EEN118 LAB THREE

The purpose of this lab is to gain some experience with interactive programming. Inputs from the user (you) will determine the behaviour of your programs. You are at a very important point in this course; the next topics to be introduced will make a lot of things much more convenient, and will open up whole new ways of programming. It is absolutely vital to be sure that you have really completely mastered the early material first. These exercises should help.

1. *Interactive Cannons*

(You may like to refer back to your solution to the last part of lab 2 to speed this up). Recall that if a cannon shoots straight upwards, the position of the cannonball is given by

$$h = vt - \tfrac{1}{2}gt^2$$

where h is the height in feet at time t seconds from firing, v is the muzzle velocity in feet per second, and g is the acceleration due to gravity, 32.174 feet per second per second.

Write a program that asks the user to enter a value for the cannon's muzzle velocity, then prints a table showing the height above ground of the cannon ball at 0, 0·1, 0·2, ..., 0·9, and 1·0 seconds after firing.

2. *Adaptive Scaling.*

The total time the cannonball spends in the air before crashing to the ground is found by setting h to zero in the above equation and solving for t, which gives:

$$t_{max} = 2v/g$$

Modify your solution to exercise 1 so that it prints the time and altitude of the cannonball at ten or eleven equally spaced times between 0 and t_{max}. That is, it should describe the whole trajectory from launch to rearrival.

3. *Bombing Someone Else.*

Sensible artillerymen do not fire their cannons straight up. They always set their cannon at an angle so that the projectile will move horizontally as well as vertically. If the angle between the cannon and the ground is α (0°=horizontal, 90°=vertical), then the position of the cannonball is given by two formulæ:

$$h = vt\sin(\alpha) - \tfrac{1}{2}gt^2$$
$$d = vt\cos(\alpha)$$

(remember that the C++ functions `sin` and `cos` take their parameters in radians, not degrees. There are π radians in 180°) Here, h is the height above ground again, and d is the horizontal distance.

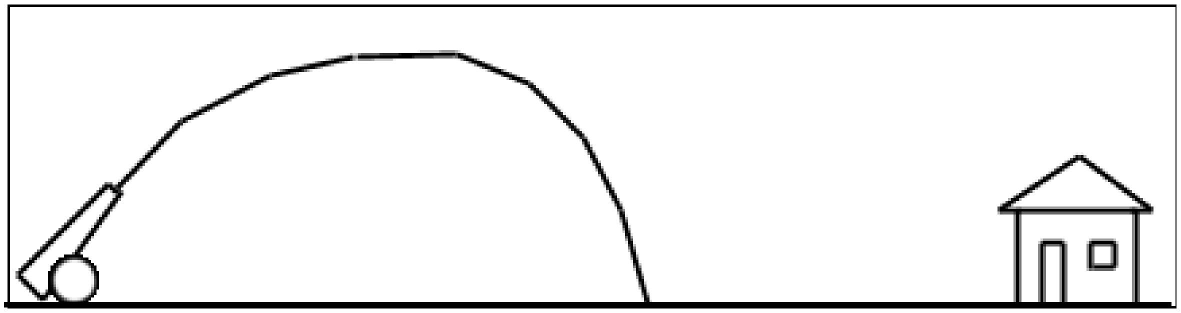
Modify your solution to part 2 so that it asks the user for two inputs: the muzzle velocity and angle (in degrees) of the cannon, then prints the time, height, and horizontal distance travelled, at ten (or more) evenly spaced times throughout the flight of the cannonball.

4. *Draw a Picture.*

   Modify your solution to exercise 3, using graphics functions to draw the trajectory of the cannonball as it would be seen by an observer standing off to the side. The results may look like the results from the end of lab 2, but in that exercise the horizontal axis was time, you were just plotting a graph. This time both axes represent position, you are drawing things as they would actually be seen.

5. *Make it a Game.*

   Modify your solution to exercise 4, so that it draws a simple representation of a cannon with the barrel at the right angle near one end of the window, and a simple representation of some kind of building near the other end of the window, so the user can attempt to select the right muzzle velocity and angle to blow up the building. There is no need to duplicate this picture, it is only provided to give an idea of what is intended.



6. *Make it a Better Game.*

   The function `random_in_range(`*`a`*`,`*`b`*`)` delivers a random number between *a* and *b*; use it to make the house appear at a slightly different position for each run.

   Make it so that each time the program is run, the user is given a few (3 to 5 seems right) chances to get the right muzzle velocity and elevation (i.e. angle). For each attempt the cannon should be erased and redrawn at the new angle, but the old trajectories should be left visible for reference.

   Make it work in *Real Time*. You have already worked out the time interval between the individual position plots; use the library function `pause(`*`d`*`)`, which makes a program pause in execution for *d* seconds (*d* is a double), to make the cannonball fly at a realistic speed.

   After each shot of the cannon, print out a message saying “Hit” or “Miss” depending on whether or not the house was hit by the cannonball.

7. *Make it Look Good (OPTIONAL for extra credit).*

   Each time the game is started, make it pick a random wind speed, and make the wind have some appreciable effect on the trajectory. You will have to do some experiments (i.e. trial and error) to get the wind effects looking right.

   Use your imagination: some scenery might be nice; perhaps colour things in too, but always remember your work is judged primarily on the cleanness and clarity of the design

8. *OPTIONAL FOR EXTRA CREDIT.*

   This was a popular game on Macintosh computers in the mid-1980s. Sell your version of it to Nintendo and become a millionaire. My cut is only 15%.