Implementation of RAYAV audio engine in a stealth video game.

Autorzy: Maciej Doniec, Karol Matyasik
Kierunek studiów: Electronics and Telecommunications
Opiekun pracy: dr inż. Bartosz Ziółko

Kraków, rok akademicki 2014/2015
Oświadczamy, świadomi odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonaliśmy osobiście i samodzielnie i nie korzystaliśmy ze źródeł innych niż wymienione w pracy.
Serdecznie dziękujemy Panu Doktorowi Bartoszowi Ziółko oraz Zespołowi DSP AGH, w szczególności panu Tomaszowi Pędzimąż, panu Bartoszowi Miga oraz pani Paulinie Bugiel za wszelką pomoc udzieloną w pracy z silnikiem RAYAV. Chcieliśmy także serdecznie podziękować naszym bliskim za wsparcie, zwłaszcza Marcinowi, za niestrudzone testowanie kolejnych wersji gry oraz dzielenie się programistycznym doświadczeniem. Na koniec dziękujemy wszystkim, którzy w jakikolwiek sposób przyczynili się do tego, że ten projekt ujrzał światło dzienne. Bez Was nie dalibyśmy rady!
Contents

1. Introduction .................................................................................................................................... 7
   1.1. Motivation .............................................................................................................................. 7
   1.2. Goals ...................................................................................................................................... 7
   1.3. Thesis content ........................................................................................................................ 7

2. Gaming industry and sound .......................................................................................................... 8
   2.1. In-game sound ....................................................................................................................... 8
   2.2. Audio engine - RAYAV ......................................................................................................... 10

3. Implementation of the game ......................................................................................................... 11
   3.1. Game engine - Unity ............................................................................................................. 11
   3.2. Concept behind the game ...................................................................................................... 12
   3.3. Plot ......................................................................................................................................... 12
   3.4. Technicalities ......................................................................................................................... 12
      3.4.1. Map ............................................................................................................................ 12
      3.4.2. Environment and assets .............................................................................................. 14
      3.4.3. Objects ....................................................................................................................... 14
         3.4.3.1 Physics .................................................................................................................... 14
         3.4.3.2 Interactive objects ................................................................................................. 15
      3.4.4. Player ........................................................................................................................ 16
      3.4.5. Enemy ........................................................................................................................ 19
      3.4.6. Sound ......................................................................................................................... 22
      3.4.7. Working process ......................................................................................................... 25

4. Summary ......................................................................................................................................... 27
   4.1. Conclusions ........................................................................................................................... 27
   4.2. Future ..................................................................................................................................... 27
M. Doniec, K. Matyasik  Implementation of RAYAV audio engine...
1. Introduction

1.1. Motivation

Our choice of diploma thesis topic was dictated by our great interest in constantly developing gaming industry. Since gaming in general is one of our hobbies we have looked to somehow transfer this passion into our academic work. We have wagered all possibilities and agreed on cooperating with dr Bartosz Ziółko and exploiting the RAYAV audio engine invented by AGH Digital Signal Processing team [1].

1.2. Goals

In fact, before getting to work our knowledge about game development was purely theoretical. The main goal of our thesis was to present RAYAV audio engine in the environment it was designed to prosper in. We had been hoping to produce a fully-working video game with the engine as its integral part and not necessarily the main feature. As mentioned at the beginning we were familiar with neither the audio aspect of the project nor with the actual game design. Thanks to the amount of work sacrificed to develop our game we have gained significant knowledge on the topic. The following pages of this document contain insight on our thought process and project work-flow as well as brief introduction to the domain of gaming audio.

1.3. Thesis content

1. Chapter II - thing on gaming industry and the current state of sound engineering, description of RAYAV and its principles of work.

2. Chapter III - description of the game including both conceptual and technical aspects.

3. Chapter IV- summary and conclusions, future of the project.
2. Gaming industry and sound

Ever since the birth of gaming industry the sound has played an important role in the gameplay, however never the most important one. Some may wonder why, but truth is as simple as it gets. Obviously, we are more visually than acoustically oriented and it takes its toll in various areas of life [2].

From the most simplistic of games such as arkanoids through tetrises and pacmans up to most up-to-date creations like Crysis, graphics have been key factor in the players’ perception of the game [3]. Among multiple genres defining the gameplay stereotypes and cliches we have chosen to represent the most dependent on audio which is survival horror. Current generation of the genre mixes the concept of stealth and adventure games with added fear factor [4].

2.1. In-game sound

Concerning sound, the technologies responsible for processing of what we hear were not evolving as fast as the graphics ones. In the cradle of video gaming, the hardware possibilities were largely limited restricting the developers to decide, what aspect of the game to focus on. Often, it ended with expenses in the audio domain making the technologies fall years behind the graphics ones.

There exist two major approaches to the matter of audio. The first, hereinafter referred to as passive relies on pre-processing the sound beforehand or even taking preferred audio effects into account at the early recording stage. Scarcely any hardware power is required at the playback time since everything is pre-configured, usually with the use of filters. They manipulate the sound to achieve desired results [2].

The second approach onward called active is quite different. It requires specific hardware to realize the audio in the real-time. The sound is adjusted on the go depending on geometrical and material parameters of the scene. Such technology is more demanding in terms of computational power and in its wake it required extension audio cards to work effectively. Analogous solutions simplify the production due to reduction of labor in the sound preparation phase. The engine performs the job for the developer [2].

Historically, the two ways of thinking have been competing to conquer the market. The passive one has always been considered viable and can often be encountered in present developments. The active approach, on the other hand, had to undergo some revolution. The ever-popular Creative Labs engineered technology abbreviated as EAX [5], Environmental Audio Extensions, which has been in prosperity from mid 90s up until now. It is served by series of Soundblaster cards and used to make the 3D audio sound...
in more realistic manner. In fact mostly focusing on reverberation, which means a persistance of sound after it was emitted. This does not reflect all the acoustic phenomena, but increases slightly the realism of 3D audio. In the 90s Creative Labs used to have a rival in a company named Aureal. The engineers from Aureal invented a dedicated extension card to serve the acoustic effects in real-time, much as RAYAV software does. Their A3D [6] engine was capable of detecting the directions of sound coming towards listener and handling the waves reflections. Aureal’s solution is still remembered across the Web, as one of the best audio technologies to ever exist. Unfortunately, it came to an abrupt end when Creative sued Aureal over patent issues. Despite winning in the court room the costs of the process led Aureal to bankruptcy. Creative did not waste time buying Aureal and burying their proprietary solutions, at the same time ensuring EAX monopoly.

![Gamer amazed with the Oculus Rift VR goggles experience.](http://cdn.bgr.com/2014/03/white-guy-oculus-rift.jpg)

Figure 2.1: Gamer amazed with the Oculus Rift VR goggles experience.

*source: [http://cdn.bgr.com/2014/03/white-guy-oculus-rift.jpg](http://cdn.bgr.com/2014/03/white-guy-oculus-rift.jpg)*

Nowadays given the potency of modern hardware few people bother with dedicated sound cards. The audio can now be served by constantly growing in power CPUs. Most vibrant software include OpenAL [7] which is a universal tool for generating 3D audio simulation with available hardware. Another solution is AMDTrueAudio [8] which shifts parts of processing to graphics unit. Thanks to additional hardware resources (built-in DSP processor) it can serve more audio at the same time improving in-game effects. However, none of the technologies offers much better outcome than the scripted audio. Introducing software like RAYAV into gaming market could really revolutionize the treatment of sound. Thanks to the real-time 3D audio technologies developers are able to achieve better reality and nonlinearity in their games. Finally, audio can be on par with graphics since the computational power is not as limited as it used to be. The industry is getting into a different dimension thanks to inventions like Oculus Rift [9], virtual reality goggles throwing player directly into the 3D game environment. Combining it with realistic 3D audio could provide experience unknown in the past.
2.2. Audio engine - RAYAV

Answering the market demand in certain way the RAYAV technology comes into action. RAYAV is an audio engine invented by engineers from AGH DSP team to utilize a technique known from computer graphics called ray- and ultimately beam-tracing. The similarities between audio and video processing result from the wave nature of both sound and light. Ray-tracing relies on following the propagation of sound wave from its source to the listener. On their way the sound waves stumble upon different obstacles. To reflect this the engine simulates changes to the traced ray based on the properties of materials and geometry of the acoustic scene. Triggered sound phenomena like reflection or diffraction occur simultaneously splitting the ray in the process. The resulting rays are tracked until faded. Listener hears the mix of rays coming from specific direction [10]. To improve the effects quality and reliability of the engine, instead of emitting a single one the source originates multiple rays combined into beams. That way the engine ensures that whatever the size of the obstacle is, to some extent of course, its influence on the sound will not be omitted and excluded from the listener’s perception. RAYAV is capable of mimicking most acoustic phenomena including Doppler’s effect that is the change in sound while its source is moving in reference to the receiver [11, 12].

![Image of RAYAV used in popular 3D FPS "Quake 3".](source: www.youtube.com)

RAYAV uses beam-tracing allowing developers to create real-like 3D soundscapes based on input environment parameters and geometry. The goal of its inventors was to use it in fully-featured video games, where player can experience not only advanced physics, but also major acoustic phenomena.
3. Implementation of the game

3.1. Game engine - Unity

By definition, the game engine is the heart of every single video game. On its foundation something bigger and more complex can be built. It coordinates all the important aspects of the game such as graphics, physics, audio and "the brain" which are scripts governing most of the behaviors. The engine supplies developer with necessary design tools and workspace to shine in. Nowadays there are multiple engines available. Bigger developer studios expanded their own proprietary technologies while others use some commercially accessible solutions like Unreal Engine (currently in version 4), Unity or Frostbite [13].

![Exemplary view of Unity workspace.](image)

One of the simplest and the most user-friendly of them is Unity [14], available either in free Basic version or the paid Pro one. It is capable of handling development of a game from the very beginning till the end. Moreover it boasts an enormous community producing assets and addons distributed via the online asset store. Part of them is downloadable free off charge although those more complex are paid. All those traits contributed to Unity’s success on the gaming market, making it a top choice for developers. It is currently one of the most popular game engines used by both amateurs to create various kinds of simple games as well as large companies in most complex projects. Even the biggest developer,
known as Blizzard Entertainment, has recently utilized Unity to produce a collectible card game called Hearthstone [15] which now holds a playerbase of around 20 million. Taking into account all the pros and cons, Unity turned out to be the perfect engine to power our game.

3.2. Concept behind the game

Our game is a stealth-heavy survival horror. Player is thrown into paranormal surroundings where nothing is obvious and predictable. The game features full 3D graphics seen from first person perspective. Its vital feature is the utilization of RAYAV engine to realistically model sound behaviour in dynamic 3D environment. It drastically improves player’s experience and helps building an atmosphere of danger which is key in horror games. Additionally, the game takes advantage of sound propagation assuring new gameplay opportunities.

3.3. Plot

The main character has to explore two floors of oddity in a trial to escape his insanity and avoid the inevitable end. Equipped with only a flashlight and being able to interact with objects he embarks on a journey full of terror. Venturing through corridors and rooms filled with darkness he does not expect the enemy lurking restless in the shadows. In the end, it turns out the protagonist is trapped within his own mind. No matter how hard he tries and how bravely he fights, he cannot prevail being placed in an endless loop of his personal trauma.

3.4. Technicalities

3.4.1. Map

Map is a crucial element in most if not all video games. In usual Unity powered products the map is designed entirely within the engine’s workspace since it possesses the components needed to arrange a complete viable scene. In our case the situation forced us to take a different path, because RAYAV demands being fed with 3D map modeled in external design tool i.e. Blender [16] which allows to assign parameters and material data to map’s parts.

Our goal was to create a complete 3D environment in a shape of labyrinth containing rooms which player can explore without any limits. What is worth noting is that the map consists of three floors, stairs, hidden passages and a lift shaft. It is the base for all interactions of physical objects in the game world and defines the game environment keeping the player enclosed inside.

One issue with Blender’s output is that normally the generic planes are transparent from one side. While this is not a problem for RAYAV it could cause some confusion in-game. Fortunately, Blender offers the remedy in its built-in functions, rigify and solidify which apply a specified level of thickness to the planes [16].
At the same time the map for RAYAV input, onward referred to as acoustic map, remained intact to limit the computations the engine had to carry out. However it required adding elements representing large in-game objects like statues, doors or pipes. The acoustic map includes information about materials which certain walls and objects are made of. Over 15 different fabrics were used with the likes of wood, sand, stone, grass and cork among them. Finally, the excessive number of triangles was cut out and mismatched connections between planes edges were fixed using appropriate Blender functionality.
3.4.2. Environment and assets

The current version of the map is divided into two separate floors. Both consist of carefully designed rooms contributing to the spooky climax. We have devoted a remarkable amount of time to prepare diverse looking rooms and unique passages. They do not bear much logical relation with each other since everything is a creation of the protagonist’s insanity.

Whereas we are not graphic designers the Unity store has come to aid offering wide range of freely available assets [17]. We have also used Blender models found over the Internet, as there is a possibility to import them into Unity from .fbx format. The map was covered wholly with wood texture. All other fabrics visible on the walls and floors excluding round carpet assets were made using an addon called Prototype. It allowed us to create sprites and paint them with whatever textures we like. That way some walls are stony or marked with paintings. The changes in physical material were mirrored on the acoustic map to deliver correct settings for the audio scene. We have ensured that there are many acoustically different environments in the game. There are wooden and stone floors, there is sandy beach and fluffy carpet. Such configurations enable more gameplay possibilities while giving the player unique experience in every room.

3.4.3. Objects

Majority of the objects found in-game are using Physics [18] which lets the user assign weight through Rigid Body and treat collisions thanks to applied mesh colliders. Furthermore Physics adds support of major gravity laws.

3.4.3.1. Physics

Physics is a 3D engine integrated into Unity to handle motion, collisions and gravity phenomena. There are two basic components responsible for physics simulation. The first is RigidBody and the second is Collider.

Rigidbody describes physical properties of an object such as Mass, Drag and Angular Drag, which are then used in convenient computations. Moreover, it can be further specified if the object will be susceptible to world’s gravity or if its movement should be restricted to some axes.

Collider determines which parts of the 3D object visually present in the game scene should collide with the others. Usually it is preferred for the entire object to collide with the environment, however on some occasions this does not apply i.e. hollow objects. To achieve custom collider shape, independent of actual object’s look and size, Unity provides a variety of colliders ranging from simple box, through cylinders up to mesh colliders able to take most complex shapes, often adapting to the real in-game objects.

Physics engine carries out calculations for every game item present in the scene taking into account its current velocity and torque. It measures all the forces applied to rigid bodies and serves hundreds of collisions taking place every second.
Exploiting Physics engine let us imitate real motion, gravity and collision principles helping enrich the gameplay. Thanks to the engine we are able to throw and operate with objects, open doors, fall down, jump, and walk properly. Even sounds created by collisions could not have been generated without Physics since it is capable of providing RAYAV with collision parameters like velocity, mass and force. This is also crucial for interaction between player and the enemy, as the player is encouraged to use all objects equipped with Rigidbody in an attempt to fool the enemy with sounds emitted by throwing them.

3.4.3.2. Interactive objects

Player is able to interact with most of the objects located in the game. There are three kinds of those: pickables, switches and doors. Having a script assigned player is able to pick up an object tagged as pickable and throw it where the power of the throw depends both on player’s strength and given object’s weight.

On the other hand, the hidden switches are used to unlock doors placed all over the map. When clicked the switch governed by script removes the freeze applied originally to the doors’ Rigidbody
position. In the process a sound is played to inform player the switch was activated.

![Figure 3.6: Player opening the door.](image)

Doors are another type of clickable objects equipped with Rigid Body and pickup tag to grab the handle. They utilize special Physics components known as joints. Hinge joint allows to join two RigidBody objects imitating door hinges. Such joint can have many parameters defined to ensure smooth rotation of the doors and apply many corrective forces. They can be utilized to achieve additional features like self-closing or latches.

All of the interactive objects are sound sources. Some of them emit sounds when clicked, some during movement, others on collision or when the player is in proximity. The collision ones have the biggest share among all of the game sounds and are handled by the RAYAV engine as described in chapter 3.4.6 Sound.

### 3.4.4. Player

Our game is seen from First Person Perspective (FPP) so visible character model is truly non-existent. However there is a collider responsible for emulating hero’s body and being subject to Physics. There is an arm attached to the body used to hold the picked objects along with a flashlight on the right hand side. The hero grasps it to lighten up the grim location. To properly interact with objects and fulfill gravity laws the body has a weight assigned.

Player’s movement is governed by Player Movement script. As the player character is basically a rigid body, its motion is implemented by adding proper forces along global axes. The script takes full advantage of Physics engine navigating the player in all available directions with different velocities. At the same time it allows him to dynamically interact and collide with other rigid bodies found in the game world. Such approach is rarely used in the game design. Commonly, the player character is not a rigid body and only mimicks interactions via scripts. The Physics solution is more demanding in terms of development time and overcoming the encountered hardships, but it proved to be very effective in advanced functionalities like manipulating objects and interacting with the Enemy.

Main camera, which is actually player’s "head" is connected to the main body with a hinge joint. Such setup ensures more realistic motion of the player as a whole. There is the main body rotating around
vertical axis and the head tilting up and down. Since the Main Camera is also a subject to Rigidbody it takes advantage of forces. Script Force Torque manages camera’s torque allowing player to look around smoothly. Rigidbody applied to camera resolves multiple gameplay issues concerning interaction with objects. Like in real life the character only has certain amount of force in disposal. Lower mass objects can be manipulated with ease, while heavy crates may cause trouble. Beside Rigidbody, there is also audio listener attached to the Main Camera performing as "ears" for both RAYAV and Unity audio systems, ensuring proper reception of incoming signals.

What is more, the character bears human-like traits like breathe and heartbeat either with changeable tempo and loudness varying together with health pool and movement speed. Those features are described in Player Health script. The character starts the game with maximum amount of health, his goal being to escape imprisonment and not let the health level drop below zero. The pool can lower in frightening situations, especially while looking at the Enemy or getting hit. Health regenerates over time, however every second is precious while in danger of engaging the Enemy. The numerical amount of health is not visible to the player. Instead, he is able to see and hear other signs of his well-being. One of them is the constantly playing heartbeat sound. Its rate increases whenever the main character feels terrified and his health drops. Also player’s breathing sound changes along with it, going from regular one, through fast gasping up to coughing when he reaches his limits. Another indicator is player’s vision being more blurred and darkened while losing health points. Blood stains appear on-screen to warn the player that he should immediately find shelter and regenerate. Those mechanics help build up wanted tension and augment the reality. Similar solutions were introduced some years ago in Call of Duty game series by Activision and have since then become commonly used replacing casual health bars and medpacks.

Another matter of importance is the sound generated by the character. Besides the heartpounding and breathing he emits footstep sounds while moving. Their timbre depends on the floor material. For example, while walking on a carpet the sound is more muffled compared to stepping on a wooden or stony

Figure 3.7: The Enemy roaming the map. Player health pool drop indicated by blood-covered screen.
floor. This behaviour is crucial for player’s interaction with the Enemy, as he will always try to remain quiet. Character’s footsteps are processed by RAYAV resulting in real-like modelling of those sounds. It can be particularly noticeable in certain moments of the game, for example while moving through narrow pipes.

![Figure 3.8: Player moving through narrow pipes.](image)

There are many more features utilizing player object. Scary Sounds script manages appearance of uneasy sounds at some points to further develop the climate. It is also used to provide a simple tutorial for new players explaining the basics of gameplay and controls. The script is key at the end of the game directing its "Grande Finale" awaiting the player on the top floor.
3.4.5. Enemy

One of the main features of our game is an intelligent opponent known as the Enemy or the Blind Man. His inability to see comes from the initial concept of the game which puts most emphasis on its audio aspect. Instead, the Enemy sports powerful hearing sense used to detect his victim based on perceived noise. Such trait can backfire too, as the player is able to distract the antagonist and lure him around the location by causing clamor in distant places of the map. The model found in final version of the game is "Speedball Player" free asset from Unity Asset store. While it was not very "monstrous", we used the asset due to the lack of eyes which went well with the name "Blind Man". Later in the development, the model proved well-suited for the Enemy’s role matching the supernatural climate of the game.

The Enemy character model is animated to imitate walking through game scene in search of loud noises originators. Currently it is the only animation applied as he is always in the walking state. To make it happen generalized humanoid animation pack for Unity mecanim was used [19].

Despite being blind, the enemy boasts perfect hearing sense following every suspicious noise such as player’s footsteps and multiple sounds generated by the environment. Once a sound reaches his ears, he compares its level with the actual threat level. If it is greater, then the emitter becomes his next target. The threat level takes the value of arriving sound and the enemy proceeds towards new destination. Over time the threat decreases forcing player to make loud noises every few seconds to keep Enemy distracted. If he fails, he will become the target again.

Another feature of the Enemy is characteristic loud shriek he lets out having detected the target with his sense of hearing. Each focus change is announced with a scream. Player can exploit it to predict whether the current target is the player himself or the object he has just thrown. Learning to utilize this...
mechanic properly can give experienced players advantage over the opponent. The shriek sound itself is a subject to RAYAV technology as described in chapter 3.4.6 Sound.

The core of Enemy’s Artificial Intelligence (AI) is the usage of NavMesh to manage his walking routine. NavMesh is Unity-based technology which allows real-time computation of point-to-point paths with specified conditions in 3D environment. NavMesh map was prepared to define areas of movement by setting certain constraints like Enemy’s height and radius. The game objects were marked either as walkable or non-walkable surfaces. Having set parameters, the engine calculates shortest paths for the enemy to travel between two points in any given time.

![Figure 3.10: View of the NavMesh with walkable terrain marked in blue.](image)

The Enemy is equipped with NavMeshAgent component, which takes information from the scripts described above to acquire current target’s location. After that, NavMesh calculates shortest path to the destination letting the Enemy proceed avoiding obstacles. NavMesh technology prospers well in our game, allowing Enemy to effectively track and engage the player. The Enemy AI and movement scripts had been rigorously tested and after hundreds of playthroughs we are confident that in his current state, the Enemy will offer a challenge even to battle-hardened veteran players.
3.4. Technicalities

The player is discouraged from looking directly upon Enemy, as it will cause his health drop because of severe headache. Such mechanic enforces even more the use of sound signals whenever possible and treat visual contact as a last resort in dire situations. Once being targeted by the enemy, the player must run away to avoid being killed and ending the game prematurely. It is really tricky, because the faster the player runs, the louder his footsteps become and the easier it is to locate him. Only proper balancing between crawling and running combined with the use of distant sound sources will guarantee overall success.
3.4.6. Sound

There are two basic ways of treating audio in the game. The first relies on the built-in Unity audio system which utilizes multiple sound sources connected to objects in the scene, and the listener joined with the character. Objects have assigned component named audio source to be able to play anything. Some of them are scripted as well to emit under certain conditions for example having collided with specific surface. Audio sources hold many adjustable parameters allowing modification of audio samples to achieve preferred effects. The developer can manipulate its volume and pitch, mute it, loop it or even filter it in different ways. What is more, Unity offers means to imitate 3D sound propagation thanks to volume roll-off with unit distance options. With the tool, sound propagation can be largely modified not necessarily to achieve real-like outcome.

Despite all those options, recognition of popular audio formats and its ease of use, Unity Audio sources have one clear disadvantage. Sounds they generate are propagated in 3D environment, however they do not collide with any objects in the game scene making output of an Unity audio source hearable through a concrete wall the same way as in the open space. This is a major flaw when it comes down to stealth games, especially horror ones, where player is bound to exploit sounds to locate the enemies and gain additional information about surroundings.

RAYAV resolves problem mentioned above thanks to ray-tracing processing of the sound signals before they are perceived by the player. Its work principles resemble Unity Audio by utilizing sources and listeners. Within scripts the developer is able to preset positioning of sound source and listener in the 3D soundscape. During gameplay both are updated in real-time. When the game starts, RAYAV initializes itself using configuration files containing properties of sound propagation at the same time it loads the Acoustic Map used to track the audio on the go. After that the samples are buffered and all the sources and the listener are injected into the soundscape. As described in 2.2 Audio engine - RAYAV sounds propagate through the environment taking into account objects placed on the acoustic map - walls, monuments, large crates and pipes - as well as the materials they are made of. In the current version of the game over 15 materials of different acoustic properties were implemented including wood, stone, sand
and even grass. RAYAV engine proved its worth in the real-time application, handling efficiently multiple sources and providing the player with realistic experience. It enriched the gameplay with phenomena like reverb, Doppler effect or audio propagation through walls.

![Figure 3.13: Surfaces made of different materials: carpet, sand, stone.](image)

On the other hand, using RAYAV posed many challenges. Implementation of the engine in our game proved more difficult and time-consuming than initially thought. If not for the continuous support from AGH DSP team, we would not have been able to resolve encountered compatibility issues and after countless hours integrate RAYAV into our project. Another hardship was mastering the operation of RAYAV and all its functionalities through C-sharp scripting. Comprehension of the RAYAV functions and their proper usage along with all other game mechanics turned out a demanding, but also creative process.

During the development we realized that RAYAV was not the best solution for every in-game application. While it is perfect for sounds generated by colliding objects or other moving sources, like player’s footsteps and the enemy, it meets with serious problems having to deal with numerous sound sources. In principle, our game gives player a lot of freedom to interact with almost every existing object. The vast majority of them could emit sound while colliding with others. Because of that the number of sources initially in the game reached over three hundred.

It was almost impossible for RAYAV to operate with that many of them, as it was prepared to serve only few present simultaneously in the scene. After lengthy testing we came up with simple solution to the problem. A global array of sound sources available for colliding objects was created to playback their respective sounds. Depending on the array’s dimensions RAYAV operation is altered to match the current needs. Such solution allowed for more sound sources to be present at the same time resulting with augmented realism, although at the expense of performance. Utilizing excessive number of sources causes strange background noises and distortions. However, despite the drawbacks more consistent experience is achieved that way and the mismatched sound is audible only during multiple simultaneous collisions. We decided to take advantage of RAYAV capabilities while maintaining gameplay freedom and hundreds of sound sources in the game. Another major disadvantage of RAYAV is that it’s not able to adjust itself to the dynamic changes in the environment. So moving an object during the gameplay
won’t influence the scenes acoustic properties since all of them have to be hardcoded into acoustic map beforehand. This may cause discrepancies between what the player sees and what he hears.

Both enemy and character movement generated sounds are modelled using RAYAV. Thanks to such setup player can locate the Enemy only with the sense of hearing. As for character footsteps, their volume varies depending on current floor material. It delivers new gameplay possibilities for the player providing a way to avoid being caught and stay silent.

It is worth noting that traditional Unity-based audio sources are still implemented to serve certain applications. When ray-tracing technology is not necessarily required, they offer reliable and sufficient solution for audio playback. Many static sound sources like a piano, telephone booth, door joints and switches as well as character’s internal sounds - heartbeat and breathing are handled by Unity audio engine. In addition, all the music present in the game is realized by standard sources. We took advantage of it reducing the number of parallely working RAYAV sources, and in consequence increasing overall performance.

Most sounds heard in-game were prepared using samples provided by DSP AGH team [20]or found over the Internet. Some of them required a bit of editing done in popular program called Audacity [21]. It provides all the key functionalities needed to create and modify a recording.
3.4.7. Working process

Working in a team enforced us to utilize a tool which is crucial to cooperation on any project, keeping it synchronized and up-to-date. Such tool is called GIT repository and is responsible for carrying out so-called version control at the same time allowing to create parallel branches and more independent forks of the project. Usually it is sufficient for comfortable and simultaneous work although in case of Unity things tend to get more complicated. The program adds some temporary files to the project and changes its file structure with every opening and closing. In the end we had to work sequentially to remedy this issue. The project was synced using cloud server called github.com [22] which offers a free account to create repositories and a simple tool to keep track of any changes in files updating the online version on user demand. This approach smoothened the work and made the cooperation more feasible.

During game development we had to share workload within our team of two. We developed initial project plan with core responsibilities assigned, keeping in mind that many features could be subject to change. Karol was in charge of preparing maps used in the game as well as the Acoustic map with the use of Blender software. At that time he was also responsible for populating the maps with intractable objects like doors and switches. Maciej was handling level design, including preparation of room themes, importing appropriate assets and positioning them inside the scene. He was also creating new assets from scratch using Prototype tool. In the meantime, we tried to come up with a story to go well with intended game genre. Karol wrote most of the scripts implemented in the game to manage interactions between objects, physics, light, sound, character and camera movement, Enemy AI and many more. The scripting part was one of the most time consuming and it involved our strict cooperation and devoting many hours to parallel solution testing. Maciej stayed in charge of our communication with AGH DSP team and management of git repository.

![Figure 3.15: Exemplary view of Github version control tool.](image-url)
Audio part of the game was developed largely in cooperation. As Maciej was looking for the best samples suiting the game needs, Karol made sure that they were correctly implemented into the game. Karol was also using Audacity to ensure that all sounds had compatible formats and fitted proper timings. Together we handled the music which is playing during the playthrough and other various sound effects, in particular the collision sounds.

Implementing RAYAV engine into the game is worth mentioning as separate point as it was a very hard task. We have encountered multiple problems on our way to get RAYAV integrated correctly into Unity environment. It required spending tens of hours on debugging process together with AGH DSP team members. Having assured that all engine’s dynamics libraries were loaded by the system, and that the Acoustic map was correctly set-up and rotated by RAYAV, we were able to take advantage of its ray-tracing abilities in the game. Karol coded additional scripts containing RAYAV functions to allow in-game audio to be handled by the external engine itself. At those final stages of the project Maciej focused on describing it and preparing the documentation using Latex [23], "a high-quality typesetting system".
4. Summary

4.1. Conclusions

Summing up the project, we really enjoyed having to deal with something of our true liking. The amount of work required to make the game up and running was huge but the results seem more than satisfactory. Despite the initial hardships we managed to implement RAYAV engine successfully at the same time producing an interesting and absorbing gameplay. Time spent in Unity workspace was not wasted since as of now we can admit knowing its environment inside out. We are hoping that the obtained experience will allow us to further expand our project and other upcoming ideas. We have learned a lot about creating scripts in C-sharp and Javascript, as well as about using various tools practical in game development - meshes, physics, animations, light and sound processing and many more. As game development is a complex process, it requires wide knowledge from the designers, especially while working in a small team. Beside Unity, we are now capable of operating in other environments convenient during game designing like Blender (3D objects, Acoustic map), Audacity (sound processing/conversion and creation) and GitHub (project repository management).

What is most important is that our game is a fully working product. All initially planned objectives were fulfilled. The game has been described in our thesis as best as it could have been, but nothing compares to playing it yourself. Moreover, it has been intensively tested by many external users - both veteran players and complete novices to ensure gameplay entertaining for everyone. Having said that, anybody can download the game from our cloud repository at: https://github.com/mdoniec/Project-Stealth/tree/master/RayAV.

4.2. Future

The game could still use some tweaking and polishing concerning both technical and conceptual aspects. The graphics together with the audio could consist of more custom made assets like specially crafted 3D models or exclusively recorded samples. Also the animations present in the game need serious improvements, as animation is very time consuming process and issues concerning them were often overlooked during development. Gameplay-wise there is also place for improvement and expansion. The current map contains one more floor which hasn not been shown yet. Furthermore, multiple mechanics could be added like a true combat system making use of melee weapons or guns.
Another matter is the plot which is not as complex and nonlinear as it should be. Thanks to the illogical concept of the story it could be easily bolstered. Time will verify whether the project development will be continued. Further improvement in described areas of the game could result in a more polished product of higher class. The future of the project looks bright. RAYAV team is lively interested in showing our game as a demo of RAYAV engine during upcoming conventions. The first occasion to present capabilities of the engine as well as its implementation in a Unity-powered game could occur during Gamescom 2015 held in Cologne, Germany.
Bibliography

[1] AGH DSP - RAYAV
    http://www dsp.agh.edu.pl/pl:resources:rayav#.VLlRm0eG-ao [20.01.2015].

    *Game Sound. An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design.*

    *The Complete Guide to Game Audio For Composers, Musicians, Sound Designers, and Game Developers.*
    Oxford, Focal Press 2009

[4] Rogers S.
    *Level Up! The Guide to Great Video Game Design.*
    Chichester, John Wiley Sons 2010

[5] EAX
    http://www.creative.com/oem/technology/eax.asp [05.01.2015]

    http://en.wikipedia.org/wiki/A3D [05.01.2015]

[7] OpenAL
    http://www.openal.org/documentation/openal-1.1-specification.pdf [05.01.2015]

[8] AMD TrueAudio
    http://www.amd.com/pl-pl/innovations/software-technologies/trueaudio [05.01.2015]

[9] Oculus Rift
    https://www.oculus.com/ [05.01.2015]


Visibility determination in beam tracing with application to real-time sound simulation.

[12] Sz. Pałka, T. Pędzimąż, B. Ziółko

A system and a method for determining approximate set of visible objects in beam tracing.
Zgłoszenie patentowe 110731 w USA, Kanadzie, Japonii i EPO.

[13] Game Engines
http://www.moddb.com/engines [18.01.2015]

[14] Unity

[15] Unity Card Life - Hearthstone

[16] Blender Wiki

[17] Unity Asset Store

[18] Unity Physics

[19] Unity Animation


[21] Audacity Wiki
http://wiki.audacityteam.org/wiki/ [03.01.2014]

[22] GitHub Guides

[23] Tex Help Forums
http://tex.stackexchange.com/ [19.01.2014]
[24] Learn With Unity
   http://unity3d.com/learn [10.10.2014]

[25] Unity Script Reference
   http://docs.unity3d.com/ScriptReference/ [02.01.2014]

[26] Unity Forum
   http://forum.unity3d.com/ [19.01.2014]