Project Increment 5
(with XACT)

These steps assume you're using XACT to implement the sound effects in the project. Although our company used XACT for the sound effects in all our commercial projects, implementing the game audio using the SoundEffect and SoundEffectInstance classes (in other words, not using XACT) is more similar to the way things are done in Unity (for example). In addition, I've had students run into more XACT problems in the recent past than I like. I recommend you implement the sound effects in the project without XACT (see the other pdf for implementation steps), but if you decide you want to use XACT, use the instructions here.

Visual Studio 2010 Users

The classes we need to use to play our sounds are contained in the Microsoft.Xna.Framework.Xact dll, which isn't included as a reference in our template Windows Game project. To add a reference to that dll, right click References in the code project, then select Add Reference. Select the .NET tab (all the way on the left) in the Add Reference dialog box that pops up, scroll down to the Xna namespaces, select the Microsoft.Xna.Framework.Xact namespace, and click the OK button.

Visual Studio 2013 Users

The classes we need to use to play our sounds are contained in the Microsoft.Xna.Framework.Xact dll, which isn't included as a reference in our template Windows Game project. To add a reference to that dll, right click References in the code project, then select Add Reference. Select Assemblies > Extensions on the left in the Reference Manager dialog box that pops up, scroll down to the Xna namespaces, select the Microsoft.Xna.Framework.Xact namespace, be sure to click the checkbox that appears, and click the OK button.

Step 0: Create and add an XACT project

Create a new XACT project and save it in the sounds folder where you have all the wav files for the game project I called mine GameAudio). Add all the wav files to the Wave Bank, drag all the wav files from the Wave bank into the sounds pane of the Sound Bank, then drag all the sounds from the sounds pane into the cues pane of the sound bank. Save your XACT project.

Right click the sound folder in your GameProjectContent project, select Add > Existing Item …, select the GameAudio XACT project, and click Add. Build your project.

Step 1: Add burger and teddy bear shooting sound effects

It's time to start adding sound effects to our game. For this step, you're adding sound effects when the burger or the teddy bears shoot.

1. At the top of the `Game1` class, delete all the `SoundEffect` fields and add fields for `AudioEngine`, `WaveBank`, and `SoundBank` fields
2. Add code to the `Game1` `LoadContent` method to load all the sound effects for the game. Notice that the sound content is contained in a sounds subfolder in the GameProjectContent project; you'll need to include that subfolder in your path to the audio content when you load it
3. In the `Burger` class, change the `SoundEffect` field to a `SoundBank` field and refactor the name of the field to `soundBank`. In the `Burger` constructor, change the `SoundEffect` parameter to a `SoundBank` parameter and refactor the name of the parameter to `soundBank`
4. In the `Game1` `LoadContent` method, change the last argument in your call to the Burger constructor from null to the sound bank field
5. Add code to the `Burger` `Update` method to use the `soundBank` field to play the appropriate cue when the burger shoots a projectile

6. In the `TeddyBear` class, delete one of the `SoundEffect` fields. Change the remaining `SoundEffect` field to a `SoundBank` field and refactor the name of the field to `soundBank`. In the `TeddyBear` constructor, delete one of the `SoundEffect` parameters. Change the remaining `SoundEffect` parameter to a `SoundBank` parameter and refactor the name of the parameter to `soundBank`. Fix any compilation errors that occur in the constructor
7. In the `Game1` `SpawnBear` method, delete one of the null arguments in the call to the `TeddyBear` constructor. Change the last argument in your call to the `TeddyBear` constructor from null to the sound bank field
8. Add code to the `TeddyBear` `Update` method to use the `soundBank` field to play the appropriate cue when the teddy bear shoots a projectile

When you run your game, you should hear sound effects when the burger and the teddy bears shoot projectiles.

Step 2: Add teddy bear bounce sound effects

For this step, you're adding sound effects when the teddy bears bounce off the sides of the game window or off each other.

1. Add code to the `TeddyBear` `BounceTopBottom` method to use the `soundBank` field to play the appropriate cue for the teddy bear bouncing sound when the teddy bear bounces off the top or bottom of the game window
2. Add code to the `TeddyBear` `BounceLeftRight` method to use the `soundBank` field to play the appropriate cue for the teddy bear bouncing sound when the teddy bear bounces off the left or right of the game window
3. Add code to the `Game1` `Update` method to use the `soundBank` field to play the appropriate cue for the teddy bear bouncing sound when two teddy bears collide with each other

When you run your game, you should hear sound effects when the teddy bears bounce off the sides of the game window or off each other.

Step 3: Add burger damage sound effects

For this step, you're adding sound effects whenever the burger take damage.

1. Add code to the `Game1` `Update` method to use the `soundBank` field to play the appropriate cue when the burger takes damage from colliding with a teddy bear or a projectile

When you run your game, you should hear sound effects whenever the burger takes damage.

Step 4: Add explosion sound effects

Explosions should also have sound effects. For this step, you're adding sound effects when an explosion is created.

1. Add a parameter to the `Explosion` constructor for a `SoundBank` that will be used to play a cue at the end of the constructor. You'll need to add a using statement for the appropriate namespace to get this to compile
2. Use the sound bank parameter to play the appropriate cue at the end of the `Explosion` constructor
3. Add code to the `Game1` `Update` method to pass the `soundBank` field as the final argument whenever a new `Explosion` object is created

When you run your game, you should hear sound effects when the explosions start.

Step 5: Add losing sound effects

For this step, you're adding a sound effect when the burger's health reaches 0.

1. Add code to the `Game1` `Update` method to call the `CheckBurgerKill` method whenever the burger takes damage
2. Add code to the `Game1` `CheckBurgerKill` method that checks if the burger's health is 0 (using the burger's `Health` property) and the burger isn't dead yet (using the `Game1` `burgerDead` variable). If both those conditions are true, set the `burgerDead` variable to `true` and use the `soundBank` field to play the appropriate cue for a burger death

When you run your game, you should hear a sound effect when the burger's health reaches 0.

Step 6: Add health display

For this step, you're adding a text display for the burger health.

1. Add code to the `Game1` `LoadContent` method to load the Arial20 font into the `font` variable
2. Add code to the `Game1` `LoadContent` method to set the `healthString` variable to the `GameConstants` `HEALTH_PREFIX` constant concatenated with the current burger health (using the burger's `Health` property). Make sure you do this AFTER creating the burger object!
3. Add code to the `Game1` `Update` method to set the `healthString` variable to the `GameConstants` `HEALTH_PREFIX` constant concatenated with the current burger health (using the burger's `Health` property). For efficiency, you should ONLY do this whenever the burger takes damage.
4. Add code to the `Game1` `Draw` method to draw the `healthString` at the location given by the `GameConstants` `HEALTH_LOCATION` constant

When you run your game, you should see the burger's health displayed and modified as the burger takes damage.

Step 7: Add scoring

For this step, you're adding scoring to the game.

1. Add code to the `Game1` `LoadContent` method to set the `scoreString` variable to the `GameConstants` `SCORE_PREFIX` constant concatenated with the current score
2. Add code to the `Game1` `Update` method that adds the value of the `GameConstants` `BEAR_POINTS` constant to the score variable when a collision between an active french fries projectile and a teddy bear is detected (you already have a block of code that detects this collision from your previous work). Also, set the `scoreString` variable to the `GameConstants` `SCORE_PREFIX` constant concatenated with the current score
3. Add code to the `Game1` `Draw` method to draw the `scoreString` at the location given by the `GameConstants` `SCORE_LOCATION` constant

When you run your game, you should see the score displayed and modified as you hit teddy bears with french fries.

Step 8: Change burger control scheme

Change the game so the player no longer uses the mouse to control the burger. Make it so WASD are used to move the burger and space is used to make the burger shoot. You'll have to change the second parameter of the `Burger` `Update` method to a `KeyboardState` parameter to make this work properly. The `GameConstants` class contains a `BURGER_MOVEMENT_AMOUNT` constant you should use for the movement part. You do NOT have to avoid the Doom Strafe 40 bug in your code.

<u>Step 9: Celebrate!</u>

That's it – you finished the optional project. Shoot the teddy bears to your heart's content, try to best your high score, and celebrate however you see fit!