IMPLEMENTATIE VAN GAME AUDIO IN UNITY

ZONDER MIDDLEWARE KAN HET OOK!

Danae Dekker
Muziek en Technologie
Hogeschool voor de Kunsten Utrecht

INTRODUCTIE

Er wordt wel eens gezegd dat een game zonder audio geen game genoemd mag worden. Audio speelt een belangrijke rol in het manipuleren van de emoties van een speler en kan feedback geven wanneer een bepaalde actie heeft plaatsgevonden of wanneer een actie plaats gaat vinden (Hirst, 2008). De eerste game met geluid was Pong uit 1972 en sindsdien heeft geluid in games een heuse ontwikkeling meegemaakt. Het is geëvolueerd van simpele piepjes naar simpele synthesizer-geluiden uit het 8-bit-tijdperk en vervolgens naar de epische soundtracks van tegenwoordig die niet misplaatst zouden zijn in een blockbuster-film. Er is zelfs een heel scala aan muziekgames ontstaan uit gamemuziek met bekende titels als Guitar Hero en SingStar.

Audio heeft dus een heel nieuwe dimensie toegevoegd aan games. In de kleine 50 jaar dat game audio bestaat, heeft het ook nieuwe functies in het bedrijfsleven ontwikkeld, waaronder de (technische) game audio designer. Deze specialist ontwikkelt over het algemeen alleen de geluiden en muziek voor een game, maar zeker in kleinere gamebedrijven komt het voor dat hij ook de technische implementatie in de game op zich neemt. Dit kan vaak door het gebruiken van audio middleware, zoals FMOD of Wwise. Dit zijn tools die het op een intuïtieve manier mogelijk maken om een audiosysteem te ontwikkelen en hebben veel weg van een digital audio workstation.

Het komt ook voor dat het niet mogelijk of wenselijk is om middleware te gebruiken, bijvoorbeeld wanneer een licentie voor middleware te duur is of de game te complex is om middleware te gebruiken, zoals bij multiplayer VR-games. De audio designer moet dan direct aan de broncode van de game gaan sleutelen. Dit artikel focust op manieren om audio te integreren in de game engine Unity zonder gebruik te maken van middleware.

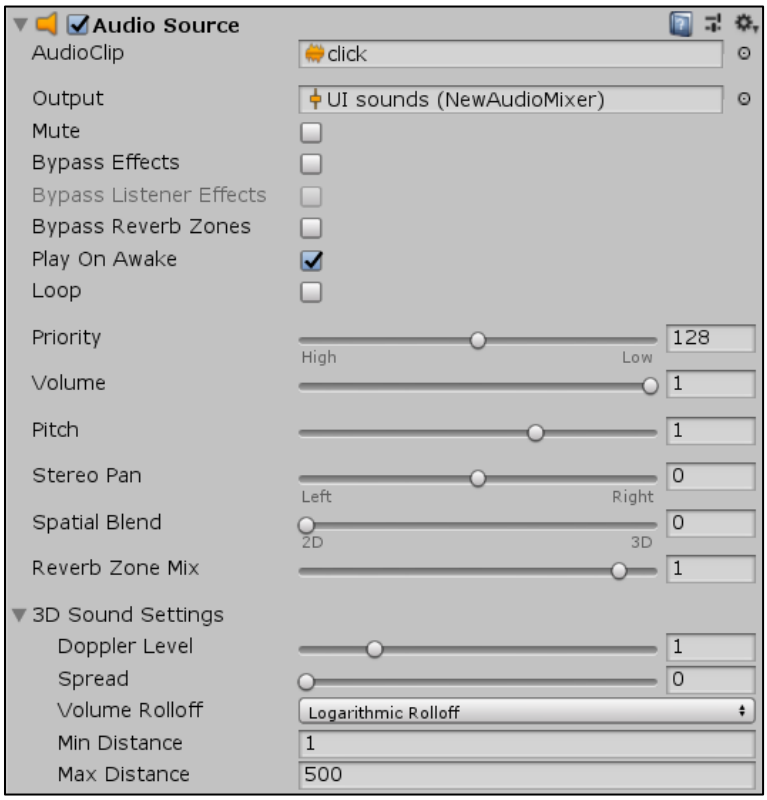
Unity bevat een audio engine met daarin een aantal componenten voor het integreren van audio in een game, die ik hieronder zal bespreken. Tevens geef ik twee aanvullingen op deze audio engine die het afspelen van geluiden gevarieerder kan maken en kan centraliseren.

STRUCTUUR VAN DE AUDIO ENGINE

De structuur van de audio engine in Unity is gebaseerd op het uitzenden van geluiden door geluidsbronnen (sources) die ontvangen worden door luisteraars (listeners), zoals dit ook in het echte leven gebeurt. Op welke manier een luisteraar een geluid waarneemt, is onder andere afhankelijk van de afstand, richting en snelheid van het geluid. Dit wordt in Unity gesimuleerd door het plaatsen van sources en listeners in de virtuele ruimte (scene) van Unity. Door positionering en beweging van deze objecten kunnen real-life-factoren zoals de afstand tot de bron en het Dopplereffect worden nagebootst.

AUDIO SOURCES EN LISTENERS

De twee belangrijkste componenten van de audio engine zijn de Audio Source en Audio Listener. Een source speelt een geluidsfragment (Audio Clip) af in de scene die wordt opgevangen door een listener. Een source biedt onder andere opties voor instellen van het volume en de afspeelsnelheid (pitch) van de clip en een optie om het geluid te herhalen. Figuur 1 geeft een overzicht van de instellingen van een Audio Source zoals die in het inspectorvenster (sneltoets Ctrl+3) worden weergegeven.



Figuur 1: Een overzicht van de instellingen van een Audio Source.

Een source bevat ook de instelling Spatial Blend. Deze instelling bepaalt of de source een clip in 2D of 3D moet afspelen, of een mix tussen deze twee. In 3D-modus wordt de clip door de audio engine in de virtuele ruimte geplaatst en op basis van de positie ten opzichte van de listener gemixt over de beschikbare kanalen (mono, stereo of surround afhankelijk van de beschikbare kanalen op het apparaat), terwijl in 2D-modus het geluid wordt afgespeeld zoals het op de oorspronkelijke kanalen is gemixt. In 3D-modus kan tevens met de instellingen Volume Rolloff, Min Distance en Max Distance worden bepaald met welke mate het volume van de clip daalt naarmate de source zich verder weg bevindt van de listener.

Een listener gedraagt zich als een soort microfoon: het vangt de geluiden op die worden afgespeeld door de sources in de scene en speelt deze af door de luidsprekers van het apparaat. Een listener heeft geen instellingen, maar wordt slechts geplaatst op een object. Voor de meeste games ligt het voor de hand om de listener op het object dat de speler representeert te plaatsen, Unity plaatst standaard een listener op de main camera. Er mag

zich altijd maar één actieve listener in de scene bevinden om de engine goed te laten werken.

Om sources en listeners toe te voegen aan een bestaand object in de scene, gebruik je het menu **Component > Audio > Audio Source/Listener**.

AUDIO CLIPS

Een source speelt een geluidsfragment af dat is opgeslagen in een Audio Clip. Deze clip representeert een geluidsbestand op je harde schijf (WAV, MP3 of een van de andere formaten die door Unity worden ondersteund). De clip bevat verschillende instellingen die het laden en afspelen van het geluid kunnen optimaliseren, maar het valt buiten de strekking van dit artikel om hier dieper op in te gaan.

Een clip kan aan het project worden toegevoegd door een bestand vanuit de verkenner in de lijst met assets te slepen of via het menu **Assets > Import New Asset**.

DE AUDIO MIXER

Het geluid dat op de verschillende sources in de scene wordt afgespeeld, wordt gemixt in de Audio Mixer. Deze mixer is vergelijkbaar met mixer-interfaces zoals die in veel digital audio workstations gebruikelijk zijn. In de mixer kunnen verschillende groepen (analoog aan mixer tracks) worden aangemaakt en vervolgens kan per source worden ingesteld naar welke groep deze zijn uitvoer moet sturen. Per groep kan het volume worden ingesteld en kunnen er verschillende filters en effecten worden toegevoegd, zoals een delay. Daarnaast kunnen verschillende toestanden van de mixer worden opgeslagen in snapshots, zodat eerder ingestelde waarden makkelijk terug te halen zijn.

Een Audio Mixer wordt aan het project toegevoegd via het menu **Assets > Create > Audio Mixer**. De mixer verschijnt daarna in de lijst met assets die beschikbaar is via het projectvenster (Ctrl+5).

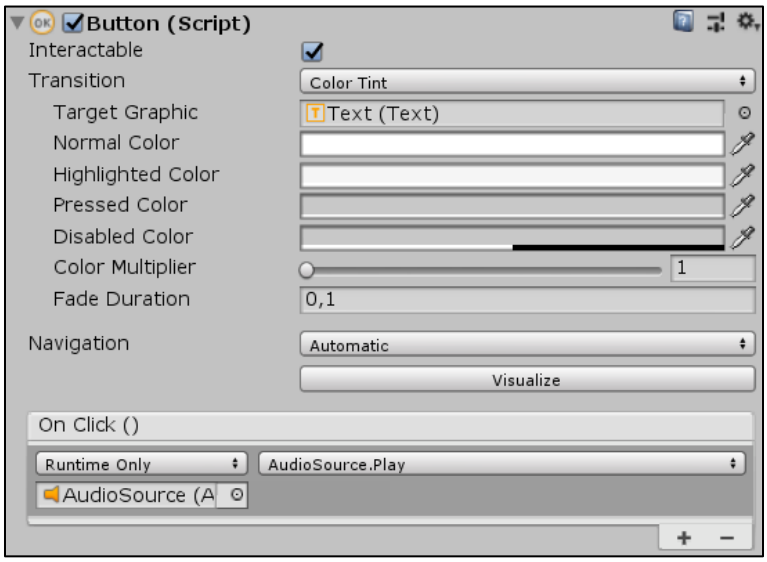
GELUIDEN AFSPELEN

Er zijn twee manieren om een geluid via een Audio Source af te spelen: via een event trigger op objecten of in een script.

AFSPELEN VIA EEN EVENT TRIGGER

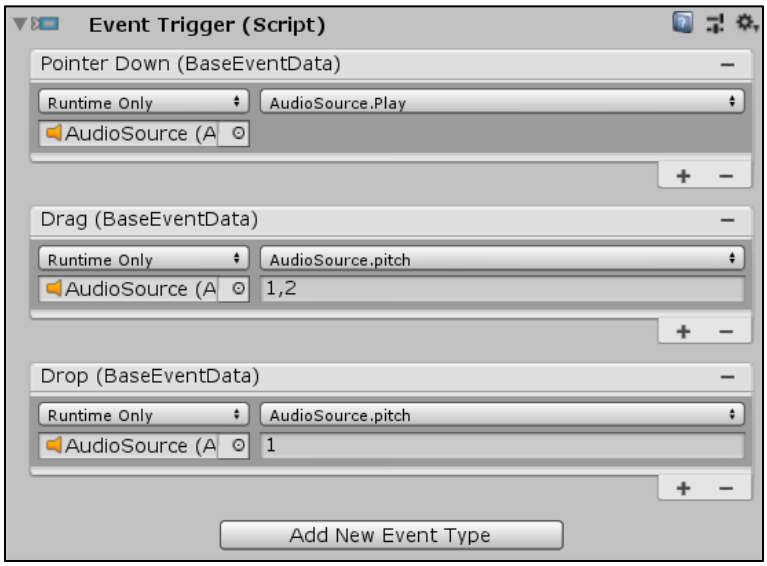
Veel user interface-componenten, zoals tekst, knoppen en sliders, hebben de mogelijkheid om acties uit te voeren wanneer een bepaald event plaatsvindt, zoals het klikken op zo'n object. Dit heet een event trigger. Op deze manier kunnen functies van een Audio Source worden aangeroepen wanneer iemand op een knop klikt of kunnen variabelen zoals het volume van de source veranderd worden zonder een regel code te schrijven. Dit werkt echter alleen als er een Event System aan de scene is toegevoegd, maar dit is in de meeste gevallen al nodig als er een interactieve user interface wordt gebruikt.

Als voorbeeld nemen we een Button. Dit component heeft een ingebouwde event trigger voor als erop geklikt wordt, het zogeheten On Click-event. In de instellingen van een Button kan een trigger worden toegevoegd bij het kopje **On Click ()** en hier kan een functie of variabele van een object gekozen worden die respectievelijk moet worden aangeroepen of veranderd. In figuur 2 is te zien hoe een Audio Source wordt afgespeeld wanneer op de Button wordt geklikt.



Figuur 2: Een overzicht van de instellingen van een Button.

Voor objecten die geen ingebouwde event triggers hebben, kan er een apart component met de naam Event Trigger op het object worden toegevoegd. In dit component kunnen events toegevoegd worden waarnaar moet worden geluisterd. In figuur 3 zie je dat een Audio Source wordt afgespeeld wanneer er op het object geklikt wordt. Wanneer het object versleept wordt, verandert de pitch van de source naar 1,2 keer zijn originele waarde en als het object vervolgens weer wordt neergezet, dan wordt de pitch weer op de oorspronkelijke waarde van 1 ingesteld.



Figuur 3: Een overzicht van de instellingen van een Event Trigger

AFSPELEN IN EEN SCRIPT

Als een object wat complexer wordt, is het soms wenselijk om de audio via een script te laten afspelen. Dit kan makkelijk door een referentie naar de Audio Source op te nemen in het script en deze vervolgens aan te roepen. Als voorbeeld nemen we een bal die een geluid afspeelt wanneer deze met een hoge snelheid botst met een ander object (zie figuur 4, voorbeeld uit de Unity-handleiding).

```
using UnityEngine;

public class SphereAudio : MonoBehaviour
{
  private AudioSource audioSource;

  void OnEnable()
  {
    audioSource = GetComponent<AudioSource>();
  }

  void OnCollisionEnter(Collision collision)
  {
    if (collision.relativeVelocity.magnitude > 2)
      audioSource.Play();
  }
}
```

Figuur 4: Dit script speelt een Audio Source af wanneer de bal botst met een ander object, bijvoorbeeld een vloer.

In de functie **OnCollisionEnter**, die als een event trigger functioneert wanneer de bal botst, wordt als eerste de snelheid van de botsing gecontroleerd. Als deze groter is dan de waarde 2, dan wordt het geluid in de Audio Source afgespeeld. De referentie naar die Audio Source wordt gemaakt op het moment dat het script actief wordt. Het script kiest de source die op hetzelfde object aanwezig is als het script.

Dit is een relatief simpel voorbeeld van geluid afspelen in een script, maar er kunnen in scripts allerlei logica en condities verbonden worden aan het afspelen van geluid. Deze manier is aanzienlijk veelzijdiger dan het aanroepen van een event trigger.

VARIATIES IN EEN AUDIO CLIP

De audio engine van Unity kan dus geluiden afspelen op Audio Sources die door een Audio Listener wordt opgevangen en wordt afgespeeld op de luidsprekers van een apparaat. Deze theorie is makkelijk toepasbaar, maar biedt weinig creativiteit op het gebied van de gebruikte geluidseffecten.

Een manier om Audio Clips creatiever te gebruiken is het aanbrengen van variaties in een bepaald geluidseffect. Zeker wanneer het een effect betreft dat veelvuldig wordt afgespeeld, kan het wenselijk zijn om hier variatie in aan te brengen. Dit kan op een aantal manieren:

- Het volume van de afgespeelde Audio Clip wordt gevarieerd, waardoor het geluid soms harder en soms zachter klinkt dan het origineel. Dit is het meest effectief voor non-diëgetische geluiden, zoals geluidseffecten van het drukken op een knop.
- De afspeelsnelheid (pitch) van de clip wordt gevarieerd, waardoor het geluid soms hoger en soms lager klinkt dan het origineel.
- Er wordt telkens een andere variant van het geluidseffect afgespeeld. Deze varianten zijn verschillende Audio Clips waarvan er willekeurig één gekozen wordt. Dit kunnen onder andere varianten met een ander volume of pitch zijn, maar deze parameters worden in tegenstelling tot de vorige punten niet programmatisch ingesteld.

Vanzelfsprekend kunnen deze drie manieren ook gecombineerd worden: er kan bijvoorbeeld een geluidseffect zijn met twee varianten waarvan de pitch ook nog programmatisch wordt bepaald.

EEN OBJECT VOOR VARIATIES

Om te voorkomen dat je in elk script dat geluid afspeelt logica moet inbouwen om een variatie te kiezen, is het handig om dit door een losstaand object te laten doen dat herbruikbaar is. Zo'n Sound-object is het beste te implementeren als een scriptable object. Dit is mogelijk omdat het Sound-object alleen maar instellingen bevat die worden doorgespeeld naar een Audio Source en zelf geen Audio Source bevat. Een Sound gedraagt zich op die manier eigenlijk hetzelfde als een Audio Clip: je plaatst ze net als clips in de assets van het project in plaats van ze als een object toe te voegen aan de scene.

Om bovenstaande drie variatiepunten te kunnen uitvoeren, moet het Sound-object in ieder geval de volgende variabelen bevatten:

- Het minimale en maximale volume van het geluidseffect. Tijdens het afspelen wordt een willekeurige waarde tussen deze twee getallen gekozen en ingesteld in de Audio Source.
- De minimale en maximale pitch van het effect. Qua logica werken deze waarden hetzelfde als het volume.
- Een lijst waarin de verschillende Audio Clips aan worden toegevoegd. Wanneer een geluid wordt afgespeeld wordt er een willekeurige clip uit deze array gekozen.
- Een boolean die aangeeft of dezelfde clip twee keer achter elkaar mag worden gekozen als er meerdere clips zijn.
- Een boolean die aangeeft of het afgespeelde geluid zich moet herhalen (loopen).

De logica die het Sound-object uitvoert is vrij simpel. Deze logica wordt uitgevoerd in een functie **PlayAtSource**, waarin je een Audio Source als argument meegeeft die wordt gebruikt om dit geluidseffect op af te spelen. Eerst kiest het object een willekeurige clip uit de array met Audio Clips. Bij het kiezen wordt rekening gehouden met of de vorige afgespeelde clip nog een keer mag worden gekozen. Daarna wordt deze clip in de source ingesteld, net als de loop-variabele en een willekeurige waarde voor het volume en de pitch, die begrensd wordt door de ingestelde minima en maxima. Hierna wordt de **Play**-functie van de source aangeroepen en speelt het geluid.

In figuur 5 is een implementatie van het Sound-object te zien. Vanwege de leesbaarheid is de logica voor het selecteren van een clip in een aparte functie **SelectClip** ondergebracht. Let ook op de derde regel, die met **CreateAssetMenu** begint. Deze regel maakt het menu **Assets > Create > Scriptable Objects > Sound** aan, waarmee je Sound-objecten aan de assets van het project toevoegt.

```
using UnityEngine;

[CreateAssetMenu(fileName = "Sound", menuName = "Scriptable Objects/Sound")]
public class Sound : ScriptableObject
{
  public AudioClip[] clips;
  private AudioClip lastSelectedClip = null;

  public bool loop = false;
  public bool preventSameClips = false;

  public float minVolume = 1.0f;
  public float maxVolume = 1.0f;
  public float minPitch = 1.0f;
  public float maxPitch = 1.0f;

  public AudioClip SelectClip()
  {
    if (clips.Length == 0)
      return null;

    else if (clips.Length == 1)
      return clips[0];

    else
    {
      AudioClip selectedClip = null;
      do
      {
        selectedClip = clips[UnityEngine.Random.Range(0, clips.Length)];
      } while (selectedClip == null || (preventSameClips && selectedClip == lastSelectedClip));
      return lastSelectedClip = selectedClip;
    }
  }

  public void PlayAtSource(AudioSource source)
  {
    AudioClip selectedClip = SelectClip();
    if (selectedClip == null)
      return;

    source.clip = selectedClip;
    source.loop = loop;
    source.volume = Random.Range(minVolume, maxVolume);
    source.pitch = Random.Range(minPitch, maxPitch);
    source.Play();
  }
}
```

Figuur 5: Het script voor het Sound-object

In figuur 6 is het script van de stuiterende bal (figuur 4) aangepast zodat het object een Sound gebruikt dat het stuitergeluid afspeelt. In de inspector moet je nu wel een referentie maken naar het Sound-object dat moet worden gebruikt als stuitergeluid. Omdat dit Sound-object alle logica afhandelt, hoef je geen clip meer bij de Audio Source in te stellen.

```
using UnityEngine;

public class SphereAudioWithVariations : MonoBehaviour
{
  public Sound bounceSound;

  private AudioSource audioSource;

  void OnEnable()
  {
    audioSource = GetComponent<AudioSource>();
  }

  void OnCollisionEnter(Collision collision)
  {
    if (collision.relativeVelocity.magnitude > 2)
      bounceSound.PlayAtSource(audioSource);
  }
}
```

Figuur 6: Dit script speelt een Sound af op de Audio Source van het object

Voor simpele events zoals het klikken met de muis op een object kun je ook een event trigger gebruiken, zoals in figuur 7 is weergegeven.

Figuur 7: Een Event Trigger voor het afspelen van een Sound

GELUIDEN BEHEREN

Wanneer er veel audio in een project gebruikt wordt op verschillende plekken in de scene, is het handig om het afspelen van deze audio te centraliseren in een speciaal object. Zulke objecten worden audio managers genoemd.

EEN SIMPELE AUDIO MANAGER

Een simpele versie van een audio manager bevat referenties naar de verschillende Sound-objecten en de sources waar deze op worden afgespeeld. Daarnaast zijn er een aantal functies die de logica uitvoeren om een bepaald geluid af te spelen. Als een scene bijvoorbeeld een UI-element bevat en geluiden afspeelt wanneer een speler op knoppen in deze UI drukt, dan is er in de audio manager een functie **PlayClickSound** die het klikgeluid afspeelt op een audio source die aan de UI gekoppeld is. Zo kan er voor de meeste globale

geluiden een functie worden gemaakt als deze niet specifiek bij een object horen (zie figuur 8).

```
using UnityEngine;

public class AudioManager : MonoBehaviour
{
  public Sound clickSound;
  public Sound confirmSound;
  public Sound cancelSound;
  public Sound moveSound;

  public AudioSource uiAudioSource;
  public AudioSource movementAudioSource;

  public void PlayClickSound()
  {
    clickSound.PlayAtSource(uiAudioSource);
  }
  public void PlayConfirmSound()
  {
    confirmSound.PlayAtSource(uiAudioSource);
  }
  public void PlayCancelSound()
  {
    cancelSound.PlayAtSource(uiAudioSource);
  }
  public void PlayMoveSound()
  {
    moveSound.PlayAtSource(movementAudioSource);
  }
}
```

Figuur 8: Een simpel audio manager-script

Deze audio manager kun je vervolgens als een script op een bestaand object plaatsen, zoals de main camera, of je kunt er een nieuw object voor maken die alleen de manager bevat.

Het voordeel van zo'n audio manager is dat je in het object dat de audio afspeelt alleen maar een referentie naar de audio manager hoeft op te nemen en de bijbehorende functie aan te roepen via een event trigger of in het script. In figuur 9 is te zien dat het afspelen van een geluid via de manager nu gereduceerd is tot enkele simpele regels code.

```
using UnityEngine;

public class SphereAudioWithManager : MonoBehaviour
{
  public AudioManager audioManager;

  void OnCollisionEnter(Collision collision)
  {
    if (collision.relativeVelocity.magnitude > 2)
      audioManager.PlayClickSound();
  }
}
```

Figuur 9: Dit script speelt een geluid af via de audio manager

IDEEËN VOOR VEELZIJDIGE MANAGERS

Een audio manager kan zo simpel of complex zijn als nodig is. Een voorbeeld van een verbetering van de simpele manager is het voorkomen dat langere geluidseffecten door elkaar heen gaan spelen of elkaar afkappen. Daarnaast vereisen sommige games, waaronder multiplayer-games, complexere manieren om het afspelen van geluid te beheren.

Een handig middel is bijvoorbeeld een prioriteitsvariabele voor een geluidseffect. Afhankelijk van deze prioriteit kan een geluidseffect het al spelende geluid stoppen en meteen worden afgespeeld of in de wacht gezet worden om op een later moment te worden afgespeeld. Een geluidseffect kan ook in zijn geheel optioneel worden gemaakt en alleen afgespeeld worden wanneer er niets anders belangrijks wordt afgespeeld. Een prioriteit is bijvoorbeeld handig voor voice-overs, die onduidelijk worden als ze door elkaar heen klinken.

Daarnaast is een instelling die aangeeft voor welke speler het geluid moet klinken handig bij multiplayer-games. In het geval van voice-overs wil je bijvoorbeeld niet dat een tegenspeler hoort wat een speler voor acties uitvoert. In het andere geval wil je dat algemene aankondigingen wél voor alle spelers hoorbaar zijn.

Voor audio die aan een vast object gekoppeld is het vaak handiger om een apart script te schrijven dat aan dat specifieke object gekoppeld is. Een globale audio manager kan het afspelen van specifiek geluid alleen maar ingewikkelder maken, terwijl een toegespitst script veel makkelijker werkt.

REFERENTIES

Lanham, Micheal (2017), Game Audio Development with Unity 5.x
Birmingham, UK: Packt Publishing Ltd.

Wijnmalen, Milan (2016), Adaptieve muziek: De samenwerking tussen componisten en programmeurs

Hirst, Tony (2008), Listening to the Game – A Brief History of Game Audio
https://digitalworlds.wordpress.com/2008/04/12/listening-to-the-game-a-brief-history-of-game-audio/, 5 mei 2019

McDonald, Glenn (2005), A History of Video Game Music
https://www.gamespot.com/articles/a-history-of-video-game-music/1100-6092391/, 5 mei 2019

Salvadori, Lorenzo (2014) Audio Design and C# Scripting
http://www.lorenzo-salvadori.com/audio-design-and-c-scripting/, 29 april 2019

https://unity3d.com/learn/tutorials/topics/user-interface-ui/ui-events-and-event-triggers, 6 mei 2019

https://docs.unity3d.com/Manual/AudioOverview.html, 29 april 2019

https://docs.unity3d.com/Manual/class-AudioClip.html, 29 april 2019

https://docs.unity3d.com/Manual/class-AudioListener.html, 29 april 2019

https://docs.unity3d.com/Manual/class-AudioSource.html, 29 april 2019

https://docs.unity3d.com/Manual/class-ScriptableObject.html, 12 mei 2019

https://docs.unity3d.com/ScriptReference/Collider.OnCollisionEnter.html, 6 mei 2019