# 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

Double-click the index file in the Help folder and click the ExplodingTeddies link in the pane on the left; this is the documentation for the classes I provided to you in the ExplodingTeddies dll.

If you run into problems, you should look at the Section 5.4 in the book before asking someone else for help. Chapter 5 is provided as a download for lecture 5.2.

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

Start up the IDE and create a new MonoGame Windows Project (or appropriate MonoGame project for your OS) named Lab10. Save the project in a reasonable location on the computer.

Use the Pipeline tool to build content for the two teddy bear images and the explosion image from the zip file and add the content to your 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.

Copy the ExplodingTeddies.dll file into the Lab10 project folder (the folder that contains the Game1.cs file). Don't try to add the dll as an existing item, we need to do this a different way.

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:

Windows Users

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 button near the lower right in the dialog that appears
4. Browse to the dll you copied into the Lab10 project folder
5. Double click the dll to add it and click the OK button

Mac/Linux Users

1. Right click References under the Lab10 project in the Solution Explorer pane at the upper left corner of the IDE
2. Click Edit References …
3. Click the .Net Assembly tab near the upper left of the dialog that appears
4. Click the Browse button near the lower right in the dialog
5. Browse to the dll you copied into the Lab10 project folder
6. Double click the dll to add it
7. Make sure the check box next to the dll is checked in the pane on the left and click the OK button

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

To actually use any of that code 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 dll.

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

```
const int WindowWidth = 800;
const int WindowHeight = 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 = WindowWidth;
graphics.PreferredBackBufferHeight = WindowHeight;
```

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 Documentation as needed.

Caution: You need to provide 2 more arguments to the constructors for the `TeddyBear` class than you've provided before: the window width and the window height. When I was building a dll for the `ExplodingTeddies` namespace classes, I couldn't access `Game1.WindowWidth` and `Game1.WindowHeight` from within the `TeddyBear` class to make teddy bears bounce off the edges of the window. That's why you need to provide that information to the constructors.

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.