Physics Assignment 3 – Week 3.5 Newton's Laws

Introduction

With this programming assignment, 5 *bonus points* can be gained which count towards the final course grade (see the course manual for grading details). For each assignment part, the points are indicated.

These bonus point can be gained by *signing off your solutions during the labs, at the latest during the fifth week* (week 3.7).

In addition, these assignments prepare for the final assignment, which should include most of these aspects.

For this assignment, you may not use the GXPEngine's MoveUntilCollision or TimeOfImpact methods.

Note: the difficulty increase in the assignment parts is steep: doing 3.1 is recommended, but 3.3 should be considered an optional extra challenge.

Assignment 3.1 Implementing a bouncing square – 1 point

For this assignment, you should start with the code sample 001_bouncing_squares_setup. As a first step, insert your own latest *Vec2* struct. You only need to work in the *Block* class for this assignment. (For uniformity and easy testing, you should not change too much in *MyGame*). Note that many debug tools are already built in to this project, e.g. by pressing space you can slow down the movement, by pressing D you can draw the trajectory, etc. For full details, see the information printed to console.

Take the following steps in the given order, and test your implementation thoroughly. These elements were all discussed in Lecture 3.

1. Detect and resolve collisions of a block with the boundaries. For now, it is sufficient to inverse the x or y component of the velocity.
2. Reset the block position when a boundary collision is detected: the block should never be shown outside of the boundaries. (But exact point-of-impact calculation is not necessary yet.)
3. Use the *bounciness* property of blocks for resolving the collision: when the value is 1, the block keeps bouncing forever, and when it is 0, the block slides along the boundary until hitting the next boundary, and then rests in a corner. (Note that a static *bounciness* field is already part of the Block class.)
4. Add (gravity based) *acceleration.* Note that *Block* already contains a static *acceleration* field, which you should use. (You can then enable and rotate the gravity using the arrow keys.)

Assignment 3.2 Point of impact – 2 point

When a block collides with the boundary, reset the position correctly. So use correct *point of impact* calculation, based on triangle ratios.

Note: when combining this with gravity, you do not need to ensure that blocks continue to slide on the ground after they stop bouncing. (Although that would be better...)

* Assignment 3.3 Multiple bouncing squares – 2 points in total

In this part of the assignment, you extend the project to correctly work with multiple bouncing squares, which bounce against each other. Note that using the number keys, you can start different test setups (with multiple blocks). It is recommended to first make a backup of your solution of Assignment 3.2. Take the following steps (for background, see the lecture slides):

1. Make sure blocks can bounce against each other: Add collision detection and resolve with other blocks. You should compute the correct point of impact, and reset the block to that position. It is ok to use *discrete* collision detection, and to resolve the collision simply by reversing the x or y component of the velocity, without considering the mass of the other object. (Recall that you may not use the GXPEngine's MoveUntilCollision or TimeOfImpact methods.) **(1 point)**
2. Use Newton's laws (preservation of momentum) to resolve block/block collisions correctly. Note that since we use (axis aligned) rectangles, you can use the one dimensional formula as explained in the lecture, either applied to the x component or the y component of the block velocities. Use the given *Mass* property for this calculation. **(1 point)**

Note: it is *not* necessary that your project works correctly when combining multiple blocks with gravity. (But in lecture 4 hints will be given on how to make that combination work.)