The Role of Audio in Games

Engagement

- **Entertains** the player
  - Music/Soundtrack

- Enhances the **realism**
  - Sound effects

- Establishes **atmosphere**
  - Ambient sounds
The Role of Audio in Games

Feedback

- **Indicate** off-screen action
  - Indicate player should move
- **Highlight** on-screen action
  - Call attention to an NPC
- **Increase** reaction time
  - Players react to sound faster
History of Sound in Games

Basic Sounds

- Arcade games
- Early handhelds
- Early consoles
Early Sounds: *Wizard of Wor*
History of Sound in Games

- Arcade games
- Early handhelds
- Early consoles

Basic Sounds

- Starts w/ MIDI
- 5th generation
  (Playstation)
- Early PCs

Sample = pre-recorded audio
History of Sound in Games

Basic Sounds

Recorded Sound Samples

Some Variability of Samples

- Arcade games
- Early handhelds
- Early consoles

- Starts w/ MIDI
- 5th generation (Playstation)
- Early PCs

- Sample selection
- Volume
- Pitch
- Stereo pan
History of Sound in Games

- Basic Sounds
- Recorded Sound Samples
- Some Variability of Samples
- More Variability of Samples

- Arcade games
- Early handhelds
- Early consoles
- Starts w/ MIDI
- 5th generation (Playstation)
- Early PCs
- Sample selection
- Volume
- Pitch
- Stereo pan
- Multiple samples
- Reverb models
- Sound filters
- Surround sound
History of Sound in Games

- Basic Sounds
- Recorded Sound Samples
  - Starts w/ MIDI
  - 5th generation (Playstation)
  - Early PCs
- Some Variability of Samples
  - Sample selection
  - Volume
  - Pitch
  - Stereo pan
- More Variability of Samples
  - Multiple samples
  - Reverb models
  - Sound filters
  - Surround sound

- Arcade games
- Early handhelds
- Early consoles

LibGDX is here
CUGL is here
LibGDX
CUGL
The Technical Challenges

- Sound **formats** are not (really) cross-platform
  - It is not as easy as choosing MP3
  - Different platforms favor different formats

- Sound playback **APIs** are not standardized
  - LibGDX & CUGL are layered over many APIs
  - Behavior is not the same on all platforms

- Sound playback crosses **frame boundaries**
  - Mixing sound with animation has challenges
## File Format vs Data Format

<table>
<thead>
<tr>
<th><strong>File Format</strong></th>
<th><strong>Data Format</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• The data storage format</td>
<td>• The actual audio encoding</td>
</tr>
<tr>
<td>• Has data other than audio</td>
<td>• Basic audio codec</td>
</tr>
<tr>
<td>• Many have many encodings</td>
<td>• Bit rate (# of bits/unit time)</td>
</tr>
<tr>
<td>• .caf holds MP3 and PCM</td>
<td>• Sample rate (digitizes an analog signal)</td>
</tr>
<tr>
<td>• <strong>Examples:</strong></td>
<td>• <strong>Examples:</strong></td>
</tr>
<tr>
<td>• .mp3, .wav, .aiff</td>
<td>• MP3, Linear PCM</td>
</tr>
<tr>
<td>• .aac, .mp4, .m4a (Apple)</td>
<td>• AAC, HE-AAC, ALAC</td>
</tr>
<tr>
<td>• .flac, .ogg (Linux)</td>
<td>• FLAC, Vorbis</td>
</tr>
</tbody>
</table>
# Game Audio Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>File Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear PCM</td>
<td>Completely uncompressed sound</td>
<td>.wav, .aiff</td>
</tr>
<tr>
<td>MP3</td>
<td>A popular compressed, lossy codec</td>
<td>.mp3, .wav</td>
</tr>
<tr>
<td>Vorbis</td>
<td>Xiph.org’s alternative to MP3</td>
<td>.ogg</td>
</tr>
<tr>
<td>FLAC</td>
<td>Xiph.org’s compressed, lossless codec</td>
<td>.flac, .ogg</td>
</tr>
<tr>
<td>MIDI</td>
<td><strong>NOT SOUND</strong>; Data for an instrument</td>
<td>.midi</td>
</tr>
<tr>
<td>(HE-)AAC</td>
<td>A lossy codec, Apple’s MP3 alternative</td>
<td>.aac, .mp4, .m4a</td>
</tr>
<tr>
<td>ALAC</td>
<td>Apple’s lossless codec (but compressed)</td>
<td>.alac, .mp4, .m4a</td>
</tr>
</tbody>
</table>

MP3 historically avoided due to patent issues
# Game Audio Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>File Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear PCM</td>
<td>Completely uncompressed sound.</td>
<td>.wav, .aiff</td>
</tr>
<tr>
<td>MP3</td>
<td>A popular compressed, lossy codec.</td>
<td>.mp3, .wav</td>
</tr>
<tr>
<td>Vorbis</td>
<td>Xiph.org’s alternative to MP3</td>
<td>.ogg</td>
</tr>
<tr>
<td>FLAC</td>
<td>Xiph.org’s compressed, lossless codec</td>
<td>.flac, .ogg</td>
</tr>
<tr>
<td>MIDI</td>
<td><strong>NOT SOUND</strong>; Data for an instrument</td>
<td>.midi</td>
</tr>
<tr>
<td>(HE-)AAC</td>
<td>A lossy codec, Apple’s MP3 alternative</td>
<td>.aac, .mp4, .m4a</td>
</tr>
<tr>
<td>ALAC</td>
<td>Apple’s lossless codec (but compressed)</td>
<td>.alac, .mp4, .m4a</td>
</tr>
</tbody>
</table>

MP3 historically avoided due to patent issues
# Game Audio Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>File Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear PCM</td>
<td>Completely uncompressed sound</td>
<td>.wav, .aiff</td>
</tr>
<tr>
<td>MP3</td>
<td>A popular compressed, lossy codec</td>
<td>.mp3, .wav</td>
</tr>
<tr>
<td>Vorbis</td>
<td>Xiph.org’s alternative to MP3</td>
<td>.ogg</td>
</tr>
<tr>
<td>FLAC</td>
<td>Xiph.org’s compressed, lossless codec</td>
<td>.flac, .ogg</td>
</tr>
<tr>
<td>MIDI</td>
<td><strong>NOT SOUND</strong>; Data for an instrument</td>
<td>.midi</td>
</tr>
<tr>
<td>(HE-)AAC</td>
<td>A lossy codec, Apple’s MP3 alternative</td>
<td>.aac, .mp4, .m4a</td>
</tr>
<tr>
<td>ALAC</td>
<td>Apple’s lossless codec (but compressed)</td>
<td>.alac, .mp4, .m4a</td>
</tr>
</tbody>
</table>

MP3 historically avoided due to patent issues
Which Formats Should You Choose?

- **Question 1:** Streaming or no streaming?
  - Audio gets large fast; music often streamed
  - But streaming creates overhead; bad for sound fx
  - Few engines support WAV streams (LibGDX & CUGL do)

- **Question 2:** Lossy or lossless compression?
  - Music can be lossy; sound fx not so much
  - Only FLAC and WAV are standard lossless

- **Question 3:** How many channels (speakers) needed?
  - Standard MP3 support is *stereo only*
  - Others support many channels (e.g. 7.1 surround)
Which Formats Should You Choose?

• **Question 1:** Streaming or no streaming?
  - Audio gets large fast; music often streamed
  - But streaming creates overhead; bad for sound fx
  - Few engines support WAV streaming (CUGL does)

• **Question 2:** Lossy or lossless compression?
  - Music can be lossy; sound fx not so much
  - Only FLAC and WAV are standard lossless

• **Question 3:** How many channels (speakers) needed?
  - Standard MP3 support is *stereo only*
  - Others support many channels (e.g. 7.1 surround)

**Sound FX:** Linear PCM/WAV

**Music:** OGG Vorbis
Linear PCM Format

- Sound data is an array of **sample** values

| 0.5 | 0.2 | -0.1 | 0.3 | -0.5 | 0.0 | -0.2 | -0.2 | 0.0 | -0.6 | 0.2 | -0.3 | 0.4 | 0.0 |

- A sample is an **amplitude** of a sound wave

- Values are normalized -1.0 to 1.0 (so they are floats)
Linear PCM Format

- Sound data is an array of **sample** values

| 0.5 | 0.2 | -0.1 | 0.3 | -0.5 | 0.0 | -0.2 | -0.2 | 0.0 | -0.6 | 0.2 | -0.3 | 0.4 | 0.0 |

- A sample is an **amplitude** of a sound wave

- Sometimes encoded as shorts or bytes MIN to MAX
Linear PCM Format

- Sound data is an array of sample values

| 0.5 | 0.2 | -0.1 | 0.3 | -0.5 | 0.0 | -0.2 | -0.2 | 0.0 | -0.6 | 0.2 | -0.3 | 0.4 | 0.0 |

- Magnitude of the amplitude is the volume
  - 0 is lowest volume (silence)
  - 1 is maximum volume of sound card
  - Multiply by number 0 to 1 to change global volume
Linear PCM Format

- Sound data is an array of **sample** values

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>-0.1</td>
<td>0.3</td>
<td>-0.5</td>
<td>0.0</td>
<td>-0.2</td>
<td>-0.2</td>
<td>0.0</td>
<td>-0.6</td>
<td>0.2</td>
<td>-0.3</td>
<td>0.4</td>
</tr>
</tbody>
</table>

- Magnitude of the amplitude is the volume
  - 0 is lowest volume (silence)
  - 1 is maximum volume of sound card
  - Multiply by number 0 to 1 to change global volume
Linear PCM Format

- Samples are organized into (interleaved) **channels**

<table>
<thead>
<tr>
<th>0.5</th>
<th>0.2</th>
<th>-0.1</th>
<th>0.3</th>
<th>-0.5</th>
<th>0.0</th>
<th>-0.2</th>
<th>-0.2</th>
<th>0.0</th>
<th>-0.6</th>
<th>0.2</th>
<th>-0.3</th>
<th>0.4</th>
<th>0.0</th>
</tr>
</thead>
</table>

- Each channel is essentially a **speaker**
  - Mono sound has one channel
  - Stereo sound has two channels
  - 7.1 surround sound is **eight** channels

- A **frame** is set of simultaneous samples
  - Each sample is in a separate frame
Linear PCM Format

- The sample rate is frames per second

- **Example**: 0.5 seconds of stereo at 44.1 kHZ
  - \(0.5 \text{ s} \times 44100 \text{ f/s} = 22050 \text{ frames}\)
  - \(2 \text{ samples/frame} \times 22050 \text{ frames} = 44100 \text{ samples}\)
  - \(4 \text{ bytes/sample} \times 44100 \text{ samples} = 176.4 \text{ kBytes}\)

- 1 minute of stereo CD sound is 21 MB!
Playing Sound Directly

Game Loop → PCM data buffer → Sound Card
Playing Sound Directly

Game Loop

Write PCM chunk to buffer

PCM data buffer

Sound Card

Speakers
Direct Sound in LibGDX: AudioDevice

- /**
  * Writes the array of float PCM samples to the audio device.
  *
  * This method blocks until they have been processed.
  */
  
  void writeSamples(float[] samples, int offset, int numSamples)

- /**
  * Writes array of 16-bit signed PCM samples to the audio device.
  *
  * This method blocks until they have been processed.
  */
  
  void writeSamples(short[] samples, int offset, int numSamples)
Direct Sound in LibGDX: AudioDevice

- /**
  * Writes the array of float PCM samples to the audio device.
  *
  * This method blocks until they have been processed.
  */
  
  void writeSamples(float[] samples, int offset, int numSamples)

- /**
  * Writes array of 16-bit signed PCM samples to the audio device.
  *
  * This method blocks until they have been processed.
  */
  
  void writeSamples(short[] samples, int offset, int numSamples)
- Buffer is really a *queue*
  - Output from queue front
  - Playback writes to end
  - Creates a *playback delay*

- **Latency**: amount of delay
  - Some latency must exist
  - Okay if latency $\leq$ framerate
  - Android latency is $\sim 90$ ms!

- Buffering is a necessary evil
  - Keeps playback smooth
  - Allows real-time *effects*
Playing Sound Directly

Choice of buffer size is important!

- **Too large**: long latency until next sound plays
- **Too small**: buffers swap too fast, causing audible pops
Playing Sound Directly

- Choice of buffer size is important!
  - Too large: long latency until next sound plays
  - Too small: buffers swap too fast, causing audible pops

- Windows: 528 bytes (even if you ask for larger)
- MacOS, iOS: 512-1024 bytes (hardware varies)
- Android: 2048-4096 bytes (hardware varies)
How Streaming Works

- All sound cards **only** play PCM data
  - Other files (MP3 etc.) are decoded into PCM data
  - But the data is *paged-in* like memory in an OS

- Why LibGDX/CUGL can stream WAV files too!

![Diagram of streaming process](image)
How Streaming Works

- **Sound**: Sound asset that is *preloaded* as full PCM
- **Music**: Sound asset that is *streamed* as PCM pages
Handling Multiple Sounds

pcm data
pcm data
pcm data
pcm data
pcm data

Literally!

Sound Card
Handling Multiple Sounds

- Can create values outside of -1 to 1
  - This causes clipping/distortion
  - Common if many simultaneous sounds
- Audio engineer must balance properly

PCM Data

Literally!
Why is Mixing Hard?

- Playback may include **multiple sounds**
  - Sounds may play simultaneously (offset)
  - Simultaneous sounds may be same asset
  - **Asset** (source) vs. **Instance** (playback)

- Playback crosses **frame boundaries**
  - It may span multiple animation frames
  - Need to know when it stops playing
  - May need to stop (or pause) it early
We Want Something Simpler!

- Want ability to **play** and **track** sounds
  - Functions to load sound into card buffer
  - Functions to detect if sound has finished

- Want ability to **modify** active sounds
  - Functions for volume and pitch adjustment
  - Functions for stereo panning (e.g. left/right channels)
  - Functions to pause, resume, or loop sound

- Want ability to **mix** sounds together
  - Functions to add together sound data quickly
  - Background process for dynamic volume adjustment
We Want Something Simpler!

• Want ability to **play** and **track** sounds
  • Functions to load sound into card buffer
  • Functions to detect if sound has finished

• Want ability to **modify** active sound
  • Functions to adjust volume and pitch
  • Functions for stereo panning (e.g., left/right channels)
  • Functions to pause, resume, or loop sound

• Want ability to **mix** sounds together
  • Functions to add together sound data quickly
  • Background process for dynamic volume adjustment

This is the purpose of a **sound engine**
Cross-Platform Sound Engines

- OpenAL
  - Created in 2000 by Loki Software for Linux
  - Was an attempt to make a sound standard
  - Loki went under; last stable release in 2005
  - Apple supported, but HARD deprecated in iOS 9

- FMOD/WWISE
  - Industry standard for game development
  - Mobile support is possible but not easy
  - Not free; but no cost for low-volume sales
Proprietary Sound Engines

• Apple AVFoundation
  • API to support modern sound processing
  • Mainly designed for music/audio creation apps
  • But very useful for games and playback apps

• OpenSL ES
  • Directed by Khronos Group (OpenGL)
  • Substantially less advanced than other APIs
  • Really only has support in Android space
  • Google is deprecating in 2022
Proprietary Sound Engines

- Apple AVFoundation
  - API to support modern sound processing
  - Mainly designed for music/audio creation apps
  - Directed by Khronos Group (OpenGL)
  - Substantially less advanced than other APIs
  - Really only has support in Android space
  - Google is deprecating in 2022

- OpenSL ES
  - And many competing 3rd party solutions
What Does LibGDX Use?

- LibGDX support is actually OS specific
  - Recall the core/desktop package distinction
  - Because LibGDX supports mobile and computer

- Different platforms have different backends
  - All desktop platforms are built on OpenAL
  - The android backend uses android.media

- Needs an abstraction bringing all together
  - This is done with the Audio interface
The LibGDX Audio Interface

- LibGDX provides an audio **singleton**
  - One global object referencing audio device
  - Access via `GDX.audio` (static field of `GDX`)
  - Same principle as `System.out`

- Singleton implements the **Audio** interface
  - Use it to access `AudioDevice` for direct sound
  - Use it to allocate new `Sound`, `Music` instances
  - But do not use it for much sound manipulation
The LibGDX Audio Interface

- LibGDX provides an audio **singleton**
  - One global object referencing audio device
  - Access via GDX.audio (static field of GDX)
  - Same principle as System.out

- Singleton implements the **Audio** interface
  - Use it to access the AudioDevice
  - Use it to allocate new Sound, Music instances
  - But do not use it for much sound manipulation

Essentially a factory for other classes
# The LibGDX Sound Classes

<table>
<thead>
<tr>
<th>Sound</th>
<th>Music</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Primary method is play()</strong></td>
<td><strong>Primary method is play()</strong></td>
</tr>
<tr>
<td>• Returns a long integer</td>
<td>• This is a <strong>void method</strong></td>
</tr>
<tr>
<td>• Represents sound <em>instance</em></td>
<td>• Only allows <strong>one instance</strong></td>
</tr>
<tr>
<td>• <code>loop()</code> is a separate method</td>
<td>• <code>loop</code> is an attribute of music</td>
</tr>
<tr>
<td><strong>Has no public constructor</strong></td>
<td><strong>Has no public constructor</strong></td>
</tr>
<tr>
<td>• Use <code>Audio.newSound(f)</code></td>
<td>• Use <code>Audio.newMusic(f)</code></td>
</tr>
<tr>
<td>• Audio can cache/preload</td>
<td>• Audio can cache the file</td>
</tr>
<tr>
<td><strong>Must dispose when done</strong></td>
<td><strong>Must dispose when done</strong></td>
</tr>
</tbody>
</table>
Playing a Sound

- Playback may include **multiple sounds**
  - Sounds may play simultaneously (offset)
  - Simultaneous sounds may be same asset
  - **Asset (source) vs. Instance (playback)**

- Playback crosses **frame boundaries**
  - It may span multiple animation frames
  - Need to know when it stops playing
  - May need to stop (or pause) it early
Playing a Sound

• Playback may include **multiple sounds**
  • Sounds may play simultaneously (offset)
  • Simultaneous sounds may be same asset

• Requires an understanding of OpenAL

• Playback crosses frame boundaries
  • It may span multiple animation frames
  • Need to know when it stops playing
  • May need to stop (or pause) it early
Classic Model: Playback Slots

Engine has fixed number of slots (historically 24)

Mixer
Classic Model: Playback Slots

- Classic Model: Playback Slots
- Engine has fixed number of slots (historically 24)
- Load sound into a slot to play it
Classic Model: Playback Slots

- Engine has fixed number of slots (historically 24)
- Load sound into a slot to play it
- Queue to follow after
Playing a Sound with Slots

- **Request** a playback slot for your asset
  - If none is available, sound fails to play
  - Otherwise, it gives you an id for the slot

- **Load** asset into the slot (but might stream)

- **Play** the playback slot
  - Playing is a property of the slot, not asset
  - Playback slot has other properties, like volume

- **Release** the slot when the sound is done
  - This is usually done automatically
Application Design

Slot
Slot
Slot
...
Slot

Sound

Need to remember the slot id

Volume is property of a slot!
The Sound API

- /**
  *  @return channel id for sound playback
  *
  *  If no channel is available, returns -1
  *  @param volume  The sound volume
  *  @param pitch   The pitch multiplier (>1 faster, <1 slower)
  *  @param pan     The speaker pan (-1 full left, 1 full right)
  */
  
  public long play(float volume, float pitch, float pan);

- public void stop(long audioID);

- public void resume(long audioID);

- public void setLooping(long audioID, boolean loop);

- Public void setVolume(long audioID, float volume);
The Sound API

- /**
   * @return channel id for sound playback
   *
   * If no channel is available, returns -1
   * @param volume  The sound volume
   * @param pitch     The pitch multiplier (>1 faster, <1 slower)
   * @param pan       The speaker pan (-1 full left, 1 full right)
   */
   public long play(float volume, float pitch, float pan);

- public void stop(long audioID);
- public void resume(long audioID);
- public void setLooping(long audioID, boolean loop);
- public void setVolume(long audioID, float volume);

Need to remember channel id

Returns available channel id
Why This is Undesirable

- Tightly couples architecture to sound engine
  - All controllers need to know this playback slot id
  - Playback must communicate id to all controllers

- Instances usually have a *semantic meaning*
  - **Example**: Torpedo #3, Ship/crate collision
  - Meaning is independent of the slot assigned
  - Would prefer to represent them by this meaning

- **Solution**: Refer to instances by *keys*
Idea: SoundController Class

- A **SoundController** is essentially a hashmap
  - Map strings (keys) to integers (slot ids)
  - Only stores a key when instance is playing

- This class needs to be a **singleton**
  - So we can access this anywhere at all time
  - **Demo:** See the class provided with this lecture

- To work, the map must be **up-to-date** at all times
  - We use this controller to play the sounds
  - And it must be notified when a sound is done
Stopping Sounds

- Would like to know when a sound is finished
  - To free up the slot (if not automatic)
  - To stop any associated animation
  - To start a follow-up sound

- Two main approaches
  - **Polling**: Call an `isPlaying()` method
  - **Callback**: Pass a function when play

- Default LibGDX cannot do *either* of these
Stopping Sounds

• Would like to know when a sound is finished
  • To free up the slot (if not automatic)
  • To stop any associated animation
  • To start a follow-up sound

• Two main approaches
  • **Polling**: Call an isPlaying() method
  • **Callback**: Pass a function when play

• Default LibGDX cannot do *either* of these
Solution: AudioEngine

- You are all making desktop games
  - This means you are always using OpenAL
  - Just need a way to expose OpenAL features
  - This is the purpose of GDIAC audio backend

- Basic interface is AudioEngine
  - Upcast GDX.audio to this interface
  - Now have access to SoundBuffer, MusicBuffer
  - These classes give extra features you need

- Note: AssetDirectory handles this automatically
The **GDIAC** Sound Classes

---

**SoundBuffer**

- Works just like `Sound`
  - Primary method is `play()`
  - Returns a long integer
- But has **playback control**
  - Can poll if still playing
  - Can add listener to monitor
- Exposes **OpenAL features**
  - Elapsed playback time
  - Panning between speakers
  - Sound pitch control

**MusicBuffer**

- Works just like `Music`
  - Primary method is `play()`
  - This is a **void method**
- But has a **playback queue**
  - Can add `AudioSource` to it
  - Provides gapless playback
- Methods **manage the queue**
  - Add or remove music
  - Swap out music at position
  - Skip over current music
Problem with the Slots Model

• All controls are embedded in the slot
  • Example: Volume, looping, play position
  • Restricted to a *predetermined* set of controls

• Modern games want *custom sound-processing*
  • User defined sound filters (low pass, reverb)
  • Advanced equalizer support
  • Support for surround and 3D sound
  • Procedural sound generation
DSP Processing: The Mixer DAG

Source → Effect → Effect → Main Mixer

Source → Effect → Mixer → Effect

Source → Mixer → Effect
Example: UDK Kismet
Example: FMOD
Example: Pure Data
The Slot Model is a Special Case

Calling `play()` assigns an input slot behind the scenes.
The Slot Model is a Special Case

Theoretically input should accept any *audio subgraph*
The Slot Model is a Special Case

Even **OpenAL** cannot do this.
The Slot Model is a Special Case

Even OpenAL cannot do this.
Summary

• Audio design is about creating soundscapes
  • Music, sound effects, and dialogue
  • Combining sounds requires a sound engine

• Cross-platform support is a problem
  • Licensing issues prevent a cross-platform format
  • Very little standardization in sound APIs

• Best engines use digital signal processing (DSP)
  • Mixer graph is a DAG supporting sound effects
  • Unfortunately, we cannot do this in LibGDX