Lab 10 – Teddy Bear Explosions

Instructions: Complete each problem. If you're struggling with a problem, feel free to ask questions on the class forum.

This lab is optional, but it gives you valuable programming experience. You should definitely complete the lab if you can.

Getting Started

Download the zip file for your IDE (Lab10Materials.zip or MonoDevelopLab10Materials.zip) from the Labs course page. Unzip the file somewhere on your computer. Copy the ExplodingTeddies Help chm file from the Help folder onto your desktop.

Double click the ExplodingTeddies Help file on the desktop to open the file.

Windows Users: If you get an error message in the right-hand pane instead of documentation links, it means you're currently blocking access to the documentation. To fix this, right-click on the Exploding Teddies Help file on the desktop, select Properties, and select the General tab. Click the Unblock button near the lower right corner of the popup.

If you run into problems, Section 5.4 of the book might help.

Problem 1 – Create a project, add content, add moving teddy bears

XNA Users

Start up the IDE and create a new Windows Game (4.0) project named Lab10. Save the project in a reasonable location on the computer.

Copy the two teddy bear images and the explosion image into the Lab10Content project folder and add them to the project.

You're going to be using an `ExplodingTeddies` namespace that I wrote for you. Remember, namespaces give us a library of classes that someone else wrote for us to use. Up to this point in the course, I've been giving you the source code for the classes I wrote for you, but that's really not the way it's actually done most of the time. In practice, people distribute code like this using dynamic link libraries (dlls). That's how we're doing it for this assignment.

To give your project access to the dll you extracted from the zip file you downloaded, the first thing you need to do is add a reference to your project. To do that, do the following:

1. Right click References under the Lab10 project in the Solution Explorer pane at the upper right corner of the IDE
2. Click Add Reference …
3. Click the Browse tab in the dialog that appears
4. Browse to the dll you extracted
5. Select the dll and click the OK button

The above steps make the code in the dll available to the code in your project.

MonoGame Users

Start up the IDE and create a new MonoGame solution named Lab10.

Mac Users: Read the "Creating a New Mac MonoGame Solution" document on the MonoDevelop Resources course page to learn how to do this.

Linux Users: Read the "Creating a New Linux MonoGame Solution" document on the MonoDevelop Resources course page to learn how to do this.

Save the project in a reasonable location on the computer − and remember where that location is!

Read the "Adding Content to a MonoGame Project" document on the MonoDevelop Resources course page to learn how to add content to your project. I've provided compiled xnb assets for both Mac and Linux users in the zip file; make sure you use the right ones!

Copy the Explosion.cs and TeddyBear.cs files into the Lab10 project folder.

Open up the IDE. If you don't see the pane on the left listing all the folders and classes in the project, select View -> Debug.

Add the files to the Lab10 project in the IDE by selecting the project name on the left (the name should be bold), picking Add > Add files … from the dropdown to the right of the project name, selecting the files, and clicking the Open button.

All Users

To actually use the `Explosion` and `TeddyBear` classes from the `ExplodingTeddies` namespace in your `Game1` class, though, you need to use the namespace. To do that, add the following code below the `using` statements already in the class:

```
using ExplodingTeddies;
```

Now you'll be able to use the classes I included in the namespace.

Add two constants to the `Game1` class just above the declaration of the `graphics` variable:

```
const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 600;
```

Just below the line that says `SpriteBatch spriteBatch;` near the top of the `Game1` class, add variable declarations for two `TeddyBear` objects and an `Explosion` object.

Just below the line that says `Content.RootDirectory = "Content";` near the top of the `Game1` constructor, add the following two lines of code:

```
graphics.PreferredBackBufferWidth = WINDOW_WIDTH;
graphics.PreferredBackBufferHeight = WINDOW_HEIGHT;
```

In the `Game1 LoadContent` method, replace the comment that says

```
// TODO: use this.Content to load your game content here
```

with a comment and the code to create two new teddy bear objects using the `TeddyBear` constructor that gives the teddy bears a specific velocity. Set the locations and velocities of the teddy bears so they'll collide. The teddy bears should use different sprites so we can tell them apart. Use the ExplodingTeddies Help file as needed.

To generate a `Vector2` for a velocity that has a positive x component only, you could use:

```
new Vector2(1, 0)
```

Add code to create an `Explosion` object using the `Explosion` constructor.

Add code to the `Game1 Draw` method to have the two teddy bears draw themselves.

Add code to the `Game1 Update` method to have the two teddy bears update themselves.

Problem 2 – Explode the teddy bears when they collide

When a pair of `TeddyBear` objects collides, start an `Explosion` animation at that location and make the `TeddyBear` objects inactive. You make a `TeddyBear` inactive by setting its `Active` property to false.

To do this properly, you need to answer the following questions before you write any code:

- Do teddy bears need to be active to collide?
- How do we know that the teddy bears have collided?
- What do we do to resolve the collision?

- How do we determine where the collision occurred? (Hint: Look at the methods available in the `Rectangle` class)
- How do we play the explosion at (approximately) the collision point? (Hint: Look at the properties available for `Rectangle` objects)

Add code to the `Update` method just above the line that says `base.Update(gameTime);` to complete all these actions. Test your code.

Add code to the `Game1 Draw` method to have the explosion draw itself.

Problem 3 – Update the explosion

Add code to the `Game1 Update` method to have the explosion update itself.