Hector Medina

Professor Berthiaume

Gam 377 Individual Project Write up

P2 Engine Sound System

**Overview:**

The individual project that I have chosen is to create a sound system so that users can have sound when they make games using the engine. The sound system was designed so that users can create sounds ( either Sound2D or Sound3D) and then simply call functions on them, such as play(), stop(), and other relevant sounds. At the start of the class, Azul had no sound system and I thought that sound is a critical part of any engine, thus I had decided to add in the Sound System.

**Research:**

**Finding an External Sound Library:**

A lot of my research was on trying to find a good Sound library that was easy enough to use, and flexible enough to be incorporated into another system (the P2 Engine.) There were a few sound Libraries that I had looked into initially. Most of these had a lot of added features that I did not need, such as SFML (<http://www.sfml-dev.org/>.) Another library I ended up looking into was the CLAM audio library, but the feature list was huge, and I was not sure I would be able to fully understand all of it with the scope of the project at hand and the due date (<http://clam-project.org/doc/CLAM-doxygen/annotated.html> .) The last and final external sound engine that I looked into was the Irrklang Audio library, which has been used in games such as World of Good. This sound library seemed fairly straight forward to use and I believed that this sound engine would be the best choice, so I ended up choosing Irrklang ( <http://www.ambiera.com/irrklang/>) and then started to get familiar with how it works.

**Getting Familiar with the Irrklang:**

The first thing I did to get familiar with Irrklang was to do the following tutorials, which are found on their website: *Hello World*, and *3D Sound*. These pretty much helped me understand how to play my first sound as well as realize how Irrklang Works. In Irrklang, there are three main classes that are responsible for actually playing sounds; the types are ISound ,ISoundSource, and ISoundEngine. The minor functions have been omitted, since they are readily available in Irrklang’s api (<http://www.ambiera.com/irrklang/docu/index.html> ).

**ISoundEngine:**

In a nutshell, the ISoundEngine is the core system for irrklang; without ISoundEngine, you cannot play sounds. Having stated that, ISoundEngine is responsible for playing both 2D and 3D sounds, as well as giving back a reference to a playing sound in the form of an ISound. Playing a sound takes in either a cstring or an ISoundSource as a parameter for playing 2D Sounds ( Play2D() ) and playing 3D Sounds ( Play3D). When one of these functions is called, an ISound is returned, which can then be manipulated, such as pausing, stopping, setting volume, etc. Setting the position of the listener, which is the “ears” of the system, when regarding 3D Sound.

**ISoundSource:**

The ISoundSource is pretty much the data of how an audio file is represented in Memory. Irrklang has procedures so that an ISoundSource is only loaded once from a file. An ISoundSource can be used as a parameter in the ISoundEngine’sPlay2D() and Play3D(), so that it gives back an ISound that is a “copy” of the audio as specified in the ISoundSource. For example, if I created a soundsource from a file called “TankShot.flac,” then it would return an ISound that, if played, will sound like the TankShot.flac. It is important to note that changing the members of the ISoundSource and ISound are independent of each other.

**ISound:**

An ISound is an instance of the actual audible sound being played. This is the object that actually causes audio to occur in the speakers of the computer. An ISound can be stopped, paused, set to a play position (a time in the overall length of the sound file,) be set to a volume, etc… In addition, an ISound can be set to call an ISoundStopEventReceiver’s OnSoundStopped, via SetSoundEventReceiver.

**ISoundStopEventReceiver:**

Although this class is not a Main contributor to playing sounds, it is important in taking actions when a sound has ended by itself, not having a user call Stop() on it before hand. This class can be derived from, in order to make a user-made callback.

Overall, the ISoundEngine is an instance of the irrklang engine itself, the ISoundSources are a representation of the audio file in memory that are used to create ISound instances, and the ISounds are the actual audible sounds that can be manipulated, while they are playing.

**Investigating Sound System Models:**

The biggest influence on how I designed my engine was through previous experience while using Unity3D’s (https://unity3d.com/) sound system. This sound system has a few rules, as follows which I have adhered to.

* A Sound Object (in unity’s case called a SoundSource (<http://docs.unity3d.com/460/Documentation/Manual/class-AudioSource.html>)) can only have one audible sound at a time. Therefore, calling play on an already playing sound causes the audible sound to reset instead of creating two audible sounds.
* A Paused sound can only be un-paused by calling Play on it after it has been paused. If a sound is stopped during a pause, then calling play on it, will essentially cause the sound to start from the beginning instead of where it left off.
* Volume can be changed at any point in the sounds life.

In essence, unity’s 2D and 3D system plays from these AudioSources. If an AudioSource is 3D, it also keeps track of its position, so that the listener (only one allowed in unity’s Sound System), can calculate how it sounds. It is important to note that in Irrklang, an AudioClip is more like an ISoundSource (tells the information of how the sound should audibly sound), and an AudioSource is more like my system’s Sound2D.h and Sound3D.h, which a player can call functions on such as playing , stopping, etc...

**P2 Engine Sound System Features:**

Considering the scope of the individual project and the allotted time to create the system, I had decided on implementing both 2D and 3D sounds with minimal features. The following is the set of features for each Sound Type and the listener features.

* Sound2D
  + Playing
    - Single burst (playing a sound once)
    - Looped (playing a sound in a loop)
  + Stopping
  + Pausing
* Sound3D
  + Playing
    - Single burst(playing a sound once)
    - Looped
  + Stopping
  + Pausing
  + Setting Position
* Listener
  + Setting a SoundConscious object to be the system’s Listener.
  + Automatic positioning of the set listener when the GameObject is moved.

**P2 Engine Sound System Model**

**Note: (See doxygen for UML Diagram!!)**

**The following is a quick reference to the functions of the sound engine as well as a small summary of what they do. Official API is found in the Doxygen.**

**SoundSystem:**

In my design, I will have a SoundSystem class that will be a singleton and will be responsible for holding the irrklang sound engine as well as being responsible for actually calling the play on an ISoundSources. There will be a few functions that are listed below.

**Important Members:**

* **map<Sound\*,ISound\*> currPlayingLoopedSounds** - holds all of the active sounds (will change name)
* **BurstSoundEndReceiver\* sndEndReceiver and BurstSoundUserEndedReceiver\* sndEndedUsrReceiver** - the derived soundendReceivers that call different processes when attached to ISounds depending on if it is looped or not.
* **SoundConscious\* systemListener**  - the sound system's listener.
* **ISoundEngine\* soundEngine** - pointer to the irrklang sound engine Instance.

**Main functions:**

* **StopAllSounds()** - Stops all of the currently registered active sound.
* **StopSound(Sound\*)** - Takes in a Sound2D or Sound3D and calls stop on the associated ISound as well as removing it from the active Playing Sounds map (currPlayingLoopedSounds).
* **Play2D(Sound2D\* media**) - Plays a Non-looped 2D sound by feeding the irrklang engine instance the Sound's mySoundSource as a parameter and registering it to the active sounds (currPlayingLoopedSounds). This also sets the ISound's StopSoundReciever to the BurstSoundUserEndedReciever, since only the player can cause it to stop, unless changed to a non loop sound while playing, which changes it to BurstSoundEndReceiver, since it takes different actions if it stops by itself.
* **Play2DLooped(Sound2D\* media)** - Plays a looped 2D sound by feeding the irrklang engine instance the Sound's mySoundSource as a parameter and registering it to the active sounds (currPlayingLoopedSounds). This also sets the ISound's StopSoundReciever to the BurstSoundEndReceiver, since it can stop itself, unless changed to a non loop sound while playing, which changes it to BurstSoundUserEndedReciever, since it takes different actions if it is a looped sound (stopped by user).
* **Play3D(Sound 3D\* media)** - Plays a non-looped 3D sound by feeding the irrklang engine instance the Sound's mySoundSource as a parameter and registering it to the active sounds (currPlayingLoopedSounds). This also sets the ISound's StopSoundReciever to the BurstSoundEndReceiver, since it can stop itself, unless changed to a loop sound while playing, which changes it to BurstSoundUserEndedReciever, since it takes different actions if it is a looped sound (stopped by user).
* **Play3DLooped(Sound 3D\* media)** - Plays a looped 3D sound by feeding the irrklang engine instance the Sound's mySoundSource as a parameter and registering it to the active sounds (currPlayingLoopedSounds). This also sets the ISound's StopSoundReciever to the BurstSoundUserEndedReciever, since it is only stopped by the user, unless changed to a non loop sound while playing, which changes it to BurstSoundEndReciever, since it takes different actions if it is a non-looped Sound.
* **PauseSound(Sound 3D\* media)** - pauses the sound media if it is a registered active playing sound. Works with Sound2D and Sound 3D.
* **bool CheckSoundPlaying (Sound\* media)** - returns if a sound is playing.
* **SetMainVolume (float)** - Sets the overall Volume of the irrklang sound engine.
* **GetSoundEngine()** - returns the instance of the irrklang engine.
* **SetActiveSoundsLoop(Sound\*,bool)** - Sets an active sound to either looping or not looping.
* **SetListener (SoundConscious\*)** - Sets the soundConscious to be the overall Sound System's listener. When a soundConscious is moved, it will update Irrklang's listener's position and direction to its own.
* **ProcessSoundDrops()** - This is called at every frame to release any ISounds that were stopped by themselves (user did not call Stop() on them but sound ended.) Note: this only applies to BurstSounds, since user ended sounds are called on a frame basis. Looped sounds can end on any frame, so this forces them to be released on a designated frame.
* **AddToDropList(Sound\* target)**  - called when a burst sound ends by itself via the BurstSoundEndReceiver.
* **StartSoundEngine()** - Initializes the Irrklang Engine.

**SoundManager:**

**Important Members:**

* **map<string, ISoundSource\*,comparer> soundSourceDataBase** - map of all of the loaded sounds (as ISoundSource).

**Main Functions:**

* **LoadSound(const char\* const fileName,const char\* const assetName)** - Loads a sound file with the name fileName and stores it as an ISoundSource named assetName;
* **Create2DSound(const char\* const assetName)** - returns a Sound2D object with the ISoundSource specified via the assetName. So if ShotSound.flac was loaded as "ShotSound" then a Sound2D created with parameter "ShotSound" will sound like ShotSound.flac when played.
* **Create2DSound(const char\* const assetName)** - returns a Sound3D object with the ISoundSource specified via the assetName. So if ShotSound.flac was loaded as "ShotSound" then a Sound3D created with parameter "ShotSound" will sound like ShotSound.flac when played.

**Sound:**

**Important Members:**

* **ISoundSource\* mySoundSource** - points to what the Sound audibly sounds like
* **PlayModeBase\* myPlayingMode** - points to the play mode the sound is set to. This could be looped or not looped.

**Main Functions:**

* **Play() = 0** - plays the Sound.
* **Stop()** - Calls SoundSystem's StopSound(sound\*) on this sound.
* **SetVolume()** - Sets the Sounds individual Volume Level.
* **Bool IsPlaying()** - Returns if the sound if playing or not.

**Sound2D (Derived from Sound):  
  
Main Functions:**

* **All functions that are derived from sound are included.**
* **(overridden) Play()** - Plays the sound in either Play2D or Play2Dlooped depending on its myPlayingMode.
* **(Overridden) SetLooped(bool)** - set the myPlayingmode to either Looped2DPlayMode or Burst2DPlayMode.

**Sound3D(Derived from Sound):**

**Important Members:**

* **All derived Members**
* **Vect audiblePos** - the position of the sound in 3D space

**Main Functions:**

* **All functions that are derived from sound are included.**
* **(overridden) Play()** - Plays the sound in either Play3D or Play3Dlooped depending on its myPlayingMode.
* **(Overridden) SetLooped(bool)** - set the myPlayingmode to either Looped3DPlayMode or Burst3DPlayMode.
* **Vect& SetSoundPosition()** - returns the position of the sound in 3D space.
* **SetSoundPosition(Vect& newPos)** - Sets the position of the sound to the newPos parameter.
* Vect& GetSoundPosition() - Returns the position of the sound in 3D space.

**SoundConscious:**

**Important Members:**

* **Vect sndConPos** - the position of the SoundConscious entity.
* **Vect sndConDirection** - the direction of the SoundConscious entity.
* **SoundConsciousListeningState\* myState** - state of the SoundConscious entity. If it is the listener it will be listening and will have the ListeningState. If it is not the listener, it will have the NotListeningState.

**Main Functions:**

* **SetSndConsciousPos(Vect& newPos)** - Sets the position of the SoundConscious entity to the newPos.
* **SetSndConsciousDir(Vect& newDir)** - Sets the direction of the SoundConscious entity to the newDir.
* **SetToNotListening()** - changes the SoundConscious's SoundConsciousListeningState to the NotListeningState.
* **SetToListening()** - changes the SoundConscious's SoundConsciousListeningState to the ListeningState.
* **UpdatePosAndDir()** - updates Irrklang's Listener to this listener's position and direction. **Note:** if the State is not listening, then this will do nothing. If it is set to listening, it will change Irrklang's listener's position and Direction. Only one SoundConsious entity can be listening at all times.