Project Increment 3 Steps

Step 1: Have teddy bears collide with projectiles

After: Chapter 10

French fries should of course kill teddy bears. For this step, you're making the teddy bears disappear when they collide with a projectile.

1. Add code to the `Game1` `Update` method to check for a collision between each teddy bear and each projectile in the game. A reasonable approach would be to have a foreach loop over the `bears` list with a nested foreach loop over the `projectiles` list inside it. This will make sure you check every teddy bear/projectile pairing. In the inner loop, check to see if the current projectile is french fries (teddy bears don't kill teddy bears!) and the current teddy bear and the current projectile collide by checking if they're both active and if their collision rectangles intersect. If they do, set the `Active` property for the bear to `false` and the `Active` property for the projectile to `false`.
2. Add code to the `Game1` `Update` method to remove all the inactive teddy bears from the `bears` list. The code just after the test cases after Figure 10.1 in the book shows one way to do this:

```
for (int i = bears.Count - 1; i >= 0; i--)
{
    if (!bears[i].Active)
    {
        bears.RemoveAt(i);
    }
}
```

3. Add code to the `Game1` `Update` method to remove all the inactive projectiles from the `projectiles` list. This will remove the projectiles you deactivated earlier in this step and will also remove the projectiles you handled in Project Increment 2 when they left the game window.

When you run your game, the teddy bear should disappear when hit by french fries and the french fries involved in that collision should also disappear.

Step 2: Have teddy bears blow up

After: Chapter 10

We'd like teddy bears to blow up when they're destroyed by french fries rather than just disappearing. For this step, you're making the teddy bears blow up when they collide with a projectile.

1. Add code to the `Game1` `LoadContent` method to load the texture for the `explosionSpriteStrip`
2. Add code to the `Game1` `Update` method just after the code that deactivates a teddy bear and a projectile involved in a collision. Your new code should create a new `Explosion` object using the `explosionSpriteStrip` and the bear's `Location` property and add the new object to the `explosions` list.

When you run your game, the teddy bear should explode when hit by french fries.

Step 3: Remove finished explosions

After: Chapter 10

At this point all the explosions we ever create in the game stay in the explosions list. For this step, you're removing explosions that have finished playing from that list.

1. Add code to the `Game1` `Update` method to remove explosions that have finished playing from the `explosions` list.

When you run your game, it should work just like it did after the previous step. Even though you can't see a difference, though, this is going to be important for efficiency.

Step 4: Include multiple teddy bears

After: Chapter 10

It will be way more interesting if we have multiple teddy bears in the game. For this step, you're adding more teddy bears to the game.

1. Add code to the `Game1` `LoadContent` method to add multiple teddy bears to the game. Move your call to the `SpawnBear` method inside a for loop that uses the `GameConstants` `MaxBears` constant to spawn the appropriate number of bears.

When you run your game, you should now have multiple (precisely `MaxBears`) teddy bears in the game.

Step 5: Bounce teddy bears off each other

After: Chapter 10

You probably noticed that the teddy bears pass right through each other, but we'd rather have them bounce off each other instead. For this step, you're making the teddy bears bounce off each other.

CAUTION: This is harder to do than it probably seems. I've put a lot of collision detection and resolution code inside the `CollisionUtils` `CheckCollision` method to protect you from that complexity. The method returns a `CollisionResolutionInfo` object you need to use, though, and you need to use that object properly for everything to work correctly.

1. Add code to the `Game1` `Update` method to check and resolve collisions between the teddy bears. You should start with two for loops. The `CheckAndResolveBearCollisions` method in the *Write the Code* area in Section 10.7 of the book shows how to structure the for loops (don't use the bodies of the loops, just the loops).

```
for (int i = 0; i < bears.Count; i++)
{
    for (int j = i + 1; j < bears.Count; j++)
    {
        <your code goes here>
    }
}
```

2. In the body of the inner for loop, make sure both bears you're checking are active; if they are, call the `CollisionUtils` `CheckCollision` method and save the `CollisionResolutionInfo` object it returns in a variable. The first argument you need to provide to the method is a time step; this is just the number of milliseconds that have elapsed since the last time the `Update` method was called. You should be able to figure out what to pass for the rest of the arguments using values from the `GameConstants` class and the properties of the two bear objects you're checking
3. If the returned object isn't null, there's a collision you need to resolve. There are two possibilities for each of the two teddy bears, though, so let's look at the possibilities for the first teddy bear. If the `FirstOutOfBounds` property of the collision resolution info variable is `true`, the first teddy bear ended up at least partially outside the game window as a result of the collision resolution. In this case, you should set the `Active` property for that teddy bear to `false` to remove that teddy bear from the game (because it was "forced out of the game world" by the collision). Otherwise, you should set the `Velocity` and `DrawRectangle` properties for the first teddy bear to their new values using the `FirstVelocity` and `FirstDrawRectangle` properties of the returned object. You'll of course need to do the same processing for the second teddy bear.

When you run your game, the teddy bears should bounce off of each other.