Modern Computer Games
COMP 521, Winter 2012

Assignment 2: Physics

Due date: Wednesday, March 7, 2012, by 6:00pm

Note: Late assignments will only be accepted with prior **written** permission of the instructor. You must **explain** all answers and **show all work** to get full marks! Please make sure your code is in a professional style: **well-commented,** properly structured, and appropriate symbol names. Marks will be very generously deducted if not!

Description

In this assignment you will investigate basic game physics as well as simple terrain generation. For this you will need to develop a 2D graphical application to display the result. You may use a language and graphical environment of your choice, but it must work on the teaching system (Linux), and you may not use any built-in features of your language for collision detection (other than basic primitives, such as line and/or box intersections if you find that useful).

1. First, you need to produce a game terrain. This will be a 2D profile of two mountains separated by a water-filled valley. The mountain profile itself should have 1D Perlin noise added to give it a more natural look (the water level can be simply represented with a straight line) **8**

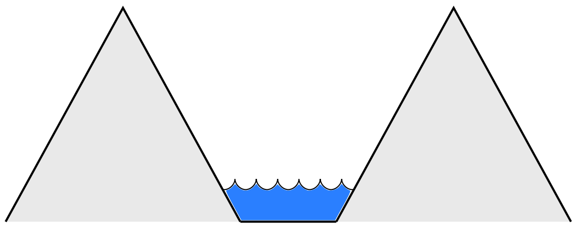

Figure 1: Basic shape without noise. You may modify proportions.

Position a cannon on each of the mountains, about 2/3 up the mountain slope and on the side facing the other mountain, with the cannon pointing (roughly) toward the other cannon. Your cannons do not have to be nicely drawn, but the elevation of the cannon barrel should be clear.

One of the cannons will be player controlled, and so you will need to listen for keyboard input. The up/down arrows should raise/lower the barrel, with the cannon barrel position restricted to the $90°$ quadrant facing up and toward the other mountain. The left/right arrows will be used to modify the initial speed of the cannonball.

2. Add to your environment the ability to fire one of the cannons when the player presses the space-bar. The cannonballs emitted as a result should be drawn as small circles (not just points) with motion modelled using projectile physics, incorporating barrel angle, gravity, wind, and wind-resistance. **5**

Determine appropriate gravity, cannonball mass, and wind-resistance parameters that allow your cannonballs to reach the corresponding point on other mountain with a $45°$ barrel angle and a default/medium muzzle speed. Cannonball flight time should be apparent to the user, taking at least $0.5s$ of real-time for a cannonball to each the other side.

A new cannonball can be fired each time the space-bar is pressed, with a short delay sufficient to ensure sequentially fired cannonballs are not immediately overlapping.

At this point there should be no wind value/direction, and cannonballs should not interact with the other mountain, cannon, or other cannonballs (ie let each ball pass through all obstacles, disappearing once off-screen).

3. Create a time-varying wind parameter, and indicate it in some fashion within the simulation. Wind strength and direction should change over (real-)time and should affect the motion of the cannonballs (both newly launched cannonballs and the ones in-flight). **4**

4. Add collision detection and responses (as detailed below) to your environment. Cannonballs can collide with the mountains, cannons, or other cannonballs.

   (a) A direct hit on a cannon with a cannonball that has not yet contacted a mountain slope should destroy the cannon (game over). **2**

   (b) A collision with a mountain slope should not result in the cannonball bouncing on impact. Instead, it should reach a full stop and then proceed to roll down the mountain slope, as accelerated by gravity. As it rolls it should collide with and bounce off imperfections in the mountain slope. Include some amount of friction to keep the cannonballs from accelerating too rapidly. A cannonball in this state should pass through (behind/in-front) of any cannon it encounters without impact. **6**

   (c) A collision with any other cannonball at any point should result in the cannonballs bouncing off of each other assuming a perfect rigid-body response (and ignoring friction). **5**

   (d) Bonus: consider conservation of momentum in cannonball collisions and correctly model head-on collisions. **(2)**
   nb: bonus marks cannot raise your total mark over 30.

   Note: All cannonballs that enter the valley water or go off-screen disappear.

   Provide a demonstration mode for this question that ensures each of the 3 (or 4) kinds of collision can be clearly visualized.

Note: As discussed in class, it is important that your code aim at the goals of efficiency and believability. This is especially important for your motion, collision detection and response calculations—assume that while you are developing this on a PC, you may need to deploy it on a machine with less computational power. Up to 25% of the total marks may be deducted for inefficiency.

What to hand in

For assignment submission use moodle: `http://moodle.cs.mcgill.ca/moodle`
For each of the above questions you should provide a separate application or clearly described options to a single application that demonstrates appropriate behaviour.

For programming questions hand in your source code code only, and include directions (readme.txt) if compilation and usage is not trivial and obvious. For non-programming questions you can provide an ASCII text file with your answers, or a .pdf file *with all fonts embedded*. Do not submit .doc or .docx files. Images (plots or scans) are acceptable in all common graphic file formats.

Note: by submitting an assignment you are declaring that it represents your own, exclusive efforts, and that all ext and code have been written by you.

This assignment is worth 15% of your final grade. **30**