Name: Noah Hart
UID: u1537288
Umail: u1537288@umail.utah.edu

Project Download (MayaExporter can be safely disabled):
https://drive.google.com/file/d/1MgJ8m49vitdqi1Vg06SMzZgFpDegGgU5/view?usp=drive_link
The Audio and Collision/Physics System consists of two projects and 2 files:

1. Audio
2. AudioBuilder
3. Collision.cpp/h

Setup Audio System

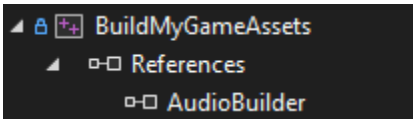
- To begin, start by making the Audio Project in your Engine, which is a Static Library
    - Don't forget to add the necessary property sheets.
- Next, you will need these files, which contain the functions from the audio system and the audio builder tool (which we will get to next)
- You will then make another project (empty project) similar to Meshbuilder called AudioBuilder in the Tools folder (with added property sheets) and add the EntryPoint and Builder files to it, deleting any extra files that are generated. **Make sure precompiled headers are disabled!**
- You will then need to update your AssetBuildFunctions.lua, under your mesh and shader asset types, add one for audio, and in the BuildAsset function, add the copy files code snippet right before the return not wereThereErrors call

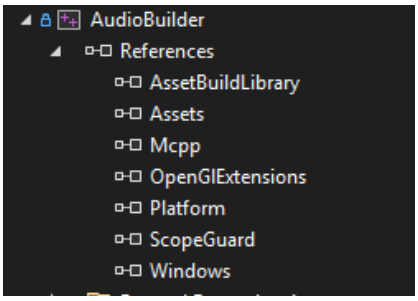
```lua
-- Audio Asset Type
--------------------
NewAssetTypeInfo("audio",
{
    GetBuilderRelativePath = function()
        return "AudioBuilder.exe"
    end,

    -- Convert "audio/bgmusic.wav" -> "audio/bgmusic.audio"
    ConvertSourceRelativePathToBuiltRelativePath = function(i_sourceRelativePath)
        -- Extract directory and filename
        local relativeDirectory, file = i_sourceRelativePath:match("(.-)([^/\\]+)$")
        local fileName, extensionWithPeriod = file:match("([^%.]+)(.*)")

        -- If the source isn't already under "audio/", force it
        if not relativeDirectory:find("audio") then
            relativeDirectory = "audio/"
        end

        -- Build the output path
        return relativeDirectory .. fileName .. ".audio"
    end,
})
```

```lua
    -- Copy Audio Files (only if no .audio builder output exists)
do
    if assetsToBuild.audio then
        for _, audioAsset in ipairs(assetsToBuild.audio) do
            local sourceRelativePath = audioAsset.path or audioAsset
            local targetAudioPath = GameInstallDir .. "/data/" .. sourceRelativePath:gsub("%.wav$", ".audio")

            if not DoesFileExist(targetAudioPath) then
                print("No .audio built file found for " .. sourceRelativePath .. ", copying raw .wav instead")
                CopyFile(GameSourceContentDir .. sourceRelativePath, targetAudioPath)
            end
        end
    end
end
```

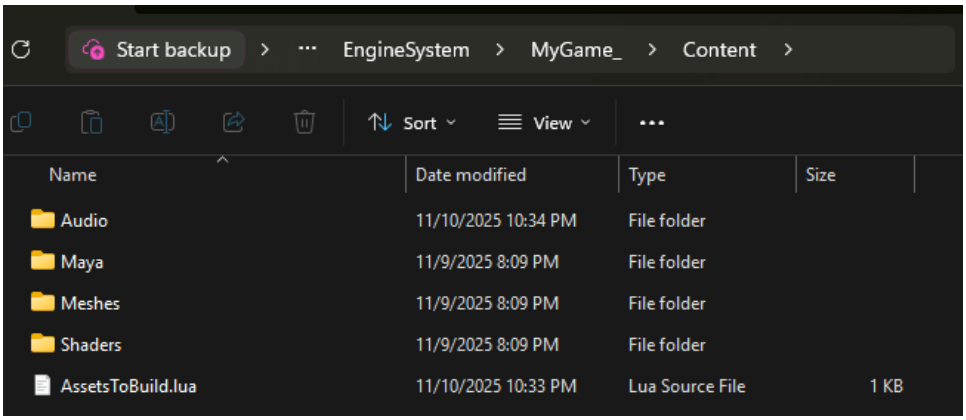
Name: Noah Hart
UID: u1537288
Umail: u1537288@umail.utah.edu

- After that, add these references:

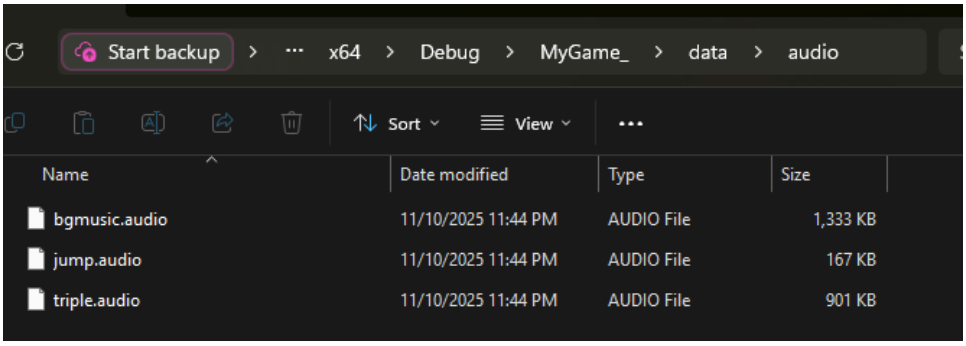




- The Audio will get converted to a binary file with the .audio extension, so make sure you create a folder in your content under MyGame_ called “Audio” and drop your files in there (**Please only use .wav files)**



- Once built, it should look like this in your data/audio/sound.audio path



- Don't forget to add your files in AssetsToBuild.lua

```
-- Audio
audio =
{
    { path = "Audio/jump.wav" },
    { path = "Audio/bgmusic.wav" },
    { path = "Audio/triple.wav" },
},
```

Name: Noah Hart
UID: u1537288
Umail: u1537288@umail.utah.edu

Usage Audio System

```
public:
    // Initialize / Shutdown
    static void Initialize();
    static void Shutdown();

    // Background music controls
    static void PlayMusic(const char* path, bool loop = true, float volume = 1.0f);
    static void PauseMusic();
    static void ResumeMusic();

    // Sound effects (can play on top of music)
    static void PlaySoundEffect(const char* path);
```

- What you will notice is that there are 6 key functions, all of which are used in MyGame.cpp
- First, you want to add an instance of your class near the top that you can call

```
static eae6320::Audio::cAudio g_Audio;
```

- Initialize()
    - Will be called in the MyGame Initialize Function

```
eae6320::cResult eae6320::cMyGame::Initialize()
{
    auto result = Results::Success;

    eae6320::Audio::cAudio::Initialize();
```

- Shutdown()
    - To be called in your MyGame CleanUP

```
g_Audio.Shutdown();
```

- PlayMusic()
    - Should be used in your UpdateBasedOnTime or Initialize function to play background music (or anything you want to be persistent.
- PlaySoundEffect()
    - Should be in Update(Simulation)BasedonInput when using a physics action like playing a Jump SFX when hitting the space bar, and the player character also jumps. (**Here is also an Example of the path you use**)

```
eae6320::Audio::cAudio::PlaySoundEffect("data/audio/jump.audio");
```

Name: Noah Hart
UID: u1537288
Umail: u1537288@umail.utah.edu

- Pause/ResumeMusic()
    - Similar to SoundEffect, you can bind a key to call the function to pause all sounds

```
// pause music
if (UserInput::IsKeyPressed(UserInput::KeyCodes::P))
{
    static bool paused = false;
    if (!paused)
        eae6320::Audio::cAudio::PauseMusic();
    else
        eae6320::Audio::cAudio::ResumeMusic();
    paused = !paused;
}
```

Setup and Usage Collision and Physics System

- For this system, all you will need are these files added to the Physics project in the Engine, as a bulk of the code will be set up in MyGame UpdateBasedonTime
- First thing you want to add to your MyGame in UpdateBasedonTime are the Colliders for your gameobjects (**Scale is not needed, but I will show you in a future step what that does**)

```
// ======================================
// 3. Build simple colliders
// ======================================
eae6320::Physics::AABB colliderPlayer;
colliderPlayer.center = m_gameObject.m_rigidBody.position;
colliderPlayer.halfSize = m_gameObject.m_colliderHalfSize * m_gameObject.m_scale;

eae6320::Physics::AABB collider2;
collider2.center = m_gameObject2.m_rigidBody.position;
collider2.halfSize = m_gameObject2.m_colliderHalfSize * m_gameObject2.m_scale;

eae6320::Physics::AABB collider3;
collider3.center = m_gameObject3.m_rigidBody.position;
collider3.halfSize = m_gameObject3.m_colliderHalfSize * m_gameObject3.m_scale;

eae6320::Physics::AABB collider4;
collider4.center = m_gameObject4.m_rigidBody.position;
collider4.halfSize = m_gameObject4.m_colliderHalfSize * m_gameObject4.m_scale;
```

Name: Noah Hart
UID: u1537288
Umail: u1537288@umail.utah.edu

- You then want to add checks between whichever game objects you want to have be collision with each other

```
// ======================================
// 4. Collision checks (track grounding)
// ======================================
bool groundedOnPlatform1 = false;
bool groundedOnPlatform2 = false;

// Collider vs each platform
groundedOnPlatform1 = eae6320::Physics::ResolveOverlap(
    m_gameObject.m_rigidBody.position,
    m_gameObject.m_rigidBody.velocity,
    colliderPlayer,
    collider3
);

groundedOnPlatform2 = eae6320::Physics::ResolveOverlap(
    m_gameObject.m_rigidBody.position,
    m_gameObject.m_rigidBody.velocity,
    colliderPlayer,
    collider2
);
```

- For adding **Gravity,** if you are going with 2D you want to remove any up and down input. I have added this condition in my UpdateSimulationBasedonInput to stop the player from sliding

```
else
{
    // no input = stop horizontal movement
    m_gameObject.m_rigidBody.velocity.x = 0.0f;
}
```

- You then want to apply gravity on the Y axis and make sure all objects that you want to have it get properly defined, and the others will just be set to static

```
// ======================================
// 1. Apply gravity (only to dynamic player)
// ======================================
const float gravity = -9.81f;
if (!m_isGrounded)
{
    m_gameObject.m_rigidBody.acceleration = Math::sVector(0.0f, gravity, 0.0f);
}
else
{
    m_gameObject.m_rigidBody.acceleration = Math::sVector(0.0f, 0.0f, 0.0f);
}

// Static objects
m_gameObject2.m_rigidBody.acceleration = Math::sVector(0.0f, 0.0f, 0.0f);
m_gameObject3.m_rigidBody.acceleration = Math::sVector(0.0f, 0.0f, 0.0f);
m_gameObject4.m_rigidBody.acceleration = Math::sVector(0.0f, 0.0f, 0.0f);
```

Name: Noah Hart
UID: u1537288
Umail: u1537288@umail.utah.edu

- If you would like to add jumping, you can set a jump height and a bool in your header file

```
// Jump logic
static bool wasSpacePressed = false;
bool isSpacePressed = UserInput::IsKeyPressed(UserInput::KeyCodes::Space);
if (isSpacePressed && !wasSpacePressed && m_isGrounded)
{
    m_gameObject.m_rigidBody.velocity.y = 7.0f; // jump height
    m_isGrounded = false;

    eae6320::Audio::cAudio::PlaySoundEffect("data/audio/jump.audio");

}
```

- Adding this check to make sure the player can't spam jump, and also a fallspeed

```
// ========================================
// 5. Update grounded state
// ========================================
bool wasGrounded = m_isGrounded;
m_isGrounded = groundedOnPlatform1 || groundedOnPlatform2;

// If we just landed, zero Y velocity
if (!wasGrounded && m_isGrounded)
{
    m_gameObject.m_rigidBody.velocity.y = 0.0f;
}

// ========================================
// 6. Clamp falling speed
// ========================================
const float maxFallSpeed = -25.0f;
if (m_gameObject.m_rigidBody.velocity.y < maxFallSpeed)
    m_gameObject.m_rigidBody.velocity.y = maxFallSpeed;
```

- **BETA FEATURE -** If you would like to add scaling, you can create this matrix function in your cMatrix_transformation.inl and in your game object, add a m_scale vector variable

```
constexpr eae6320::Math::cMatrix_transformation eae6320::Math::cMatrix_transformation::CreateScale(const sVector& i_scale)
{
    return cMatrix_transformation(
        i_scale.x, 0.0f, 0.0f, 0.0f,
        0.0f, i_scale.y, 0.0f, 0.0f,
        0.0f, 0.0f, i_scale.z, 0.0f,
        0.0f, 0.0f, 0.0f, 1.0f
    );
}
```

```
Math::sVector m_scale = Math::sVector(1.0f, 1.0f, 1.0f);
Math::sVector m_colliderHalfSize = Math::sVector(0.5f, 0.5f, 0.0f);
```

Name: Noah Hart
UID: u1537288
Umail: u1537288@umail.utah.edu

- Then, however, you are submitting your data for rendering you would just call that scale matrix

```
void cGameObject::SubmitForRendering(float i_elapsedSecondCount_sinceLastSimulationUpdate) const
{
    if (m_mesh && m_effect)
    {
        auto localToWorld = GetLocalToWorldTransform(i_elapsedSecondCount_sinceLastSimulationUpdate);

        const auto scaleMatrix = Math::cMatrix_transformation::CreateScale(m_scale);
        const auto scaledTransform = scaleMatrix * localToWorld;

        Graphics::SubmitDrawCall(m_mesh, m_effect, scaledTransform);
    }
}
```

Feel Free to contact me directly if you have any issues with setting up these systems!