Homework 3: Artillery Golf & Superman®

**Due Sunday, October 1st by 11:59pm**

There are two activities for this assignment. Please make certain you do each one and submit your work as described in the “deliverables” section.

# Activity 1: Artillery Golf

One of the earliest uses of computers was to compute ballistic tables for artillery during World War II. Nowadays you still see the use of ballistics in games. For example, in a golf or basketball game, the motion of the ball may be calculated using standard physics formulas. You will be doing a similar calculation in this homework. Note that the calculation ignores many of the actual variables in a real ballistic calculation, such as air friction, wind, and the rotation of the Earth. (For long range artillery, the rotation of the planet actually has to be factored in, due to the longer flight time of the shell.)

You are going to write the code to simulate a game of “Artillery Golf”. Given a distance to the hole (in meters), and the user’s guess of an angle and speed of the cannonball, your code will calculate how far away the cannonball will land. This requires you to take the formulas on the next page and re-write them as C# code.

The user will get a maximum of 10 tries to hit the target. If they succeed early (before the 10th attempt), of they run out of chances, the program should end.

Don’t get scared by the math on the next page! You do **NOT** have to actually solve any formulas yourself! We’re giving them to you. You simply have to make them work in C#. Your program will do the heavy lifting.

# Task Overview

This is a brief overview of the tasks you must complete for this assignment. Specifics are given in the corresponding sections later in this document. You should ***read the entire document*** before starting.

* Start with a **new C# project**
* **Prompt** the user for:
  + The **angle of the cannon** from the ground
  + The **initial velocity** of the cannonball
* Perform the **ballistics calculations** outlined on the next page
* Determine if they **hit the target**
* Allow the user to **try again** until they “win”, or until they make a total of 10 attempts

## The “Game”

Start by creating a new C# Console Application. The user will be entering values to attempt to hit a target which is 751 meters away. Prompt the user for the following two values:

* The initial **velocity** of the cannonball (in meters/sec)
* The **angle** of the cannon (**in degrees**)

Valid values for the **velocity** of the cannonball are **positive numbers** (anything greater than zero). Valid values for the **angle** of the cannon are **between** **0 and 90 degrees**. If the user gives you invalid values for either variable, keep prompting them until you get valid values. This will require looping.

## Formulas

Once you have valid values, you’ll need to plug them into the calculations outlined below:

* The formula for the *time of flight*, the result of which is used in the next step:

**t = (v \* sin(θ) + (v2 \* sin2(θ) + 20.0 \* L \* sin(θ) )1/2 ) / 10.0**

where:

v is the velocity of the cannonball (user input)

θ is the angle to the ground (the user input)

L is the length of the cannon barrel (always use the value 2.0 for this)

* The formula for the distance travelled: **d = v \* cos(θ) \* t**

The “t” variable in this distance formula is calculated by the “time of flight” formula above. This means you’ll have at least two “steps” to the overall calculation. You may want to break it up into even more steps if it helps you keep your code organized!

To save you time: **IT IS NOT POSSIBLE TO REARRANGE THE FORMULAS GIVEN ABOVE TO GET VELOCITY BY ITSELF SO YOU CAN EASILY SOLVE USING THE ANGLE.**

Some more information related to the variables in the formulas above:

* Cannonball velocity is in meters/sec, which is what the second formula requires.
* The angle of the cannon (θ) is the angle relative to the ground
  + A value of 0 means parallel to the ground
  + A value of 90 means straight up in the air
  + You’re prompting the user for the angle in degrees, but it must be **converted to radians** for the calculation. See the next page for details.
* The length of the cannon barrel should always be 2.0 meters

## Doing Math in C#

In order to do the calculations above, you will need to use several methods from C#’s built-in Math class. This class contains trigonometric operations like sine & cosine. Feel free to look on [MSDN for more information about what’s available in the Math class](https://msdn.microsoft.com/en-us/library/system.math(v=vs.110).aspx).

Some examples of the Math methods:

double result = Math.Cos(3.14159); // This finds the cosine of 3.14159 (PI)

double exponentResult = Math.Pow(5, 3); // This takes 5 and raises it to the 3rd power

One important thing to note is that the trig functions use **radians**, not degrees. You must convert any degree values to radians. The conversion is 1 degree = ∏/180 radians, so simply multiply by PI (which you can get from the Math class) and then divide by 180.

## Determining a Hit

Once you get the actual distance that the cannonball travels (from the formulas on the previous page), compare that distance to the target’s distance (751 meters). You can assume the target is a meter wide, so anything that hits from 750.5 to 751.5 is successful. Report back to the user if they hit the target or not. There are multiple sets of correct values to get a valid distance.

For instance: an angle of **51 degrees** and a velocity of **87.55 m/s** should result in almost exactly 751 meters (751.006915407272 to be exact). Don’t forget to convert from degrees to radians! Using 51 radians will not work correctly.

## Trying Again

Once you have the code above working, put it in a loop so the user can continue guessing without having to restart the program. Each time through the loop, report which attempt the user is taking by printing a number. The first attempt is 1, the second is 2, etc. See the sample output on the next page for an example of what this should look like.

The user should be given a maximum of 10 tries. After the 10th attempt, the program should end.

Each time through the loop, the user should (hopefully) try a different velocity and/or angle to see if those values result in a hit. The loop should end when the user’s input is within the appropriate target distance listed above.

Be sure to print the result of the shot each time, including the difference between where the shot hit and the target.

## Sample Run

Welcome to Artillery Golf!   
Your goal is to hit a target 751 meters away.

Attempt 1 ----------------------------------

Enter the cannon’s angle (between 0 and 90 degrees): **96.7**That is an invalid angle.  
Enter the cannon’s angle (between 0 and 90 degrees): **62.7**

Enter the cannonball’s initial velocity (a positive number): **-22.6**  
That is an invalid velocity.  
Enter the cannonball’s initial velocity (a positive number): **100.0**

Thank you.

A cannonball fired with an initial velocity of 100.0 m/s, at an angle of 62.7 degrees from the ground, will strike the ground 816.044064877365 meters away.

You’re 65.044064877365 meters from the target. Try again!

Attempt 2 ----------------------------------

Enter the cannon’s angle (between 0 and 90 degrees): **51.0**

Enter the cannonball’s initial velocity (a positive number): **87.55**

Thank you.

A cannonball fired with an initial velocity of 87.55 m/s, at an angle of 51.0 degrees from the ground, will strike the ground 751.006915407272 meters away.

You’re 0.006915407272 meters from the target. A successful hit!

Thanks for playing!

Press any key to continue . . .

If the user runs out of guesses:

Ran out of attempts, but thanks for playing!

Press any key to continue . . .

# act001sActivity 2: Leaps Tall Buildings in a Single Bound

As long as we are talking about physics, let’s consider physics as seen in comic books. In the original Superman comic books in the 1930s, he could leap tall buildings in a single bound. This required that he be able to jump very quickly, since Superman sort of followed the laws of physics. (Later, in the 50’s the writers came up with a variety of other powers, including flight, X-ray vision, time travel, etc. that had no physical basis.)

You’re going to write a small program that determines the required velocity for Superman to clear a tall building on various planets with different gravitational constants.

# Task Overview

This is a brief overview of the tasks you must complete for this assignment. Specifics are given in the corresponding sections later in this document. You should ***read the entire document*** before starting.

* Start with **another new C# project**
* Prompt the user for a **gravitational constant** (in feet/sec2)
* **Calculate** Superman’s jump velocity to clear a 660 foot tall building
* Ask theuser if they’d like to **enter a new value** or **quit**

## The Program

Start by creating another new C# Console Application. No need to make a copy of the first activity – this should be a separate Visual Studio project.

Start by prompting the user for a gravitational constant, reminding them that it’s in feet/sec2.

Valid values for gravitational constants are **positive numbers** (anything greater than zero). If the user gives you an invalid value, keep prompting them until you get a valid one.

Once you have their input, use it in the formula on the next page.

## Formula

Calculate the required initial velocity for a successful leap over a tall building, using the following formula:

**v2 = 2 \* g \* h**

where:

g is the gravitational constant for the planet

h is the height in feet of the building (660 feet)

v is the initial velocity required to clear the building in feet/sec.

Note that you will also need to use the Math class again. Look through the Math class for an appropriate method to get the correct velocity value; the formula solves for velocity squared, but how could you isolate velocity?

## Displaying the Results

Once you’ve calculated the initial velocity, display it to the user. Below you’ll find a table of gravitational constants (in feet/sec2) of various planets and moons, and the corresponding initial velocities for leaping over a building 660 feet high. Use these to test your code.

|  |  |  |
| --- | --- | --- |
| **Planet** | **g (feet/sec2)** | **Initial Velocity** |
| Venus | 29.1 | 195.989795652733 |
| Earth | 32.1 | 205.844601580901 |
| Moon | 5.3 | 83.6420946653059 |
| Mars | 12.4 | 127.937484733756 |
| Jupiter | 85.1 | 335.159663444156 |

## Trying Again

Once you have the code above working, put it in a loop so the user can continue entering new values without having to restart the program.

After displaying the results, ask the user if they’d like to enter a new value. Be sure to denote what kind of input you expect from them (“yes”/”no”, “Y”/”N”, etc.). If they respond yes, they should be taken back to the top of the loop to enter a new value. If they respond no, then the loop (and the overall program) should end.

## Sample run

Welcome to the Superman Jump simulator!

Your target building height is 660 feet.

Please enter the gravitational constant for the planet on which Superman is currently attempting this jump. Remember that the units should be in feet/second^2.

Gravitational Constant: -3.1415  
That is an invalid value. It must be a positive number.

Gravitational Constant: 29.1  
Superman must jump with an initial velocity of at least 195.989795652733 feet/second.

Would you like to try another value? (Y/N): Y

Gravitational Constant: 32.1  
Superman must jump with an initial velocity of at least 205.844601580901 feet/second.

Would you like to try another value? (Y/N): N

Thanks!

Press any key to continue . . .

# Deliverables:

Turn in all of the homework materials in a single zip file.

|  |  |  |
| --- | --- | --- |
| **Activity #** | **What to submit** | **Grading** |
| 1 | A folder in the zip with your code for Activity 1 | 60% |
| 2 | A folder in the zip with your code for Activity 2 | 40% |