Computer Science Taster Day

XNA Crazy Balls!

Hello everyone and welcome to Royal Holloway!

Introduction

Today we are going to be working on an XNA application. XNA is used for making high-speed graphical games and simulations in 2d and 3d. Anything you write in XNA can be deployed to a real Xbox!! XNA is actually part of the Microsoft .Net Framework which is a software development platform and the programming language used is C# which is similar to Java. Microsoft Visual Studio Express is the piece of software we will be using to edit, compile and run the code.

We have already created a cool starting XNA application template for you to play around with. The application models a physical system with some bouncy balls. In this session we are going to get you to change all kinds of things about this physical system like the mass, size and colour of the balls, and also the underlying forces and attributes of the physical system.
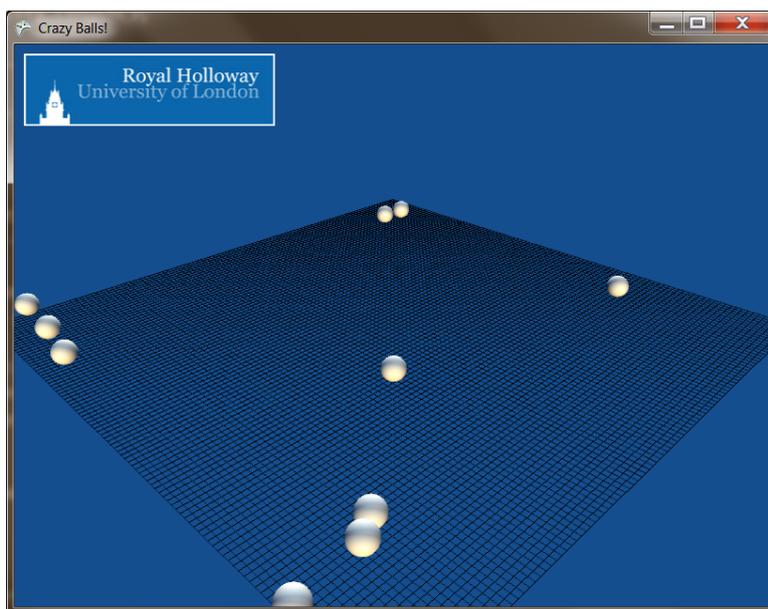
Getting the application up and running

1. Please log into the Windows machines with your given username and password.
2. When you logged in you should find a folder called **PoolGame** on your desktop. Go into this folder and double click the **PoolGame.sln.**
3. If you can't find the folder, you can download it from here: http://www.developer-x.com/dropbox/CrazyBalls.zip-- download and extract this folder to continue.
4. Visual Studio Express will fire up and load your solution.
5. Take a look at the "Solution Explorer" on the right hand side; this contains all the code files that make up the program. The files that you need to make changes in are **Game1.cs, Ball.cs, PoolTable.cs**

6. To run the application *with the debugger attached*, press **F5** or the green arrow in the tool bar at the top. **CTRL+F5** will run the application without the debugger attached which means faster performance, but obviously you can't debug code.
7. Go ahead and run the application now -- you should see the image above.
8. The cool thing is that you can make any changes you like to the code and when you run the program, these changes will be reflected immediately!
9. Start to read through the **Game1.cs, Ball.cs, PoolTable.cs** files to get a feel for what they do. I have made lots of "comments" in the files. In C# if the line starts with a double blackslash i.e. \\ or \\\ it means that the line isn't part of the program. So I have used it to explain everything that the program does.

Controlling the Application

You can change the position of the camera using the **WASD** keys on the keyboard. And you can zoom in and out using the **ZX** keys. You might find that the balls come to a natural stop -- well you can shake them back into life by pressing the **R** key!

High Level Description

The Game1.cs file is the main game class and it contains 1 important workflow or "Method" as we shall call them:

1. **SetupTable**() -- this gets called from the LoadContent method, and crucially is responsible for adding the balls onto the table. It uses the looping semantics of C# to perform the same task many times i.e. "Add a ball with these characteristics, 5 times". So play around in here to change the colours, sizes, and number of balls etc. Note that if you don't explicitly set a ball colour, a random one will be generated which might look cooler than having all the balls the same colour!

You don't need to worry about anything else in the Game1.cs. Most of it is related to how we have set up the 3d view, or boring details that you don't need to know about.

The **Ball.cs** is very important. We are using a concept called Object Oriented Programming here. And that means we have modelled everything about a ball in the *abstract*. The Game1.cs class comes along and initialized *instances* of our ball class. Think of a class as being a template, or blueprint for an object. It will set a bunch of default values, but these can be overridden. All of the state and behaviour of every ball is controlled by *properties* and *methods* in this file.

1. **public Ball(GraphicsDevice device, PoolTable table, float radius)** this is the "constructor" this is where all the default values get set for every instance of the ball class
2. **public void Update(GameTime gametime)** This is a critical method, this is where all the forces get modelled and most importantly the current **velocity** and **position** get set for the ball instance. The position is interpolated using position+timestep*velocity. The Velocity is interpolated using velocity + timestep * acceleration.
3. **PerformCollisionDetectionResponse(List<Ball> balls, PoolTable table)** This is a static method which means it's not a ball-instance level method, but rather a game-level method that deals with all of the balls at once. This method deals with checking when the balls intersect each other or the walls. When that happens, a reflection vector is calculated and the balls are bounced away by modifying their velocities.

4. **BallBounce** this sets the elasticity of the ball bounces off each other. 1 would be perfectly elastic i.e. |v_before| == |v_after|, 0.5 would mean |v_after| = |v_before|/2
5. **BallFriction** how much friction the balls encounter all the time, higher value will bring them to a halt quicker
6. **InitialSpeed** self explanatory!
7. **Mass** self explanatory but bear in mind this is just a default value. Most often it will be overridden by the value given to the instance in the Game1.cs.

In the PoolTable.cs there is one property you need to know about:

1. **WallBounce** this is the elasticity of the bounce from the side walls.

Challenges (Starting easy, getting harder!)
You don't have to do these in order, feel free to play around and have a stab at any you like.

1. Experiment with different colours on the balls. White is so boring!
2. There are 2 unused "for" loops in **SetupTable.** Why not put the other 2 to good use and create different groups of balls with different characteristics i.e. different mass, radius, colours etc. You may notice that when you change the sizes, that a third dimension is introduced! When a heavy small ball crashes into a bigger one, it is thrown up into the air! Luckily we have implemented gravity so it can come back down to earth.
3. There is a feature to get the balls to display random colours, see if you can make that work.
4. Experiment by changing the constants i.e. the initial speed, elasticity, friction, and gravitational constant.
5. Find the method where the initial random velocity is set, and add random components in three dimensions (it is currently just 2 dimensions).
6. Now getting more tricky! Try to add another force acting on the balls. First look at how we implemented gravity in the Update method in the Ball class. We have a Vector3 which represents all the forces acting on the balls (currently the friction force and gravity). Try adding in a force that makes all the balls get pulled to the side of the table.
7. Remove the dummy gravity force and implement a real gravitational force between the balls. You may want to add in a ceiling wall in the **PerformCollisionDetectionResponse** to stop them from flying too far up!
8. Try adding in an electric force that causes some balls to attract each other, and other to repel each other.