A Tool to Implement Adaptive Audio Techniques on Unity Games

Miguel F. Rodrigues
DECOM - Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais
Belo Horizonte, Brazil
miguelfeliperod@gmail.com

Flávio R. S. Coutinho
DECOM - Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais
Belo Horizonte, Brazil
fegemo@cefetmg.com.br

***Abstract*—Developing an effective and simple audio system that can adapt to in-game events can take time and become complex, specially to beginner developers that don't know how to use adaptive audio techniques. To make this less challenging, we developed a tool to implement adaptive audio into Unity games. The Adaptive Audio Manager tool for Unity is an accessible package that provides classes and methods which help developers implement a functional audio system in their games, using two popular adaptive audio techniques: vertical layering and horizontal re-sequencing. Along with the framework code, we provide documentation and a Unity prototype project with examples to keep the learning process on how to use this framework simple.**

***Index Terms*—adaptive audio, framework, unity**

I. Introduction

Each year, the digital games industry grows more (1) whether with the participation of large companies or with independent and small developers. The billionaire industry of games surpassed even the music and movies business, in terms of annual revenue (2). Computer, console and mobile games together formed a large community of fans of electronic games, both casual and hardcore players, with different preferences when deciding the type of game they want to play. In a competitive market like this, such intense growth of the industry stimulated the development of techniques and technologies that help to improve the quality of games in order to attract players by providing the best experience as possible. Game developers apply such techniques to different aspects of a game, such as narrative, mechanics, graphics and audio (3).

In the last years, the importance of the soundtrack in the immersion of players has become clear, which led to the evolution of techniques that enhance the soundtrack's complexity and its influence to the player's experience. For example, audio can be used to incite different emotions and sensations to players (4). Pong, released in 1972 by Atari, is recognized as the first game that included audio in its working formula (5), being a 2D game with a square to represent the projectile and two vertical bars to represent the players, which bounce the ball in opposing directions, simulating a Ping Pong game. Sound effects are played whenever one of the bars touches the projectile. In the years following Pong's release, many games were released and the audio elements grew popular and improved. Soundtracks conquered a fundamental role in digital games so, nowadays there are games with mechanics focused on the rhythm of the songs (rhythm games) and even full albums and orchestras dedicated to game soundtracks (6).

One of the techniques which emerged with the game audio evolution is the use of dynamic audio. In the movie industry there's always the same sequence of events that occur in the same order and frequency, which makes the accompanying audio always the same (static audio). However, thanks to the interactive nature of games, players may take different actions leading to different situations, which can incur in some variation of the game music and sound effects (dynamic audio). When the game soundtrack reacts to player's interactions, we call such technique adaptive audio (7).

In this study, we developed a tool to aid developers to implement adaptive audio techniques in their games, focusing on Unity engine games due to the possibility of being a free platform and one of the most popular engines for game development (8), having been used in the production of famous games. Named Adaptive Audio Manager for Unity (AAMU), the tool is an accessible unity package which provides classes and methods that helps implementing two popular adaptive audio techniques: vertical layering and horizontal re-sequencing. Near the end of its development we conducted a preliminary evaluation regarding its ease of use with 7 programmers with different levels of expertise with Unity, and the collected feedback identified some pain points and also showed promising benefits from using the tool in games. We made AAMU available[1] as a Unity package and expect it to encourage developers to implement adaptive audio techniques in their games.

II. Adaptive Music Techniques

There are different approaches to develop a game with adaptive audio experience. Each can have different results for different game environments, depending on the proposal of the developer and his/her goal.

A. Vertical Layering

The vertical layering technique is the segmentation of an audio in individual layers, each composed by one or more

[1]https://drive.google.com/drive/folders/1-o8wCZIs1hGG69EHqosJCdjf5DU5Zei-?usp=sharing

instruments/sound effects. When playing a song using this approach, the user can activate or deactivate any layer, introducing or removing the desirable sounds according to in-game events.

For example, this technique can be used in a situation where the player is on a peaceful state, along with a calm music, composed by a soft piano and ambience sound. The moment some enemies approach or are on sight, the game state changes from peaceful to action/danger, adding some electric guitar, bass and percussion layers. Those added instruments can give the player a sense of danger, providing him the information that something changed (4).

B. Horizontal Re-sequencing

Horizontal re-sequencing is a technique used to reproduce queued audio tracks, like a playlist. Each track can have a different duration, long tracks or short fragments, as long as they can musically connect with each other (9). Each track fragment is played according to the current state of the game. As an example, consider a game character that is moving away from a safe area, like a town, and entering a dangerous area, like a forest or a cave. Using the horizontal re-sequencing technique, the audio can transition along with the character, from a calm track to a more tense track without an abrupt interruption, as the start of a new segment can be delayed until the current one is in a state which can harmonically transition into the new one.

That said, a transition between tracks can happen in many ways, some of them are: fade-in (gradually incrementing the volume of the track being inserted), fade-out (gradually decrementing the volume of the exiting track) and cross-fade (simultaneous fade-in and fade-out). Those fades have straightforward implementation, but they might cut a track in the middle of the execution, before the music piece has ended, what can sound off. (10).

Bridges are another option of transition between tracks. They consist of intermediate tracks that connect two different ones to avoid discontinuation, making the transition sound more natural.

C. Organization and Segments Variation

Organization and segments variation focus on avoiding the excessive repetition of audios. In many games, some sounds occur repeatedly, which can cause ear fatigue to the player. To avoid reproducing the same audio in quick succession, it's possible to create variations of the same audio by changing its volume, tempo, and frequency or providing enough audio variations to randomize their reproduction. As an illustration, shooter games frequently use this technique for the sounds of firing guns and footsteps.

III. Related Works

There are various projects and researches regarding adaptive music techniques and their effects on video game player's behavior. For example, Alves, Silva, and Araújo (11) researched the development of audio for games using the middleware FMOD. To synchronize in-game events, the authors create Unity variables and link them to FMOD's audio parameters.

For example, in a car racing game, the faster the car gets, the louder the engine noise. To represent that, FMOD can increase the sound level and the pitch variation to mime the engine's sound. Middlewares can help avoid sound repetition, too, creating a wider variety of sounds in real-time, without the need to repeat a preset.

Similarly, Gungormusler, Paterson-Paulberg, and Haahr (12) evaluated the impacts that music causes on player's feelings. Using Unity, they developed an adaptive audio system that can create different melodies based on two very high-level inputs: energy and stress. The user can tweak such inputs through sliders, and the combination of values results in a different mood represented by a procedurally generated song. Optionally, the user can choose one of the presets combinations of inputs, being them: exciting, happy, tender, neutral, depressed, sad and angry.

Grumble Labs (13) created the Adaptive Music Player for Unity, a package that helps developers implement the horizontal re-sequencing audio technique along with sample audio tracks and an illustrative implementation. The Adaptive Music Player offers an interface and some methods to manage the execution of the tracks. There are two main structures: layers and songs. Songs comprise at least one layer, and layers are the audio track variations of the song that can play during different game states.

The software provides some methods, including play, pause, stop layer or song, fade-in, fade-out, and even cross-fade between layers. However, the limitation of that tool is that only one layer can be played simultaneously, not allowing the developers to control which instrument they want to be played at each game state.

IV. AAMU Development

The Adaptive Audio Manager for Unity (AAMU) is a tool to help implement the vertical layering and horizontal re-sequencing audio techniques on games developed using the Unity engine. It provides classes, methods, and a user interface that assist developers in implementing such adaptive audio strategies. Within the same package, there are the framework, documentation, and a demonstrative prototype.

A. Vertical Layering Development

Firstly, to understand the vertical layering technique, some games that use this audio technique were analyzed. The first one was Nier: Automata (14), winner of The Game Awards (2017) in the Best Score/Music category. The game has a big map with different environments and enemies. When a player discovers a new area, he hears a simple version of that place's music theme. As he advances the game story, some new instruments and voice tracks are added, evoking a feeling of progression and discovery of the world that is being explored. Additionally, some other player actions also trigger the addition or removal of tracks, such as the presence or absence of enemies nearby.

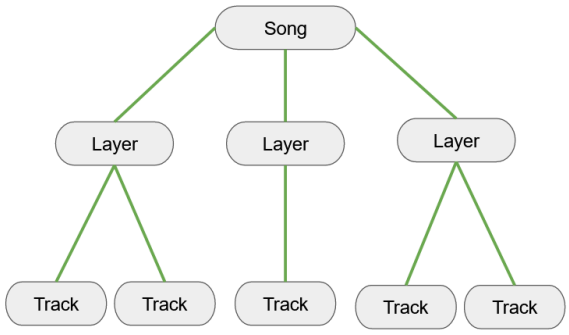

Fig. 1. Basic structure of the vertical layering hierarchy technique

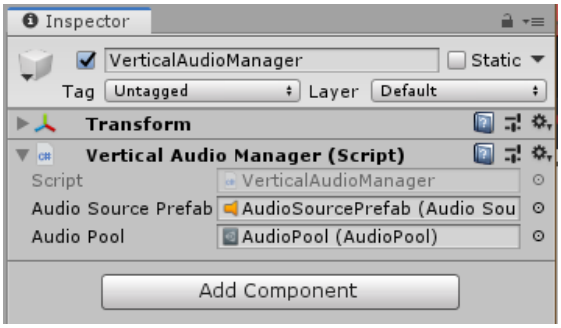

Fig. 2. `VerticalAudioManager` structure on Unity's inspector example

Portal 2 (15) is a puzzle-like game that uses the vertical layering technique too. The game shows many different interactive elements, such as lasers and cubes that are part of the puzzle solutions. When a player interacts or even approaches such objects, a new audio track is added, playing along with the current environment music. This technique can help the player detect essential game elements from elements that he can't interact with.

Crypt of the NecroDancer (16) is a rhythm game where the player's goal is to explore and descend levels inside a procedurally generated dungeon. Each level has different enemies, non-player characters (NPCs), chests, and items, letting the player move only on the right moments according to that level music rhythm. Among the map elements, there is a merchant NPC that starts to sing that level music as approached by the player. From the vertical layering perspective, what is happening is that when the player is sufficiently close to the merchant, a new audio track with the NPC's voice is added to the level music, communicating the player's proximity to the NPC.

Lastly, The Legend of Zelda: Skyward Sword (17) is an action and adventure game that uses the vertical layering technique in a place called Bazaar, which is a market with some merchants that can interact with Link, the main character controlled by the player. The Bazaar has a music theme that is played while the player is there. Whenever the player approaches a merchant, the audio tracks of the Bazaar theme fade out, while the specific merchant's theme tracks fade in. Each merchant has a melody similar to the Bazaar theme but using different instruments and somewhat resembling the NPC's mood, attitude, or profession. When a fade-in and a fade-out co-occur, we call that a cross-fade. From the vertical layering perspective, the music being played is always the same. What is really changing is the activation or deactivation of audio tracks to compose that music according to the player position and interaction.

Before implementing the vertical layering technique, we gathered the basic requirements based on observations of the previously mentioned games:

1) control reproduction (play, stop, pause, resume) of single audio tracks;
2) control fade transitions (fade-in, fade-out, cross-fade) of audio tracks;
3) control individual track volume;
4) control reproduction (play, stop, pause, resume) of multiple audio tracks at the same time.

As shown in Fig. 1, we defined three basic structures that represent the audio tracks that will be reproduced when using the vertical layering technique considering our previous observations and our requirements:

- **track:** is the structure that represents a single instrument/sound, being the most granular audio structure that the user can individually control. Each track needs a name and an `AudioClip` (Unity component);
- **layer:** is the structure representing a set of tracks that can be controlled simultaneously. It belongs to the same song but does not necessarily comprise all of its instruments. For each layer, the user has to define a name and a list of tracks;
- **song:** is a structure that represents a full song, including all instruments, sounds, and sets of instruments. For each song, the user has to define a name and a list of layers;

To have control over those structures, there is a singleton (18) class called `VerticalAudioManager` (Fig. 2) that holds the methods and resources needed to implement vertical layering. The `VerticalAudioManager` requires two components as parameters: a Unity component called `AudioSource` and an `AudioPool`. The `AudioPool` (Fig. 3) is responsible to keep a reference and name to each song, layer and track. It's possible to define multiple `AudioPool` for different moments in the game.

The `VerticalAudioManager` class has 27 methods that can be classified into three categories: basic, transition, and support. Basic methods are responsible for playing, stopping, pausing, and resuming individual tracks, entire layers, or entire songs. `Stop` and `Pause` methods take no additional parameters. The `Play` method can take some parameters that define the start volume, whether or not the track, layer, or song should loop, whether or not other tracks should stop playing, and the start time of the track, layer, or song. Lastly, `Resume` can take the parameter that defines whether the track, layer, or song will loop or not.

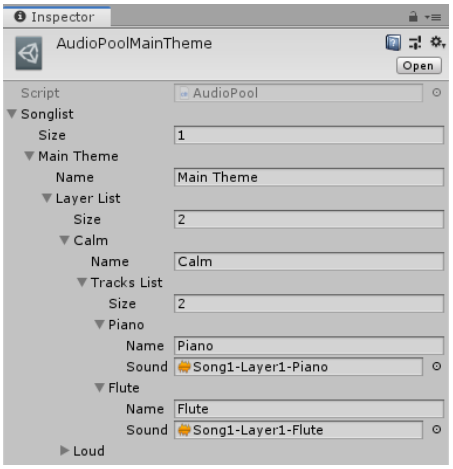

Fig. 3. `AudioPool` structure on Unity's inspector example

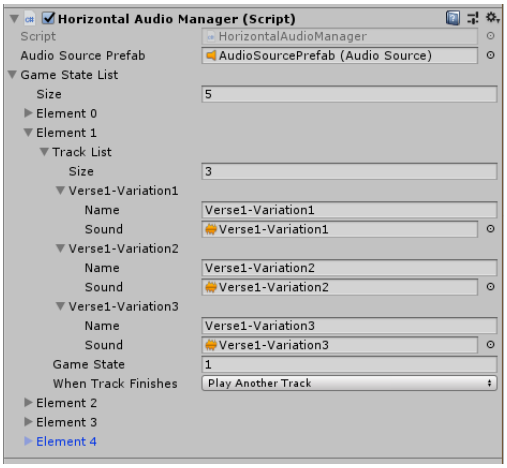

Fig. 4. `HorizontalAudioManager` structure on Unity's inspector example

Transition methods are responsible to smoothly start or stop reproducing tracks, layers, and songs using Fade techniques, and the `VerticalAudioManager` has three methods to apply those techniques. The `FadeIn` method can smoothly introduce a track, layer, or song progressively, from volume zero to the target volume provided by the user. The user can also provide parameters that will define whether or not the track, layer, or song should loop, a fade duration in seconds, the start time of the `FadeIn`, and a Unity `AnimationCurve` defining the interpolation function for the volume to go from zero to the target value. Likewise, a `FadeOut` method smoothly stops a track, layer, or song by progressively decreasing the volume to zero. The `FadeOut` method can also receive as parameters the fade duration in seconds and the `AnimationCurve`. Lastly, the `CrossFade` method lets the user fade-in and fade-out simultaneously two tracks, layers, or songs. Also, the user can choose different `AnimationCurve` for each fade, define if the track, layer, or song will loop, the fade duration, the time, and the final volume for the fade-in.

Support methods implement utilities that should set parameters and aid in the use of the other `VerticalAudioManager` methods. For example, three search methods receive a name parameter from the user and return the related track, layer, or song from the `AudioPool`. Also, two methods can be applied to each track, layer, and song, one of them can set the current volume, and the other get the current time of the corresponding track, layer, or song.

B. Horizontal Re-sequencing Development

Similar to how we elicited requirements to implement vertical layering, we analyzed some games that use horizontal re-sequencing. First, in Assassin's Creed II (19), the player controls a character that needs to accomplish assassination missions that involve exploration, combat, and escape. Each of those moments represents a game state followed by a specific soundtrack to convey the player's adequate intensity feeling. While in the exploration state, the player is not being harassed by other characters, which results in a calm soundtrack. When detected by an enemy, the game state switches from exploration to combat, changing the soundtrack to a more intense one. Lastly, by fleeing and hiding from the enemies, the game eventually returns to the initial exploration state. Those state transitions happen between fades that result in different segments of the soundtrack that connect with each other according to the game states.

In Spider-Man (20), the player controls the superhero that explores the streets of New York, trying to stop crimes from bandits and villains with superpowers. The main character can travel through the city using his spider webs to move among the buildings. That way, the game's audio system uses a set of variables like the main character's speed, altitude, time swinging on the web, among others, to determine which soundtrack should be played. The game state varies in three intensity levels according to the mentioned parameters. When the main character's speed drops to zero, the game state changes to the passive state, and the intense music played before is concluded. In this example, we can notice the different audio tracks that replace each other in sequence according to the game state.

Furi (21) is an action and combat game in which a prisoner equipped with a sword and a gun defies diverse bosses in sequence, trying to reclaim his freedom. The game state varies according to the boss's health bar and the combat style used between the main character and the enemy: long-range or melee combat. Each time the boss loses a health bar, the combat style changes, and the soundtrack alters accordingly. If the player loses all health, the game reverts to the previous state along with the music. This technique allows the transition between states to happen in determined segments so that it doesn't interrupt a segment of the song before it ends.

We identified two requirements for the horizontal re-sequencing technique based on observations of the previously mentioned games, which we list next:

1) control the current state of the game;

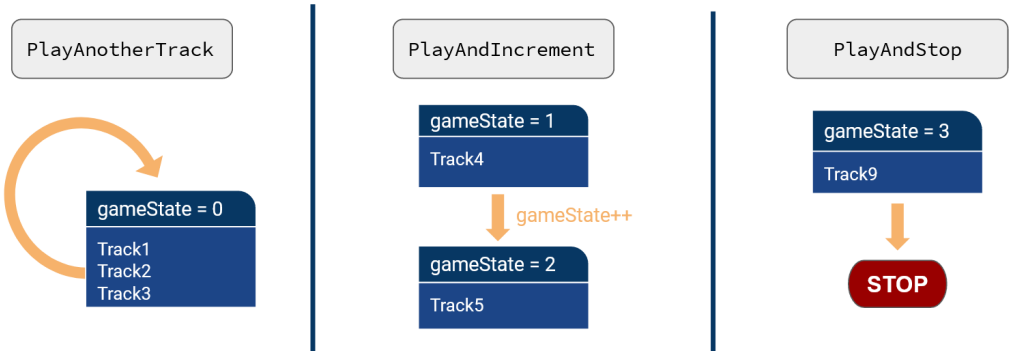

Fig. 5. Possible behaviors of the `whenTrackFinishes` parameter

2) randomize audio tracks according to the current state of the game.

We defined an architecture based on states to depict a soundtrack that can be responsive to the game's current state. To manage the states of the game, we created a `HorizontalAudioManager` (Fig. 4) structure that keeps the current state of the game according to in-game events. It determines the state of the soundtrack as well as the transitions that should play between each state. `HorizontalAudioManager` is also responsible for listing all possibles game states.

Each game state has a list of tracks (the same structure from vertical layering), a variable to determine the action to be performed when that track finishes, and a unique value representing that state. When there are multiple tracks in one state, the tool randomly picks one, creating the possibility to generate different compositions of the same music, avoiding ear fatigue. Plus, the `whenTrackFinishes` enumerator, shown on Fig. 5, determines what happens when that track finishes. There are three possible values for that parameter:

1) `PlayAnotherTrack`: randomly chooses a track from the current game state to play;
2) `PlayAndIncrement`: randomly chooses a track from the current game state to play and adds 1 to the current game state value;
3) `PlayAndStop`: randomly chooses a track from the current game state to play and suspends its reproduction in the end.

The `HorizontalAudioManager` has a method that starts playing the game soundtrack according to the game state. Finally, there is a method that forces the audio reproduction of the `HorizontalAudioManager` to stop.

V. Evaluation

We conducted a preliminary evaluation by gathering feedback about using AAMU from 7 developers. At the time of evaluation, the participants were between 22 and 31 years old, two of which were computer engineering students, three software developers, one game engineer and a game development professor. All of them had some experience with Unity. They were given four options to classify their own game development skills experience as: very experienced, experienced, not much experienced or no experience at all. Three participants classified themselves as not much experienced, three of them as experienced, one as very experienced and none of them claimed to have no experience.

Each user received a detailed activity guide with two activity descriptions, one for each of the techniques implemented by the tool, the user's manual, and a Unity project with seminal code and assets for the proposed activities. After accomplishing the tasks, we asked each user to answer a short survey regarding their difficulties and thoughts on using the tool. Each activity had its guidelines and an individual Unity Scene with the audio and visual assets necessary to implement the solution. A pilot test was conducted before the actual experiment to validate the evaluation methodology.

A. Proposal

On the first activity, corresponding to the vertical layering technique, we gave users a Unity scene with 2D assets, like in Fig. 6. The setting is divided into five areas arranged from left to right, and a character initially positioned on the far left of the scene (first area). The player could move in four directions using the keyboard arrows. Given that, we gave five tasks to the users (represented by Fig. 7):

1) T1-1: when inside the first area (Camp), only the guitar track should play;
2) T1-2: when inside the second area (Desert), only the guitar, bass and percussion tracks should play;
3) T1-3: when inside the third area (Rain), only the bass, flute, guitar and percussion tracks should play. Between the second and third areas, the new tracks had to fade-in and the removed had to fade-out;
4) T1-4: when inside the fourth area (Mud), only the guitar base, guitar solo and drums tracks should play. The transition between the third and fourth areas had to occur through a cross-fade or mixing a fade-in and a fade-out;
5) T1-5: when inside the fifth area (Red Area), all music should instantly stop.

On the second activity, corresponding to horizontal re-sequencing, users received a Unity scene with a background

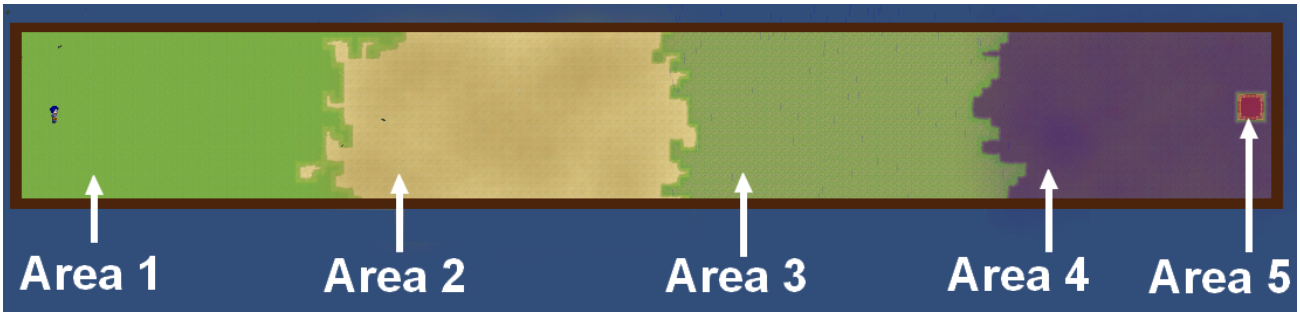

Fig. 6. Overview of the scene for the first activity of the evaluation

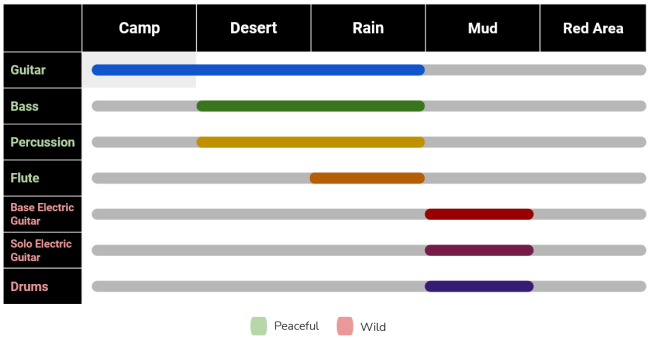

Fig. 7. Vertical layering example

image and a green bar representing the enemy health points which starts complete (with a hundred points), shown in Fig. 8. The player can reduce the enemy's health by pressing the space bar until the minimum level of zero. Given this scenery, we asked five tasks from the users (represented by Fig. 9):

1) T2-1: at the beginning of the scene, start with the game state 0 (zero), playing the track "Intro" just once;
2) T2-2: when the track "Intro" finishes playing, play the tracks "Verse1-Variation1", "Verse1-Variation2" and "Verse1-Variation3" repeatedly and randomly (one at a time);
3) T2-3: when the enemy's life drop below 50%, wait for the end of the current track and then play the track "Bridge" just once;
4) T2-4: when the track "Bridge" finishes playing, play the tracks "Verse2-Variation1", "Verse2-Variation2" e "Verse2-Variation3" repeatedly and randomly (one at a time);
5) T2-5: when the enemy's life reaches zero, play the track "Ending" then stop playing any track.

B. Users' activities conclusions

As shown in Fig. 10, all seven participants accomplished all tasks with no or only minor errors, summing 56 (80.0%) full correct tasks and 14 (20.0%) tasks completed but with mistakes. When testing the implementation results for activity one, we observed two common mistakes:

1) two users did not use the time parameter to introduce a new track or layer while another track or layer was still playing, as requested on T1-1 T1-2, T1-3 and T1-4, and just played all the tracks from the beginning;
2) two other users did not play the proposed tracks while moving from area 5 to area 4, as stated by T1-4;

When testing the implementation results for activity two, we observed only two mistakes:

1) two users played the "Ending" track at the right time, but twice, when the T2-5 asked to reproduce it just once;
2) one user delayed the reproduction of both "Bridge" and "Ending" tracks, with the delay of one execution (another track played before the asked track);

C. User's feedback

After solving the proposed tasks, the users answered a survey about their difficulties and impressions regarding AAMU. On the first three questions, we surveyed the participants' perceived difficulty to use the tool to include vertical layering (activity 1), horizontal re-sequencing (activity 2), and to read the user manual. They could choose a number from a 1 to 4 scale, with 1 meaning very easy and 4 meaning very hard, along with an optional input field which allowed additional comments. The participants' perceived difficulty is shown in Fig. 11.

Most (five) users classified the first activity as easy and two classified it as very easy. On the comments field, two users stated that the concepts of track, layer and song were not clear to them, which made the tasks more difficult.

On the second activity, the majority of users (five) classified it as very easy, one as easy and one as hard. The user that stated this activity was hard added a comment saying that "Although seeming a simpler system than the vertical layering, the horizontal re-sequencing did not seem to work intuitively", while other three users stated that the system was "intuitive and easy to use". Also, one user stated that the `gameStateEndAction` (older name for `whenTrackFinishes`) name was not intuitive.

As for the user manual, participants classified it was either easy or very easy to find the relevant information. On the comments section, four users stated that the documentation was clear and simple while one of them asked for more details regarding the horizontal re-sequencing implementation.

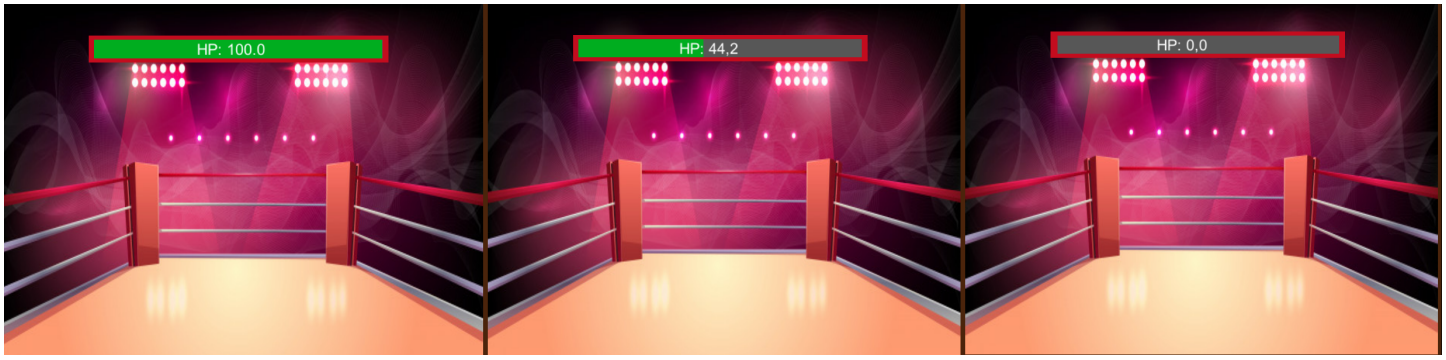

Fig. 8. Overview of the second activity of the evaluation

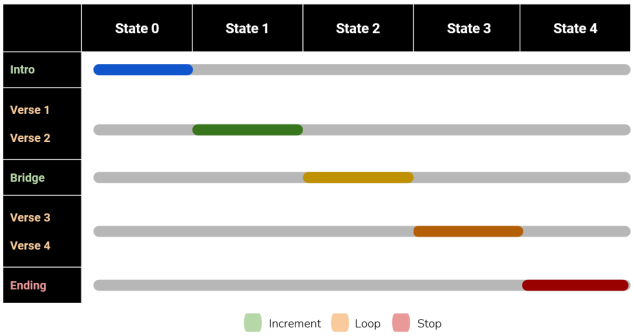

Fig. 9. Horizontal re-sequencing example

After that, users where questioned if they considered that AAMU could help developing adaptive audio techniques in games, and all of them agreed. There was also an input field for comments. Six participants mentioned that the tool made the implementation of adaptive audio technique easier or more simple than it would have been if they had to implement it from scratch. Two of them highlighted the time gain using the tool and other two recognized the importance of implementing these techniques in games.

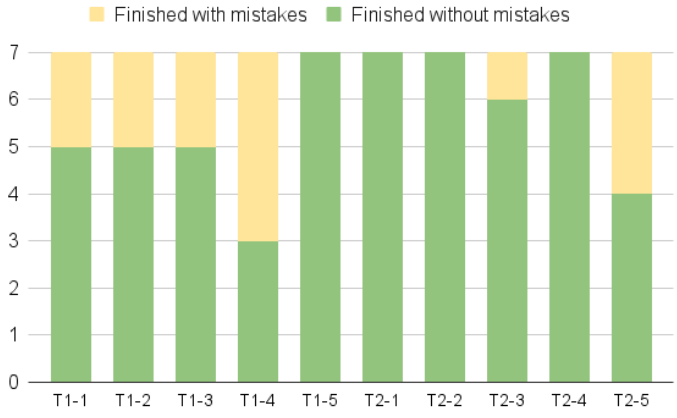

Fig. 10. User's activity conclusion result

When questioned whether they would use AAMU in a personal project, five participants replied affirmatively, and two stated maybe. No one fully denied it. Finally, users where given space to comment, criticize and suggest changes. One participant stated that the manual could have more images or examples of use, especially dealing with the `HorizontalAudioManager` use. Two participants suggested taking advantage of the Unity editor and creating Unity Prefabs (preset structures that work like templates in Unity) that use the AAMU methods to make the tool be more visual and easily used even by non-programmers. Lastly, users also suggested new methods for both techniques:

1) Pause with Fade-Out (vertical layering);
2) Resume with Fade-In (vertical layering);
3) New `whenTrackFinishes` option to subtract 1 of the current state (horizontal re-sequencing);
4) Give the option to play a track of a given game state without waiting another track to finish or executing a cross-fade (horizontal re-sequencing).

VI. Conclusion

This paper presented the development of AAMU, a tool to help game developers to implement adaptive audio techniques on their games using the Unity engine. We assessed different adaptive audio techniques, academic works, and games to gather requirements that guided the development of a framework that implemented two popular audio techniques, vertical layering, and horizontal re-sequencing, along with prototypes and a user manual.

We conducted a preliminary evaluation of the tool through potential users' perspectives with the participation of seven developers to assess both the tool and its documentation in search for errors and opportunities for improvement and to collect feedback. After analyzing the results of the evaluation and the participant's feedback, we improved AAMU's manual and code resulting in a framework suitable for use and ready to be made available[1] to the community.

[1]https://drive.google.com/drive/folders/1-o8wCZIs1hGG69EHqosJCdjf5DU5Zei-?usp=sharing

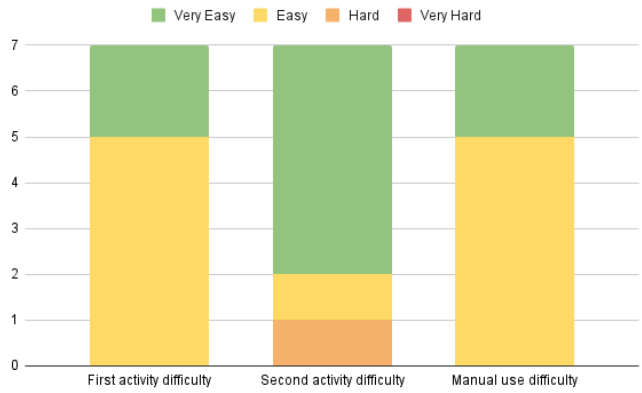

Fig. 11. User's activity and manual difficulty evaluation

With the final version of AAMU, even less experienced developers can straightforwardly implement two popular adaptive audio techniques into games made on Unity: vertical layering and horizontal re-sequencing techniques. The manual, methods and classes provided by the package allow users to avoid the work of implementing a full adaptive audio system from scratch also while avoiding to learn how to use an external tool, like a middleware.

The following steps in this research include creating new methods and classes that increase dynamic audio implementation options, whether by improving techniques already implemented or enabling the use of new techniques. Also, there are possibilities for performance evaluation and code optimization to improve the efficiency of the tool and further explore the use of Unity's Custom Editor by developing graphical interfaces that could make the AAMU simpler, friendlier to non-programmer users, and more dynamic.

REFERENCES

[1] T. Wijman. (2018) Mobile revenues account for more than 50% of the global games market as it reaches $137.9 billion in 2018. Acessado em 02 de dezembro de 2020. [Online]. Available: https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/

[2] J. Abbade. (2019) Indústria dos videogames bate recordes e fatura us 134 bilhões. Acessado em 02 de dezembro de 2020. [Online]. Available: https://jovemnerd.com.br/nerdbunker/industria-dos-videogames-bate-recordes-nos-eua-e-fatura-us-43-bilhoes/

[3] J. Novak, *Game development essentials: an introduction*. Cengage Learning, 2011.

[4] F. Coutinho, R. O. Prates, and L. Chaimowicz, "An analysis of information conveyed through audio in an fps game and its impact on deaf players experience," in *2011 Brazilian Symposium on Games and Digital Entertainment*, Nov 2011, pp. 53–62.

[5] G. McDonald. (2005) A history of video game music. Acessado em 02 de dezembro de 2020. [Online]. Available: https://www.gamespot.com/articles/a-history-of-video-game-music/1100-6092391/

[6] M. Fritsch, "History of video game music," in *Music and Game*. Springer, 2013, pp. 11–40.

[7] K. Collins, "An introduction to the participatory and non-linear aspects of video games audio," *Essays on sound and vision*, pp. 263–298, 2007.

[8] Unity. (2016) Public relations. Acessado em 02 de dezembro de 2020. [Online]. Available: https://unity3d.com/public-relations

[9] L. MENEGUETTE, "Áudio dinâmico para games: conceitos fundamentais e procedimentos de composição adaptativa," *Simpósio Brasileiro de Games, UNEB*, 2011.

[10] L. Kähärä, "Producing adaptive music for non-linear media," 2018.

[11] L. H. M. Alves, J. M. S. Junior, and C. S. de Araújo, "Desenvolvimento de áudio para jogos com unity e fmod," *Simpósio Brasileiro de Games, Curitiba*, 2017.

[12] A. Gungormusler, N. Paterson-Paulberg, and M. Haahr, "barelymusician: An adaptive music engine for video games," in *Audio engineering society conference: 56th international conference: audio for games*. Audio Engineering Society, 2015.

[13] GRUMBLE LABS. (2016) Adaptive music player. Acessado em 02 de dezembro de 2020. [Online]. Available: https://assetstore.unity.com/packages/tools/audio/adaptive-music-player-52023

[14] PLATINUM GAMES. (2017) Acessado em 02 de dezembro de 2020. [Online]. Available: https://nierautomata.square-enix-games.com/en-us/age-gate/

[15] VALVE CORPORATION. (2011) Acessado em 02 de dezembro de 2020. [Online]. Available: https://www.thinkwithportals.com/

[16] BRACE YOURSELF GAMES. (2015) Acessado em 02 de dezembro de 2020. [Online]. Available: https://braceyourselfgames.com/crypt-of-the-necrodancer/

[17] NINTENDO. (2011) Acessado em 02 de dezembro de 2020. [Online]. Available: https://www.nintendo.com/games/detail/the-legend-of-zelda-skyward-sword-wii-u/

[18] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[19] UBISOFT QUEBEC. (2009) Acessado em 02 de dezembro de 2020. [Online]. Available: https://www.ubisoft.com/pt-br/game/assassins-creed-2/

[20] INSOMNIAC GAMES. (2018) Acessado em 02 de dezembro de 2020. [Online]. Available: https://insomniac.games/game/spider-man-ps4/

[21] THE GAME BAKERS. (2016) Acessado em 02 de dezembro de 2020. [Online]. Available: https://www.thegamebakers.com/press/sheet.php?p=furi