

## Lab 2:

# Someone Set Us Up the Bomb

Due date: February 5 at the end of lab

### Overview

In this lab, you will make a simple text-based game about hitting randomly placed targets by shooting bombs from a cannon. The player will repeatedly enter an angle to fire the cannon at and an amount of gunpowder to use; your game will respond with how far away from the target the player's shot landed. You will repeatedly ask for an angle and gunpowder amount until the player's shot lands within 1 meter of the target. If a shot is too long or short, the player is told how far away the shot was and tries again.

### Game Rules

1. The game begins by asking the user to enter a positive integer seed value for the game's random number generator. Use this number when creating a `default_random_engine` variable *instead of* `random_device`. The seed value must be positive before continuing. Example:  

```
int seed;  
// input and validate seed, then:  
default_random_engine engine(seed);
```
2. The target is placed at a random distance from 1 to 1000 meters using the random number generator; the distance can contain decimals. We have only seen how to generate integers in C++, so you will need to look up how to use `uniform_real_distribution` to generate a decimal number in a certain range. Tell the user how far away the target is. **Print all decimal values using 2 decimal places.**
3. If you generate the distance correctly, your output should match mine in the Sample Output below when using the same seed value.
4. Ask the player to choose an angle between 0 and 90 degrees, and then an amount of gunpowder in kilograms. Both values can be decimal numbers. Each kg of gunpowder will produce 30 m/s of velocity.
5. The shot's total distance is calculated, and if the shot is within 1.0 m of the target, the shot is scored a hit and the player wins. Otherwise the player is told how far short or long of the target the shot landed, and must try again.
6. The game ends when the player gets a hit.

### Physics

Given an initial velocity  $v$  and angle of elevation  $A$ , the bomb's initial speed can be broken into vertical and horizontal components ( $v_y$  and  $v_x$  respectively) by the equations  $v_y = v \cdot \sin(A)$  and  $v_x = v \cdot \cos(A)$ . There are two ways to find the bomb's final position:

1. Using a formula for finding the position of a projectile given its initial velocity and angle of elevation, determine the time it takes for the bomb to return to ground level. (Hint: consider the height or y-coordinate of the bomb the instant it touches the ground.)
2. Using the vertical component of the initial velocity ( $v_y$ ), determine how long it takes for the effect of gravity to reduce that velocity to zero. This is the time it takes for the bomb to reach the apex of its arc. Double that amount for the time it takes to return to the ground.

Once you have the time it takes for the bomb to hit the ground, you can find how far it travels in the horizontal direction using the horizontal component of the initial velocity ( $v_x$ ).

## Line Limits

The entirety of your program cannot exceed **55 countable lines**. All lines of code are considered “countable” except for the following:

- Empty lines
- A line containing only a single }
- A comment.

Some hints on reducing your line count:

1. Eliminate any variables that are not used to verify user input or in calculating answers. For example, you may feel a need to use an integer variable to keep track of how many tries it takes for the user to enter a value correctly, but that is not required of the program and only inflates your line count.
2. Consider when it's appropriate to use a for loop instead of a while loop; this can condense multiple lines into a single 'for' declaration.
3. Multiple variables of the same type can be declared on the same line:  
`double angle, gunpowder;`
4. If you find yourself duplicating large blocks of code only to change a single word or number of output, consider how you might accomplish the same using only one block of code.
5. You do not need a new variable for each step of a calculation. Combine multiple operations into a single statement when possible.

## Notes and Hints

1. The `<cmath>` library has methods for calculating sines and cosines, but they operate on radians and not degrees.
2. Use  $g = 9.8 \text{ m/sec}^2$  for the vertical acceleration of the bomb, and use  $\pi = 3.141592653589793238463$  if you need it.
3. Use `double` for any decimal numbers instead of `float`.
4. Having trouble deciding where to start? Try this approach.
  - (a) Build a program that sets the target's distance to 500 every time. Ask the user how far the shell should go, and tell them whether the shot was short, long, or a hit. (Goal: get the “hit” detection working.)
  - (b) Add prompts for the user to enter the angle of elevation and gunpowder amount, without any error checking. Compute the distance from those inputs. (Goal: get your formulas correct and applied to the game.)
  - (c) Repeat the game until the user gets a hit. (Goal: general game flow.)
  - (d) Add error checking for angle and gunpowder.
  - (e) Add a prompt at the start for the random number generator seed, including error checking. Use this value to select a random starting distance for the target. This should complete your project.

## Sample Output

User input is in *italics*.

```
Please enter a positive integer seed value:
0
Please enter a positive integer seed value:
5
```

The target is 222.77m away.

Please enter an angle between 0 and 90 degrees:

91

Please enter an angle between 0 and 90 degrees:

30

Please enter an amount of gunpowder in kilograms:

1

You were 143.24m short.

Please enter an angle between 0 and 90 degrees:

30

Please enter an amount of gunpowder in kilograms:

1.8

You were 34.92m long.

Please enter an angle between 0 and 90 degrees:

40

Please enter an amount of gunpowder in kilograms:

1.6

You were 19.17m short.

Please enter an angle between 0 and 90 degrees:

36

Please enter an amount of gunpowder in kilograms:

1.6

It's a hit!