Asteroids – Session 2

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

Getting started:
This week we will continue working from the code you started with last week. If you weren't here last week, come see me and I'll help you get started![1]

What to do:
Hopefully last week everyone was at least able to implement moving a sprite around on screen using the keyboard. This week's session will look at shooting (which I left as a challenge for people last week), and then go on to adding our enemies – the asteroids – and how you can make them interact with the player, and the bullets.

If you have already implemented shooting, skip on to part **2**!

1. Last week, I hinted that implementing shooting would require creating a new class. For those of you that haven't programmed before, here we are using a class to describe an object – in this case, the bullet our ship will shoot.
   a) To create a new class, in the package explorer on the left side of the screen, right click on the package called *game* which contains the other code files. Choose New → Class and then give it a name (for example, *bullet*) and click finish.
   b) Much like our player, our *bullet* will need an Image variable to store the sprite we will render on screen. It will also need variables to store its x and y position so we can render it, and possibly additionally a variable to store its velocity, if you used that last time.

   Make sure to put these variables at the top of the file, so they will be in scope in all of the methods we will create.
   c) When we want to shoot a bullet, we will want to create a new instance of our class. To do this, classes have a special method called a constructor. A constructor is of the following form:

   ```
   public bullet( float x, float y, float rotation ) {
    // here goes code to initialise all the classes variables
   }
   ```

   As you can see, a constructor takes any arguments you want it to – in this case, we will want to define the bullet's starting x and y position - and possibly its rotation - to be the same as that of our player's space ship. By using this constructor, we can pass these values as arguments to our bullet, and then assign them to the bullet's variables!
   d) Now, we will also want to move our bullet around, and render it on screen each frame! The easiest way to do this, is add our own *update()* and *render()* methods to our bullet class, much like the ones in *GamePlayState*.

   In the *update()* method, we can move the bullet a bit in the direction our ship is facing, and in the *render()* method we can call the *draw()* method on the bullet's Image variable we set up earlier.

   By doing this, back in *GamePlayState*, all we have to do to make our bullet appear on

1 GitHub repo from last week: https://github.com/bradleypollard/GameDev1

screen and move is call the *update()* and *render()* methods of the bullet, in the *update()* and *render()* methods of our game. Simple!

e) The last challenge with bullets is being able to create more than one. You will want to create a *List* of bullets (the syntax of this is the form *List<bullet>*). Then you can iterate through the list with a FOR loop, to render and update each bullet.

This bit is tricky, so ask for help or do some googling on how lists work if you get stuck.

2. Now you can shoot bullets, we need something to shoot them at, right? It's time to make our enemy – the terrifying asteroid. If you have got your head around making bullets, making asteroids should be easy – it's basically the same!

a) Again, we will need a new class. Why not call it *asteroid*?

b) And of course, we will want a constructor which allows us to choose where we want the asteroid to appear – or even better, randomly chooses a place for it to appear for us. The asteroid will similarly need *update()* and *render()* methods (surprise surprise), but this time we can do some fun stuff, like making the asteroid image rotate to make them look like they are spinning.

c) Ideally, you will want the asteroids to start somewhere off of the screen, then come on (and possibly in the direction of the player if you want to make it really challenging!) Also, back in *GamePlayState* you will need to keep track of all the asteroids, to check you don't spawn too many, and also to render and update them each frame, much like the bullet. I'll leave that for you to figure out.

d) Finally, at some point we are going to want to make it so we can interact with these asteroids. For now, I would create a method in the asteroid which describes what it should do when hit by a bullet. We will worry about collision detection later!

i. Create a method *public void onHit()* or something similar.

ii. In my version, I'd like to have the asteroids split into two when they get shot, to make it more exciting, however, you can have them do whatever you want! Die, explode into 10 pieces, drop a powerup – whatever you can think of.

iii. To do this, I could construct two new asteroids with the same x and y position as this one, but with opposing velocities (so they fly away from each other). Additionally, I'd scale them to 50% the size of the original asteroid. Then I just need to let *GamePlayState* know somehow that this current asteroid no longer exists, and to stop rendering and updating it (perhaps by removing it from the list of asteroids?)

3. Lastly, the challenge for this week will be implementing collision detection. Collision detection is always a pain, but the general idea is you will need to work out when a bullet has hit an asteroid, or when an asteroid has hit a player. Once you can work that out, you can easily call other methods (like *onHit()* which we defined above) to carry out the appropriate action.

Here are a couple methods you can try:

a) Find the center x and y coordinates of the sprite you are interested in, and manually check if they are within a certain range of the colliding object's x and y.

b) 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.

Good luck, and I'll try and help you if I can!

2 http://slick.ninjacave.com/javadoc/org/newdawn/slick/geom/Circle.html
3 http://slick.ninjacave.com/javadoc/org/newdawn/slick/geom/Rectangle.html