Unbreaking Immersion

Audio Implementation for INSIDE

Wwise Tour 2016

Martin Stig Andersen and Jakob Schmid
PLAYDEAD

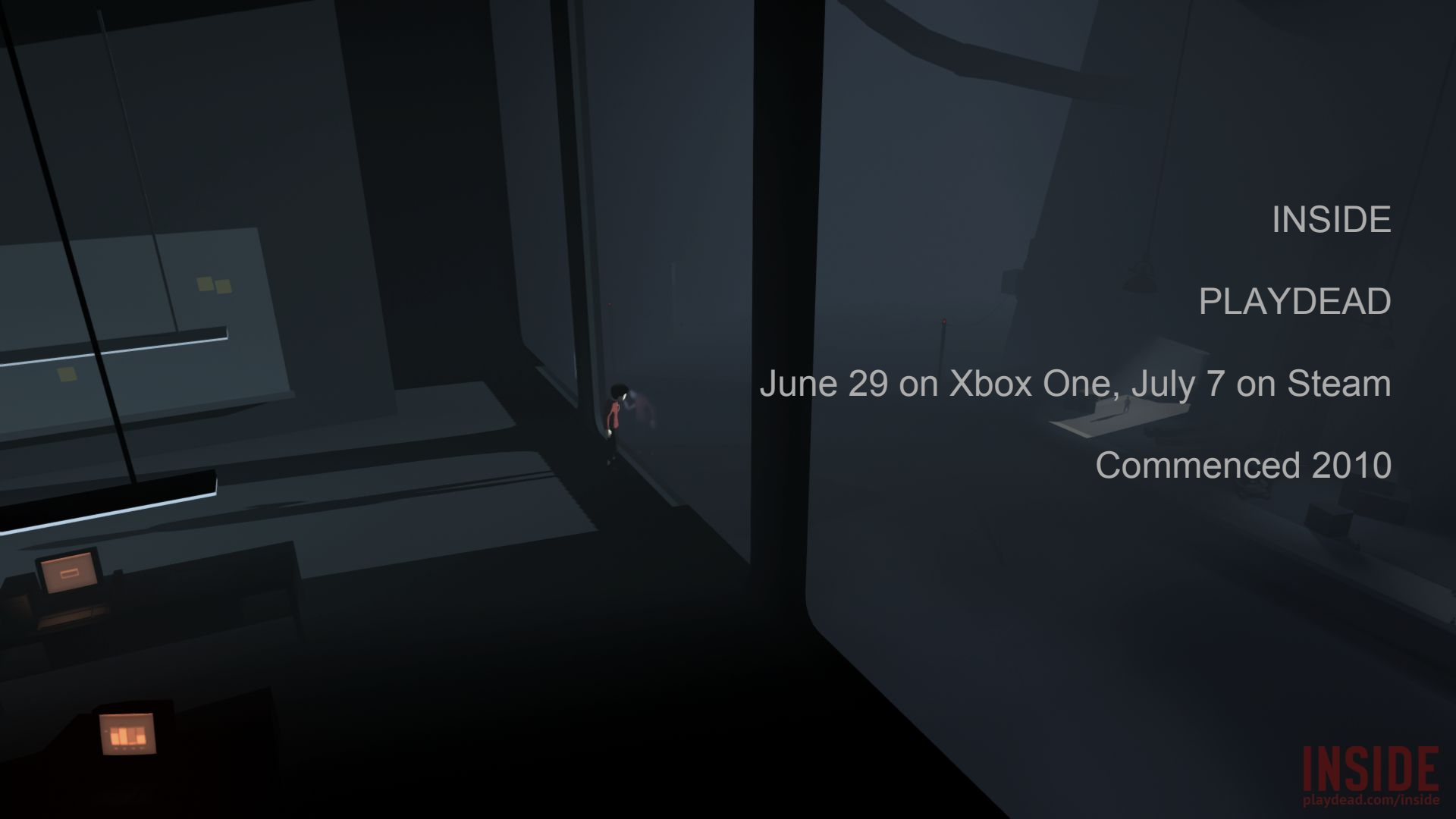
Martin Stig Andersen

Audio director, composer and sound designer

Jakob Schmid

Audio programmer at Playdead

Composer and sound designer on 140


INSIDE
PLAYDEAD
June 29 on Xbox One, July 7 on Steam
Commenced 2010
INSIDE
playdead.com/inside

Playdead Audio Team

Martin Stig Andersen	composer, sound designer
SØS Gunver Ryberg	composer, sound designer
Andreas Frostholm	sound designer
Jakob Schmid	audio programmer

Unbreaking Immersion

- Introduction
- Voice
- Scene Change
- Performance

Slides available online. Link on last slide!

Voice

Voice Concept

- Natural and adaptable audio playback
- Integration of physical and emotional states

Voice Demo

INSIDE Technology

Unity

Audiokinetic Wwise

Modified Wwise-Unity plugin

PlayMaker

Voice Sequencer

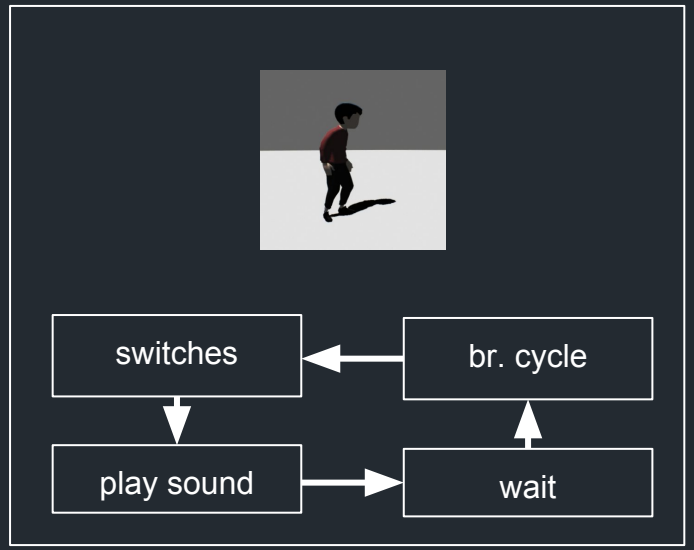
- Sequencer implemented in C# using Wwise callbacks
- Sequences voice sound events, alternating between inhale and exhale

Voice Sound Events

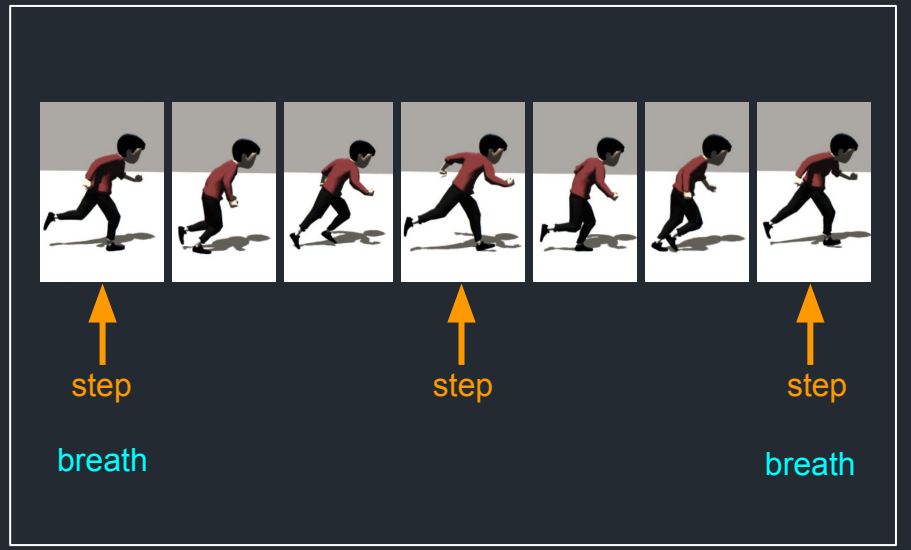
- Which sound to play is defined by switches:
  - Action
  - Emotion
  - Intensity
  - Etc.
- Intensity is a numeric value:
  - Increases with physical exertion
  - Decreases when idle

Voice Sequencer Modes

Continuous Mode

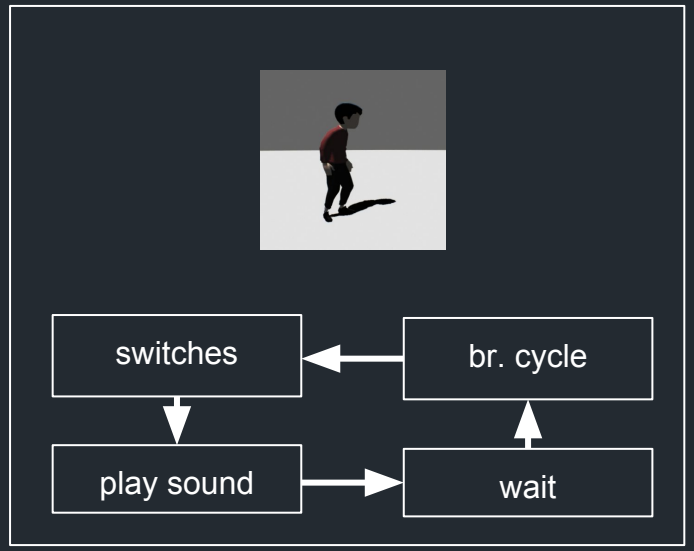


Rhythmic Breathing



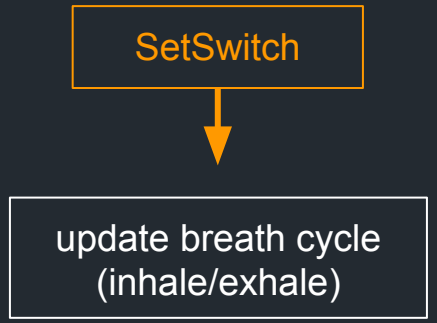
Voice Sequencer Modes

Continuous Mode

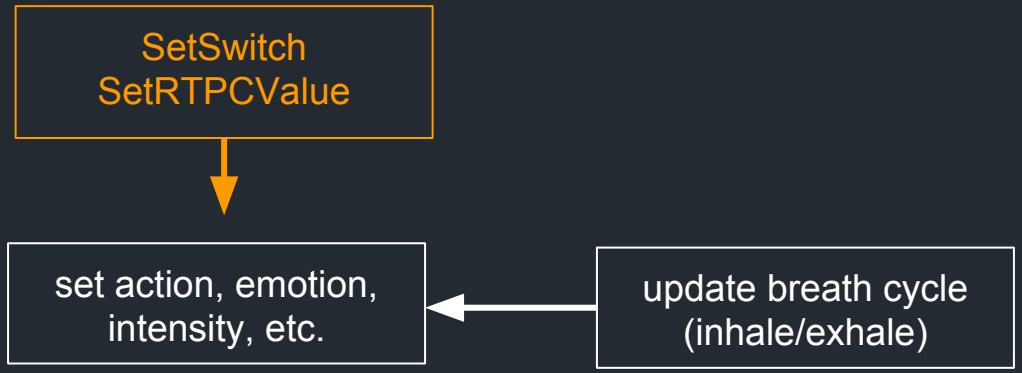


Rhythmic Breathing

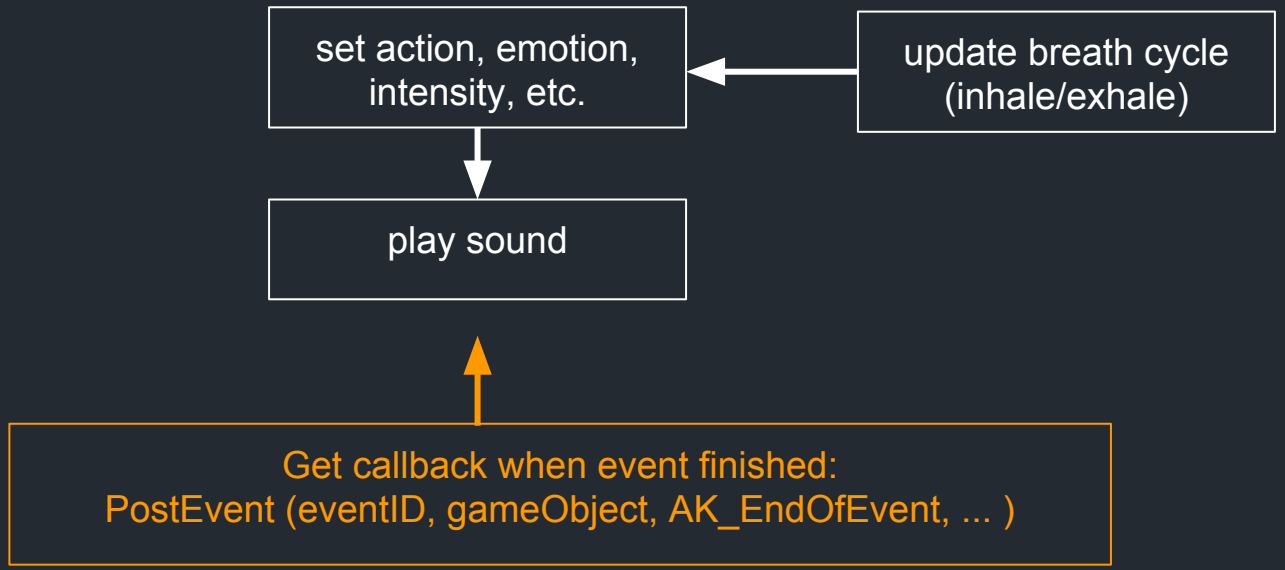
Voice Sequencer: Continuous Mode



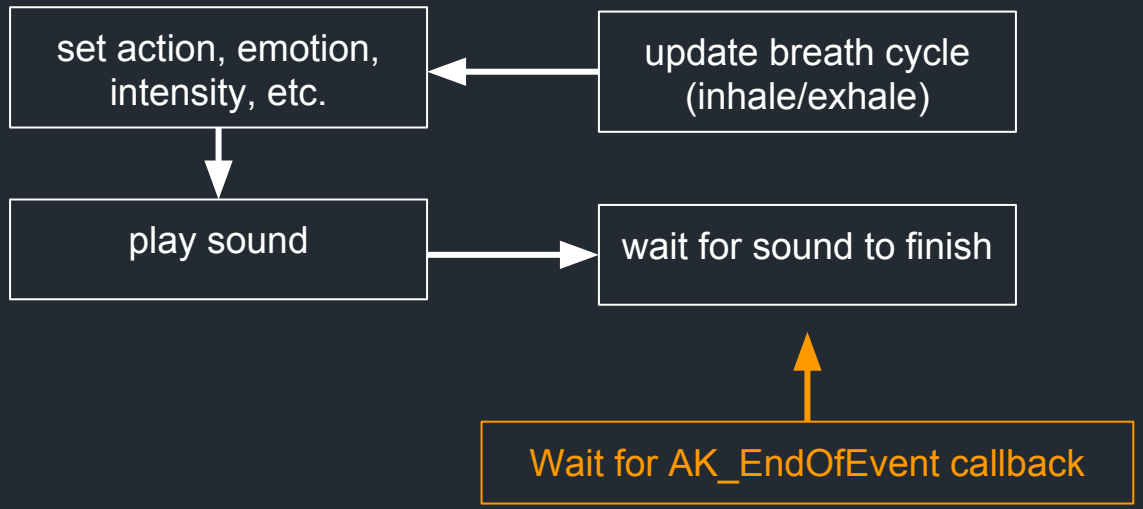
Voice Sequencer: Continuous Mode



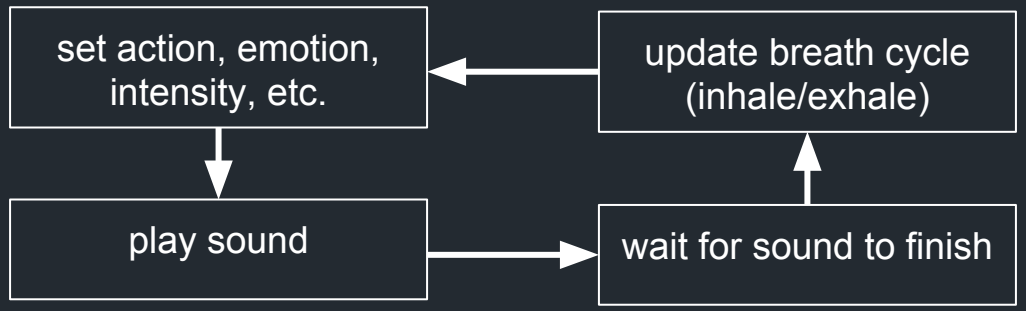
Voice Sequencer: Continuous Mode



Voice Sequencer: Continuous Mode

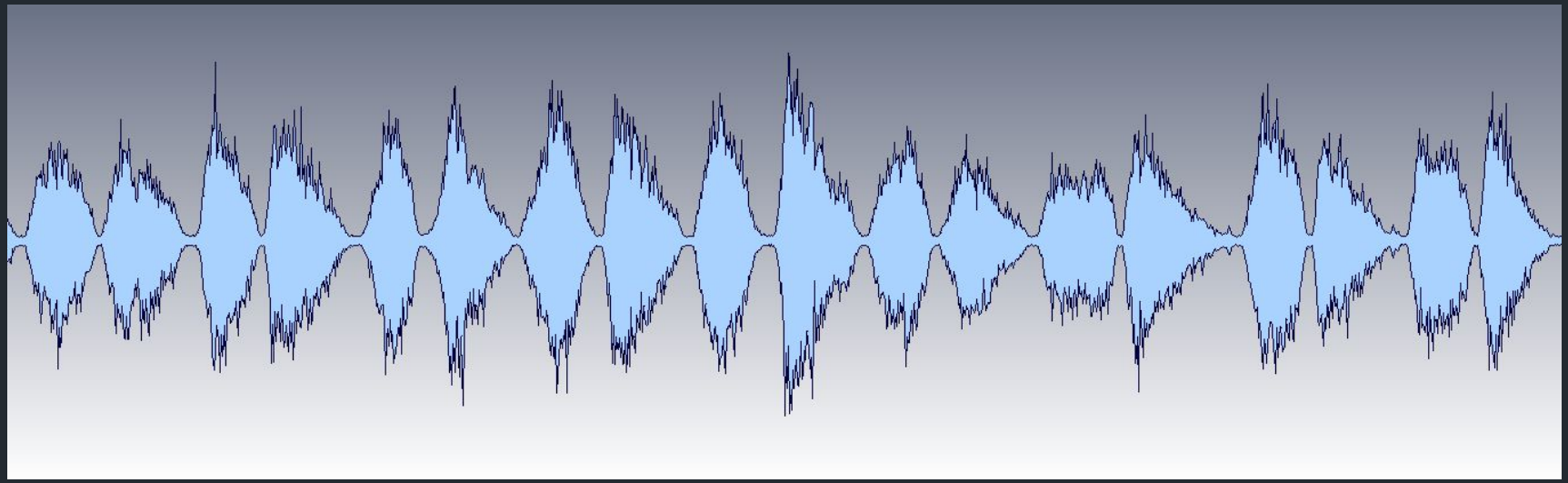


Voice Sequencer: Continuous Mode



Natural Breathing

- Recorded breath sounds have varying durations
- Continuous sequencing results in natural, uneven breathing pattern



Animation Feedback

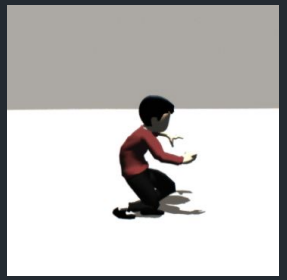
- Every breath results in a callback to the game
- Callback controls additive breathing animation, affecting boy pose

Holding Breath

On jump:

if currently inhaling, stop afterwards

if currently exhaling, do a quick inhale, then stop



On land:

restart breathing with exhale (action = land)

soft impact: normal exhale, hard impact: grunt

Engagement Actions

Special actions indicate performing work, uses different set of sounds

not engaged

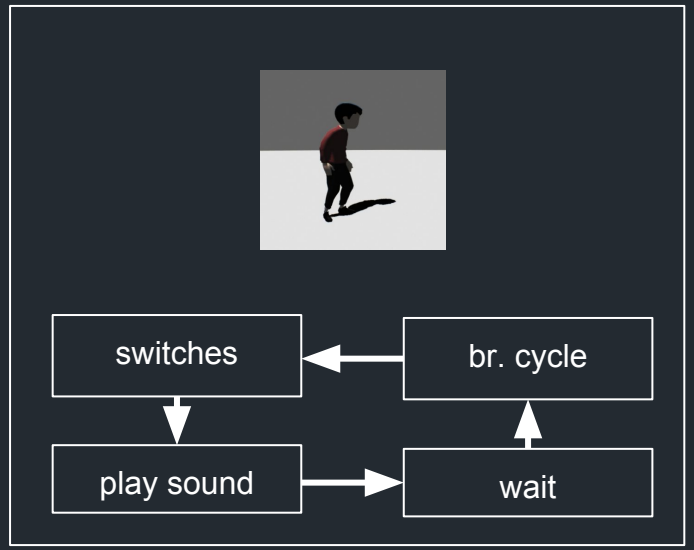
engaged passive

engaged active

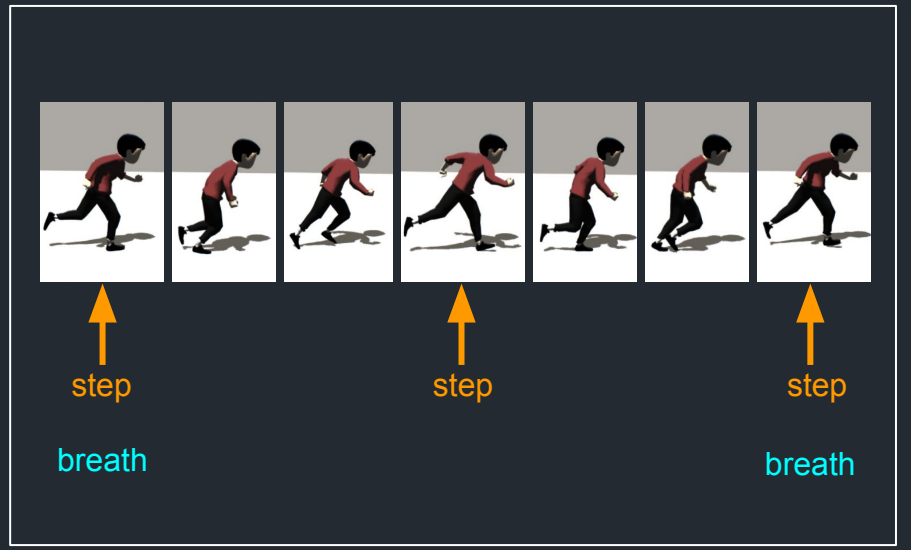
Voice Wwise Setup

Voice Sequencer Modes

Continuous Mode

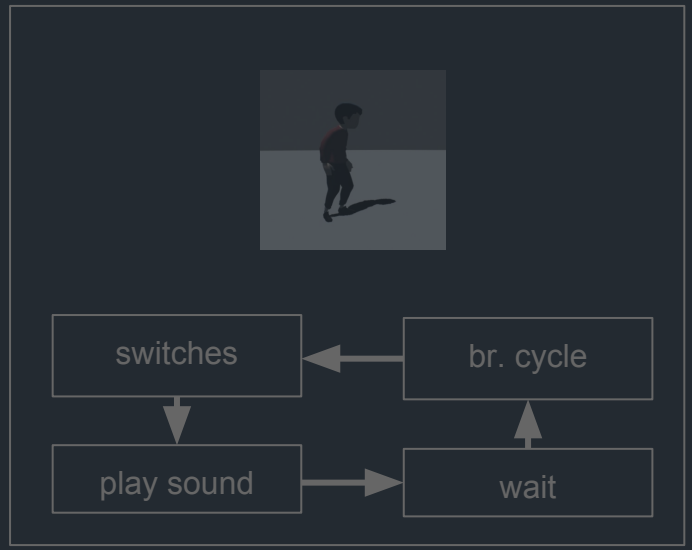


Rhythmic Breathing

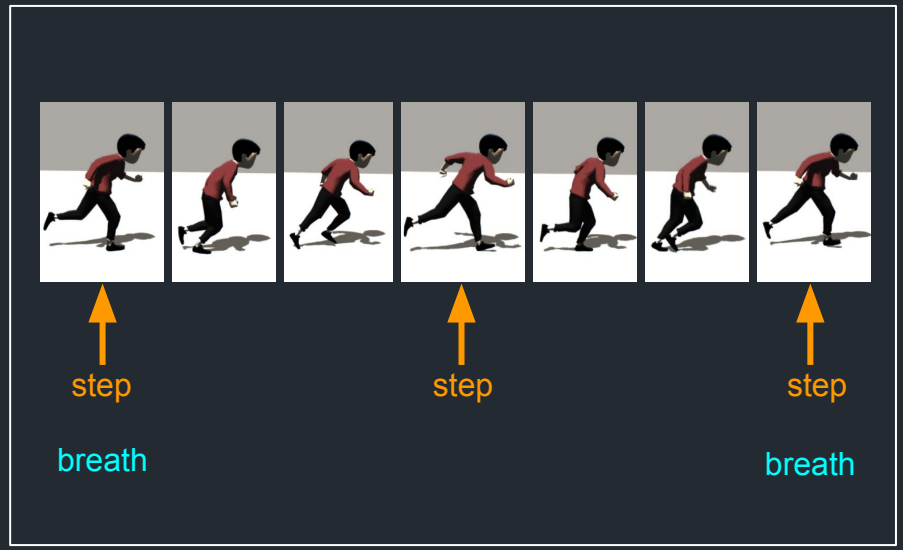


Voice Sequencer Modes

Continuous Mode



Rhythmic Breathing



Rhythmic Breathing

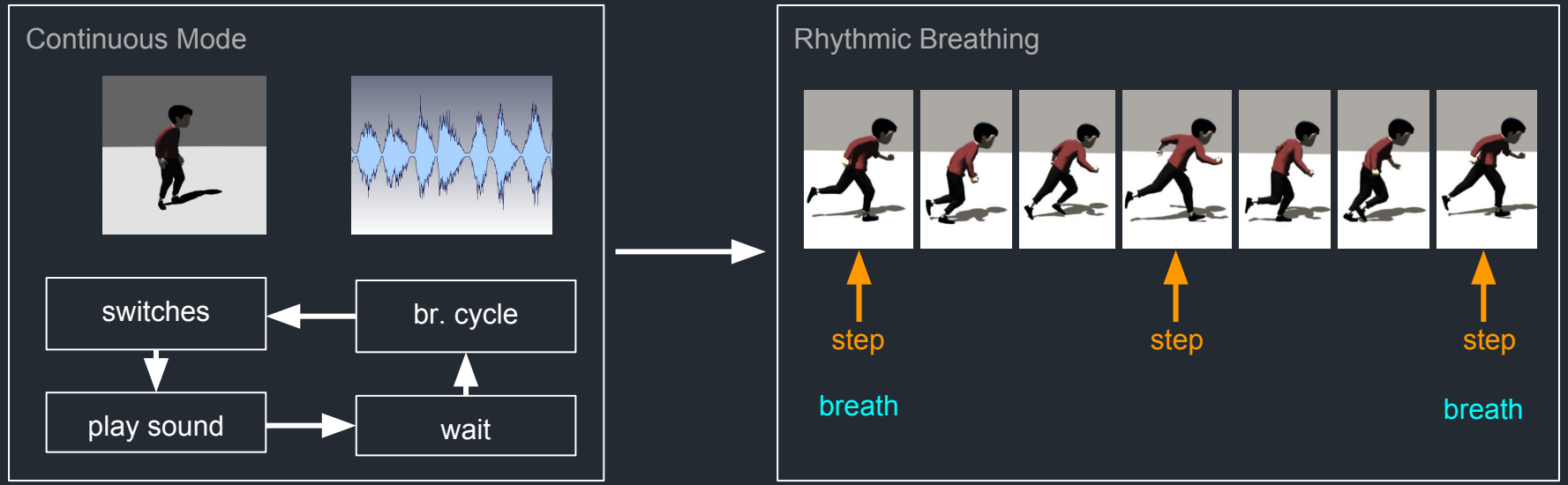
- Goal: breath should align with footsteps when running
- Non-continuous sequencing

Rhythmic Breathing

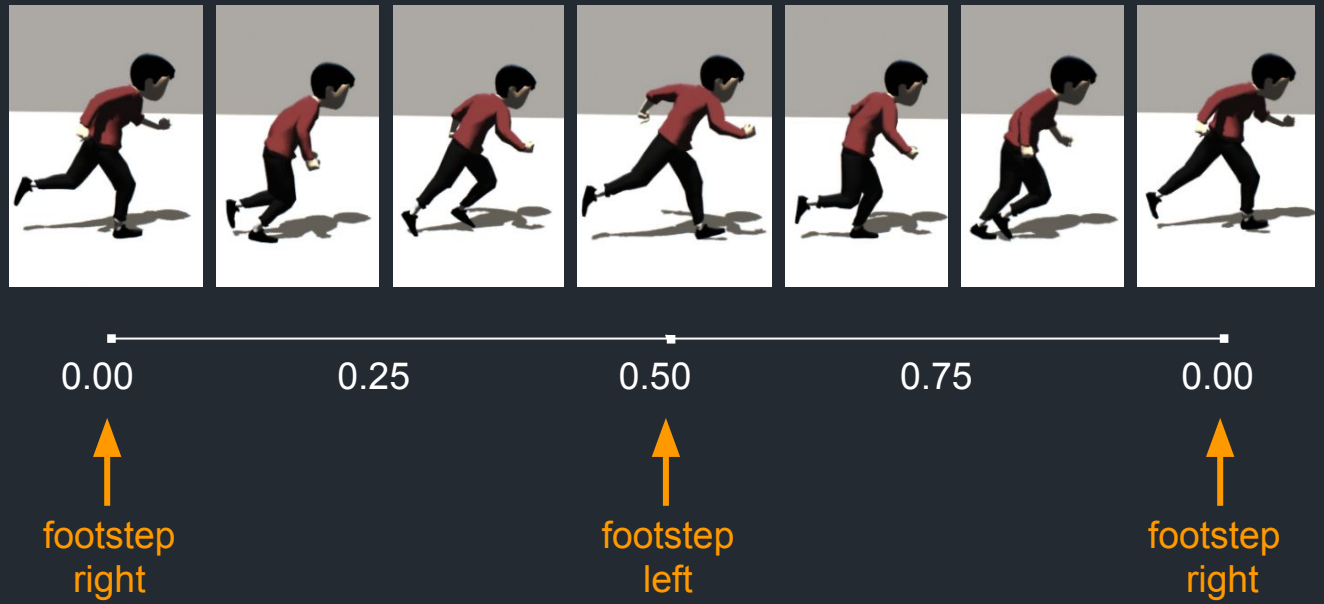
- Goal: breath should align with footsteps when running
- Non-continuous sequencing
- 1 breath for every 2 steps

Rhythmic Breathing Transition

- When not running, breath runs continuously
- When starting to run, gradually transition from continuous rhythm to footstep rhythm



Run Cycle Phase

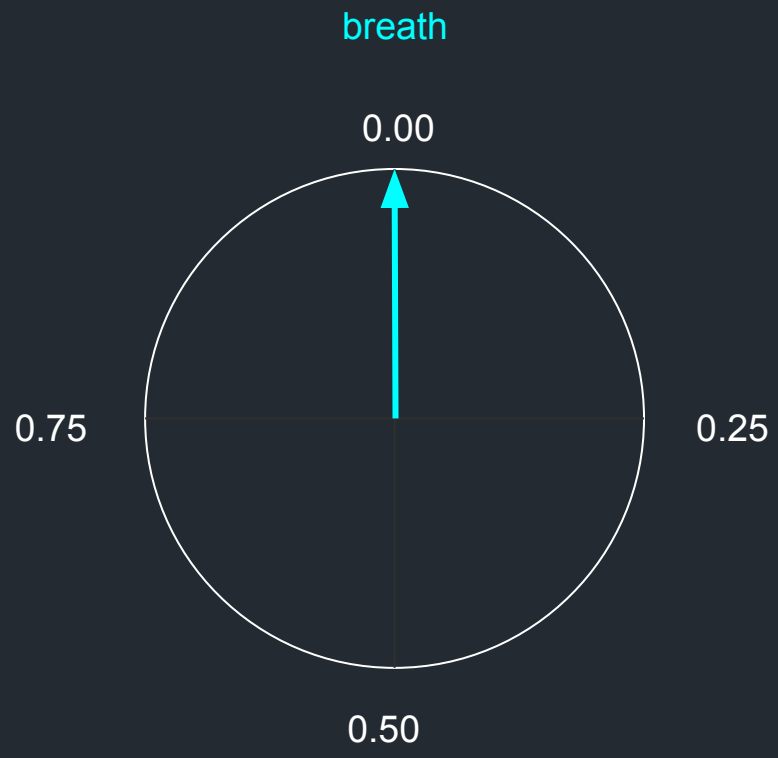


Run Cycle Phase

- Full cycle is 2 steps
- Right footstep on 0.0
- Left footstep on 0.5

Breath Phase

- Breathe when phase is 0
- Full cycle is 1 breath
- When switching from continuous to rhythmic breathing:
  - Compute frequency from last 2 breaths
  - Compute phase from frequency and last breath time



Gradual Alignment

- Gradually align breath rhythm to run cycle rhythm
- Align two **frequency, phase** pairs

Gradual Alignment Problem

- Who knows about aligning two **frequency, phase** pairs?

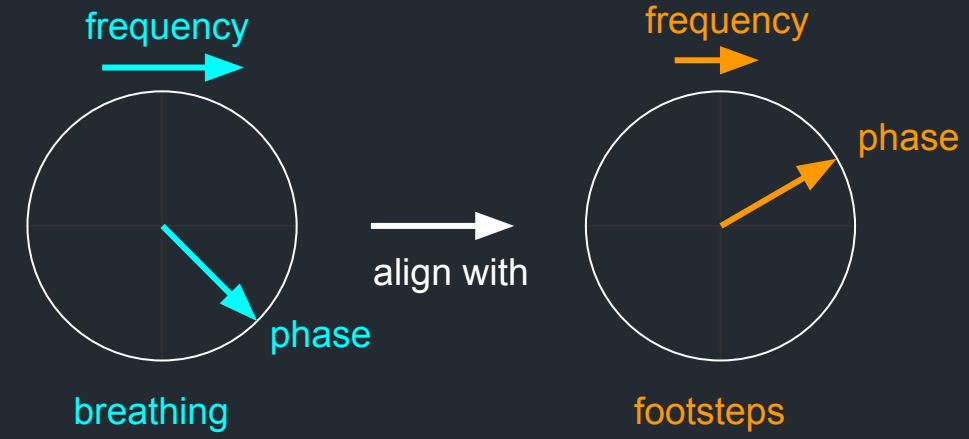
Solution: Beat Matching

- Who knows about aligning two **frequency, phase** pairs?
- DJs do.

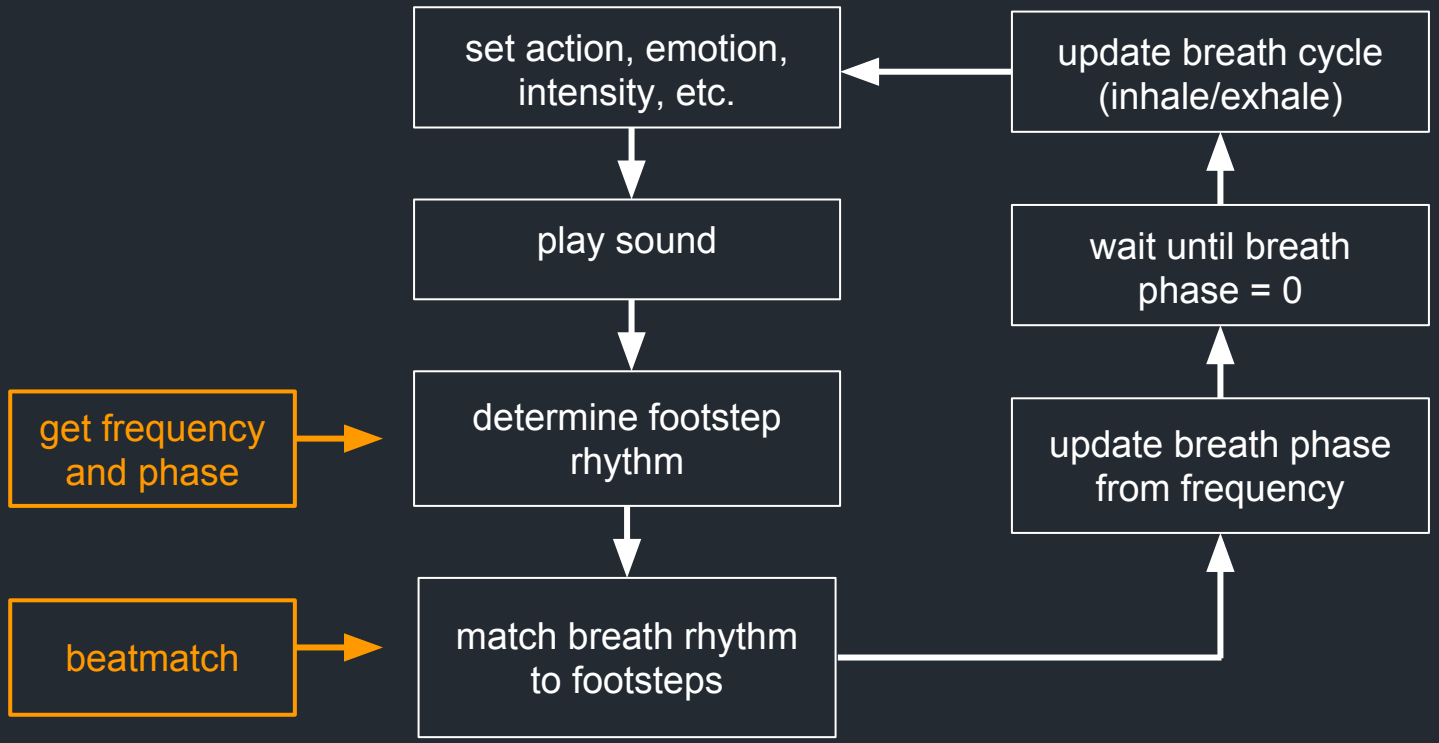
Solution: Beat Matching

- Gradually interpolate breath frequency towards run cycle frequency
- Compensate breath frequency for phase offset

- Like a DJ that uses pitch adjust without nudging the record



Voice Sequencer: Rhythmic Breathing



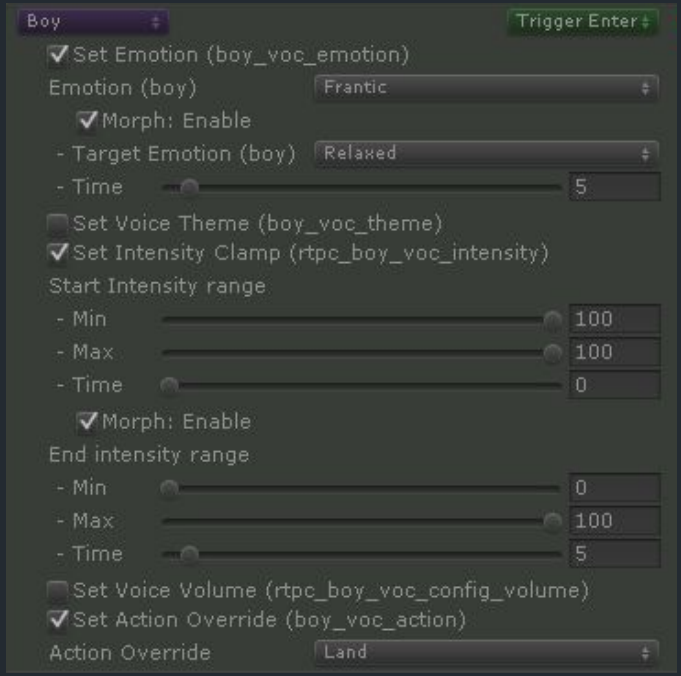
Voice Direction

- Voice direction is accomplished using our voice configuration system
- The director (Martin) instructs the actor (voice sequencer) how to emote:
  - based on location or
  - based on reacting to events

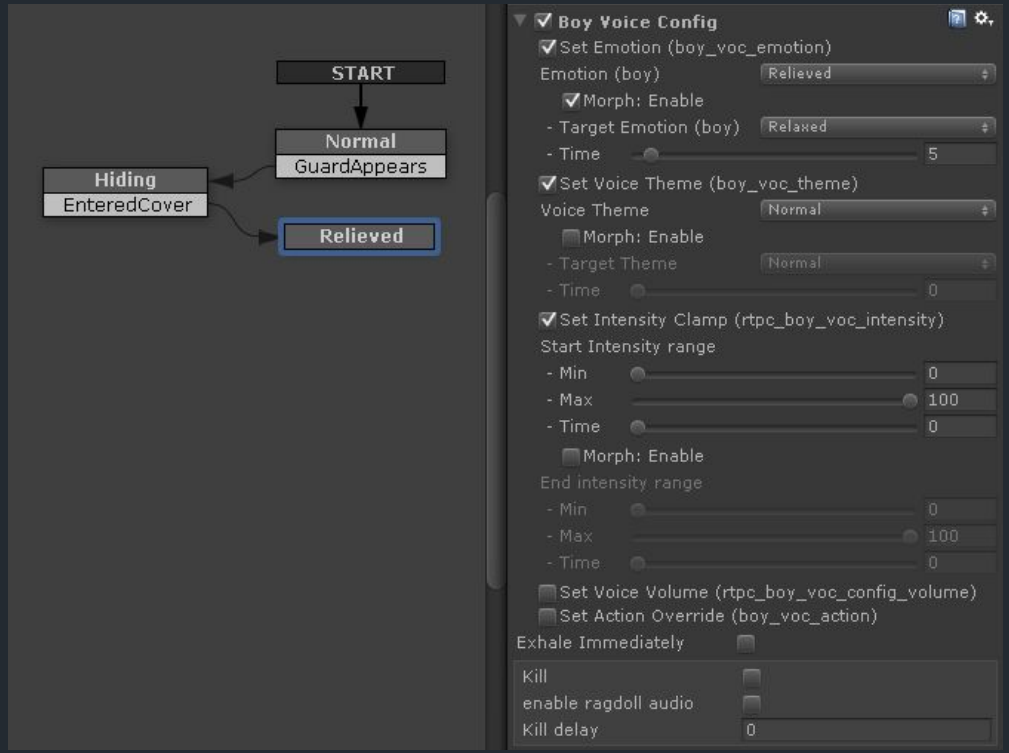
Voice Configuration

- Trigger boxes
- State machines
- Scripts
- Gives full control over voice parameters
  - action
  - emotion
  - intensity

Voice Configuration: Trigger box

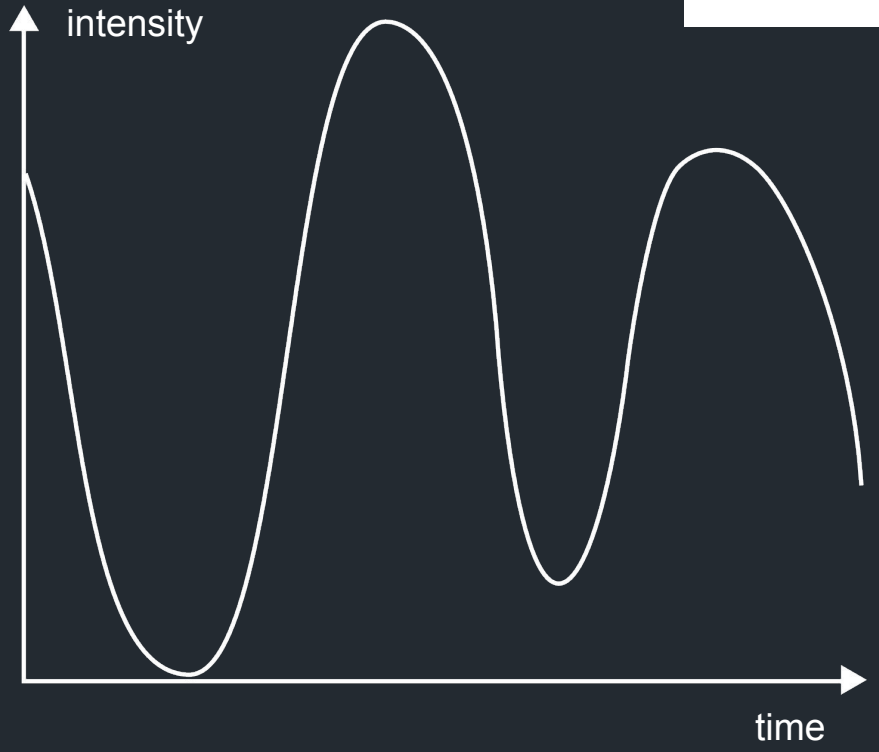


Voice Configuration: State Machine



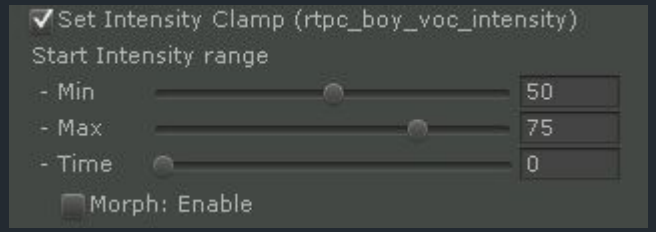
Voice Intensity

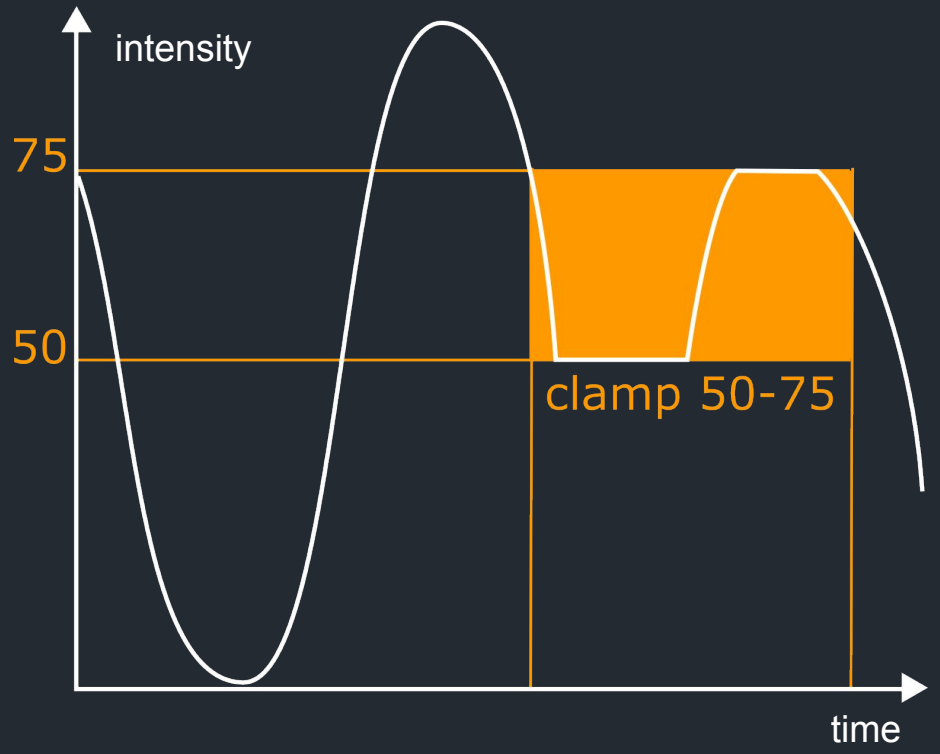
- Boy movement generates exhaustion
- Voice intensity = lowpass filtered exhaustion
- Voice Intensity selects depth and force of breathing
- Depending on the emotion parameter, intensity defines:
  - Physical exertion level
  - Intensity of character emotion



Voice Intensity Clamping

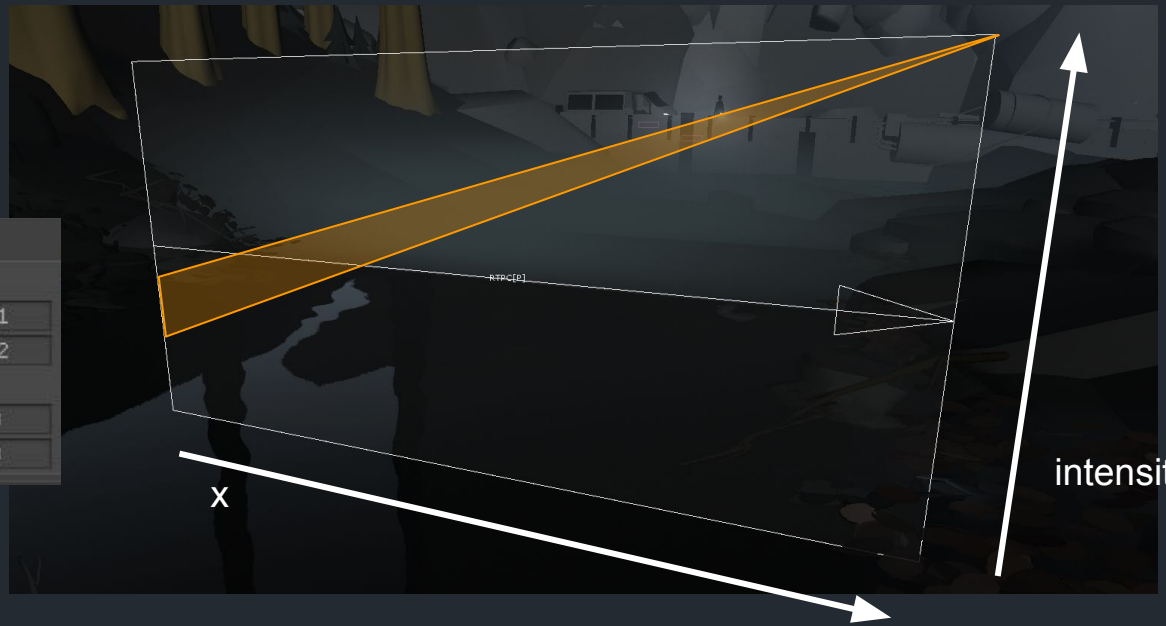
- Clamping constrains intensity to a given range





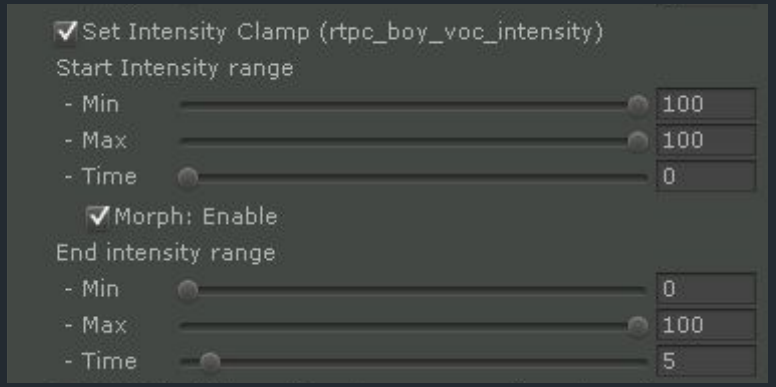
Voice Intensity Interpolation in Space

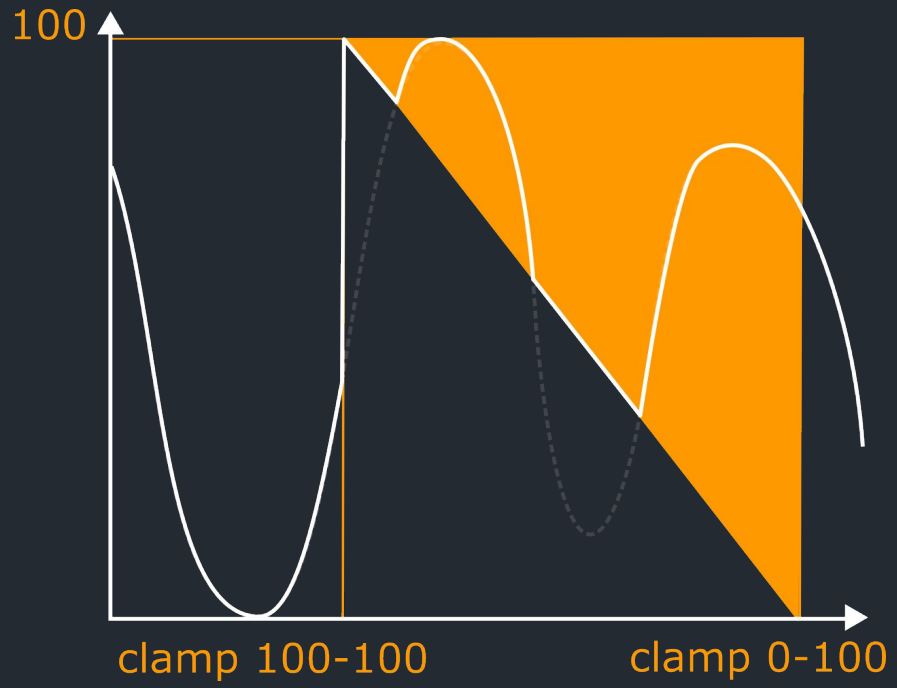
- Useful for indicating proximity to danger



Voice Intensity Interpolation over Time

- Useful for creating reactions to game events, and relaxing over time.





Voice Summary

- Single event, switch hierarchy determines sound
- Continuous sequencing using callbacks
- Rhythmic breathing uses beatmatching to align breath to footsteps
- Voice direction with trigger boxes and state machines
- Voice Intensity can be clamped
- Clamping can be interpolated in space and time

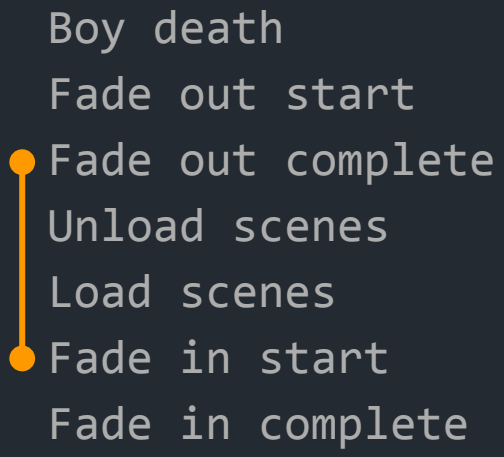
Shockwave Demo

Scene Change

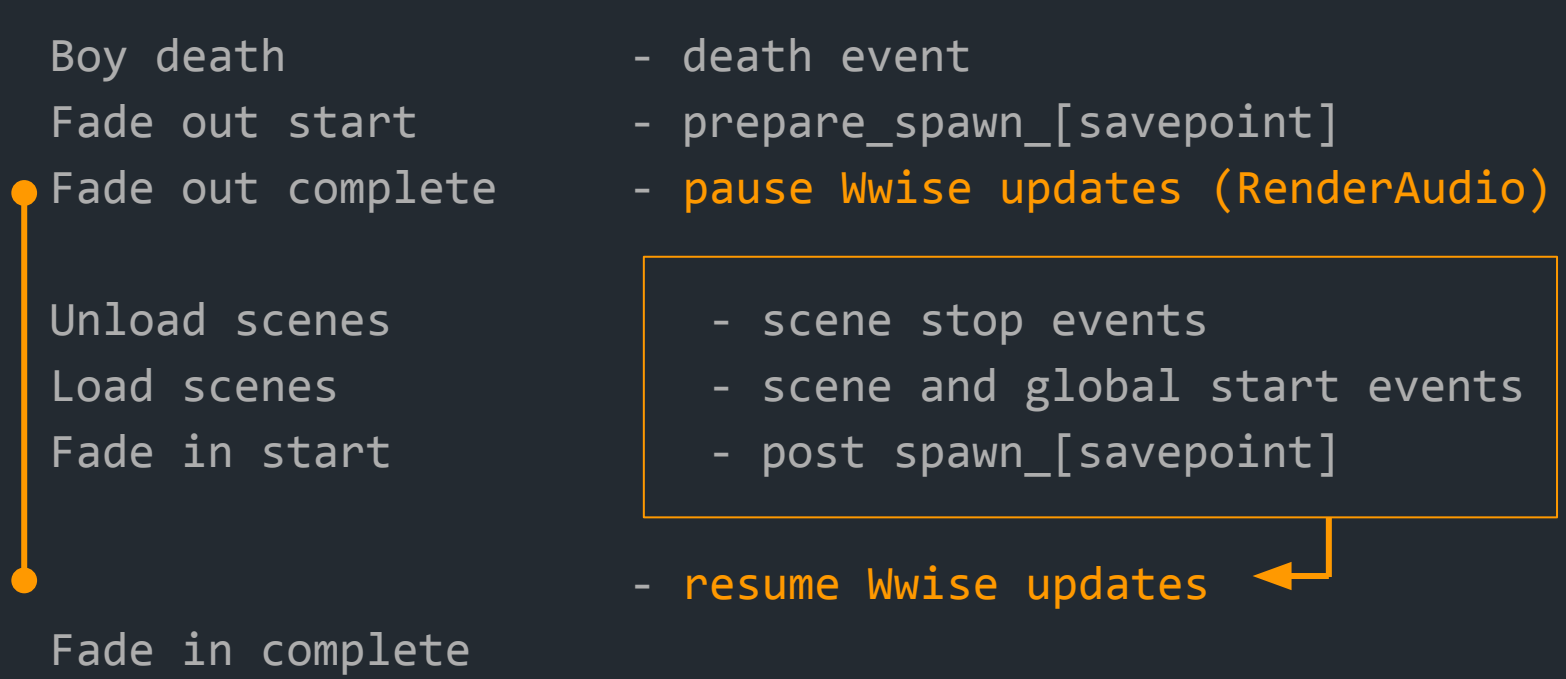
- When main character dies, scenes are reloaded
- Audio should retain state and continue during load
- When reload is complete, audio should switch to new state instantly
- We call this a scene change

Image credit: The New York Times: 'Times Lapse Video: Behind the Scenes at the Metropolitan Opera'

Scene Change Events



Scene Change Implementation



Scene Change Implementation

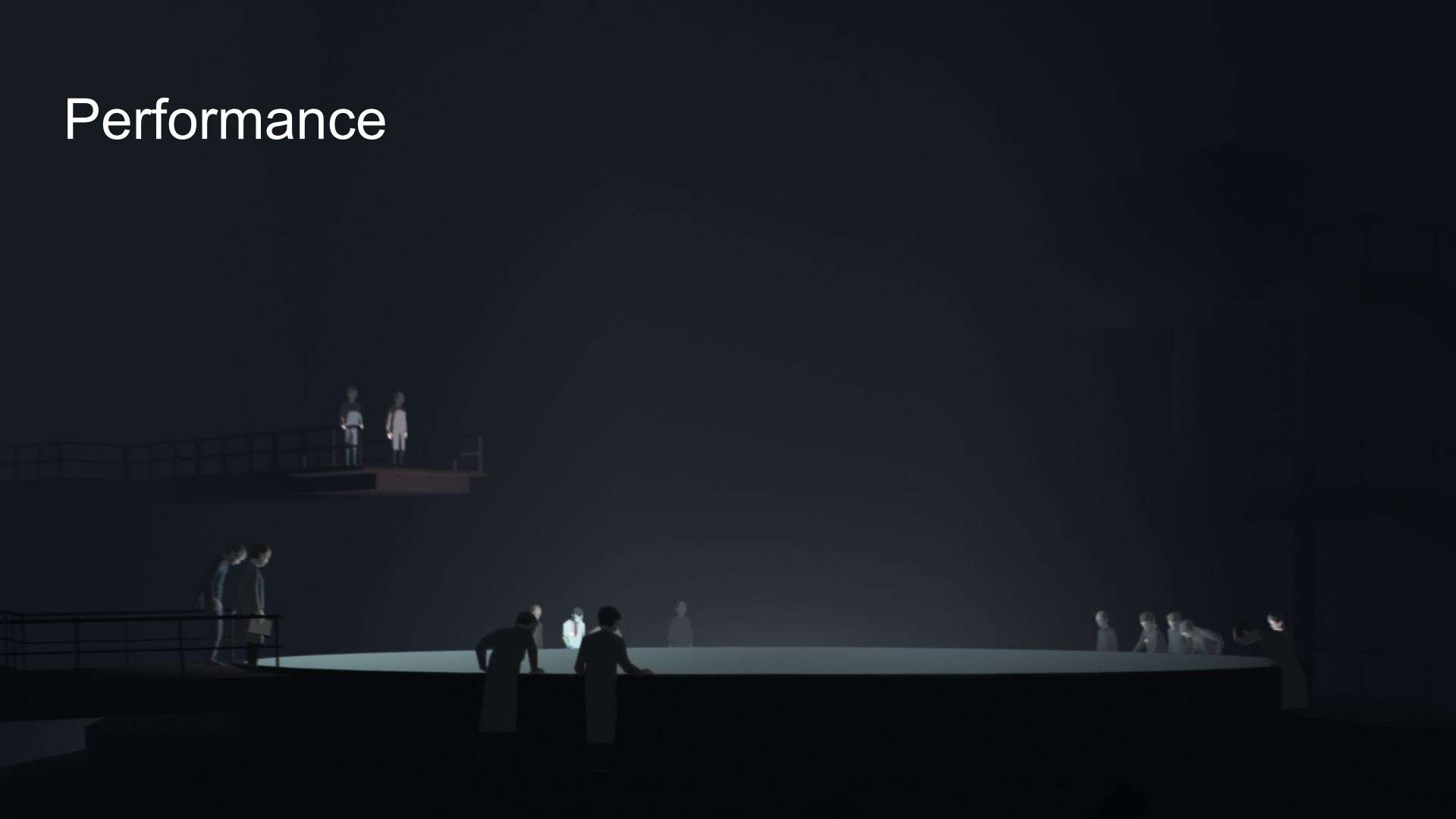
Wwise updates are paused during scene change:

- Retains audio state during scene change
- Wwise commands accumulated during load
- All commands are executed at once when scene change is complete

Image credit: www.artsjournal.com

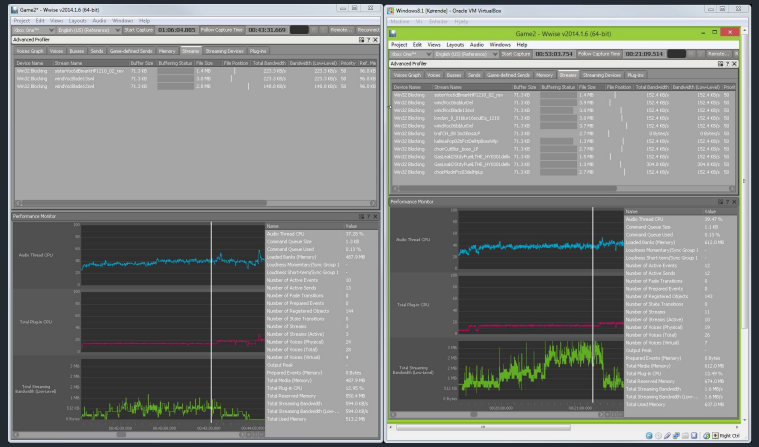
Creates the illusion of no time passing during scene change

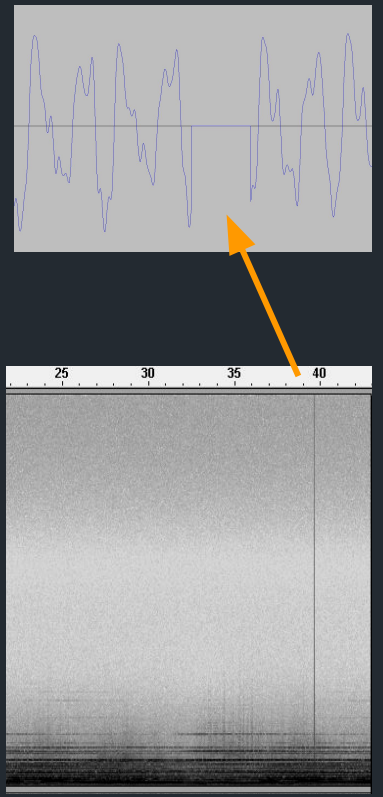
Performance



Debugging INSIDE Audio

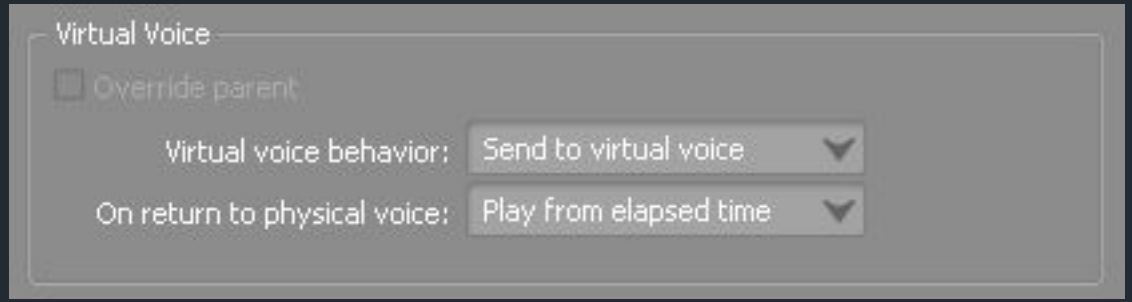
- 2D gameplay: predictable, testable performance
- Profile entire playthrough and analyze the data
- Record audio digitally and inspect for glitches





CPU Performance

- Virtual Voices are your friends
- Inaudible sounds are still updated, but not mixed



Fixing glitches

- Scene Change requires large command queue (2 MB)
- Caused glitches with standard 512 sample audio buffer
- Audio buffer size adjusted to 1024 samples

```
initSettings.uCommandQueueSize = 2048 * 1024;
initSettings.uNumSamplesPerFrame = 1024;
```

Wwise-Unity Plugin Modifications

- Wwise API wrapped in C#
- General Unity performance concerns:
  - Unity API calls are slow
  - Runtime allocations cause CPU spikes on garbage collection

C# API Optimization

Unity API calls are slow.

We removed AkGameObj check from all Wwise API calls (except in editor):

```
#if UNITY_EDITOR
if (in_gameObjectID.activeInHierarchy) {
    if (in_gameObjectID.GetComponent<AkGameObj>() == null) {
        in_gameObjectID.AddComponent<AkGameObj>();
        Debug.LogError("Missing AkGameObj", in_gameObjectID); // no AkGameObj = error
    }
}
#endif
```

Avoiding Callback String Allocation

User Cues and Markers are sent from Wwise as hashes instead of constantly allocating C# strings:

```
AkCallbackSerializer.cpp (Wwise-Unity plugin code):
    const char *s = pCueInfo->pszUserCueName;
    akCallbackInfo.cueHash = AK::SoundEngine::GetIDFromString(s);
```

And recognized based on hash in Unity:

```
Custom callback handler (Unity C# code):
    string cueNameToWaitFor = "my cue";
    // if(info.pszUserCueName == cueNameToWaitFor) { ... }
    uint hash = AkSoundEngine.GetIDFromString(cueNameToWaitFor);
    if(hash == akCallbackInfo.cueHash) { ... }
```

Performance Summary

- Virtual Voices
- Glitches caused by large command queues:
  - Audio buffer size 1024
- Wwise-Unity plugin optimized:
  - Removed slow Unity API calls
  - User cues and markers are hashed to avoid allocations

Questions?

playdead.com

Slides are here:

schmid.dk/talks/2016-06-16-wwise

Martin Stig Andersen

martinstigandersen.dk

E-mail: martin@martinstigandersen.dk

Jakob Schmid

Twitter: @jakobschmid

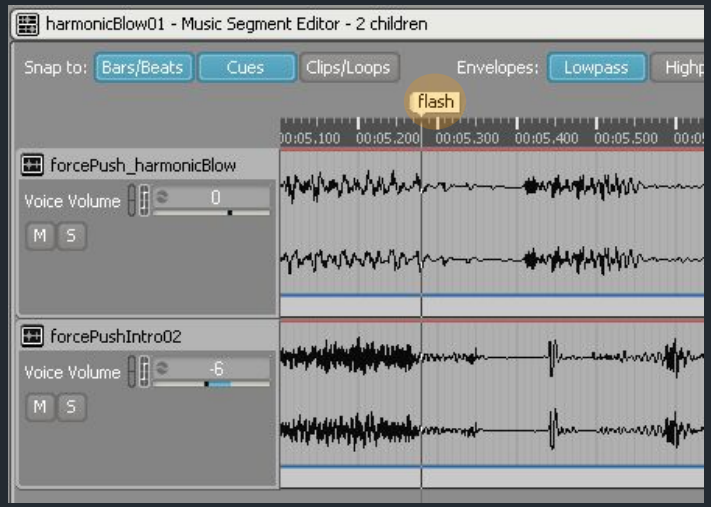
E-mail: jakob@schmid.dk

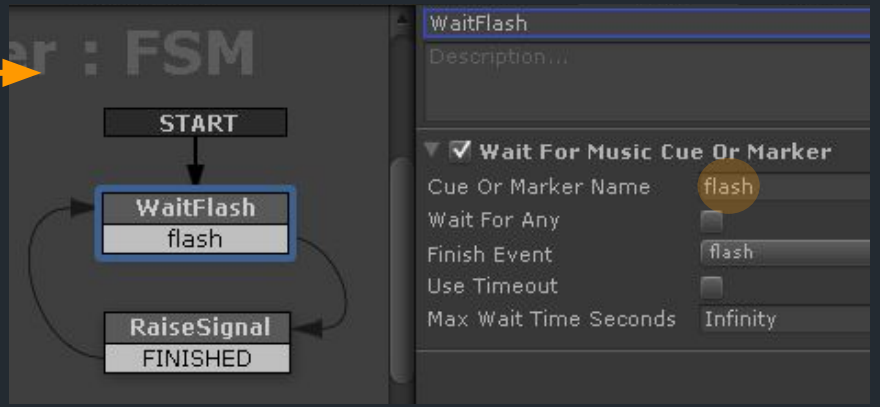
game140.com

BONUS SLIDES

Audio-driven Gameplay: User Cues

- Named User Cues can be placed in music segments
- Received in Unity as callback when AkCallbackManager.PostCallbacks is called (normally, the next frame after cue occurred).

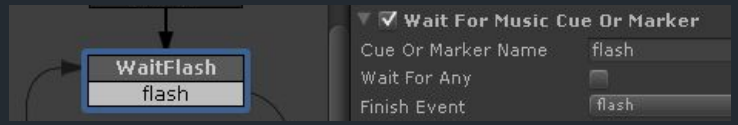




Receiving User Cues

- Receiving User Cues:

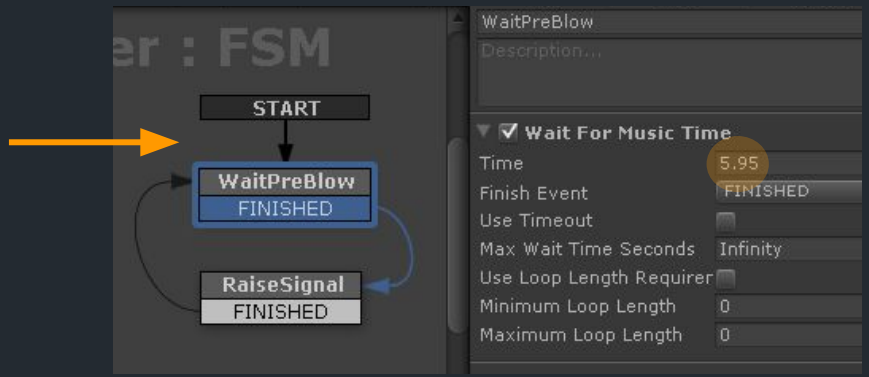
```
void HandleCallback(object akCookie, AkCallbackType akType, object akInfo) {
    if(akType == AkCallbackType.AK_MusicSyncUserCue) {
        AkCallbackManager.AkMusicSyncCallbackInfo info =
                        (AkCallbackManager.AkMusicSyncCallbackInfo) akInfo;
        if(info.pszUserCueName == cueNameToWaitFor) {
            // do stuff
        }
    }
}
AkCallbackType callbackTypes = AkCallbackType.AK_MusicSyncUserCue;
AkSoundEngine.PostEvent(eventID, gameObject, callbackTypes, HandleCallback, cookie);
```



Getting Music Time

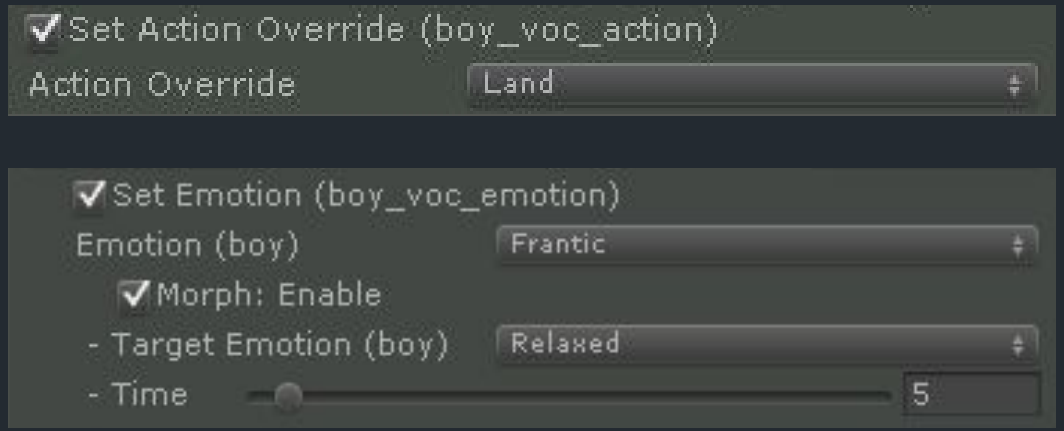
- The game can also get music time information directly from Wwise:

```
AkCallbackType flags = AkCallbackType.AK_EnableGetMusicPlayPosition;
uint id = AkSoundEngine.PostEvent(eventID, gameObject, (uint)flags);
AkSegmentInfo info = new AkSegmentInfo();
AkSoundEngine.GetPlayingSegmentInfo(id, info, doExtrapolation);
float musicTime = info.iCurrentPosition * 0.001f;
```



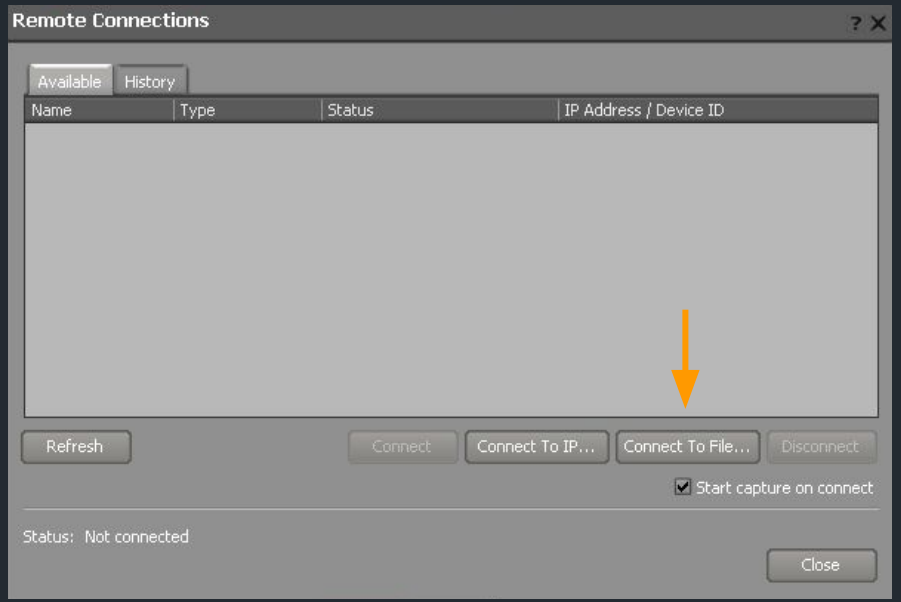
Voice Action and Emotion Override

- Action is normally determined automatically from animation
- Action and emotion can be overriden in Voice Configuration
- Enables defining voice reactions in custom situations
- Morphing allows automatically changing emotion after a specified time



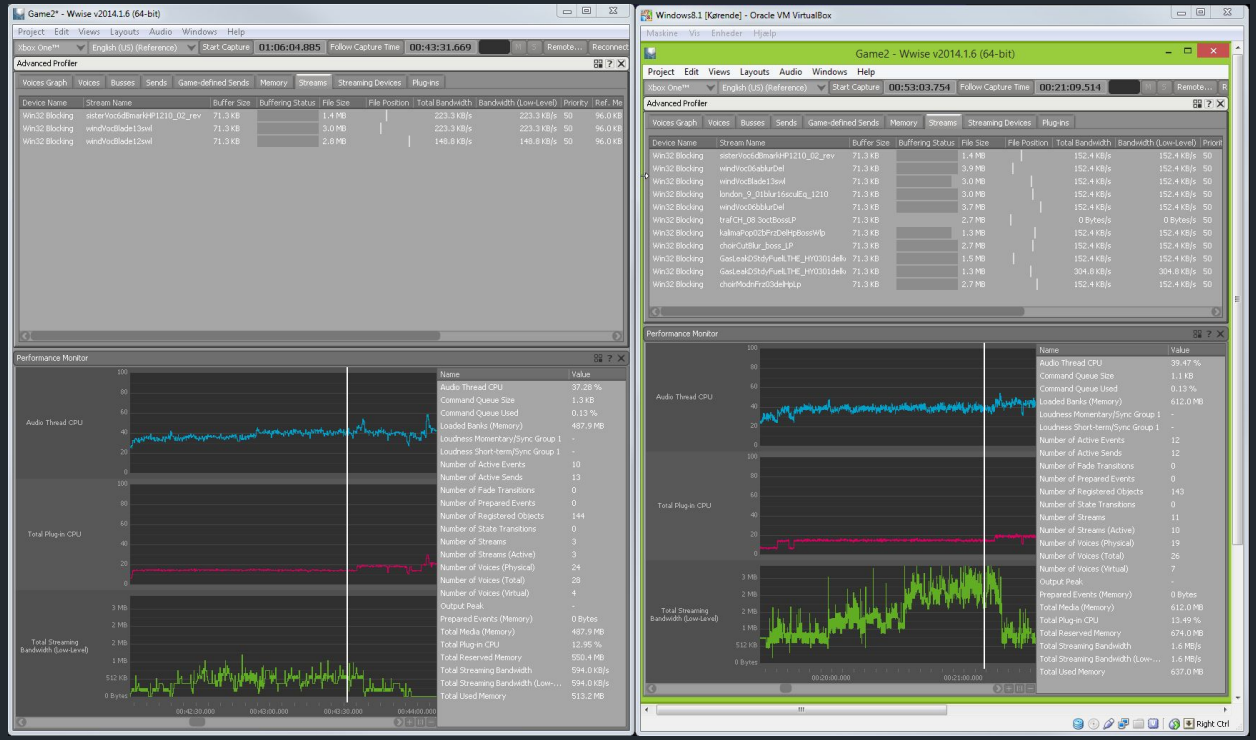
Profiling Tips: Recording

- Record playthrough and use Connect To File
- Record large profiler sessions (~ 2 hours) by setting Capture Log Max Memory Usage to 3999 MB



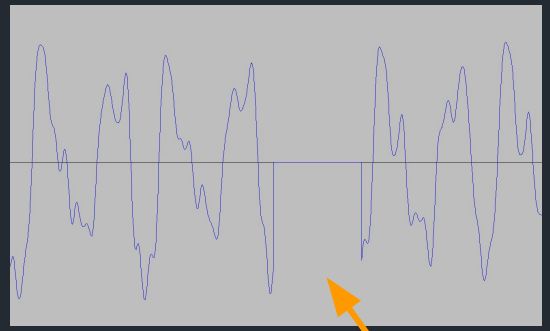
Profiling Tips: Comparing

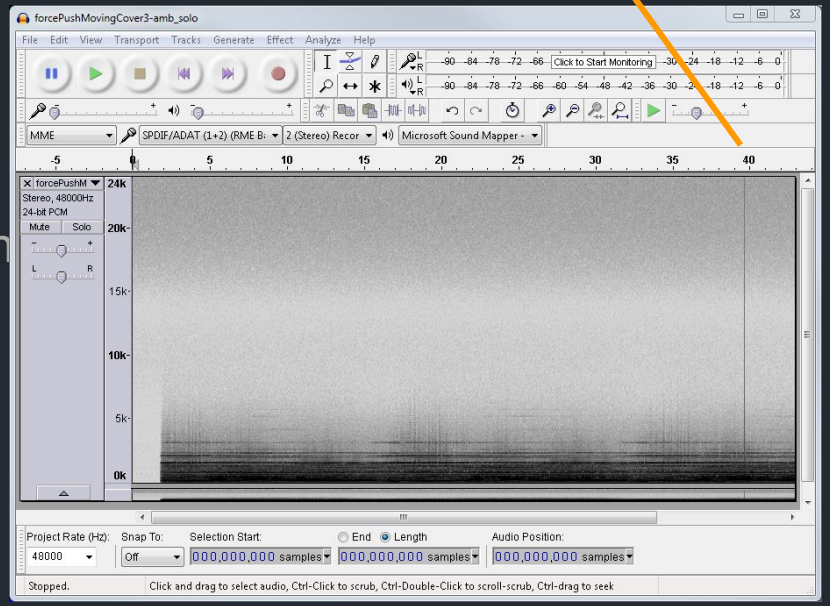
Wwise is single-instance. Compare two profile sessions by running another instance in a virtual machine (e.g. VirtualBox)



Debugging Tips: Recording Audio

- Tiny audio glitches
- 256 samples of zeros when command queue was large
- No errors in Wwise profiler, only detectable using audio recordings
- Record console output using S/PDIF
- Barely audible glitches are easy to spot in spectrogram





CPU Performance

- Wwise runs on CPU core 5, Unity worker threads run on core 2-4:

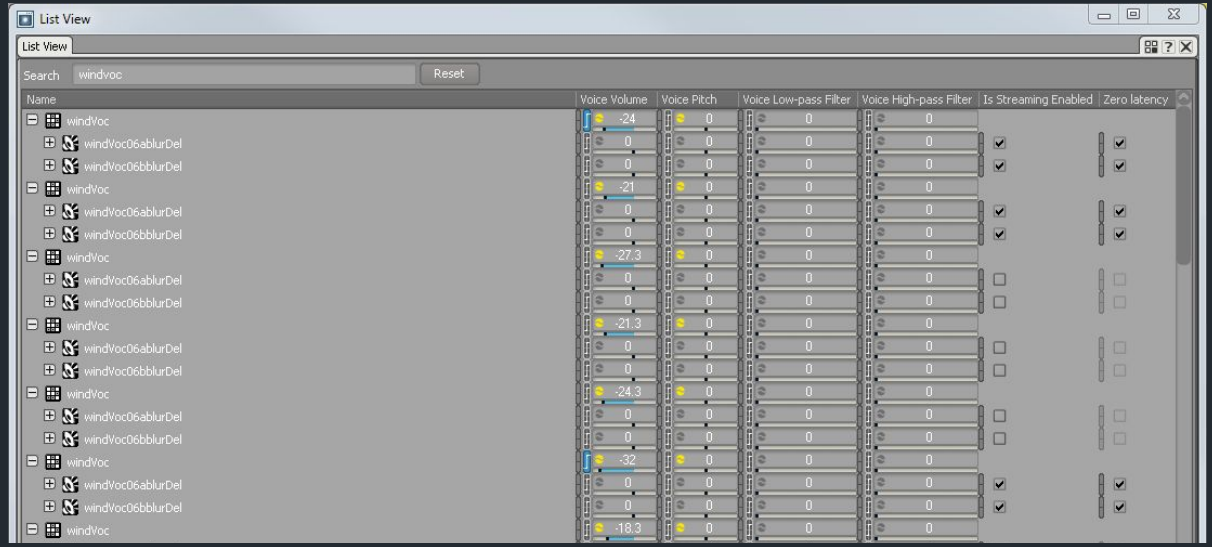
```
platformSettings.threadLEngine.dwAffinityMask = (1 << 5)
```

Avoiding Callback Allocations

Single instance of callback data structure is reused for every callback.

I/O Performance

- Look for streams that are used a lot throughout the game, and convert them to non-streams. Searching and opening in List View is useful for this.
- We used PCM for music, and all other sounds vorbis (quality 10)



Teaser and Bios

Teaser

A 5-year collaboration between sound designer Martin Stig Andersen and programmer Jakob Schmid on INSIDE, Playdead's follow-up to award-winning game LIMBO, has led to an uncompromising audio implementation, unique in its design choices and level of detail. This talk focuses on the design and implementation of foley and voice for the main character of INSIDE, and the seemless handling of the death - respawn cycle.
The talk will cover both the Wwise setup and game engine tools used for audio features, and show how Wwise can be used as a compositional tool.
Finally, performance results and considerations will be discussed in relation to the topics covered.

Bios

Martin Stig Andersen (b. 1973) has a background as a composer in the fields of acousmatic music, sound installations, electroacoustic performance, and video art, earning several international distinctions and awards. In 2009 he joined Playdead where he created the audio for the video game LIMBO which won Outstanding Achievement in Sound Design at the Interactive Achievement Awards, the IndieCade Sound Award 2010, and was nominated for best audio at the BAFTA Video Games Awards 2011. In the years following the release of LIMBO, Martin Stig Andersen has been working on Playdead’s next title, INSIDE, which is to be released 2016.
Martin Stig Andersen graduated as a composer from The Royal Academy of Music in Aarhus, Denmark in 2003, after which he went on to study electroacoustic composition under Professor Denis Smalley at City University, London.

Jakob Schmid (b. 1976) graduated as a computer scientist specialized in game development and a minor degree in music science from the University of Aalborg, Denmark in 2007. Working in the danish video game industry since 2008, Schmid has specialized in developing novel dynamic audio systems for video games.
He created the music and sound for Jeppe Carlsen’s rhythm platformer '140', which went on to win the 'Excellence in Audio' award at IGF 2013, with honorable mention in Technical Excellence, as well as the 'Sound of the Year' award at SpilPrisen 2014.
Schmid joined Playdead in 2011 as audio programmer, mainly working on the studio's next title, 'INSIDE'.