrected through the implementation of a come complex filtering pipeline, which would allow the developer to tweak the contribution of each filter to the nal salience value.

Currently, the framework is not being used by developers. In order to promote the framework, it could be published in Unity's asset store, so that it can reach its target audience.

6.2 Future Work

After the development of the framework, and in spite of the successful integration of the framework in the sample game, and the positive results obtained during the evaluation process, a few key points stood out, which require further investigation and development. Each of the following points provide improvements that will impact positively both the game developer and the players experience:

a mechanism that allows noise Emitters to implicitly update their location, as the current implementation only updates the position through the explicit emission of new noise sound messages;

implementation of a filtering pipeline that allows the developer to choose how much impact each filter has in the computation of salience;

optimize the usage of processing power by optimizing how each individual filter processes salience;

a bank of sounds descriptions, which provides normalized values for all the properties that compose a sound message in the framework, so that the developer does not have to aggregate the whole data itself;

6.3 Dissemination

Part of this work resulted in an article, which was published at the 24th Portuguese Meeting of Computer Graphics and Interaction [4]. The source code and the Unity project which were developed as part of this work were hosted online, in a repository [3].