Project Increment 4

Step 1: Spawn new teddy bear when one is destroyed

At this point you can kill all the teddy bears, then there's nothing else to do (except drive the burger around and shoot french fries, of course). For this step, you're adding more teddy bears to the game when one or more teddy bears is destroyed.

1. Add code to the `Game1` `Update` method, right after you clean out the inactive teddy bears, to spawn new teddy bears (using the `SpawnBear` method) until you have `GameConstants` `MAX_BEARS` bears in the game again. You should use a while loop for this. Why? Because you might have removed more than one inactive teddy bear, so you might need to call the `SpawnBear` method multiple times to get back up to `MAX_BEARS` bears. You could also do this with a for loop, but use a while loop since we haven't used one yet.

When you run your game, you should have a new bear (or new bears) spawn immediately after you kill one or more bears. Now the game is unwinnable! Go for a high score (score isn't included in the game yet, but it will be before we're done) instead of trying to kill all the teddy bears.

Step 2: Only spawn new teddy bear into a collision-free location

You may not have seen this, but spawning a new teddy bear into a location where it's immediately in collision can make the physics really goofy (or immediately kill the new teddy bear if it spawns on top of a projectile). For this step, you're making sure that new teddy bears only spawn into collision-free locations.

1. Add code to the `Game1` `SpawnBear` method to accomplish this. First, right after you've created the new bear, get a list of the collision rectangles for the other game objects in the game by calling the `GetCollisionRectangles` method. Next, add a while loop that keeps looping while the `CollisionUtils` `IsCollisionFree` method returns `false`; you'll need to provide the new bear's collision rectangle and the list of collision rectangles as the arguments for the method call. In the body of the while loop, set the `X` and `Y` properties of your new bear to new values using the `GetRandomLocation` method. If you don't do this correctly, you'll get an infinite loop for the while loop, so be careful!

When you run your game, you probably won't notice any difference from the previous step because we'd only see a problem when a new teddy bear spawns into a collision. Of course, even if you've never seen that problem, it would occur a gazillion times as soon as you shipped the game, so it's a good thing we fixed it.

Step 3: Add burger health and burger collisions with teddy bears

At this point, the burger never takes any damage; it's time to change that. For this step, you're adding a health property for the burger and detecting and resolving collisions between the burger and teddy bears.

1. Add code for a `Health` property to the `Burger` class. The property should have both get and set accessors. For the set accessor, make sure that the health value never goes below 0 or above 100.
2. Add code to the `Game1` `Update` method to check for a collision between the burger and each teddy bear in the game. A reasonable approach would be to have a foreach loop over the `bears` list with an if statement inside the loop that checks if the current bear is active and the current bear and the burger collide by checking if their collision rectangles intersect. If they do, subtract health from the burger's `Health` property using the `GameConstants` `BEAR_DAMAGE` constant, set the `Active` property for the bear to `false`, 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 bears should explode when they collide with the burger. How can you make sure the burger is actually taking damage? Drive the burger around running into teddy bears. If you implemented the `Burger` `Update` method properly previously, making sure the burger can only move if its health is greater than 0, at some point the burger will stop responding to the mouse. You can also just use the debugger to check this if you want.

Step 4: Add burger collisions with projectiles

We should also handle collisions between the burger and the projectiles in the game; just the teddy bear projectiles, though, because we don't want to punish players for running into the french fries they just shot (though maybe we should <grin>)! For this step, you're detecting and resolving collisions between the burger and teddy bear projectiles.

1. Add code to the `Game1` `Update` method to check for a collision between the burger and each projectile in the game. A reasonable approach would be to have a foreach loop over the `projectiles` list with an if statement inside the loop that checks to see if the current projectile is a teddy bear projectile, if the current projectile is active, and the current projectile and the burger collide by checking if their collision rectangles intersect. If they do, set the `Active` property for the projectile to `false`. You also need to reduce the burger's health, so you should use the `GameConstants` `TEDDY_BEAR_PROJECTILE_DAMAGE` constant to do that.

When you run your game, teddy bear projectiles should disappear when they collide with the burger and french fries should pass through the burger. How can you make sure the burger is actually taking damage? Drive the burger around running into teddy bear projectiles. If you implemented the `Burger` `Update` method properly previously, making sure the burger can only move if its health is greater than 0, at some point the burger will stop responding to the mouse. You can also just use the debugger to check this if you want.