Game Engine Development II

Week 12

Hooman Salamat

Audio Implementation

Sound or music?

- SFML provides two classes for playing audio: sf::Sound and sf::Music. They both provide more or less the same features, the main difference is how they work.

- sf::Sound is a lightweight object that plays loaded audio data from a sf::SoundBuffer. It should be used for small sounds that can fit in memory and should suffer no lag when they are played. Examples are gun shots, foot steps, etc.

- sf::Music doesn't load all the audio data into memory, instead it streams it on the fly from the source file. It is typically used to play compressed music that lasts several minutes, and would otherwise take many seconds to load and eat hundreds of MB in memory.

Loading a sound

- the sound data is not stored directly in sf::Sound but in a separate class named sf::SoundBuffer.
- sf::SoundBuffer class encapsulates the audio data, which is basically an array of 16-bit signed integers (called "audio samples").
- A sample is the amplitude of the sound signal at a given point in time, and an array of samples therefore represents a full sound.
- the sf::Sound/sf::SoundBuffer classes work the same way as sf::Sprite/sf::Texture from the graphics module.

Loading a sound

- You can load a sound buffer from a file on disk with its loadFromFile function:

```
#include <SFML/Audio.hpp>

int main()
{
    sf::SoundBuffer buffer;
    if (!buffer.loadFromFile("Media/Sound/nice_music.wav"))
        return -1;

        return 0;
}
```

Playing a Sound

```
sf::SoundBuffer buffer;
    if
(!buffer.loadFromFile("Media/Sound/Explosion1.w
av"))
        return -1;

    sf::Sound sound;
    sound.setBuffer(buffer);
    sound.setVolume(50.f);
    sound.setLoop(true);
    sound.play();
```

Music and Sound

```
 sf::Music music;
    if
(!music.openFromFile("Media/Music/MissionT
heme.ogg"))
        return -1; // error
    music.setVolume(10.f);
    music.setLoop(true);
    music.play();


    sf::SoundBuffer buffer;
    if
(!buffer.loadFromFile("Media/Sound/Explosi
on1.wav"))
        return -1;

    sf::Sound sound;
    sound.setBuffer(buffer);
    sound.setVolume(50.f);
    sound.setLoop(true);
    sound.play();
```

- Sounds (and music) are played in a separate thread.
- You are free to do whatever you want after calling play() (except destroying the sound or its data, of course), the sound will continue to play until it's finished or explicitly stopped.

playback

- To control playback, the following functions are available:

- play starts or resumes playback
- pause pauses playback
- stop stops playback and rewind
- setPlayingOffset changes the current playing position

```
sound.play();
   // advance to 2 seconds

sound.setPlayingOffset(sf::seconds(2
.f));

   Sleep(5000);

   // pause playback
   sound.pause();

   Sleep(5000);
   // resume playback
   sound.play();

   Sleep(5000);
   // stop playback and rewind
   sound.stop();
```

Setting the Pitch & Volume & Loop

- The pitch is a factor that changes the perceived frequency of the sound: greater than 1 plays the sound at a higher pitch, less than 1 plays the sound at a lower pitch, and 1 leaves it unchanged. Changing the pitch has a side effect: it impacts the playing speed.
- The volume is... the volume. The value ranges from 0 (mute) to 100 (full volume). The default value is 100, which means that you can't make a sound louder than its initial volume.
- The loop attribute controls whether the sound/music automatically loops or not. If it loops, it will restart playing from the beginning when it's finished, again and again until you explicitly call stop. If not set to loop, it will stop automatically when it's finished.

```
sf::SoundBuffer buffer;
    if (
!buffer.loadFromFile("Media
/Music/MissionTheme.ogg")
)
return -1;
sf::Sound sound;
sound.setBuffer(buffer);
sound.setVolume(50.f);
sound.setPitch(1);
sound.setLoop(true);
sound.play();
```

sf::Music is not copyable

- `sf::Music music;`
- `sf::Music anotherMusic = music; // ERROR`

- `void doSomething(sf::Music music) { ... }`

- `sf::Music music;`
- `doSomething(music); // ERROR (the function should take its argument by reference, not by value)`

Load Sound From Samples

- You can also load a sound buffer directly from an array of samples, in the case they originate from another source:

```
sf::SoundBuffer buffer2;
const sf::Int16* samples = buffer.getSamples();
int fileSize = buffer.getSampleCount();

buffer2.loadFromSamples(&samples[0], fileSize, 1,
44100);

sf::Sound sound;
sound.setBuffer(buffer2);
sound.setVolume(50.f);
sound.setLoop(true);
sound.play();
```

Load Music from Memory

```
std::vector<char> fileData;

std::ifstream file("Media/Sound/LaunchMissile.wav");

if (!file) return -1;

// Get the size of the file

file.seekg(0, std::ios::end);
std::streampos length = file.tellg();
file.seekg(0, std::ios::beg);

fileData.resize(length);
file.read(&fileData[0], length);


sf::Music music;


// we start with a music file in memory (imagine that we extracted it from a zip archive)
music.openFromMemory(&fileData[0], fileData.size());


music.setVolume(50.f);
music.setLoop(true);
music.play();
```

sf::Listener

- All the soundsand music in your audio environment will be heard by a single actor : the listener.
- What is output from your speakers is determined by what the listener hears.
- The class which defines the listener's properties is sf::Listener.
- Since the listener is unique in the environment, this class only contains static functions and is not meant to be instantiated.
- First, you can set the listener's position in the scene:
- If you have a 2D world you can just use the same Y value everywhere, usually 0.
- sf::Listener::setPosition(0.f, 0.f, 0.f);

SETTING LISTENER POSITION & DIRECTION

- Here, the listener is oriented along the +X axis.
- This means that, for example, a sound emitted at (15, 0, 5) will be heard from the right speaker.

```
sf::Listener::setDirection(1.f, 0.f, 0.f);
```

- The "up" vector of the listener is set to (0, 1, 0) by default, in other words, the top of the listener's head is pointing towards +Y. You can change the "up" vector if you want.
- It is rarely necessary though.
- This corresponds to the listener tilting their head towards the right(+X).

```
sf::Listener::setUpVector(1.f, 1.f, 0.f);
```

- the listener can adjust the global volume of the scene :

```
sf::Listener::setGlobalVolume(50.f);
```

Sound Position

- the sound is coming from right: `sound.setPosition(2.f, 0.f, -5.f);`

- This position is absolute by default, but it can be relative to the listener if needed.
- This can be useful for sounds emitted by the listener itself (like a gun shot, or foot steps).
- It also has the interesting side-effect of disabling spatialization if you set the position of the audio source to (0, 0, 0).
- Non-spatialized sounds can be required in various situations: GUI sounds (clicks), ambient music, etc.
- `sound.setRelativeToListener(true);`
- You can also set the factor by which audio sources will be attenuated depending on their distance to the listener.

- The minimum distance is the distance under which the sound will be heard at its maximum volume.

```
sound.setMinDistance(5.f);
```

- The attenuation is a multiplicative factor.The greater the attenuation,
- the less it will be heard when the sound moves away from the listener.
- To get a non - attenuated sound, you can use 0.
- On the other hand, using a value like 100 will highly attenuate the sound,
- which means that it will be heard only if very close to the listener.

```
sound.setAttenuation(10.f);
```

Open Audio Library

```
// Initialization
Device = alcOpenDevice(NULL); // select the "preferred device"

if (Device) {
Context = alcCreateContext(Device, NULL);
alcMakeContextCurrent(Context);
}
// Check for EAX 2.0 support
g_bEAX = alIsExtensionPresent("EAX2.0");
// Generate Buffers
alGetError(); // clear error code
alGenBuffers(NUM_BUFFERS, g_Buffers);
if ((error = alGetError()) != AL_NO_ERROR)
{
DisplayALError("alGenBuffers :", error);
return;
}
// Load test.wav
loadWAVFile("test.wav", &format, &data, &size, &freq, &loop);
if ((error = alGetError()) != AL_NO_ERROR)
{
DisplayALError("alutLoadWAVFile test.wav : ", error);
alDeleteBuffers(NUM_BUFFERS, g_Buffers);
return;
}
```

```
// Copy test.wav data into AL Buffer 0
alBufferData(g_Buffers[0], format, data, size, freq);
if ((error = alGetError()) != AL_NO_ERROR)
{
DisplayALError("alBufferData buffer 0 : ", error);
alDeleteBuffers(NUM_BUFFERS, g_Buffers);
return;
}
// Unload test.wav
unloadWAV(format, data, size, freq);
if ((error = alGetError()) != AL_NO_ERROR)
{
DisplayALError("alutUnloadWAV : ", error);
alDeleteBuffers(NUM_BUFFERS, g_Buffers);
return;
}
// Generate Sources
alGenSources(1, source);
if ((error = alGetError()) != AL_NO_ERROR)
{
DisplayALError("alGenSources 1 : ", error);
return;
}
// Attach buffer 0 to source
- 10 - alSourcei(source[0], AL_BUFFER, g_Buffers[0]);
if ((error = alGetError()) != AL_NO_ERROR)
{
DisplayALError("alSourcei AL_BUFFER 0 : ", error);
}
// Exit
Context = alcGetCurrentContext();
Device = alcGetContextsDevice(Context);
alcMakeContextCurrent(NULL);
alcDestroyContext(Context);
alcCloseDevice(Device);
```

Objectives

- Implement different music themes in the background
- Implement sound effects that correspond to game events such as explosions
- Integrate sound effects in the 2D world to convey a feeling of spatial sound

Music Themes

- We want to play background music depending on the state we are currently in
- We have two themes
  - One for the menu
  - One for the game
- To do this, we'll define an `enum`:

```
namespace Music
{
    enum ID
    {
       MenuTheme,
       MissionTheme,
    };
}
```

Music Themes (cont'd.)

- Now we'll create a class that has an interface to play music:

```
class MusicPlayer : private sf::NonCopyable
{
public:
   MusicPlayer();
   void play(Music::ID theme);
   void stop();
   void setPaused(bool paused);
   void setVolume(float volume);
private:
   sf::Music mMusic;
   std::map<Music::ID, std::string> mFilenames;
   float mVolume;
};
```

Music Themes (cont'd.)

- SFML does NOT suppose the MP3 format (gasp)
  - We'll use OGG instead
  - You can use the free Audacity to convert formats
    - Audacity ® | Free, open source, cross-platform audio software for multi-track recording and editing. (audacityteam.org)
    - Now back to our class methods:

```
MusicPlayer::MusicPlayer(): mMusic(), mFilenames(), mVolume(100.f)
{
   mFilenames[Music::MenuTheme] = "Media/Music/MenuTheme.ogg";
   mFilenames[Music::MissionTheme] = "Media/Music/MissionTheme.ogg";
}
```

Loading and Playing

```
void MusicPlayer::play(Music::ID theme)
{
    std::string filename = mFilenames[theme];
    if (!mMusic.openFromFile(filename))
          throw std::runtime_error("Music " + filename + " could not be loaded.");
    mMusic.setVolume(mVolume);
    mMusic.setLoop(true);
    mMusic.play();
}

void MusicPlayer::stop()
{
    mMusic.stop();
}

void MusicPlayer::setPaused(bool paused)
{
    if (paused)
          mMusic.pause();
    else
          mMusic.play();
}
```

Sound Effects

- We also create an enum for sound effects and a typedef for the resource holder of `sf::SoundBuffer`:

```
namespace SoundEffect
{
    enum ID
    {
          AlliedGunfire,
          EnemyGunfire,
          Explosion1,
          Explosion2,
          LaunchMissile,
          CollectPickup,
          Button,
    };
}

typedef ResourceHolder<sf::SoundBuffer, SoundEffect::ID> SoundBufferHolder;
```

Sound Effects (cont'd.)

```
class SoundPlayer : private sf::NonCopyable
{
public:
   SoundPlayer();
   void play(SoundEffect::ID effect);
   void play(SoundEffect::ID effect, sf::Vector2f position);
   void removeStoppedSounds();
   void setListenerPosition(sf::Vector2f position);
   sf::Vector2f getListenerPosition() const;
private:
   SoundBufferHolder mSoundBuffers;
   std::list<sf::Sound> mSounds;
};
```

Sound Effects (cont'd.)

```
SoundPlayer::SoundPlayer(): mSoundBuffers(), mSounds()
{
   mSoundBuffers.load(SoundEffect::AlliedGunfire,
       "Media/Sound/AlliedGunfire.wav");
   mSoundBuffers.load(SoundEffect::EnemyGunfire,
       "Media/Sound/EnemyGunfire.wav");
   ...
}

void SoundPlayer::play(SoundEffect::ID effect)
{
   mSounds.push_back(sf::Sound(mSoundBuffers.get(effect)));
   mSounds.back().play();
}
```

Sound Effects (cont'd.)

```
void SoundPlayer::removeStoppedSounds()
{
   mSounds.remove_if([] (const sf::Sound& s)
   {
       return s.getStatus() == sf::Sound::Stopped;
   });
}
```

GUI Sounds

```
class State
{
public:
   struct Context
   {
        ...
        MusicPlayer* music;
        SoundPlayer* sounds;
   };
};

Button::Button(State::Context context)
: ...
, mSounds(*context.sounds)
{
   ...
}
```

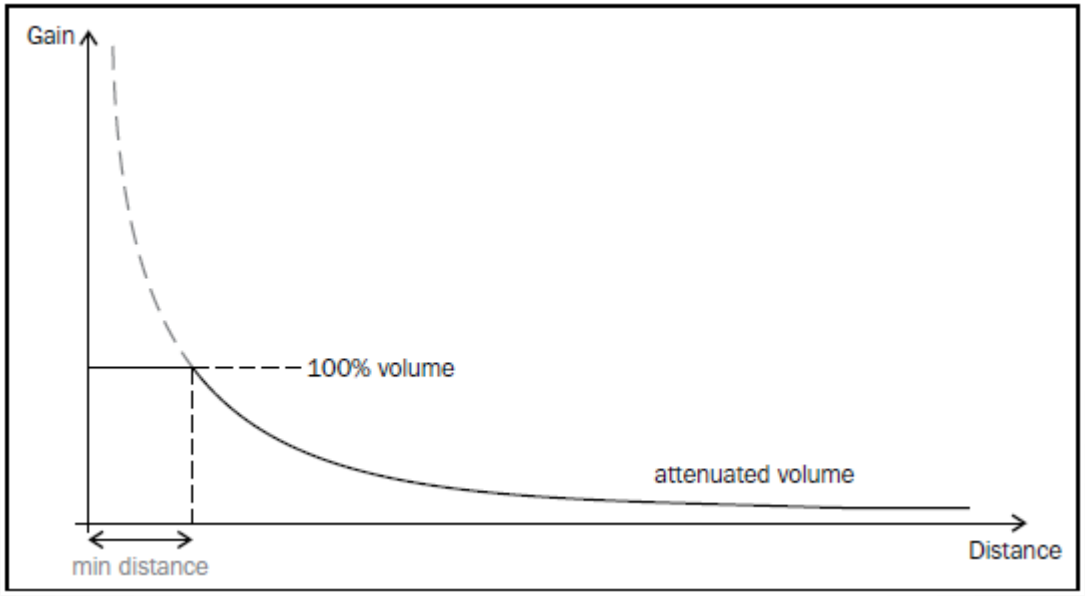
GUI Sounds (cont'd.)

```
void Button::activate()
{
   ...
   mSounds.play(SoundEffect::Button);
}

MenuState::MenuState(StateStack& stack, Context context)
: State(stack, context)
, ...
{
   auto playButton = std::make_shared<GUI::Button>(context);
   auto settingsButton = std::make_shared<GUI::Button>(context);
   auto exitButton = std::make_shared<GUI::Button>(context);
   ...
   context.music->play(Music::MenuTheme);
}
```

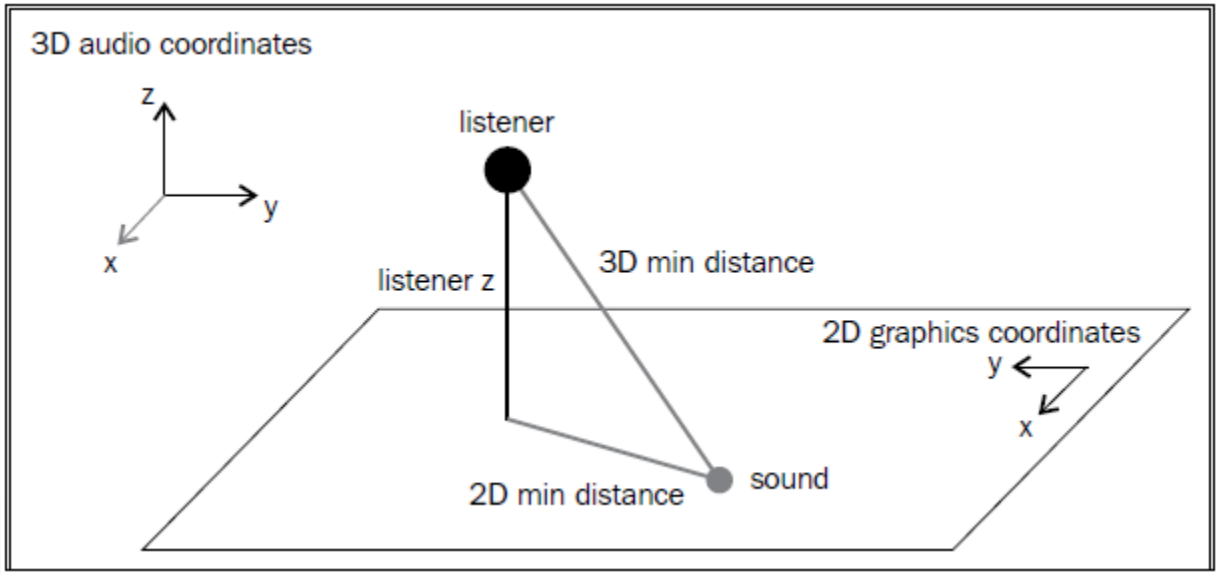
3D Sounds

- Close sounds are perceived louder than distant ones
  - Attenuation factor determines how fast a sound is attenuated depending on the distance



Positioning the Listener

- We place the listener in a plane different than the sound
  - The listener's Z coordinate has a greater value than zero



Playing Spatial Sounds

- In SoundPlayer.cpp, we create an anonymous namespace
  - For a few constants related to sound position

```
namespace
{
   const float ListenerZ = 300.f;
   const float Attenuation = 8.f;
   const float MinDistance2D = 200.f;
   const float MinDistance3D =
   std::sqrt(MinDistance2D*MinDistance2D + ListenerZ*ListenerZ);
}

void SoundPlayer::setListenerPosition(sf::Vector2f position)
{
   sf::Listener::setPosition(position.x, -position.y, ListenerZ);
}
```

Playing Spatial Sounds

```
void SoundPlayer::play(SoundEffect::ID effect, sf::Vector2f position)
{
    mSounds.push_back(sf::Sound());
    sf::Sound& sound = mSounds.back();
    sound.setBuffer(mSoundBuffers.get(effect));
    sound.setPosition(position.x, -position.y, 0.f);
     sound.setAttenuation(Attenuation);
    sound.setMinDistance(MinDistance3D);
    sound.play();
}

void SoundPlayer::play(SoundEffect::ID effect)
{
    play(effect, getListenerPosition());
}
```

Use Case

```
class SoundNode : public SceneNode
{
public:
    explicit SoundNode(SoundPlayer& player);
    void playSound(SoundEffect::ID sound,
    sf::Vector2f position);
    virtual unsigned int getCategory() const;
private:
    SoundPlayer& mSounds;
};

void Aircraft::playLocalSound(CommandQueue& commands, SoundEffect::ID effect)
{
    Command command;
    command.category = Category::SoundEffect;
    command.action = derivedAction<SoundNode>(
          std::bind(&SoundNode::playSound, _1, effect, getWorldPosition()));
    commands.push(command);
}
```

Use Case (cont'd.)

```
void Aircraft::checkProjectileLaunch(sf::Time dt, CommandQueue& commands)
{
    ...
    if (mIsFiring && mFireCountdown <= sf::Time::Zero)
    {
          playLocalSound(commands, isAllied() ?
          SoundEffect::AlliedGunfire : SoundEffect::EnemyGunfire);
          ...
    }
    if (mIsLaunchingMissile)
    {
          playLocalSound(commands, SoundEffect::LaunchMissile);
          ...
    }
}

void World::updateSounds()
{
    mSounds.setListenerPosition(
    mPlayerAircraft->getWorldPosition());
    mSounds.removeStoppedSounds();
}
```