COMP222 Tutorial 4

Fun with physics

The aim of this tutorial is to familiarise with the use of the jBullet physics engine, which is part of jME.

We pick up from where we finished last time. For your convenience, you can download the Tutorial4.zip project, which contains models and code you were supposed to develop last time from http://intranet.csc.liv.ac.uk/~konev/COMP222/tutorials/Tutorial4.zip To import the project into your workspace, go *File→Import Project→From ZIP…*

The tutorial is based on the sample code supplement to one of the core books, Ruth Kusterer: *jMonkeyEngine 3.0: Beginner's guide*. PACT publishing, 2013. The code can be found online, https://github.com/jMonkeyEngine/BookSamples Specifically, we use https://github.com/jMonkeyEngine/BookSamples/blob/master/src/chapter06/PhysicsFallingBricks.java

The final code can be found at http://intranet.csc.liv.ac.uk/~konev/COMP222/tutorials/ThrowFlowers.java however, I would encourage you follow the steps one by one rather copy from mine.

By default, jME uses a Java version of the popular Bullet physics engine, http://bulletphysics.org/wordpress/ In Bullet, rigid body entities can be of one of three kinds:

- **Static.** A Static entity can interact with dynamic entities but stays stationary. We will make the table static.
- **Kinematic.** A kinematic entity interacts with the dynamic entities but is not itself subject to physics control. We will make the dancing monkey a kinematic entity.
- **Dynamic.** Dynamic entities interact with other physics bodies and are controlled by the laws of physics. We will through dynamic flowers at the monkey.

Tutorial 4 tasks:

1. Initialise the physics engine by adding the following lines to the end of the `SimpleInitApp()` method:

   ```
   bulletAppState = new BulletAppState();
   stateManager.attach(bulletAppState);
   ```

   When the SDK highlights an error, click on the light bulb and select *Create field "bulletAppState" in mygame.ThrowFlowers*

2. Make the table a static rigid physics entity.

   ```
   RigidBodyControl tControl = new RigidBodyControl(0);
   table.addControl(tControl);
   bulletAppState.getPhysicsSpace().add(tControl);
   ```

   Notice that the `RigidBodyControl` constructor takes the mass of an object as input. The value of 0 indicates that the entity is static.

3. Make the monkey a kinematic rigid physics entity.

```
RigidBodyControl mControl = new RigidBodyControl(10);
monkey.addControl(mControl);
bulletAppState.getPhysicsSpace().add(mControl);
mControl.setKinematic(true);
```

   Notice that a monkey is a body of mass 10kg controlled as before via the `simpleUpdate()` method.

4. We will through dynamic flowers at the monkey. Copy the `ActionListener` and `shootCannonBall` code from https://github.com/jMonkeyEngine/BookSamples/blob/master/src/chapter06/PhysicsFallingBricks.java

5. Refactor the code to rename `shootCannonBall` to `throwFlower` and fix any undeclared fields and missing imports issues. We will use a flower model instead of a ball (I've included a simple Blender model into the Tutorial4.zip project, but please consider modelling a flower yourself).

```
public void throwFlower() {
      Node flowerNode = (Node)
         assetManager.loadModel("Models/flower.j3o");
      flowerNode.scale(0.1f);

      flowerNode.setLocalTranslation(
         cam.getLocation());
      rootNode.attachChild(flowerNode);

      flowerNode.setShadowMode(
         RenderQueue.ShadowMode.Cast);

      RigidBodyControl flowerPhy = new
         RigidBodyControl(.5f);
      flowerNode.addControl(flowerPhy);
      flowerPhy.setLinearVelocity(
         cam.getDirection().mult(10));
      flowerPhy.setAngularVelocity(new
         Vector3f(0, 5, 0));
}
```

6. It's time to tune physics. Notice that the current implementation may suffer from interpenetration and tunnelling.

To reduce these effects, add the following two lines to the `throwFlower` method:

```
flowerPhy.setCcdSweptSphereRadius(1f);
flowerPhy.setCcdMotionThreshold(0.001f);
```

It will make the physics engine use extruded geometry rather than collision shapes for the flower. While this reduces interpenetration and tunnelling, both might still occur. They can be further addressed by making the physics engine update faster. Add the following line to the simpleInitiApp method.

```
bulletAppState.getPhysicsSpace().setAccuracy(1/300f);
```

This will make jBullet update the physics state 300 times per second. Notice that this can drastically increase the CPU load.

7. You can debug your collision shapes by adding

```
bulletAppState.getPhysicsSpace().enableDebug(
                                        assetManager);
```

to the `simpleInitApp` method.