Asteroids – Session 3

Objective:
To make a simple Asteroids clone, and then extend it in an interesting way!

Getting started:
Again we will be carrying on with our code from the previous weeks. If you missed those sessions, come see me and I'll help you get started![1]

What to do:
Okay, so if you are all caught up to where we got to last week, you should have movement, bullets to shoot and asteroids. The challenge I left was to try and implement collision detection – easily the trickiest part. I hinted at 2 different ways to to it, however I will now explain the following method:

1. The method we will look at is this one I suggested last week:

> "Bounding your sprites with a circle[2] or a rectangle[3], and using the built in *intersects(Shape shape)* method to see if the two bounding boxes overlap."

To do this, we will have to slightly change the way we move our sprites (the player, bullets and asteroids) around. Every object we have will now need to have a "hitbox" (or more aptly, hitcircle) added to them which we can use to see when two objects intersect – like when a bullet and an asteroid collide.

a) Start by adding the following variable to your classes:

Circle hitbox;

Additionally, we will no longer need the x and y variables we were using to store our objects position – instead, the centre of the hitbox will represent where we want the sprite to appear.

b) In the constructor (the method that starts *public asteroid() or public bullet()*) add the following line to set up the hitbox:

hitbox = new Circle(x, y, r);

Where x and y are the centre coordinates of the circle, and r is the radius (size) you want the circle to be. You will need to work the radius out using the dimensions of the image you are using as the sprite! For example, an image 400px wide will need a radius of 200, and you must also account for **any scaling you apply**. So, if your image is 0.1 times the size, you must adjust the radius accordingly.

c) Next, you must make sure wherever you used the old x and y variables (for example the *render()* and *update()* methods), now use your hitbox instead. The hitbox has the following methods you can take advantage of:

hitbox.getCenterX() - Returns the centre x coordinate (also exists for y).
hitbox.getY() - Returns the y coordinate of the top left corner.
hitbox.setCenterX() - Sets a new centre x coordinate.

---

1 GitHub repo from last week: https://github.com/bradleypollard/GameDev1
2 http://slick.ninjacave.com/javadoc/org/newdawn/slick/geom/Circle.html
3 http://slick.ninjacave.com/javadoc/org/newdawn/slick/geom/Rectangle.html

2. Once you have got your game working exactly the same as it was before, but now making player, asteroid and bullet all use a hitbox instead of x and y coordinates, we can start implementing collision!

   a) With any hitbox, you can call the following line of code to see if that hitbox intersects another one:

   hitbox1.intersects(hitbox2);

   Now, if you have a list of asteroids and a list of bullets, you can iterate through your list to check when bullets collide with asteroids – and then define some cool behaviour to happen when they do!

   For example, in my game, when a bullet collides with an asteroid, the bullet immediately expires (and so is removed from my bullet list) and the asteroid is destroyed too (and removed from the asteroid list). Then, in its place, I create two brand new, half size asteroids which fly out in opposite directions. Try to come up with your own ideas for what could happen.

   b) Similarly, you can check if the player hits an asteroid, and then take action. A simple way to do this is to print to the console that the player has died.

3. Once you have all that, you pretty much have a game of Asteroids! Obviously, the hit detection isn't straight forward at all, so do please ask me or one of the other people going round helping if you are stuck (even if it's on a previous week's sheet!)

   If you do finally have all that implemented, the challenge this week will be to get started on your own unique extension. Our game is currently lacking a method of scoring (in other words: a reason to play it), so what you first might want to do is decide what the objective is. From there try and make your Asteroids different! Add new types of asteroids, power ups, mechanics, sound effects or game modes – whatever you can think of, because:

The best asteroids game will win a prize. Depending on how far people get today I may announce the deadline for this competition at the session!

The game you make has to be recognisably inspired by Asteroids, yet you can take as much artistic license as you like. Additionally, if you don't like Slick2D, feel free to work on it in whatever language and library you like.

Lastly, if you don't know how to do something you can always email me bp912@ic.ac.uk or post on the Facebook group https://www.facebook.com/groups/185348224901467/ .