ulm university universität **uulm**

Faculty for Engineering, Computer Science and Psychology
Institute for Databases and Information Systems
Ulm University

Implementation and evaluation of a mobile iOS application for auditory stimulation of chronic tinnitus patients

Bachelor thesis

Submitted by:

Stefan Mayer

stefan.mayer@uni-ulm.de

Verifier:

Prof. Dr. Manfred Reichert

Supervisor:

Marc Schickler

2015

Version 9th November 2015

Abstract

Mobile devices take in more and more space in our lifes and as tinnitus becomes a much more widespread disease, science started to develop treatment strategies. One of those treatment strategies is 'auditory training' and with this strategy in mind a game concept with spatial audio was developed to provide an easily available ambulant help for chronic tinnitus patients. In the course of this thesis a seperate game only using spatial audio is developed on mobile iOS devices and in a study it is evaluated how good spatial audio can work on different mobile platforms. At the end of thesis the work is summarized and improvements in the game are displayed for further use in the above mentioned game scenario.

Satz: PDF-LaTeX 2_ε

Contents

1 Introduction

Health is one of the most important aspects in our society and therefore a lot of effort is made to keep people healthy. One of these efforts is to overcome deseases and make people live longer. Non lethal deseases tend to be less researched, because they have a smaller impact on our lifes. Nethertheless, these non lethal deseases can be life affecting. One of these life affecting deases is tinnitus. There are two types of tinnitus, one which is subjective and another one which is objective [14]. An objective Tinnitus is caused by the body tissue, but is a lot more uncommon than the subjective tinnitus which occurs completely without any physical reason [14]. Such a tinnitus produces a subjective sound on patients with the abscene of a physical source and is one of the most common symptoms of hearing disorders. For 1-3% of the affected people, their tinnitus is loud enough to affect their life [16]. For a long time there has been no treatment to overcome tinnitus, but today there are a lot of strategies to relieve subjective tinnitus:

- **Psychoeducational treatment**. Because a lot of patients have a negative attitude towards their tinnitus, this strategy is recommended as a basement of all treatment strategies. It is based on a psychological explanation of the patient's tinnitus [13].
- **Cognitive behavioral therapy** is the best studied and evaluated treatment for tinnitus. The patients are made aware of their tinnitus and are helped by improving the modification of maladaptive patterns, causing the tinnitus [13].
- **Individualized auditory stimulation**. These strategies focus on the tinnitus of the individual patient as each tinnitus is subjective and has a different frequency range and loudness. One of these strategies is the auditory training. With this technique the patient actively works on the tinnitus. Studies have shown, that auditory training, including object identification and localication, reduces tinnitus [13].
- **Tinnitus retraining therapy** is a combination of counseling and auditory stimulation but is not evaluated very well [13].

- **Neuromodulatory treatment**. Because chronic tinnitus always results in a change of nervous activity, this treatment focuses on reverting these changes. One technique is to modify audio's frequency and then let the patients hear that audio in order to compensate the tinnitus [13].
- **Pharmacotherapy** is a strategy involving medication [13].

For application developers, the most interesting treatment strategy is probably the auditory training, which is part of the individualized auditory stimulation, because of the adaption capabilities an application and especially a mobile application can have and therefore can be used for a lot of patients.
Additionally, as the amount of smartphone users increased rapidly in the last years, a lot of game developers started developing games for smartphones [2]. The game industry also started to experiment with games depending solely on spatial audio [26]. One of this game's is 'Audio Defense' for the platform iOS [5]. The games target is to kill zombie enemies with weapons only through hearing the zombies with headphones.
As there is a need to overcome tinnitus and there are a lot of people owning mobile devices, it was only a matter of time to develop an application for mobile devices especially fitted for tinnitus patients. For keeping the patients on their training a game application works best, because the patients have a very high motivation level. Such a game is not only cost efficient but the game industry also has shown that such audio games are possible and very enjoyable.

1.1 Goals of this thesis

One goal of this bachelor thesis is to create a game, in which the main goals are to identify and locate objects. This should make it possible to evaluate how good object localization works on mobile devices. Therefore, three other implemenations on the platforms Android, Windows Phone and a Web application, of the game were made as part of other bachelor/master thesises. In the game implemented by the four platforms it is the goal to take the best possible pictures of animals, where the challenge is to find these animals only through spatial audio. The creation of the game has some pitfalls.

- **Embedding the OpenAL library**. One of the biggest pitfall is embedding the OpenAL library into the game. As the API is written in the non object-oriented C there is a need to build an object orientated wrapper around it, which needs to be written in Objective-C.

- **Implementing the game**. Another pitfall is the implementation of the game and its logic, which consits in finding a matching architecture as well as structuring the logic.
- **Designing a UI suitable for a study**. Because the game needs to be used for a study there is a need for a UI especially adjusted for this study.
- **Finding possibilities to use sensors**. For the in-game movement it was decided to use the movement sensors of the device. The challenge and goal is to interpret the motion data provided by the iOS SDK for the use in the game.

Because frameworks like OpenAL and the iOS SDK are going to be used, mathematical problems are solved differently. That is why there is a need to make the different results usable in the game. This is mainly covered in the fundamentals chapter 2. A study was done as well as part of this bachelor thesis. This is covered in chapter 6 and compares the four implementations with each other in terms of how good object localisation works on mobile devices.

1.2 Structure

Introduction
Fundamentals
Requirements Analysis
Implementation
Requirements Comparison
Study
Recapitulation

Chapter 2 covers the fundamentals, like the game basics, angle conversion, as well as coordinate systems and the head-related transfer function. In chapter 3 the functional, as well as the non-functional requirements for the application are covered. The next chapter's topic is about the implementation of the game, making an OpenAL wrapper and the structure of the code. After the implementation chapter 4 a comparison in the requirements of chapter 3 are made in chapter 5. After that a chapter covering the results of the study follows in chapter 6. And at the end a recapitulation chapter 7 summarizes this thesis.

2 Fundamentals

This chapter covers the fundamentals needed for the following chapters, especially for the implementation and study parts. In the beginning of the chapter the game mechanic is explained, followed by an explanation of the conversions needed to be made for using vectors and angles in the different subsystems. Last but not least the head-related transfer function is covered, as it is a fundamental part of the OpenAL implementation.

2.1 Game basics

The game which was implemented in this bachelor thesis, is mainly focused on spatial audio. The target of the game is to locate sounds in a three dimensional environment. Several sound targets are placed randomly around the player in a circle. The aim is to take a photo of the targets one after another. To aim on these targets the gyroscope of the iOS device is used, so the player has to rotate in the target's direction by himself. Whenever the target is in front of the player, they have to take a photo of the target by tapping on the screen. The objective is to get deviation from the sound targets position that is as small as possible (see Figure 2.1, where target and player positions are shown). The current implementation that was use in the study uses a maximum of two targets, which are a frog and a bluejay bird.

2.2 Coordinate systems

For placing sounds and a player into the game a coordinate system is needed. Because the sounds are also placed in OpenAL and sounds need to be displayed on screen. Therefore, coordinates need to be converted between the systems. Because the player has a static position, it is most intuitive to use a standard cartesian coordinate system, with the player in the origin of the system (see Figure 2.1 showing player and sounds). The UIKit of the iOS SDK used

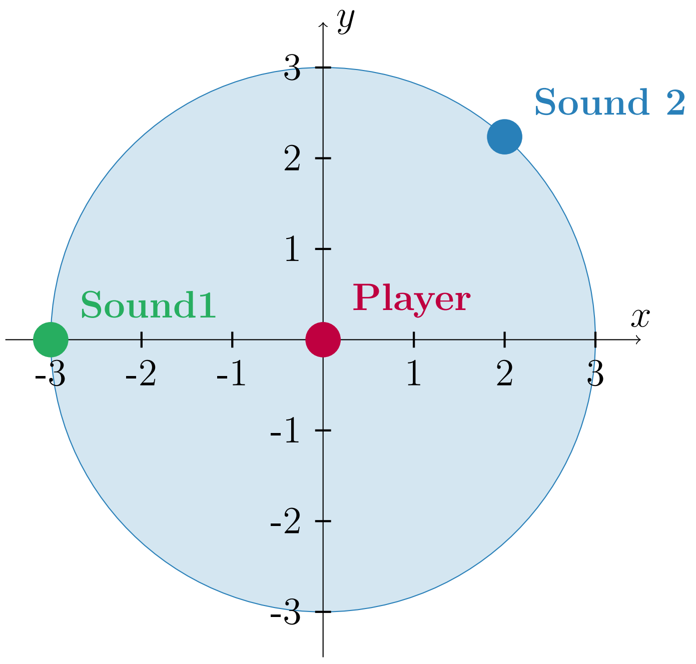


Figure 2.1: Two dimensional cartesian coordinate system.

for image creation and UI desgin uses a two dimensional cartesian coordinate system with the origin in the devices top left corner but also has the modification that the y-axis goes in the other direction, which is common not only in iOS systems but also in Android (see Figure 2.2). In contrast, the OpenAL library

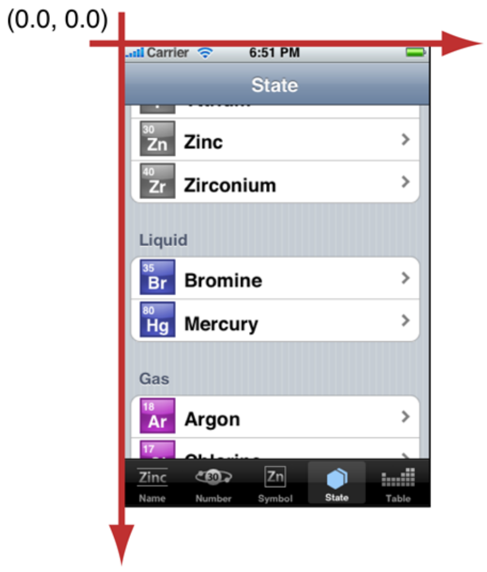


Figure 2.2: UIKIt Coordinate system sample view [1].

uses the cartesian coordinate system in 2D but in its three dimensional version, as it also can be used for three dimensional applications. Therefore, some small adjustments must be made when taking over coordinates from to game to OpenAL. Which in our case is only changing the sign of the y coordinate. Because of the choices made for the game the plane used by the game sits on top of the XY plane of the OpenAL system (see Figure 2.3).

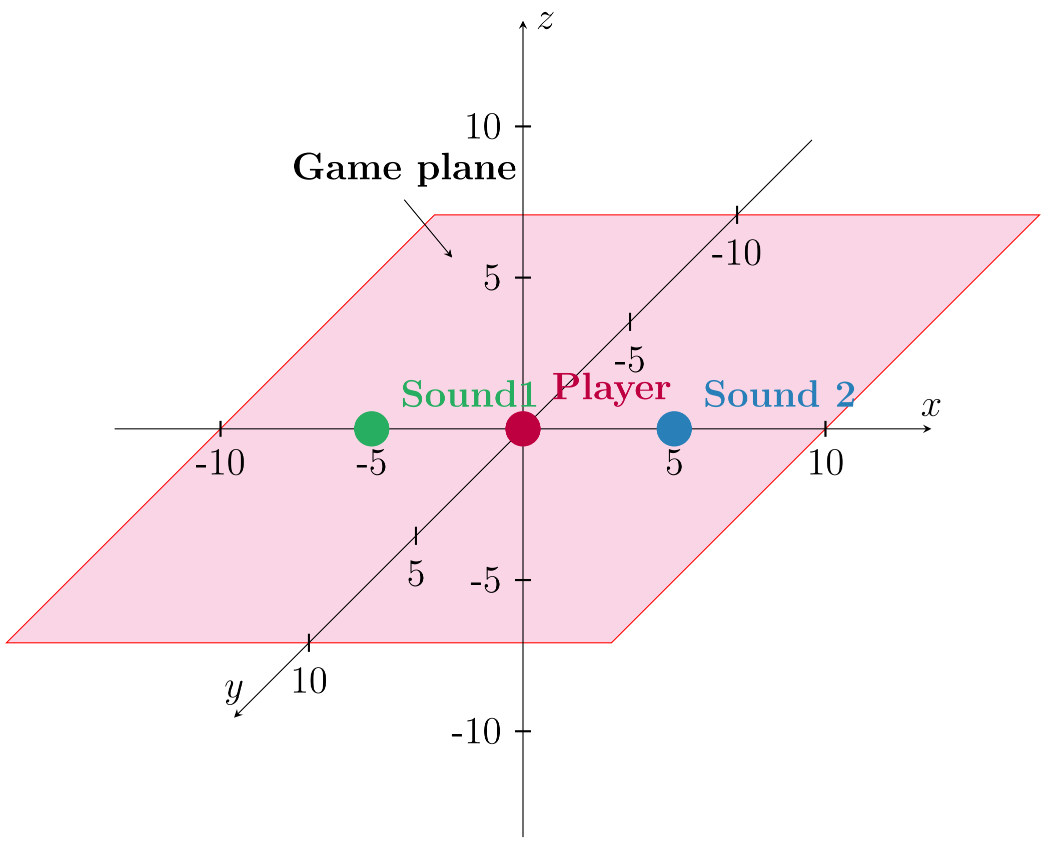


Figure 2.3: Position of game plane in OpenAL system.

2.3 Angle measurement

A very important feature of the game is stated by angle measurements, because entities are being placed on the circle line around the player with a constant distance. So the position is defined soley by an angle. The *CMDeviceMotion* iOS class, used for determining the player's position, uses cartesian angles in radians. These angles start from the inital starting position, when intitializing the class. So a rotation of 20 degrees to the left results in a value of $\pi/9$, the rotation of 20 degrees to the right results in a value of $-\pi/9$. There is also a restriction to the rotation so that it is not bigger than $180°$ causing the value to jump form $-180°$ to $180°$ if rotated close to 180 degrees. Because the starting

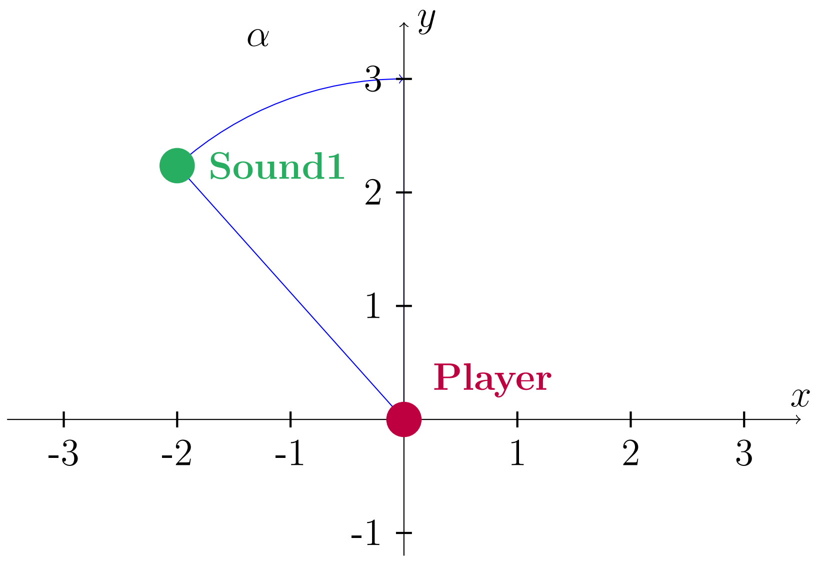


Figure 2.4: 2D cartesian coordinate system, showing angles used by CMDeviceMotion.

point of the angle in the game differs from the output of the gyoroscope, there is a need to rotate this starting point by 90 degrees to the right. Therefore, the angles from the *CMDeviceMotion* class must be converted as follows:

$$w(\alpha) = \begin{cases} (2.5\pi + \alpha) \mod 2\pi, & \alpha < 0 \\ (0.5\pi + \alpha) \mod 2\pi, & \textit{otherwise} \end{cases} \tag{2.1}$$

For points or entities placed on the game's coordinate system, an angle also must be calculated from the position vector $\vec{x}$, when evaluating results:

$$w(\vec{x}) = \begin{cases} \arccos \frac{x}{\sqrt{<x,x>}}, & y > 0 \\ 2\pi - \arccos \frac{x}{\sqrt{<x,x>}}, & \textit{otherwise} \end{cases} \tag{2.2}$$

This also has to be done the other way round, when randomly choosing an angle, where d refers to a constant distance and α to the angle.

$$\vec{x} = \begin{pmatrix} \cos\alpha \cdot d \\ y \end{pmatrix}, \qquad y = \begin{cases} -\sqrt{d^2 - x^2}, & \alpha > \pi \\ \sqrt{d^2 - x^2}, & \textit{otherwise} \end{cases} \tag{2.3}$$

For the study in chapter 6 angle deviations are logged to a file. These angles are measured clockwise in contrast to the standard counter-clockwise opening of angels.

2.4 Head-related transfer function

The head-related transfer function (HRTF) is the result of sound modifications caused by the human's head diffraction and reflection properties. The HRTF is measured for each ear and contains the amplitude changes for, in the best case, every frequency (see Figure 2.5). The human's possibility to locate sounds, is heavily depended on this HRTF and is specific for every human. That is why a working OpenAL implementation, which is used for object localization in-game, needs to use at least an approximation of a HRTF. Some implementations of OpenAL even support setting an own HRTF.

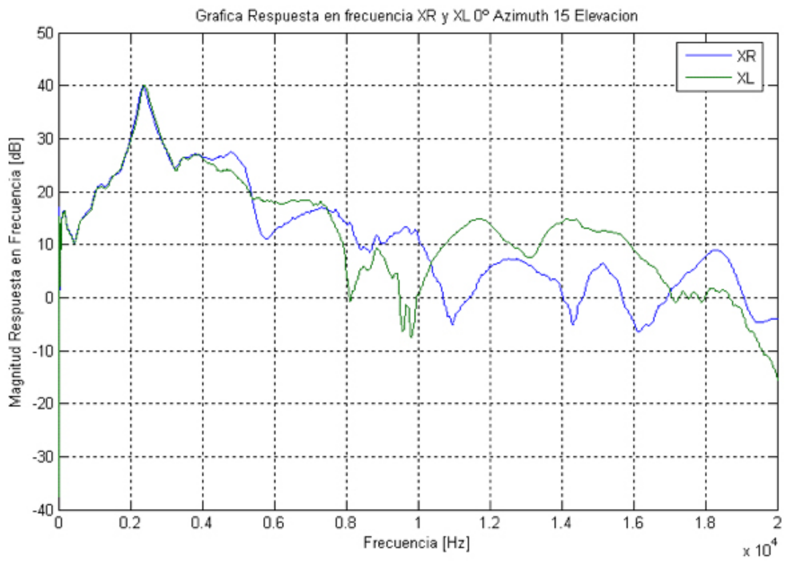


Figure 2.5: Example of a HRTF [10].

3 Requirements Analysis

For developing this application functional as well as non-functional requirements are constructed beforehand and are late compared with the final application in chapter 5.

3.1 Functional Requirements

The functional requirements in the following table describe the features the application must have.

Table 3.1: Functional requirements

	Requirement	**Explanation**
FR#1	The game should be designed like a game.	Because it is the goal of this thesis to create a game the design and architecure should be implement for a use in the game.
FR#2	The user should be able to locate sounds.	To be able to locate sounds the game must implement spatial audio.
FR#3	The game should be able to detect the player's real orientation.	To enable the user to control the game there is a need to detect the player's real world orientation.
FR#4	The game should have a main menu for starting a game.	As there are a lot of settings for example background sound and level there needs to be a main menu.
FR#5	Sounds in the game should be randomly positioned around the player.	To measure differences between study participants the sounds should be placed in a circle around the player.

FR#6	It should be possible to play several rounds of a game with the same settings.	As the game is used for a study it should be possible to have an option to play a dynamically set number of rounds.
FR#7	The game should log data at certain events.	For the study there is a need for logging data.
FR#8	The game should be able to display results of a game session.	After every round the user should be displayed with his result, including an image of the current looking direction.
FR#9	A game session should be able to be interrupted.	As some levels are seperated into several different parts the game should be interruptable and display only a part of the result.

3.2 Non-functional Requirements

The non-functional requirements in the following table, describe the qualities of the application. Especially important for mobile applications, because they depend heavily on user interaction and the resulting non-functional requirements.

Table 3.2: Non-functional requirements

	Requirement	**Explanation**
NFR#1	Displaying the end screens should not take more than $500ms$.	Because the users should not experience any stutters there is a need for the end screen to be generated in a time a user will not recognize.
NFR#2	Fluid playback without interruption.	Because the basis of the game is audio fluid playback is very important.
NFR#3	The game should not only be usable on the test device.	As the game may be used on other devices, it should work on a wide variety of iOS devices.

NFR#4	The game should log the time of the user with accuracy of more than 0.01s.	Because the study compares time it is important that times are accurate measured.
NFR#5	The game should be reliable.	No matter what user interaction happens, the game should not hang, crash or stop.

4 Implementation

This chapter handles the game application described in the previous chapter in the game basics section. The application has been programmed in Swift and Objective-C. The IDE Xcode was used, because it is the only official IDE supported for iOS programming. The game logic was programmed in Swift, whereas the OpenAL wrapper used in this game was programmed in Objective-C (see Figure 4.9). As test device an iPad Air of the first generation was used. An iOS device with version 7.0 or higher is needed, because of the usage of the Swift programming language.
The following chapter deals with the game logic and its architecture. The section afterwards gives insights in the input handling of the game followed by practical implementation details. After that the user interface is discussed. Then, the topic of concurrency, which is used almost everywhere in the game is covered. The OpenAL wrapper is explained afterwards and last but not least it is outlined how all these components interact and work with each other.

4.1 Game

This section deals with the implementation of the game logic. It covers the architecture of the game, as well as the structure of the game.

4.1.1 Architecture

The architecture of the application is based on a typical MVP(Model-View-Presenter) pattern of iOS [15]. MVP is seperated into three parts.

1. The model is the logic of the view. It handles everything from inputs to sound positioning and all the game logic.
2. The view is responsible for displaying information and passing inputs. The view is controlled by the presenter. In a modern iOS Application these views are generated within the storyboard.

3. The presenter is responsible for preparing data from the Model to prepare for display on the view. This can be angle conversions, image generation and so on.

The game is implemented in a tick based manner (see Figure 4.11). This means, that the game updates itself in constant time intervals. This is often used in continous systems where predictability is important. This makes it a lot easier to extend the game with features, which are simulating a continous system like moving sounds or physics. But even adding multiplayer where it is important to predict the game's behaviour, is possible more easily. For higher performance all operations triggered on sounds are executed concurrently. Another detail is that the sound's distance to the player is always constant and equal to the OpenAL reference distance, in order to eliminate possible distance fade problems, when sounds differ from the reference distance.

4.1.2 Game Structure

As a game can be very complex, there is a need to structure the game into a lot of small modular and exchangable parts. Therefore, a lot of interfaces are used. Every object in the game is referenced as entity. All these entities are bound to a level. This also includes actions which are triggered through certain events, implied by an input or an in-game event.

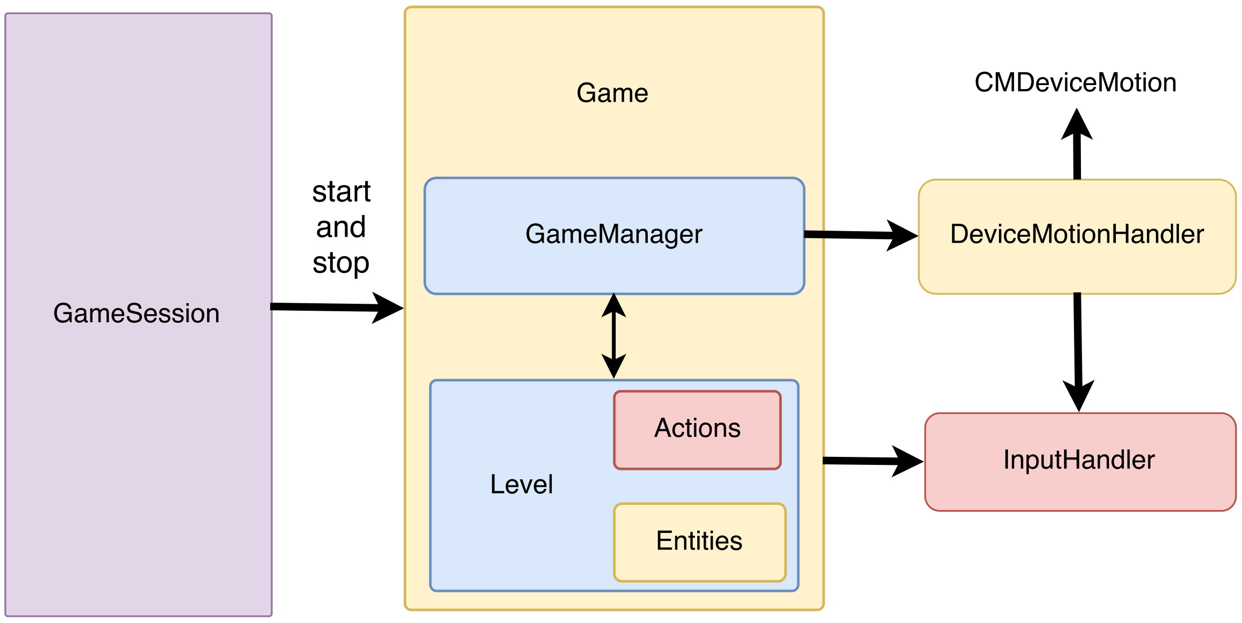


Figure 4.1: Game and its underlying structure.

Entities

All objects in the game are called entities. An entity has to implement the entity interface. Because the game is implemented in a tick based manner every tick based object has to be updated in each of those ticks, therefore an interface, having an *update* method, must be provided for all objects being in the game and handled by the *GameManager*. But not only updating is a very common task dealt within every tick, also movement is often a crucial part in a game, that is why all entities also have getters and setters for a position. Because this game is mainly focused on sounds, there is also a sound entity inherit the entity interface. A sound in addition to an updating functionality and positioning needs control over the audio connected to the classes. Therefore, this interface has an additional *play*, *stop* and *pause* method, but also methods considering audio information and current status. In the game it is also very common to randomly position entitites in a circle around the player, that is why there is a third interface further inheriting from the sound entity. This type of entity also has methods to specify a range of angle and distance, from whose ranges, random values are chosen. Because sound entities in the game, also need to be displayed, this interface also includes a getter for an image. How these interfaces inherit from each other is displayed in Figure 4.2.

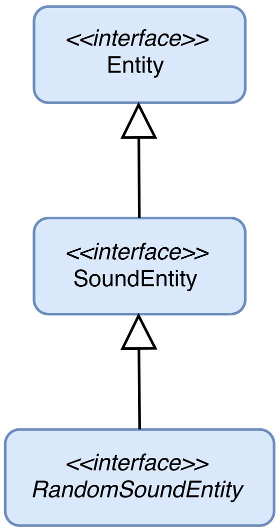


Figure 4.2: Entity interfaces.

Action

Whenever a user interaction or a game event happens, the game has to provide an action. Because there are not much user interactions happening in the game and because they are heavily dependend on the current level played the handling of actions is outsourced to the level. Every time when a user interaction happens, the game chooses an action from the level by running the *getAction* method of the current level, which then is executed from the object making the user interaction (see Figure 4.3). Every one of those actions is meant to provide a reaction to a user interaction in a specific state of the level. In the current implementation the *execute* method only accepts an angle and the current controller, as this is the only relevant data needed by an action. For later use it would be good to have an additional event class containing data obtained by the event and objects like the controller involved in the event.

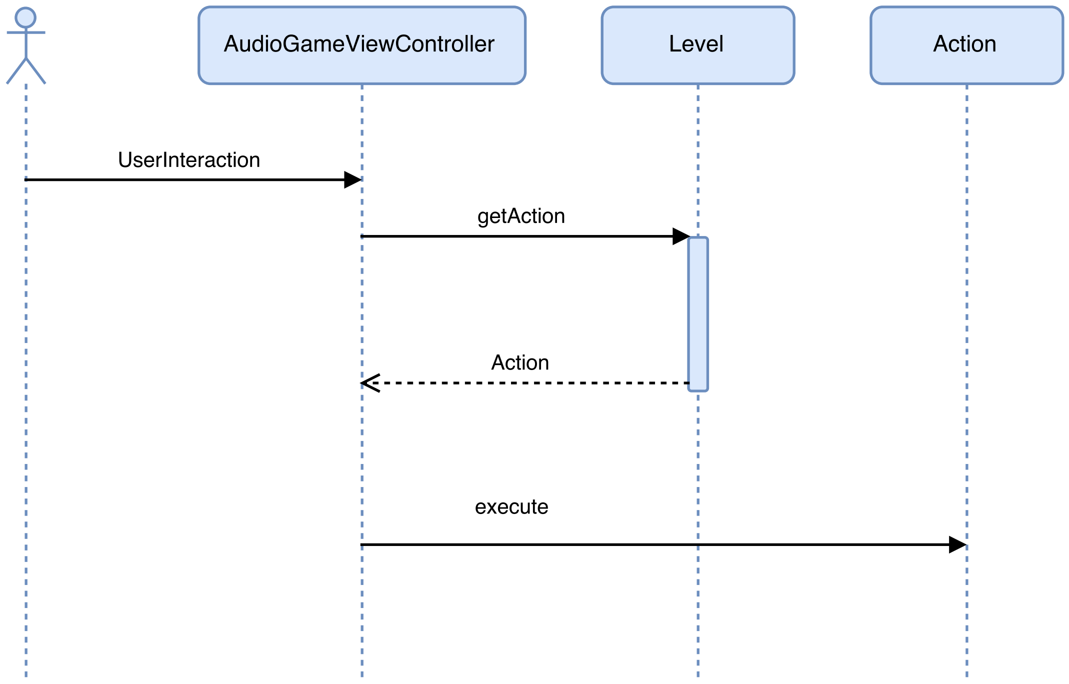


Figure 4.3: Action choosing.

Levels

The level is one central part in the game. It not only initializes entities, but also updates and manages them in its data structure. In this work it is also responsible for choosing and managing actions to a user interaction. Therefore, the level needs to maintain a condition for which it is decided which action to choose. Because there is nothing happening in the level while the game is running expect for user interactions, the level mainly acts as a data structure.

GameSession

Because the study, performed in a later chapter, runs several rounds of one level it was decided to introduce a *GameSession* class, which includes a username, a logger, a level and an amount of rounds to be played. A central part of such a session is that it can be paused and has a possibility to restart a level. This is why this class provides corresponding methods. It also handles how the user interface reacts when the session is paused, stopped, started or restarted.

4.2 Input handling

For input handling there are two classes. The *InputHandler* class and the *DeviceMotionHandler* class. The *InputHandler* is responsible for holding all inputs, for example the yaw of the device. The *DeviceMotionHandler* is responsible for handling the devicemotion, which is starting, stopping, pausing and updating the device attitude and saving it to the *InputHandler* and setting the direction vector for the listener in OpenAL.

4.2.1 Sensor

For distinguishing the player's direction, the *CMDeviceMotion* class of the iOS CoreMotion library is used because it combines the gyroscope and the accelerometer, which is more accurate then both of the methods seperately, as you can differntiate between user acceleration and gravity. Especially the accelerometer on its own is very error prone because of the ongoing error involved in the double integration for getting the traveled distance. The *CMDeviceMotion* class is fed with motion updates every $16,66ms$, which is the time a tick takes in the game. These motion updates include the attitude of the device in relation to a referenceFrame. This referenceFrame is the device's attitude at the start of the device motion updates. The attitude object has three properties, which are pitch, roll and yaw (see Figure 4.4). In our case only horizontal rotation is important, so only the yaw value is used for distinguishing the player's direction. As OpenAL requires a normalized direction vector, some calculations must be made before passing it to the listener direction function. In the equation (4.1) $\vec{x}$ is the normalized vector and α is the yaw angle [3, 4].

$$\vec{x} = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}, \qquad y = -\cos\alpha, \quad x = -\operatorname{sgn}(\alpha)\sqrt{1-y^2} \tag{4.1}$$

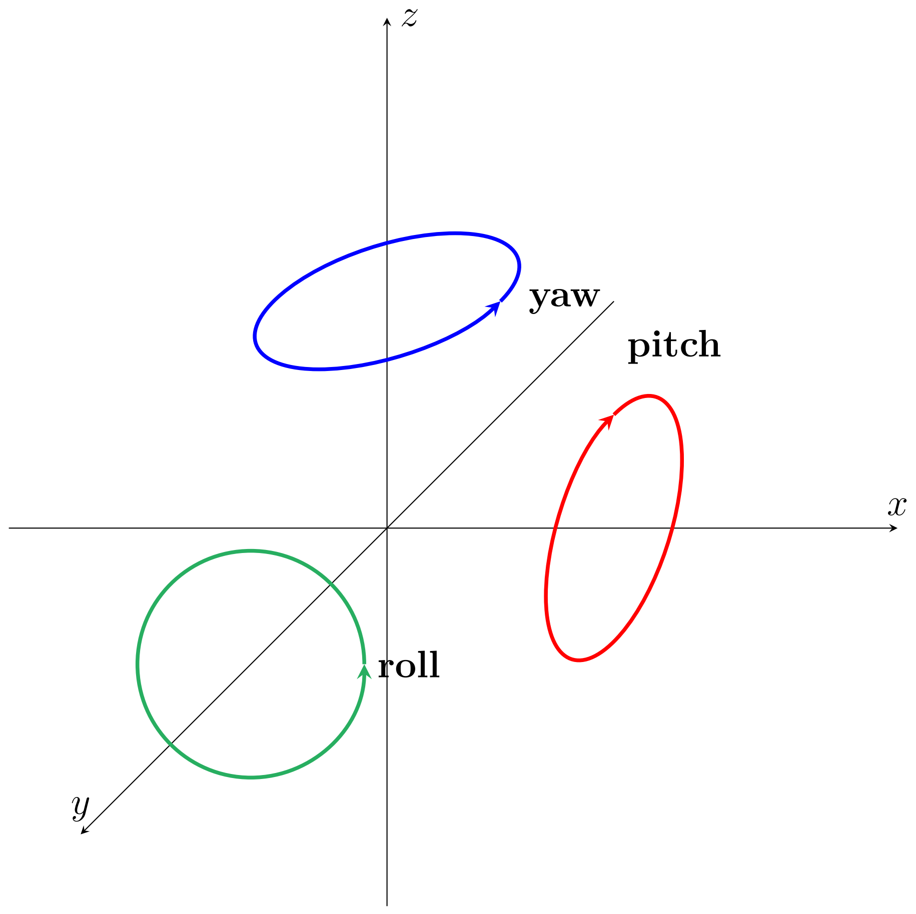


Figure 4.4: Pitch, yaw and roll.

4.3 Details

This section covers special insights in the game application and how problems were solved in the development progress.

4.3.1 Class Variables

In the current implementation it was not possible to have class variables for a class in swift, but only a custom variable with getter and setter. Therefore, a work around was needed (see Figure 1). In newer Swift versions higher or equal to version 1.2 this problem was solved [27].

```swift
class Minimal {
        // Hack for class variable
        private struct SubStruct {
                static var sharedInstance: Minimal?
        }

        // SharedInstance of Minimal
        class var sharedInstance: Minimal! {
                get { return SubStruct.sharedInstance }
                set { SubStruct.sharedInstance = newValue }
        }
}
```

Listing 1: Workaround for class variables.

4.3.2 Singleton Pattern

In the application the *Game* class is the central part. Because there is only one game needed, the singleton pattern is applied. That is why there is a need for a method which returns the current instance or creates one if none existent. In iOS it is common to use a sharedInstance class variable, to access the current instance or initialize it, so that the call Game.sharedInstance returns the current instance of the game.

4.3.3 Image Creation

Every image object of the class *UIImage* has a drawInRect method which accepts a rectangle object. This method draws the image in the provided rectangle to the current image context. As we generate an image we have to begin the context with *UIGraphicsBeginImageContext*. After the context is set you can start drawing the images onto the context. After the image has been created it can be obtained from the current context with *UIGraphicsGetImageFromCurrentImageContext* and the context can be ended with *UIGraphicsEndImageContext*.

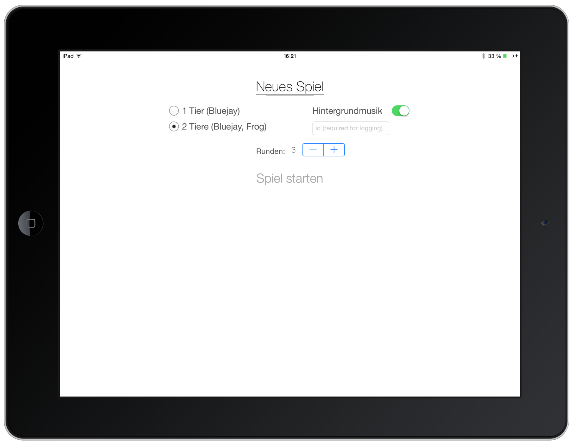
4.3.4 Factory method pattern

The factory method pattern is a creational pattern, when looking at object-oriented programming languages. In the *GameSession* a level needs to be created over and over again. For this a string of the corresponding level class is saved, because it was impossible to save a direct reference to the class. The level is then created with the factory method pattern, which exactly solves this problem of creating objects without specifying the exact class [8]. That is why the *GameSession* can create an appropriate level with one call for every level case.

4.4 UI

A UI is an important part of a mobile application, as it guides the user through the application. In iOS a view is an element in which positioning and behaviour of UI elements are combined. A view is always bound to one so called *UIViewController* which acts as a presenter and sets the content of the view, but also is the first instance when a user interaction happens. All views and their relation are saved in a storyboard file. Such a relation can be a segue which is a route to another view and includes how views are faded. A segue also can be named and be triggered programmtically, which is the technique used by the *GameSession* class to switch between views.
There are four views in the application. The first one is the main menu view (see Figure 4.5) that is bound to the *NewGameViewController*. In this view it is possible to set the game mode, the background sound setting, the user name and an amount of rounds. After clicking on the 'Spiel starten' button, a new *GameSession* is initialized. If the game is started the game moves into a running state (see Figure 4.6). The corresponding view only displays a camera and an instruction. This instruction is dynamically fetched from the current level. When tapping the camera the level chooses an action, which in this work always triggers the *GameSession* to pause or stop the game and change the view to a result view. In this view an image is provided, in which targets

are placed relative to the shooting position. This image shows a section of 45 degress. This view also provides a text area, where results are presented. There are two such result views one for an interrupted game and one for the end of a game (see Figure 4.7). The *IntermediateViewController* is responsible, whenever the Game is paused and the *EndScreenViewController* is responsible, whenever a Game is stopped or ended.



Neues Spiel

1 Tier (Bluejay) [✓ 2 Tiere (Bluejay, Frog)]

Hintergrundmusik: [✓ on]

id (required for logging)

Runden: 3 − +

Spiel starten

Figure 4.5: The UI of the main menu with settings.

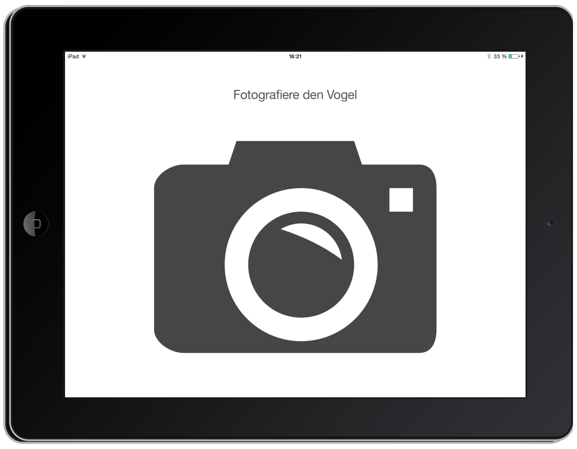
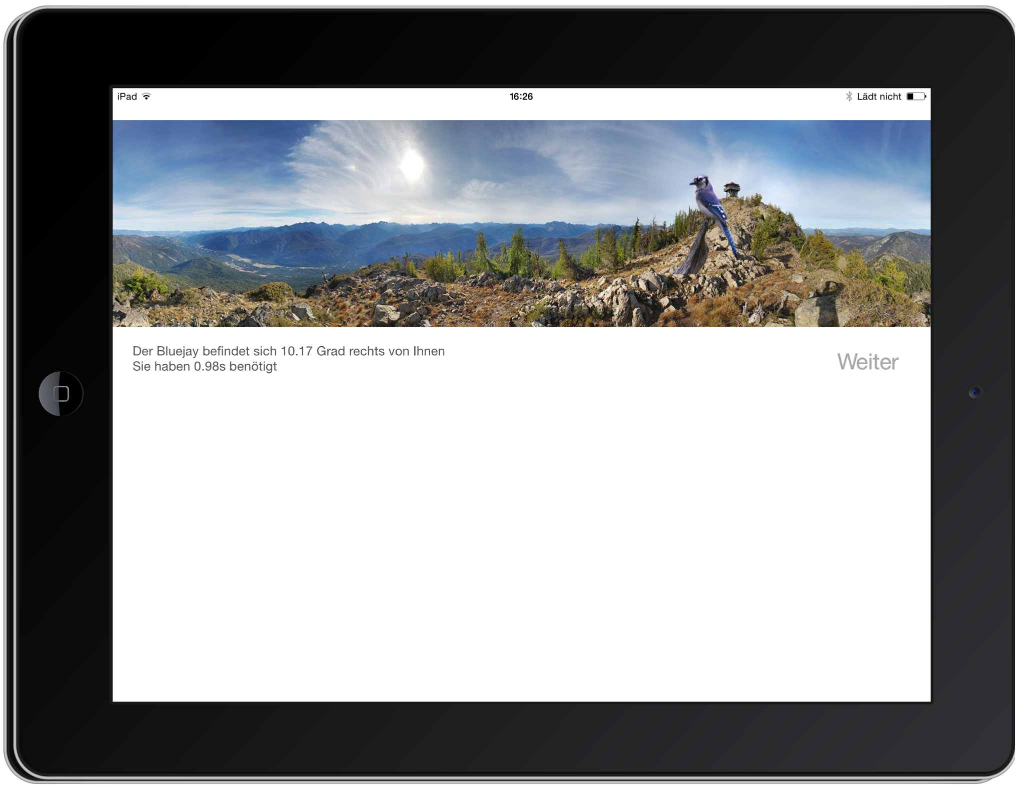


Figure 4.6: The UI state in the running game.



Der Bluejay befindet sich 10.17 Grad rechts von Ihnen
Sie haben 0.98s benötigt

Figure 4.7: The UI state for the result.

4.5 Concurrency

An important aspect of the implementation is concurrency. A lot of actions happen in parallel, like playing sounds, updating sound entities or updating the player position. This has a lot of benefits in regards of performance but also implies a lot of caution while programming. There are some ways to implement parallelism on computers, with techniques like OpenMP, OpenMPI or threads. In iOS there are not very much options to implement parallelism, it only supports threads. There are two ways to use threads in iOS, one is using the *NSThread* class, the other way is to use POSIX-Threads. In this thesis the *NSThread* class is used, as it has more options than a standard POSIX-Thread. POSIX-Threads would have had the benefit, that they can be used for almost every platform. But as only one platform is covered in the thesis, the benefits of the *NSThread* class are predominant. Threads always run with shared memory. This implies problems like conflicting resource access. So if it is the case that threads run on the same data structure, measurements like locks or atomic operations must be used. If using threads it is also not assured, that threads run on seperate cores, this heavily depends on the operating system's thread scheduler.

4.5.1 Concurrent objects in the game

In this thesis almost every object runs in seperate threads. Most important is the *GameManager* providing the tick of the tick-based architecture used in the game. But not only the *GameManager* but also all entities run in seperate threads. This rapidly increases performance on calculation or time intensive operations on entities. If not taking the right measurements, this can also destroy a continuous system, which in first place was the basis of the decision to use a tick-based system at all. Therefore, parts of the game which run in threads and depend on the *tick* of the *GameManager* must be synchronized with the *tick* of the *GameManager* to ensure that a tick by itself can always be reconstructed. Because sound entities in the current scenario are not dependend on a tick or each other and use only their own resources, they can operate completely seperate from each other and that is why none of such synchronizing elements were implemented in this work. Not only in the game itself measurements were made to ensure threads can incorporate with each other, but also in the OpenAL library. ALSource structs (see Listing 4) are thread-safe and use atomic variables as well as locks. Another part of the game which runs concurrent is the input handling. In Figure 4.8 all objects that run concurrent are displayed and connected to their responding component.

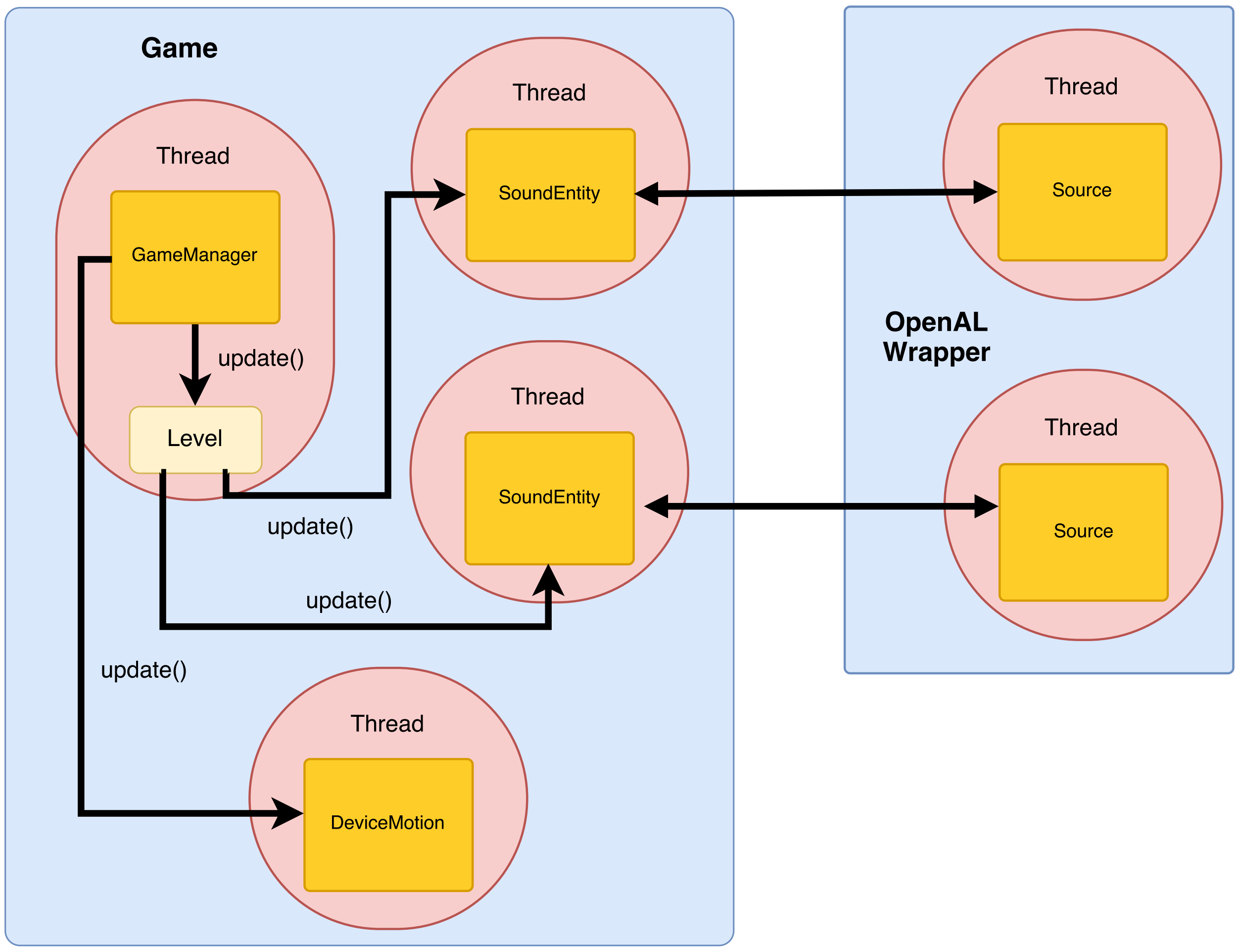


Figure 4.8: Concurrency in the game

4.6 OpenAL wrapper

For the implementation of this audio game the library OpenAL is probably best fitted, because of its big community and compatibility. Moreover because there is already a built in C-implementation in iOS. As this library is written in C it can not directly be used in an object oriented language like Swift. Therefore, there is a need for a wrapper projecting the functional interface of this library onto an object oriented one, because Swift is lacking support of directly calling C code. This wrapper needs to be written in Objective-C, which is then compatible to Swift. In Figure 4.9 it is visualized how Objective-C and

Swift Code can work with each other. Using Swift code in Objective-C can be easily achieved by importing an automatic generated header. These headers are named after there class name and have a '-Swift.h' extension to it and can be imported only in the Objective-C implementation file. If desired, Swift classes can also be referenced in Objective-C header files with a @class keyword. To use Objective-C in Swift there are no generated headers. Every Objective-C class needs to be imported in a so called bridging header. After importing the Objective-C class in the bridging header the class can be imported in a Swift class with the standard *import* keyword.

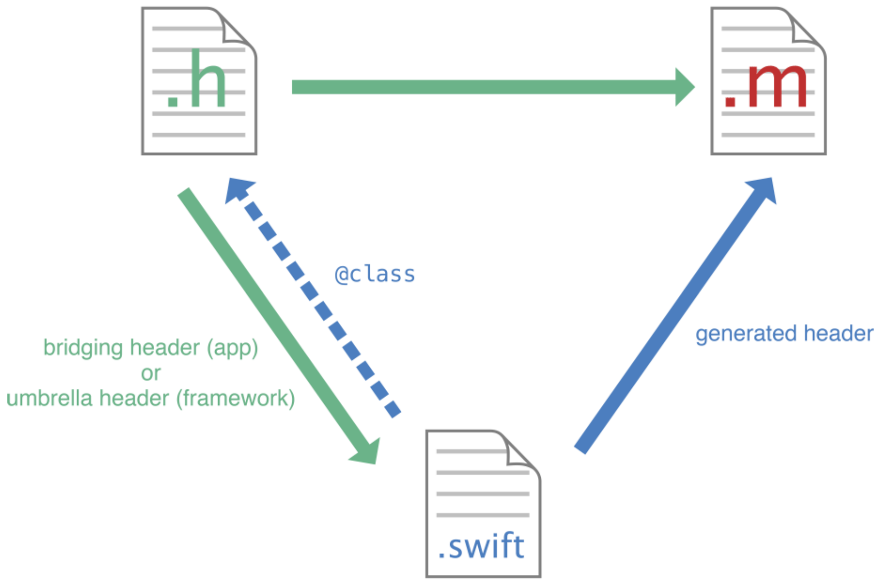


Figure 4.9: How Swift and Objective-C corporate with each other [11].

4.6.1 Structure

In OpenAL there are two basic entities. These are multiple sources and one listener. Both of these types can be three dimensionally positioned and orientated. The sources are the entities playing sounds, whereas the listener is the entity receiving these sounds. The wrapper which is built around the OpenAL API is structured into a Buffer, Source and a Soundpool. Where the Source handles positioning, the Buffer, the audio data and the SoundPool handles the rest which is listener orientation and audio initialization of the iOS audio device.

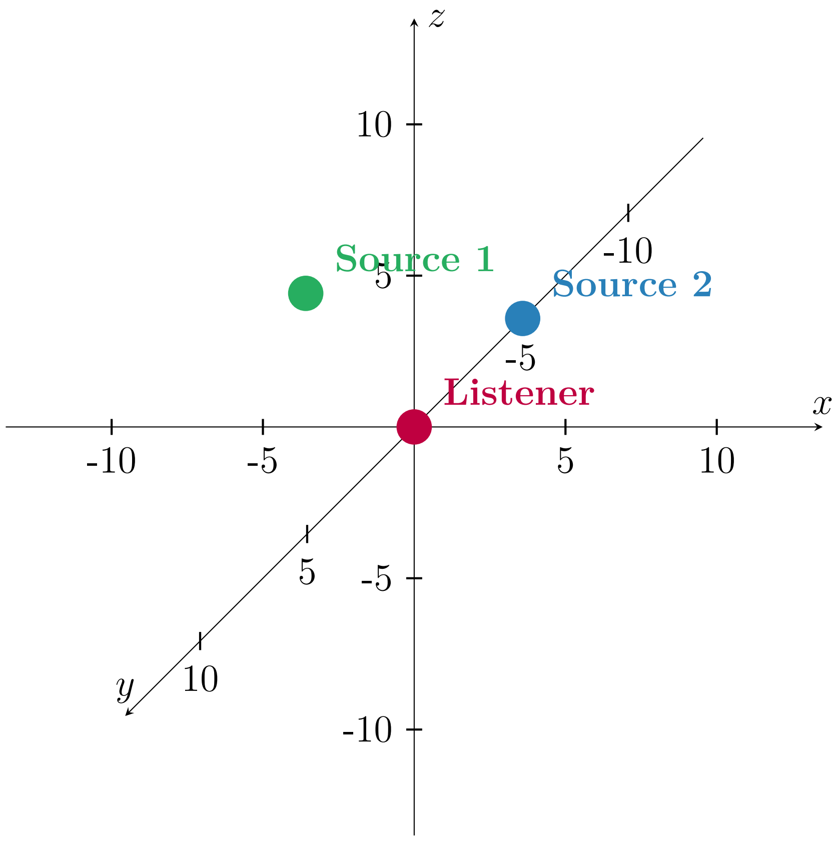


Figure 4.10: Listener and Sources in coordinate system.

4.6.2 Buffer

The buffer contains the audio data and its settings. The wrapper class is based on the OpenAL struct holding this data (see Listing 2). Those settings are frequency, format, sample length and amount of channels. Every buffer has a unique id with which it is referenced throughout all OpenAL functions. This id is also bound to the wrapper class. For the game case the native iOS audio format *CAFF* is used with mono channel and a 16 bit depth sound in little endian byte order. The sound needs be a mono sound in order for OpenAL to let the sound be positionable. A bit depth of 16 bit is very common and is used on Compact Disks, providing a high quality audio resolution which is more than enough when using headphones. The little endian byte order is used, so that the file can be directly copied to memory, as current processor architectures like ARM and x86 both support little endian byte order, whereas the big endian byte order is not support by x86 architectures. The sounds were all converted with the built in Mac OS command in listing 3. It was also required to write a C helper function for reading the file and copying its bytes

```
typedef struct ALbuffer {
        ALvoid  *data;

        ALsizei  Frequency;
        ALenum   Format;
        ALsizei  SampleLen;

        enum FmtChannels FmtChannels;
        enum FmtType     FmtType;

        ...

        /* Self ID */
        ALuint id;
} ALbuffer;
```

Listing 2: OpenAL source code ALBuffer(alBuffer.h) struct [18].

```
afconvert -f caff -d LEI16 -c 1 input.mp3 output.caf
```

Listing 3: Command generating *CAFF* file.

onto the heap and referencing its address for the OpenAL buffer. This function also sets the buffer settings like format, frequency and length.

4.6.3 Source

The Source is responsible for modifying the audio data according to position, gain and some more advanced properties (see listing 4). In our game scenario only position, gain and looping are important. So only those properties are available in the wrapper via getters and setters. The Source is also bound to a Buffer containing the audio source, required for playing a sound.

```
typedef struct ALsource {
        /** Source properties. */
        ...
        volatile ALfloat   Gain;
        ...
        volatile ALfloat   Position[3];
        ...
        volatile ALboolean Looping;
        ...
        /** Source Buffer Queue info. */
        ATOMIC(ALbufferlistitem*) queue;
        ATOMIC(ALbufferlistitem*) current_buffer;
        RWLock queue_lock;

        /** Current buffer sample info. */
        ALuint NumChannels;
        ALuint SampleSize;
        ...
        /** Self ID */
        ALuint id;
} ALsource;
```

Listing 4: OpenAL source code ALSource(alSource.h) struct [18].

4.7 Component interaction

This section handles how components interact with each other and what happens within the classes when a tick is triggered through the *GameManager* class. It also shows what happens when a user interaction is starting to happen and what steps are taken and which methods are called.

4.7.1 Tick action

To generate a tick, a loop in the *GameManager* class is used, which runs every $16,66ms$ and from there triggers all operations of a tick (see Figure 4.11). It firstly calls the *updateDeviceMotion* method of the *DeviceMotion* class. This methods retrieves the current yaw of the device and sets the listener orientation in OpenAL through a slightly modified equation (2.3) generating a normalized vector. The *GameManager* then starts updating the Entities with calling the current level's *update* method. There, sound modifying actions are made. In our scenarios nothing happens in these methods, as there is no movement or physics involved.

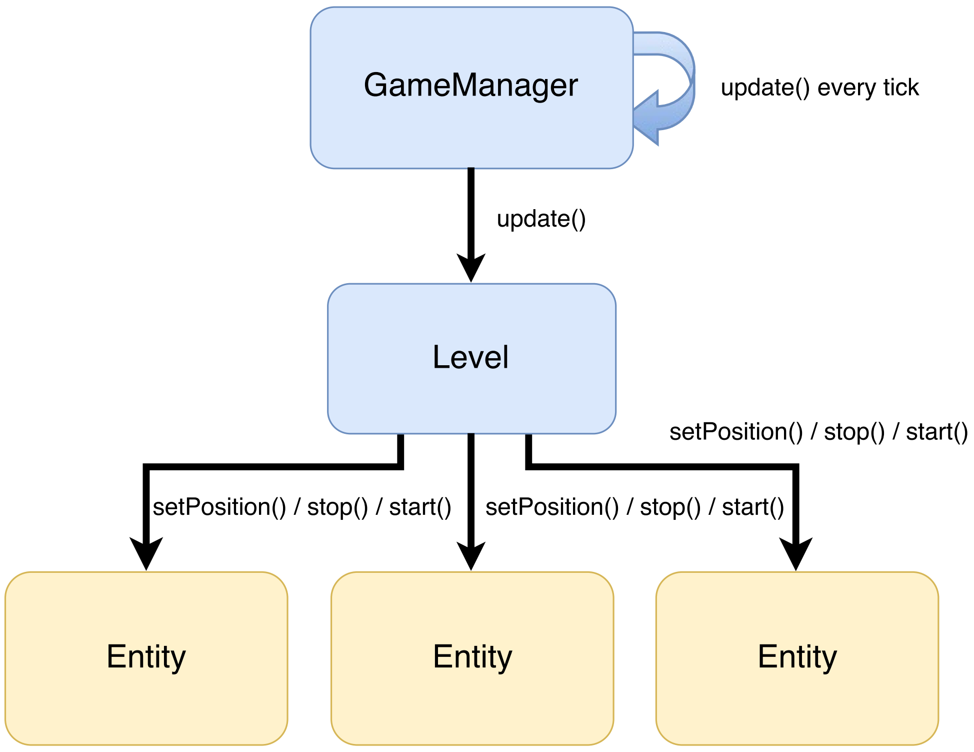

GameManager
update() every tick
update()
Level
setPosition() / stop() / start()
setPosition() / stop() / start()
setPosition() / stop() / start()
Entity
Entity
Entity

Figure 4.11: How a tick is applied.

4.7.2 User interaction

In this work, an action is called every time a user interaction happens. The actions in this thesis, always make use of the *GameSession* performing view changes.

Pausing/Stopping

Whenever a user interaction happens in a running game the *getAction* method of the current level is called and the returned action is executed, when in-game and the user interface is in the game state (see Figure 4.6). In this work all of these actions interrupt or stop the current level and therefore the current *GameSession*. These actions call the *pause* or *stop* method of the current *GameSession* instance, which has been intialized via the singleton pattern. But not only *GameSession* changes are triggered from the action but also logging takes place here. The static *log* method of the *Logger* is called and writes the results to the user's corresponding log file, which was set through the *GameSession* class at the start of current the session. Because the action called a *pause* or *stop* method on the *GameSession*, the *GameManager* is called to stop or pause the current game, with delegating the action to the underlying level and also stops the current thread. If the *pause* method is called this also means that the *DeviceMotion*'s referenceAttitude is maintained to ensure that the postions of the sound targets change to the player's direction. Also implied by the *GameSession* call is that the *GameSession* informs the current user interface controller to change the current view, appropriate to the called *pause* or *stop* method.

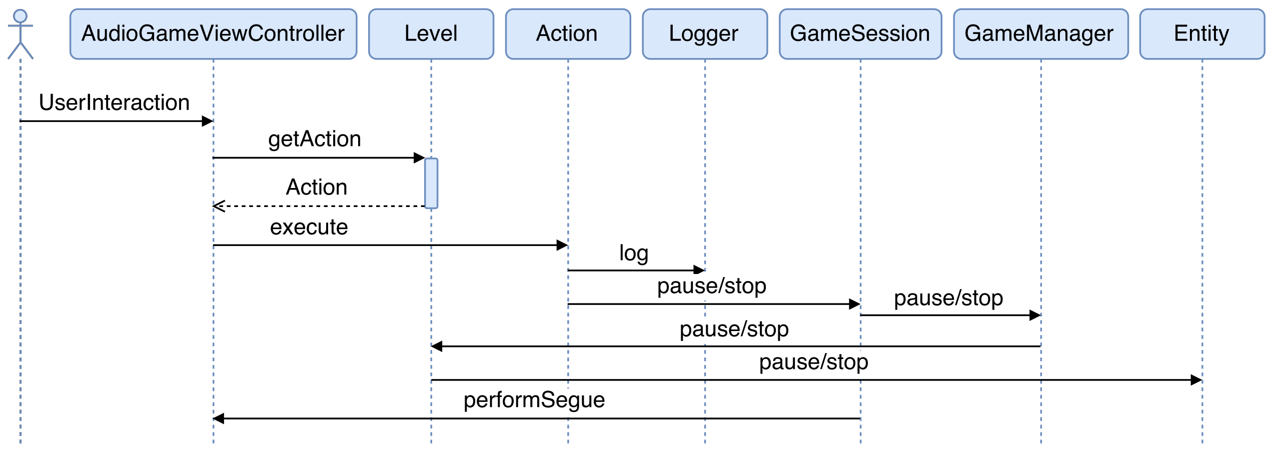


Figure 4.12: How the game is stopped or paused via user interaction.

Starting

Another user interaction happens when the user presses a button on the view corresponding to the *NewGameViewController* or the *EndScreenViewController*. The *GameSession* start method is called. This then calls the *LevelFactory*'s createInstance method to create a new Level instance based on settings and level name, which is saved in a *LevelClass* struct. The level is then passed to a new *Game* and this class starts the game in the following steps (see Figure 4.13).

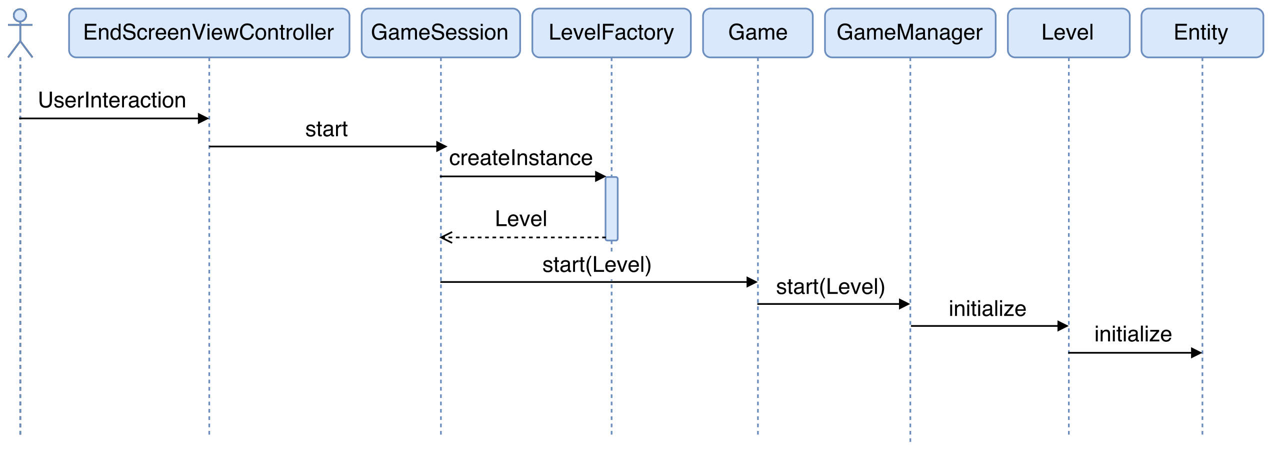


Figure 4.13: How game is started via user interaction.

5 Requirements Comparison

In this chapter the requirements from chapter 3 are compared with the end result of the thesis. All requirements are rated with [−−],[−],[o],[+],[++]. [++] meaning the best rating and [−−] the worst.

5.1 Functional Requirements

Table 5.1: Functional requirements comparison

	Requirement	**Rating**	**Comparison**
FR#1	The game should be designed like a game.	[++]	The game implements a typical tick-based systems, which can easily adept common game functionalities like physics and movement operations.
FR#2	The user should be able to locate sounds.	[+]	The sound localization was implemented with OpenAL which works very well, but as it only approximates the head translated transfer function it is not perfect, especially, if the sound is behind the player.
FR#3	The game should be able to detect the player's real orientation.	[++]	For detecting the player's direction the device direction is used. Therefore this only works if the relative position of the player to the device is not changed.

FR#4	The game should have a main menu for starting a game.	[++]	The menu has all fields required for being used in a study
FR#5	Sounds in the game should be randomly positioned around the player.	[o]	The sounds are randomly positioned around the player, but if several sounds are used, the positions are still completely random, and are not including the positions of the other sounds. This results in the problem also known as birthday paradox, forcing the participants of the study to follow extra instructions.
FR#6	It should be possible to play several rounds of a game with the same settings.	[++]	The possibility to play several rounds was implemented in a way that settings are applied on a new level a certain amount of times defined in the menu.
FR#7	The game should log data at certain events.	[++]	Every time the player starts a game, the time, deviation, current setting and target are saved in a log file.
FR#8	The game should be able to display the results of a game session.	[++]	After the user shoots, a result is displayed, therefore a image with all entities is generated and deviation and time are displayed.

FR#9	A game session should be able to be interrupted.	[++]	Every time the user shoots, the game is interrupted, therefore, the positions of sounds need to be maintained even when the player moves in the menu. This is handled in the Level class handling the current condition of the level.

5.2 Non-functional Requirements

The non-functional requirements in the following table, describe the qualities of the application.

Table 5.2: Non-functional requirements comparison

	Requirement	**Rating**	**Comparison**
NFR#1	Displaying the end screens should not take more than 500ms.	[++]	As the picture shows up instantly after the view switch animation, this operation has to take less time than the animation time which takes about 100msec
NFR#2	The sound should not stutter.	[++]	The sound is loaded into the OpenAL buffer at the start of the application, therefore the sounds can play immediatly wenn the level starts.
NFR#3	The game should not only be usable on the test device.	[+]	The UI was designed to work on iPhone and iPad devices. But there were no real world tests made and the UI only tested with the simulator.

NFR#4	The game should log the time of the user with accuracy of more than 0.01s.	[++]	Because the time is measured with the system function *mach_absolute_time()* the time has a theoretical accuracy of one nanosecond which corresponds to 0.000000001s [6].
NFR#5	The game should be reliable.	[++]	Because the game never crashed, was hanging or stopped, the game should be considerable reliable.

6 Study

To evaluate how good object localization works on different mobile platforms, every mobile application (Android, iOS, Web application and Windows Phone) was designed to provide study cases and a logging functionality additional to the game. Also data from questionnaires were obtained and evaluated. The following sections deal with the design, the results and finally the discussion of those results.

6.1 Study design

Three test cases were designed:

1. One sound is playing without background sound.
2. Two sounds are playing without background sound.
3. Works like the second case but with background sound.

All of these cases was played three times, so in sum nine rounds were played. For the study an option for a number of runs was implemented in the menu (see Figure 4.5). To compare the results, the player, the current time, the test case, the target, the time taken for the target and the angle deviation from the player to the target are logged in file. All values are saved comma seperated. Additionally, to the logged data, a questionnaire was given out to the participants to collect basic data like age, gender, phone experience, game experience and phone used on a daily basis.

6.2 Results

The quality of the results depends on the time taken per target and the resulting angle deviation. The average results for the platforms are plotted in Figure 6.1.

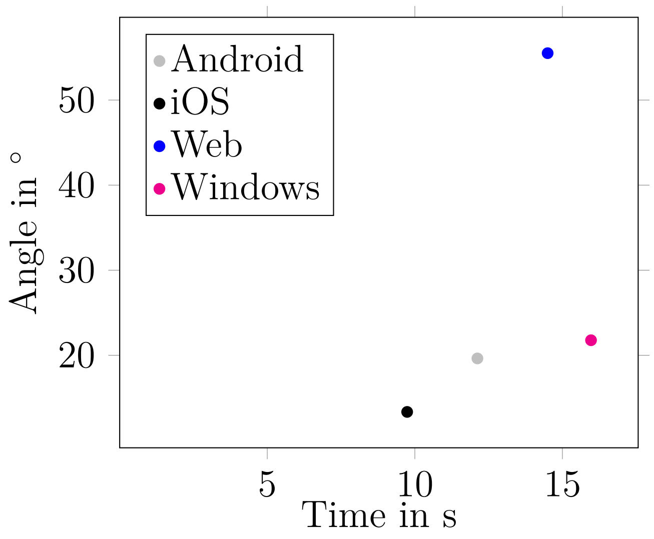


Figure 6.1: Study comparison between platforms.

As a further step p-values, comparing the targets, are calculated with the Analysis of Variance (ANOVA) and the T-Test (see Table 6.1 and Table 6.2).

Test	**p-value**
ANOVA	0.01
T-Test	0.01

Table 6.1: p-values for angle deviation.

Test	**p-value**
ANOVA	0.536
T-Test	0.536

Table 6.2: p-values for time.

For comparing the platforms it is also important how the participants performance is distributed on the specific platform (see Figure 6.2, 6.3, 6.4, 6.5).

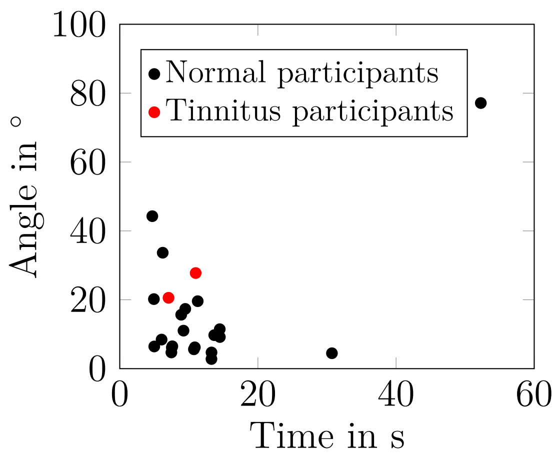


Figure 6.2: Android participants.

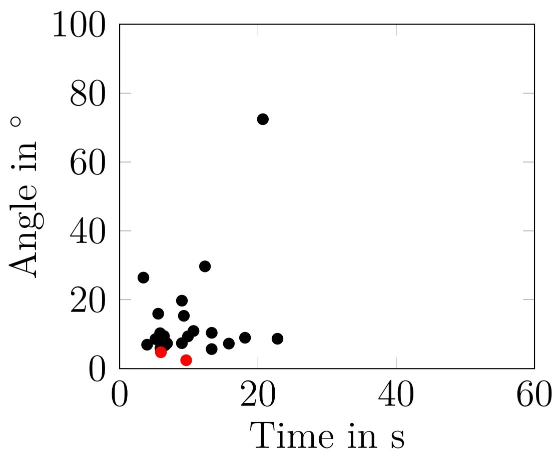


Figure 6.3: iOS participants.

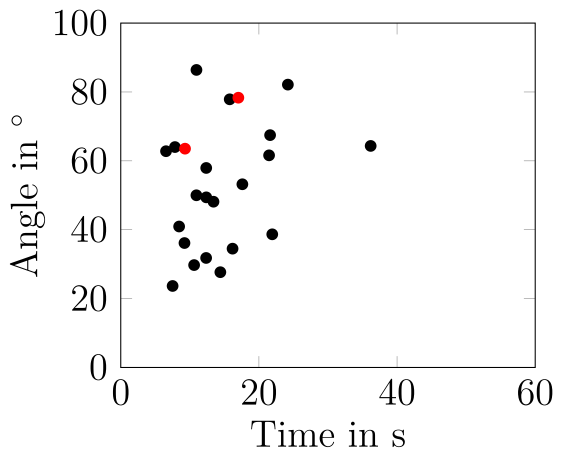


Figure 6.4: Web participants.

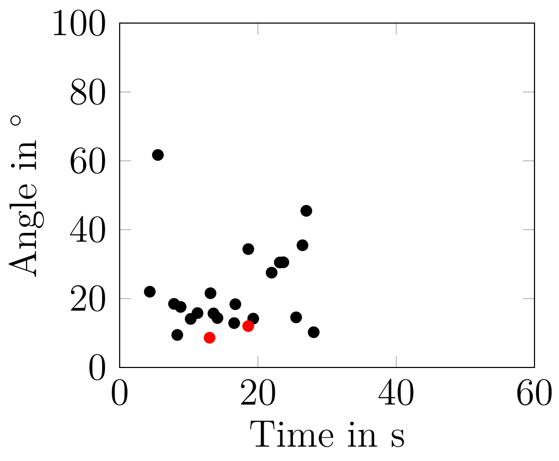


Figure 6.5: Windows participants.

To put the result in a context, the questionnaires were evaluated. The average age of the participants was at 27.8 years. The distribution of gender and personal mobile phones are examined in Figure 6.6 and Figure 6.7.

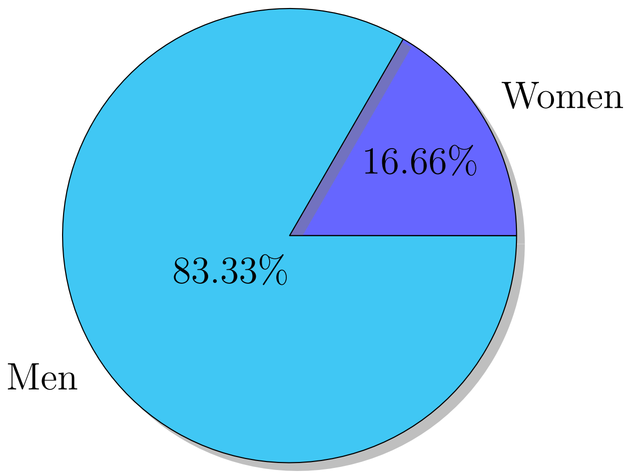


Figure 6.6: Gender of participants.

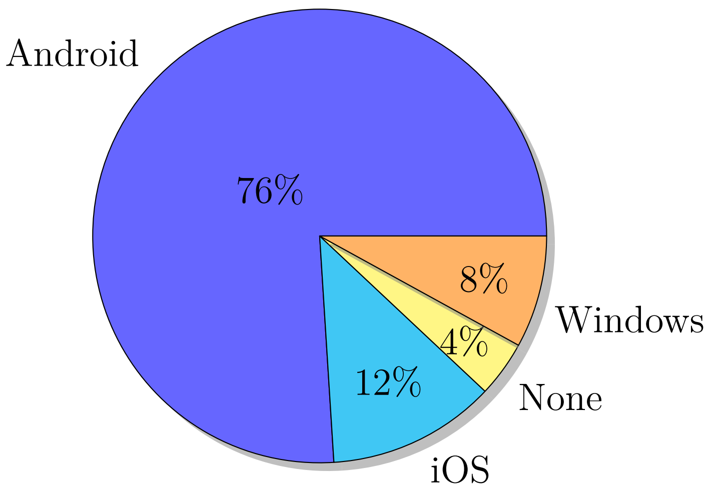


Figure 6.7: Mobile platforms of participants.

The participants were also asked how good they evaluate their mobile phone interaction skills and gaming experience in a scale from 0 to 5 (see Figure 6.8 and Figure 6.9).

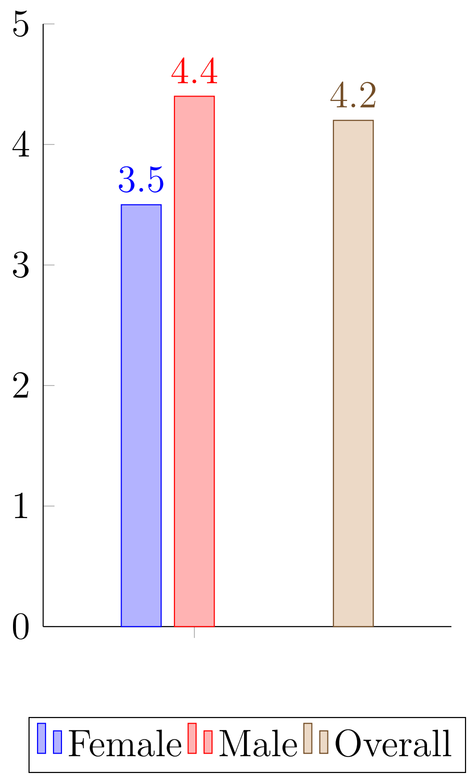


Figure 6.8: Skill level on mobile devices.

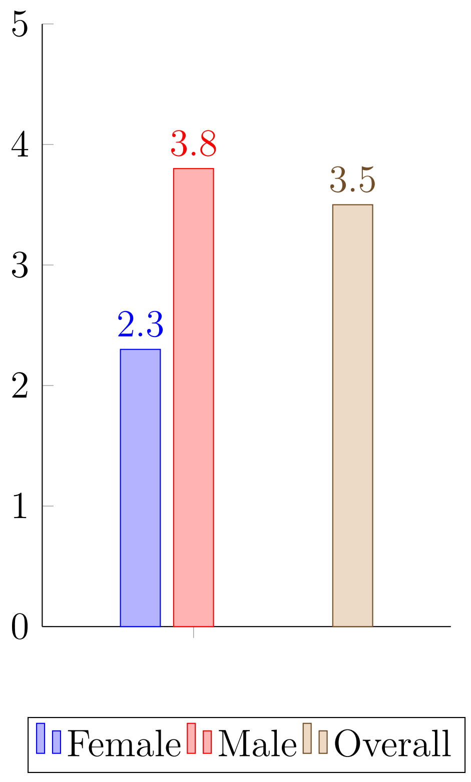


Figure 6.9: Gaming experience of participants.

6.3 Discussion

The following discussion about the results of the previous section, starts with a discussion about the platform only and how the platform and the implementation can effect the results. As a next step the psychological part is consulted. In this part it is dealt with how the participants affect the results. Also including the influence of the different platforms as well as how good tinnitus patients perform with spatial audio.

6.3.1 Platform only

All platforms are very close to each other, in terms of time and angle deviation. The small differences are probably due to the implementation specific details. One is that the sensor of the web platform is implemented with a compass that is not very accurate, causing the web platform to have the highest spike in angle deviation at about 55 degrees. Another implementation detail is that on the web and iOS platform, when two sounds are used in the second phase one sound is stopped, whereas on Android and Windows sounds still play. Also in the iOS implementation when using two sounds there is a relativly high probability caused by the birthday paradoxon, that both sounds are in the same location. This was compensated by instructing the participants to move between the two runs, but still might have a positive effect on the results for participants using the iOS platform.

6.3.2 Platform including participant data

As the participants are used to mobile devices (4.2 out of 5 see Figure 6.8) the user interface of the platforms should not affect results for the platforms. The participants are also very familiar to playing games, with a rating of 3.5 out of 5 and can very likely adopt seamlessly to the game scenario. In the native mobile applications of iOS and Android platforms the participants mostly are clustered in a small area in the left bottom corner of the graphs (see Figure 6.2 and Figure 6.3) which concludes that participants had no considerable problems when playing the game on these platforms. On the windows platform the participants are not as dense clustered as in the iOS and Android platform. This is caused by the participants taking more time in taking a photo of the targets. This can probably be traced back to spatial audio implemented in a way, that the resulting head related transfer function is not as near to the participants head related transfer function, as on the other platforms. But because the movement detection worked reasonable well the angle deviations are still very low. On the web platform, the problem probably resides in the movement detection, which can cause very random movements, making it almost impossible to hit the targets and also causing the time taken to be very high. Looking at the tinnitus patients they are doing very well on all platforms, except the web platform. It is also very interesting that these two participant with tinnitus have the two best values for angle deviation on iOS.

7 Recapitulation

The overall goal was to develop a game helping chronic tinnitus patients. Therefore this thesis was focused on evaluating if spatial audio can work on mobile devices. One goal of this thesis was to implement a game with spatial audio and evaluate its effectiveness in comparison to other mobile platforms. Therefore, a wrapper was built around the OpenAL library for the iOS platform in order to make is usable in a object-orientated environment like the Swift written game logic. The application was also extended to make it usable for the study that was performed.

7.1 Improvements

As there is a possible use case as an application for auditory training, it can be further improved. One of these improvements would be to introduce automated testing, to make it more easily possible to detect future errors. There also can be made improvements to the OpenAL wrapper, through outsourcing it in a library, the same thing can be made with the game itself, to make it reusable for more than one game scenario, like the one considered in this thesis.

7.1.1 Automated Testing

Testing, especially automated testing, is a crucial part in big software. Because of the small size of this bachelor thesis everything was tested by hand, via trial and error, but that can rapidly lead to a big amount of errors. A first step would be to write unit tests, which test every class on its own. Later on, a next step would be to write functional tests, which are meant to test a function on its own. And as a last step there would be the implementation of integration tests, including functional tests and also testing non functional requirements. The main problem when introducing testing is changing the implementation and maintaining another programming style. That is because every test should only test a very specific part of an application, so there must be possibilities to remove dependencies to other parts of the application. Therefore, techniques like dependency injection must be used beforehand when programming.

7.1.2 OpenAL Wrapper

As part of this thesis, an OpenAL wrapper was written to fit for the application. A first step would be to export the code to a library for use in other projects. With this step the wrapper must also be extended to reflect the whole range of OpenAL features and be adjusted to be functional as seperate part of the application. In regards of the aim of this bachelor thesis, which is to use this application for chronic tinnitus patients, there is a need to eliminate specific frequencies coherent to the tinnitus. Therefore, the specific iOS OpenAL extensions must be integrated, which include an equalizer for weakening specific frequencies. Also this OpenAL extensions include hardware accelerated audio manipulation like reverb, which would help simulating different environments in-game.

7.1.3 Game logic

Games and applications in general are constantly extended and changed. Therefore, concepts like 'Inversion of Control', which are often used in libraries, can be applied, making the game more modular and also more testable. So an extension to the game can be programmed without changing central parts of the program. The aim would be to use the game like a library, so that functionality sits in the library and a specific game scenario is just a use of that library functionality.
Also some minor changes need to be made beforehand. In the current implementation actions are chosen from within a level. This should be outsourced to a seperate class handling actions for levels. Also actions need to be changed. At the moment actions have an *execute* method, which accepts a yaw and the current controller. Because an action is a response to a event, a new event class should be introduced. This class should hold the data of a event, like controller source and input state. This event can be passed to an action, which then can respond accordingly. Also a tap on the screen directly triggers a action in the ViewController. This should be delegated to the *InputHandler* which then creates a new event and passing it to the action.

7.2 Closing Statement

This thesis is a starting point for tinnitus research with auditory training on mobile devices. It was outlined, if spatial audio works on mobile devices and it was successfull proven to work very well on the iOS platform. This thesis can be a basis for future work in this direction and may even relieve subjective tinnitus on some patients suffering from this desease.

8 Acknowledgement

I would like to thank my supervisor Marc Schickler for his support and his guidance through this thesis. I also would like to thank Paul Mohr for his patience in helping me with my struggles and keeping me motivated. A lot of thanks goes also to my classmates with which I worked on this awesome topic, implementing a great game on all major mobile platforms.

Another thanks goes to Apple, who created the iOS framework and the IDE Xcode used for creating the application. Also i like to thank Creative Technology for providing the OpenAL standard and Strangesoft for providing an OpenAL standard implementation.

Bibliography

[1] *2D Image Coordinates*. URL: https://developer.apple.com/library/ios/documentation/2DDrawing/Conceptual/DrawingPrintingiOS/Art/flipped_coordinates-2_2x.png (Last accessed: 24th October 2015).

[2] *Apple Inc. Q1 2015 Unaudited Summary Data*. 2015. URL: https://www.apple.com/uk/pr/pdf/q1fy15datasum.pdf.

[3] *Apple.com CMAttitude*. URL: https://developer.apple.com/library/prerelease/ios/documentation/CoreMotion/Reference/CMAttitude_Class (Last accessed: 10th October 2015).

[4] *Apple.com CMDeviceMotion*. URL: https://developer.apple.com/library/prerelease/ios/documentation/CoreMotion/Reference/CMDeviceMotion_Class/ (Last accessed: 10th October 2015).

[5] *AudioDefense*. 2014. URL: http://www.audiodefence.com/ (Last accessed: 1st November 2015).

[6] *CACurrentMediaTime*. URL: https://developer.apple.com/library/prerelease/ios/documentation/Cocoa/Reference/CoreAnimation_functions/#//apple_ref/c/func/CACurrentMediaTime (Last accessed: 7th November 2015).

[7] Jos J. Eggermont and Larry E. Roberts. *The neuroscience of tinnitus*. Paper. Department of Physiology, Biophysics, and Department of Psychology, University of Calgary, Alberta, Canada.

[8] *Factory method pattern*. URL: http://www.oodesign.com/factory-pattern.html (Last accessed: 7th November 2015).

[9] Philip Geiger, Marc Schickler, Rüdiger Pryss, Johannes Schobel and Manfred Reichert. “Location-based Mobile Augmented Reality Applications: Challenges, Examples, Lessons Learned”. In: *10th Int’l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps.* April 2014, pp. 383–394. URL: `http://dbis.eprints.uni-ulm.de/1028/`.

[10] *Image HRTF.* 2008. URL: `https://commons.wikimedia.org/wiki/File:FreqHRTF.jpg` (Last accessed: 1st November 2015).

[11] *Image Swift Objective-C.* URL: `https://developer.apple.com/library/ios/documentation/Swift/Conceptual/BuildingCocoaApps/Art/DAG_2x.png` (Last accessed: 3rd November 2015).

[12] *iOS Drawing Concepts.* URL: `https://developer.apple.com/library/ios/documentation/2DDrawing/Conceptual/DrawingPrintingiOS/GraphicsDrawingOverview/GraphicsDrawingOverview.html` (Last accessed: 24th October 2015).

[13] Peter M. Kreuzer, Veronika Vielsmeier and Berthold Langguth. “Chronic Tinnitus: an Interdisciplinary Challenge”. In: *Deutsches Ärzteblatt International* (2013).

[14] B. Langguth et al. *Consensus for tinnitus patient assessment and treatment outcome measurement: Tinnitus Research Initiative meeting.* Paper. Department of Psychiatry and Psychotherapy, University of Regensburg, 2006.

[15] *Model View Presenter.* URL: `https://de.wikipedia.org/wiki/Model_View_Presenter` (Last accessed: 5th November 2015).

[16] Hidehiko Okamoto, Henning Stracke, Wolfgang Stoll and Christo Pantev. *Listening to tailor-made notched music reduces tinnitus loudness and tinnitus-related auditory cortex activity.* Paper. Institute for Biomagnetism and Biosignalanalysis, Westfalian Wilhelms-University.

[17] *OpenAL.* URL: `https://www.openal.org/` (Last accessed: 5th November 2015).

[18] *OpenAL Soft.* URL: `http://www.openal-soft.org/` (Last accessed: 5th November 2015).

[19] Marc Schickler, Rüdiger Pryss, Johannes Schobel and Manfred Reichert. "An Engine Enabling Location-based Mobile Augmented Reality Applications". In: *Web Information Systems and Technologies - 10th International Conference, WEBIST 2014, Barcelona, Spain, April 3-5, 2014, Revised Selected Papers*. LNBIP. Springer, 2015. URL: http://dbis.eprints.uni-ulm.de/1137/.

[20] Marc Schickler, Manfred Reichert, Rüdiger Pryss, Johannes Schobel, Winfried Schlee and Berthold Langguth. *Entwicklung mobiler Apps: Konzepte, Anwendungsbausteine und Werkzeuge im Business und E-Health*. Springer-Verlag, 2015.

[21] Winfried Schlee, Isabel Lorenz, Thomas Hartmann, Nadia Müller, Hannah Schulz and Nathan Weis. *A Global Brain Model of Tinnitus*. Paper. Department of Psychology, University of Konstanz, 2011.

[22] Winfried Schlee, Martin Schecklmann, Astrid Lehner, Peter M. Kreuzer, Veronika Vielsmeier, Timm B. Poeppl and Berthold Langguth. *Reduced Variability of Auditory Alpha Activity in Chronic Tinnitus*. Paper. Department of Psychiatry and Psychotherapy, University of Regensburg, 2014.

[23] Johannes Schobel, Martina Ruf-Leuschner et al. "A generic questionnaire framework supporting psychological studies with smartphone technologies". In: *XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference*. June 2013, pp. 69–69. URL: http://dbis.eprints.uni-ulm.de/962/.

[24] Johannes Schobel, Marc Schickler, Rüdiger Pryss, Fabian Maier and Manfred Reichert. "Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned". In: *10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*. April 2014, pp. 371–382. URL: http://dbis.eprints.uni-ulm.de/1036/.

[25] Johannes Schobel, Marc Schickler, Rüdiger Pryss, Hans Nienhaus and Manfred Reichert. "Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned". In: *9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps*. May 2013, pp. 509–518. URL: http://dbis.eprints.uni-ulm.de/918/.

[26] Peter Steinlechner. *Golem.de Hörspiele*. URL: http://www.golem.de/news/hoer-spiele-games-ohne-grafik-1501-%20111884.html (Last accessed: 24th January 2015).

[27] *Swift class variables*. URL: http://stackoverflow.com/questions/24015207/class-variables-not-yet-supported (Last accessed: 24th October 2015).

[28] *Swift getting started*. URL: https://developer.apple.com/library/ios/referencelibrary/GettingStarted/DevelopiOSAppsSwift/index.html#//apple_ref/doc/uid/TP40015214 (Last accessed: 5th November 2015).

[29] *Swift programming language*. URL: https://developer.apple.com/swift/ (Last accessed: 5th November 2015).

List of Figures

List of Tables

List of Listings

Name: Stefan Mayer Student Number: 793098

Statutory Declaration

Hereby I declare that I have authored this thesis with the topic:

"Implementation and evaluation of a mobile iOS application for auditory stimulation of chronic tinnitus patients"

independently. I have not used other than the declared resources. I have marked all material which has been quoted either literally or by content from the used sources. Further I declare that I performed all my scientifical work following the principles of good scientific practice after the directive of the current "Satzung der Universität Ulm zur Sicherung guter wissenschaftlicher Praxis".

Ulm, ...

Stefan Mayer