Illes BOGNAR

NEA

Illes Bognar

Table of Contents

[Background to and Identification of the Problem 3](#_Toc95012997)

[Description of Current System 3](#_Toc95012998)

[Ophysics Simulation 3](#_Toc95012999)

[PhET simulation 4](#_Toc95013000)

[How I Want To Improve The Current Systems 5](#_Toc95013001)

[Planets 5](#_Toc95013002)

[Identification of prospective users 5](#_Toc95013003)

[Limitations 5](#_Toc95013004)

[Object Analysis Diagram Draft 6](#_Toc95013005)

[Numbered Objectives of the project 7](#_Toc95013006)

[Extension objectives 10](#_Toc95013007)

[Proposed Method of Solution 10](#_Toc95013008)

[Design 11](#_Toc95013009)

[Modular Structure of the System 11](#_Toc95013010)

[Data Dictionary 11](#_Toc95013011)

[Object Analysis Diagram 20](#_Toc95013012)

[Description of Algorithms and User Interface 21](#_Toc95013013)

[Menu Form 21](#_Toc95013014)

[Practice Form (Main Form) 23](#_Toc95013015)

[Calculations 24](#_Toc95013016)

[ShapeCreator Form 30](#_Toc95013017)

[ShapeLoad Form 34](#_Toc95013018)

[ValueLoader Form 36](#_Toc95013019)

[Technical Solution 37](#_Toc95013020)

[List of Functions and classes 37](#_Toc95013021)

[Menu 43](#_Toc95013022)

[Practice 46](#_Toc95013023)

[ShapeCreator 63](#_Toc95013024)

[ShapeLoad 79](#_Toc95013025)

[LaunchLoad 84](#_Toc95013026)

[Classes 86](#_Toc95013027)

[Testing 0](#_Toc95013028)

[Testing Plan 0](#_Toc95013029)

[Evaluation 0](#_Toc95013030)

[Evaluation against objectives 0](#_Toc95013031)

[Overall assessment of the project 0](#_Toc95013032)

[User feedback 0](#_Toc95013033)

[Analysis of User feedback 0](#_Toc95013034)

[Possible extensions and improvements 0](#_Toc95013035)

**Analysis**

## Background to and Identification of the Problem

I want to create a simulation of projectile motion. I do A-level physics and want to create a simulation for students like myself so visualise projectile motion. Projectile motion is the study of the motion of objects under gravity. Kinematics is a subfield of mechanics which revolves around the motion of objects without considering the forces that act on them. In A-level physics and Maths, the most students learn is to calculate the velocity and displacement of a dimensionless shape acting under gravity, based on time. This means that the acceleration of the objects must be constant and the only force acting on them is gravity. By ignoring the dimensions of an object, we can ignore other external forces that could act on them, such as Drag. This makes the calculations much more basic.

What I want to do is create a simulation of projectile motion that allows the user to do both dimensionless calculations where the only force is acting is gravity and calculations that involve the dimension of the shape where Drag is also accounted for. I want the user to be able to create their own shapes with their own custom values, these values will be used to calculate the velocity and displacement of the object at a given time.

I will be using current systems to identify many of the key features that they all share, such as adjustable values and an animation of the object’s motion. This will provide me with ideas for my system.

## Description of Current System

There are three systems / designs that I want to highlight. My design will mainly be based on <https://ophysics.com/k8.html> which is an interactive physics simulation of projectile motion and the PhET online interactive simulation of Projectile Motion.

### Ophysics Simulation

Chart, line chart

Description automatically generated

Firstly, the physics simulation is very simplistic. It takes in values of either the initial velocity and launch angle or the initial values of the horizontal and vertical components of the velocity. Furthermore, there are other sliders which allow you to adjust the acceleration due to gravity and add an initial height to the object. More importantly there is a slider for time which allows you to adjust the time. There are buttons which allow you to Fire, Pause and resume the motion of the object. The animation is simple with blue arrows clearly labelled as the horizontal, vertical and overall velocity of the ball and a clear green angle which shows the initial launch angle. All of this is over a graph showing horizontal and vertical displacement. Overall is a good and simplistic design however in my opinion the information displayed is too little and there is not much you can do with the simulation and not much data can be gathered from it.

From this system, I would mainly like the UI, its easy to understand and easy to use. Furthermore, the sliders are a great way to adjust values. I would like to incorporate both the sliders and a similar UI in my project. I want to have Zoom In and Zoom Out buttons similar to this project so that my user can see the full motion of the object regardless of its displacement. I would also like to have arrows for the Velocities as a visual display like this system.

### PhET simulation

Graphical user interface, application

Description automatically generated

The PhET simulation is a lot more complex. This system allows the use to adjust the values of initial speed, angle, mass, diameter of the ball and gravitational field strength, altitude (fluid density) and height. There is a target which the user can move to different locations to measure the distance travelled and an inspect gadget which can be used to see height and Range at certain times. The UI is overall too crowded and has some unnecessary features in my opinion.

The main things I want to borrow from this simulation is the Drag as the other simulation does not use drag. I want to make it easier for the user to understand what values they are adjusting and how that changes the outcome of the result. I want my program to be educational so I will include text boxes to explain each value. For example, in this program the user can adjust altitude, most people wouldn’t know that altitude changes the fluid density of air, for my program I want to make sure that the user understands each value they input.

## How I Want To Improve The Current Systems

I want to create a clean and simple user interface which will take in initial velocity, initial angle, mass and dimensions of the object as parameters and have a simple grid in the background showing displacement vertically and horizontally. I want the user to be able to launch the object at the desired values and receive accurate readings of the horizontal and vertical components of the ball, its overall velocity, its displacement in the x and y directions calculations based on these values. I want the user to be able to pause the simulation at any time for readings. I want the user to create their own shapes which can later be loaded and launched under different conditions. Most importantly I want the user to save the shapes they want and save the values that they were launched in, this is something that other system don’t allow. I want the user to be able to alter gravitational field strength depending on the planet they select.

## Planets

Only two values will depend on planets, gravity and fluid density. Gravity is easy to get a value for however fluid density is a little more complicated. Fluid density is the how close the particles in the given fluid are, a fluid can be anything from air to water to hydrogen. Fluid density is usually measured at the surface of planets, however some planets such as Jupiter do not have a surface as they are gas giants. This means values for density must be estimated based on the distance from the core of the planet. I found a great website that contains information about planets, they use density at 1 bar which means that pressure is assumed to be slightly less than the current average atmospheric pressure on Earth at sea level. With this being considered the values I will be using are...

|  |  |  |
| --- | --- | --- |
| Name | Gravitational field Strength (m/s) | Fluid Density (kg/m) |
| Mercury | -3.7 | 0 |
| Venus | -8.87 | 67 |
| Earth | -9.81 | 1.225 |
| Moon | -1.62 | 0 |
| Mars | -3.721 | 0.02 |
| Jupiter | -24.79 | 0.16 |
| Saturn | -10.44 | 0.19 |
| Uranus | -8.69 | 0.42 |
| Neptune | -11 | 0.45 |
| Pluto | -0.66 | 0 |

http://btc.montana.edu/ceres/malcolm/cd/html/orbitsfacts.html#tres

## Identification of prospective users

My core users will be A-level physics students at Xaverian College. For this reason, I want to make sure that my program is easy to use and more importantly educational. I want to make sure that my program displays accurate results for projectile motion so that students can use it to learn about projectile motion.

## Limitations

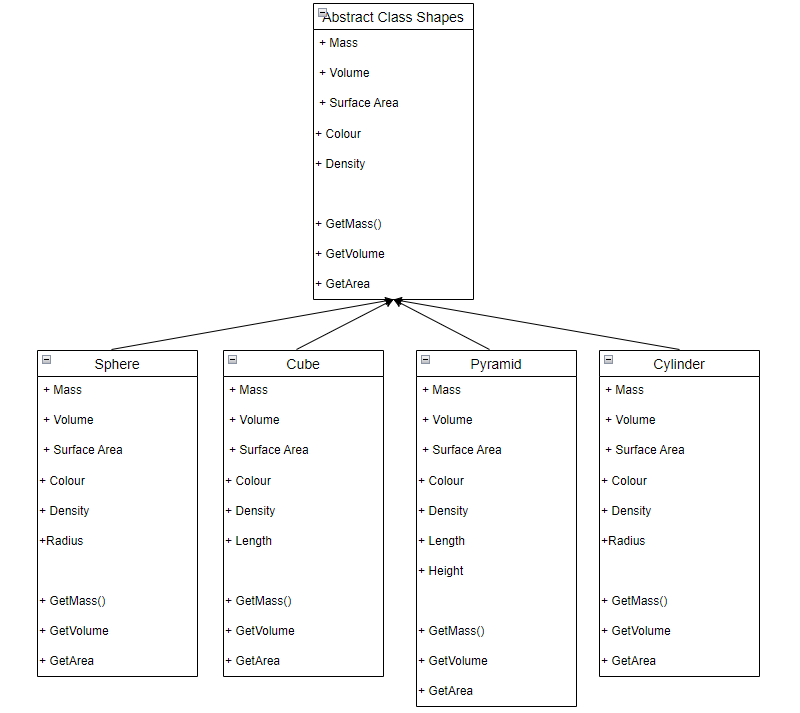
Not everything can be done, modelling our real life is extremely difficult so many aspects of may not be included. Many physics calculations at A-level are done in an ideal environment without the concepts of drag or air resistance. Many times, factors such are impact on the object landing and rebounds aren't considered. I will try to implement as many of these features as I can however my simulation cannot even come close to reality. Furthermore, I want to make program user friendly which means I may have to simplify some information on the screen. A more technical limitation is that negative values cannot be used for initial Velocity and Angle as those conditions don’t happen often in the real world. I would prefer the user to only launch the object starting from the left to the right, this means I will have to limit the angle at which they can launch the object between 0° and 90°. Furthermore, I will also have to limit the speed at which the object can travel at, velocity cannot be 0ms-1 as the object will not move and velocity cannot be greater than 299 792 458 ms-1 as that exceeds the speed of light which is physically impossible.

A technical limitation is presented by the timer in C# Forms. The timer can increment at set intervals (1 second, 0.1 second, 0.01 second). However, it also takes time for the program to run calculations and create a new value to move the object to. This means that the timer will never be exactly accurate to real life. Furthermore, if the interval is too little the animation will become noticeably slower and inaccurate. I will have to find an interval that still looks smooth for the animation and provides enough readings for the user.

## Object Analysis Diagram Draft

I want to use OOP to calculate and assign properties to shapes. I want to have an abstract base class called Shapes. Shapes will store some basic properties such as Mass, Volume, Density, frontal surface area and colour. Shapes will store some basic methods such as GetMass(), GetVolume() and GetArea(). These methods will then be inherited into more specific classes of shapes, such as Pyramid, Sphere, Cube and Cylinder. When inherited the methods will need to be overridden as the base class is abstract. When the user selects a shape to draw, the specific shape class will be instantiated and use the values entered by the user to calculate the Mass, Area and Volume of the object.

It would look something like this...

I want all the Methods and Properties to be public so they can be accessed in whatever Form i need them in. ShapeCreator will be the Form that uses these classes the most.

Data Volumes

For this project, I will store information on Shapes and Launch values based on the user they are stored under. There will be no limit on the number of Shapes and Launch Values a user can create or how many users can be created. However, I will not create many records as it will not be necessary to properly test my system and showcase all its features.

The Database is only amended when values are inserted into tables, for example, when an account is created, or values of a launch are being saved. However, when a new shape is created quite a lot of properties must be saved into 2 different tables.

## Numbered Objectives of the project

1. **When the program is first launched and database doesn’t exist, set up database with all the required tables.**
   1. Create Table Accounts which stores Usernames and Passwords.
   2. Create Table for Shapes which stores basic properties that all shapes share.
   3. Create individual tables for each shape with its own unique properties.
   4. Create table for Launch values.
2. **First Form to load is Menu, where the user can create their account.**
   1. User should be able to type in any password, when a character is typed password should display \*.
   2. If the input username already exists clear the TextBoxes and don’t store values into database when user presses save button.
   3. Input username must be longer than 7 characters and password and username must be less than 50.
   4. Ask the user to input a username that isn’t taken.
3. **Allow the user to login using an account stored in the database.** 
   1. When logging in check if Username exists in the database, if not output MessageBox saying so.
   2. If username exists check if password associated with the username matches with the input password.
   3. If password doesn’t match output MessageBox saying so.
   4. If password matches open Practice form.
4. **Practice Form should display input fields and a graph with an object on it.**
   1. Inputs such as Velocity and Angle can be typed to.
   2. Validate inputs so that only numbers can be input.
   3. Further validation to check if the initial velocity is smaller than 299 792 458 and greater than 0 below m/s.
   4. Validate angle so that it’s between 0 and 90 degrees inclusive.
5. **Practice Form should have information buttons that help the user understand physics terminology if they aren’t familiar with it.**
6. **Practice Form should have a combobox with different planets stored on it.**
   1. User can pick between different planets and the values of Gravity and fluid density adjust depending on the selected planet.
   2. User should not be able to edit any of the values that are associated with the combobox.
   3. Launch can only happen if a planet is selected.
7. **Practice Form should have a radiobutton which can toggle Drag.**
   1. When pressed display variables that drag depends on.
   2. Values such as Mass, Area, Drag Coefficient and Fluid Density are displayed.
   3. If no shape is loaded but drag is enabled, still do not use drag.
8. **When variables are filled out, the user can press LAUNCH to begin animation.**
   1. If some variables are missing inform the user to fill out all values.
   2. Based on a timer, move the graphic of the ball and display appropriate velocities and displacements.
   3. Perform suvat calculation every 0.1 seconds to calculate displacement horizontally and vertically (Extra details on Calculations section).
   4. Move location of the object until timer hits 30 seconds or the object hits the ground.
   5. Disable all input fields so that the user cannot adjust values mid animation.
   6. Values can only change when ball is reset (time = 0 seconds).
9. **When launching without drag program should calculate the Range and Max Height reached by the object.**
   1. Details on Max Height and Range section.
10. **Allow the user to stop, start and reset the animation.**
    1. After user launches show ‘PAUSE’ instead of ‘LAUNCH’.
    2. When pause is pressed stop the calculations and ball freezes with appropriate values displayed for that time.
    3. The user can press ‘LAUNCH’ again to continue the animation.
    4. At any point the user can press ‘RESET’ to set the ball back to its original position and set the timer back to 0 seconds where then the user can launch again or set new values.
11. **Allow the user to zoom in/out of the graphic.**
    1. Have a zoom function that changes the size of the ball and the distance markers on give the perception that the camera is zooming in or out.
    2. User should be able to Zoom in and out dynamically with certain limits.
    3. The minimum interval should be 10m between each marker.
    4. The maximum interval should be as close to Int32.MaxValue as possible.
    5. Position of the ball should remain the same relative to the distance markers no matter what.
    6. Zoom should depend on a zoom counter which is increased or decreased depending on if the user presses zoom in or zoom out.
12. **In ShapeCreator Form, the user can select from a variety of shapes and input custom values to them.** 
    1. When a shape is selected, the user should input the dimensions of the shape and select a material which the shape is made of.
    2. The program then calculates values for surface area, Volume and Mass of the object.
    3. The user can keep the shape’s original Drag Coefficient or make it custom.
    4. Drag Coefficient should not be greater than 100.
13. **When ShapeCreator is loaded, load colours into a combobox.**
    1. Colour can be selected from the combobox and the program will draw the shape in that colour.
14. **Once all the values for a shape are input the program draws the given shape.**
    1. Measurements of the shape should not be greater than 1000m each.
15. **Allow the user to save Properties of a shape.**
    1. If any inputs are equal to 0 tell user and don’t save the shape.
    2. If shape doesn’t have a name or name is already taken don’t input values.
    3. Save universal properties into Shapes database.
    4. Save Shape specific properties into the shape’s own table (e.g. Sphere’s radius).
    5. User can name the shape.
    6. Name of the Shape links two tables together.
16. **‘Load Shape’ button allows user to load a specific shape that was created on that account.**
    1. Allow the user to select shapes from a data table.
    2. If user selects a row from a table of shapes a ‘Load shape’ button should appear.
    3. If clicked, all the properties of the shape should be loaded into Practice Form.
17. **If a shape is loaded, then user should be able to press ‘LAUNCH’ with drag enabled.**
    1. Perform Drag calculations.
    2. Calculate Drag Force Vertically and Horizontally.
    3. Calculate vertical and horizontal deceleration due to drag.
    4. Use SUVAT every 0.1 seconds to calculate the displacement.
       1. More information on calculations section.
18. **When a shape is loaded and launched find max height and range.**
    1. Compare previous vertical and horizontal displacements to current ones.
    2. If current ones are larger than previous ones, replace values.
    3. Display new values.
19. **Once a full animation has been completed (timer hits 30 seconds or ball hits the ground), the user should be able to save input values for the given launch.**
    1. Data of the Lauch should be stored in a database.
    2. Initial velocity, angle, if it hit the ground or not, range, max height and if drag was enabled or not.
    3. If drag was enabled save the shape which was launched in the table.
    4. If drag was disabled save a dimensionless object.
20. **Saved Launch values can be later loaded and compared.**
    1. When user clicks on a row of values, a Load button should appear.
    2. Once a user has selected a row they cannot unselect only click on a different row.
    3. Allow user to click Load button which will take Launch Values from database and load them into textboxes in Practice Form.
    4. If LaunchValues contains a specific shape, program should load shape into the Practice Form.

## Extension objectives

1. Have an option for the user to toggle between 2D and 3D.
2. Add a ‘wind’ value which creates a wind force in a set direction.
3. Wind will create a rotation on 3D object which I want to resolve using quaternions.

## Proposed Method of Solution

For my project I want to use C# in Visual Studios so I can create Windows Forms. This will allow me to have a visual representation of the object being launched through the System.Drawing Graphics Class and a timer which increments regularly.

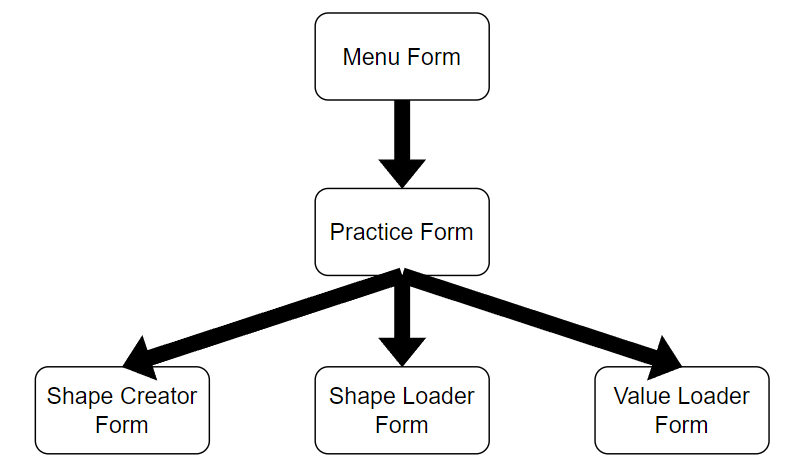
Visual studios will also provide useful ways for the user to input values and for me to display values through Labels, TextBoxes and DataGrid Views which can be linked to tables and displayed values from a database and more importantly load values from a database.

I want to use C# as it is the language, I am the most comfortable with and it allows me to create classes and utilise OOP.

For my database I want to use MS Access and use SQL statements to store and load values from tables.

# **Design**

## Modular Structure of the System



E-R Diagram

Accounts (**Username**, Password)

LaunchValues(**LaunchID**, Username, InitialVelocity, Angle, ShapeName, IsDragOn, WasGroundHit, FluidDensity, MaxHeight, MaxRange, TimeTaken )

Shapes(**ShapeName**, Username, ShapeType, Mass, Surface Area, Drag Coefficient, Colour)

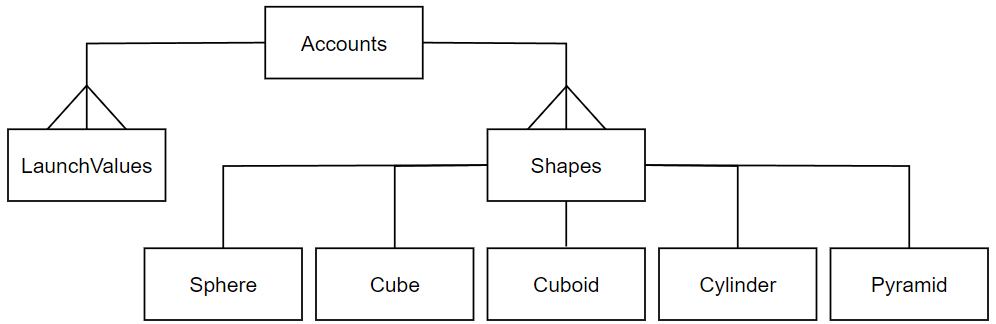
Circle (**CircleID**, ShapeName, Radius)

Cube(**CubeID**, ShapeName ,Length)

Cuboid(**CuboidID**, ShapeName, Length, Width, Height)

Cylinder(**CylinderID**, ShapeName, Length, Radius)

Pyramid(**PyramidID**, ShapeName, Length, Height)



Data Dictionary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Table name: Accounts – Stored as an Access Table*** | | | | ***Primary key: Username*** | |
| Field Name | Data Type | Length | Validation | Example Data | Comment |
| Username | VARCHAR | 50 | • Presence Check  • Length Check  • Uniqueness Check  • Length Check, must be longer than 7 characters | IllesB123 | The username of the account must be unique as it’s the primary key. |
| Password | VARCHAR | 50 | • Presence Check | Password1 | Each Account must have a password to be able to log in. |

When the user starts the program, they are prompted to create an account, all they need is a unique username and password. This is so that values for Launch and shapes can be stored under the Account. This is done so that there is no confusion between any values stored on one account and values stored on another.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Table name: Shapes – Stored as an Access Table*** | | | | ***Primary key: ShapeName*** | |
| Field Name | Data Type | Length | Validation | Example Data | Comment |
| Username | VARCHAR | 50 | • Presence Check  • Length Check  • Uniqueness Check | IllesB123 | This username must match with one in accounts, used to link the two tables together as shapes are stored under an account. |
| ShapeName | VARCHAR | 50 | • Presence Check  • Length Check  • Uniqueness Check | PurpleSphere7 | This is the Unique identifier for a shape. This is referenced in other tables. |
| ShapeType | VARCHAR | 8 |  | Cylinder | This determines what shape the object is, cube/cuboid/sphere/pyramid/cylinder. This is used to select values from other tables, so the program doesn’t have to search all tables for a name.  Cannot be CHAR |
| Colour | VARCHAR | 50 | • Presence Check | Cyan | Colour refers to an object from the Color class in C# which can be used to draw shapes. Without colour program can’t draw shape. |
| Area | FLOAT | - | • Presence Check  • Length Check must be greater than 0. | 1809.56 | Area refers to the surface area which air particles are in contact with during the flight of an object (Frontal Area). |
| Mass | FLOAT | - | • Presence Check  • Length Check must be greater than 0. | 79.32 | This refers to how heavy the object is based on its density and volume. Those two values aren’t stored in the database as they aren’t necessary for calculations. |
| DragCo | FLOAT | - | • Presence Check  • Length Check must be greater than 0 and smaller than 100. | 0.47 | the drag coefficient is a dimensionless quantity that aerodynamicists use to model all the complex dependencies of shape, inclination, and flow conditions on objects.  This value is used in calculations. |

The shapes table stores all the universal values of a shape. All shapes must be named, have a colour, Area, Mass and Drag Coefficient. Values such as density could also be stored however it is not required for calculating Drag.

Username is a foreign key here. This is because I want the user to only select shapes that were created on their account, this makes it easier to identify shapes and doesn’t make things confusing with too many shapes being in one table.

ShapeType will be set depending on what shape is created, for example a sphere or a pyramid. This can be used to only load objects that are a specific shape. It will identify which table values need to be loaded from so that the program doesn’t have to look through each table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Table name: Circle – Stored as an Access Table*** | | | | ***Primary key: CircleID*** | |
| Field Name | Data Type | Length | Validation | Example Data | Comment |
| CircleID | AUTOINCREMENT  INTEGER | - |  | 17 | This is the primary key. Its only function is to act as a unique identifier to the shape. |
| ShapeName | VARCHAR | 50 | • Presence Check  • Length Check  • Uniqueness Check | PurpleSphere7 | ShapeName is used to link the shapes properties in Shapes table so that all its necessary properties can be loaded. |
| Radius | FLOAT | - | • Presence Check  • Length Check, the value cannot be greater than 1000 or and must be above 0. | 3.88 | This value is used to display the shape accurately. This value is loaded from the table and the size of the Sphere being displayed will be according to it. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Table name: Cube– Stored as an Access Table*** | | | | ***Primary key: CubeID*** | |
| Field Name | Data Type | Length | Validation | Example Data | Comment |
| CubeID | AUTOINCREMENT  INTEGER | - |  | 21 | This is the primary key. Its only function is to act as a unique identifier to the shape. |
| ShapeName | VARCHAR | 50 | • Presence Check  • Length Check  • Uniqueness Check | GreenCube7 | ShapeName is used to link the shapes properties in Shapes table so that all its necessary properties can be loaded. |
| Length | FLOAT | - | • Presence Check  • Length Check, the value cannot be greater than 1000 or and must be above 0. | 1.42 | This value is used to draw the shape accurately. This value is loaded from the table and the size of the Cube being displayed will be according to it. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Table name: Cuboid– Stored as an Access Table*** | | | | ***Primary key: CuboidID*** | |
| Field Name | Data Type | Length | Validation | Example Data | Comment |
| CuboidID | AUTOINCREMENT  INTEGER | - |  | 2 | This is the primary key. Its only function is to act as a unique identifier to the shape. |
| ShapeName | VARCHAR | 50 | • Presence Check  • Length Check  • Uniqueness Check | BlueCuboid46 | ShapeName is used to link the shapes properties in Shapes table so that all its necessary properties can be loaded. |
| Length | FLOAT | - | • Presence Check  • Length Check, the value cannot be greater than 1000 or and must be above 0. | 5.124 | This value is used to draw the shape accurately. This value is loaded from the table and the size of the Cuboid being displayed in the horizontal plane will be according to it. |
| Width | FLOAT | - | • Presence Check  • Length Check, the value cannot be greater than 1000 or and must be above 0. | 12.14 | Width is not used in any way other than to display in the table since the shapes are represented in 2D, only length and height are used to draw the shape. |
| Height | FLOAT | - | • Presence Check  • Length Check, the value cannot be greater than 1000 or and must be above 0. | 28.53 | This value is used to draw the shape accurately. This value is loaded from the table and the size of the Cuboid being displayed in the vertical plane will be according to it. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Table name: Cylinder – Stored as an Access Table*** | | | | ***Primary key: CylinderID*** | |
| Field Name | Data Type | Length | Validation | Example Data | Comment |
| CylinderID | AUTOINCREMENT  INTEGER | - |  | 86 | This is the primary key. Its only function is to act as a unique identifier to the shape. |
| ShapeName | VARCHAR | 50 | • Presence Check  • Length Check  • Uniqueness Check | CyanCylinder9 | ShapeName is used to link the shapes properties in Shapes table so that all its necessary properties can be loaded. |
| Length | FLOAT | - | • Presence Check  • Length Check, the value cannot be greater than 1000 or and must be above 0. | 1.42 | This value is used to display the shape accurately. This value is loaded from the table and the size of the Cylinder being displayed in the horizontal plane will be according to it. |
| Radius | FLOAT | - | • Presence Check  • Length Check, the value cannot be greater than 1000 or and must be above 0. | 1.5 | This value is used to display the shape accurately. This value is loaded from the table and the size of the Cylinder being displayed in the vertical plane will be according to it. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Table name: Pyramid – Stored as an Access Table*** | | | | ***Primary key: PyramidID*** | |
| Field Name | Data Type | Length | Validation | Example Data | Comment |
| PyramidID | AUTOINCREMENT  INTEGER | - |  | 17 | This is the primary key. Its only function is to act as a unique identifier to the shape. |
| ShapeName | VARCHAR | 50 | • Presence Check  • Length Check  • Uniqueness Check | BlackPyramid4 | ShapeName is used to link the shapes properties in Shapes table so that all its necessary properties can be loaded. |
| Length | FLOAT | - | • Presence Check  • Length Check, the value cannot be greater than 1000 or and must be above 0. | 10.92 | This value is used to display the shape accurately. This value is loaded from the table and the size of the Pyramid being displayed in the horizontal plane will be according to it. |
| Height | FLOAT | - | • Presence Check  • Length Check, the value cannot be greater than 1000 or and must be above 0. | 4.96 | This value is used to display the shape accurately. This value is loaded from the table and the size of the Pyramid being displayed in the vertical plane will be according to it. |

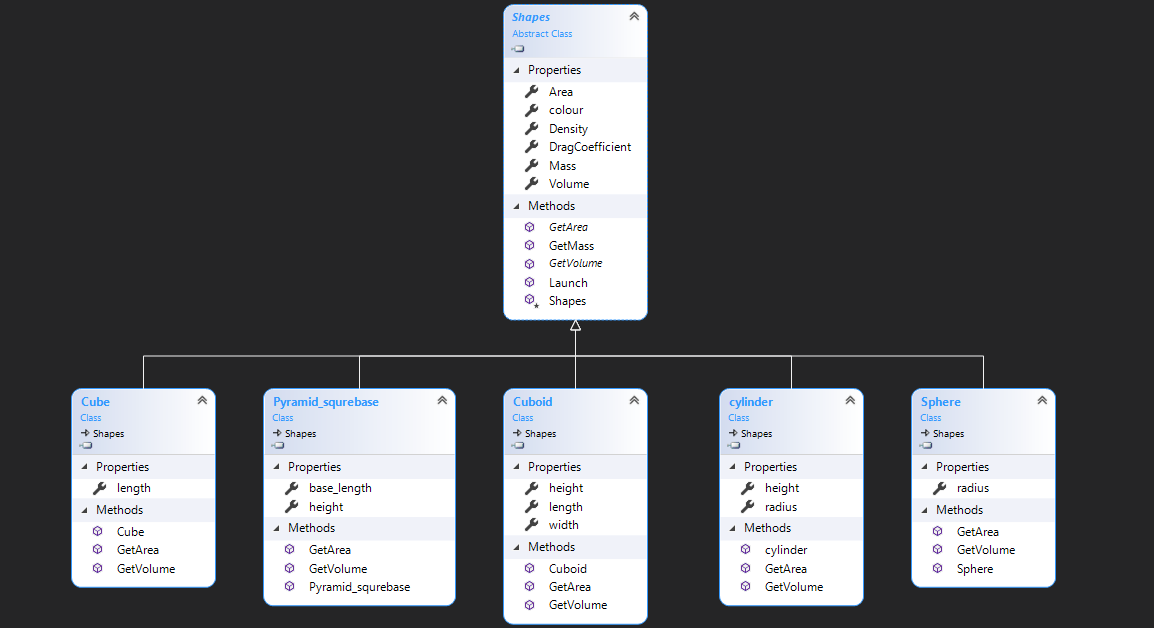
The last 5 tables are the ‘ShapeType’ tables. They store the unique information about each shape. I made these tables separate so that when a shape is being loaded the program knows which shape to load and doesn’t need to go through a large table with mainly empty fields.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Table name: LaunchValues – Stored as an Access Table*** | | | | ***Primary key: LaunchID*** | |
| Field Name | Data Type | Length | Validation | Example Data | Comment |
| LaunchID | AUTOINCREMENT  INTEGER | - |  | 64 | This is a primary key that uniquely identifies a set of values for launch. |
| Username | VARCHAR | 50 | • Presence Check  • Length Check  • Uniqueness Check | IllesB123 | This username must match with one in accounts, used to link the two tables together as Launch values are stored under an account. |
| ShapeName | VARCHAR | 50 | • Presence Check | AquaPyramid3 | This is used to link the Shapes table to the values table, the values being displayed depend on the properties of the shape that was launched, this means that a set of values must store the shape that was launched. If the launch was performed with a dimensionless object, this value becomes –DIMENSIONLESS- |
| Velocity | FLOAT | - | • Presence Check  •Length Check, must be above 0 and below 299 792 458 | 30.551 | This refers to the initial speed at which the object was launched at. It's required in both ideal calculations and ones with drag. |
| Angle | FLOAT | - | • Presence Check  •Length Check, must be above 0 and below 90 | 45.0 | This refers to the initial angle at which the object was launched at. It's required in both ideal calculations and ones with drag. |
| Gravity | FLOAT | - | • Presence Check  • Must be a negative number. | -9.81 | The value for gravity is loaded from a combobox so not much validation is needed. It must be negative as it acts against initial vertical velocity. It's required in both ideal calculations and ones with drag. |
| FluidDensity | FLOAT | - | • Presence Check | 1.225 | The object being launched is travelling in a fluid, such as air. This quantity refers to how close the particles in the given fluid are. This value is only used in in calculations for drag. |
| DragOn | BIT | - | • Presence Check | 0 (FALSE) | Determines whether to use drag, used to determine if a shape needs to be loaded or not. |
| HitGround | BIT | - | • Presence Check | 1 (TRUE) | Saves if the object hit the ground or not in the gives time period (30 seconds). It's required in both ideal calculations and ones with drag. |
| MaxHeight | FLOAT | - | • Presence Check | 824.141 | Saves the max height reached by the object in the given time (calculations with drag) or determines the max height that could have been reached (calculations without drag). |
| Range | FLOAT | - | • Presence Check | 112.865 | Saves the max distance travelled horizontally by the object in the given time (calculations with drag) if the object never hit the ground or determines the max distance from origin that could have been reached (calculations without drag). |
| Time | FLOAT | - | • Presence Check | 30 | The amount of time the object was travelling for. |

This table stores all the values for a launch. This table can be used to compare values with different calculations and determine which travels further or higher. The user will be able to load these calculations back into the Practice Form and try them out with different shapes or on a different planet.

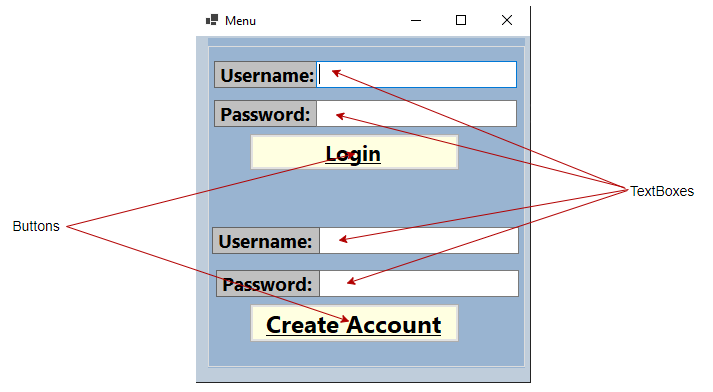
Username is a foreign key here. This is because I want the user to only select calculations that were done on their account, this also helps with identifying which shape to use if the calculations were done with a specific shape.

## Object Analysis Diagram

Here is the Object Analysis Diagram I created for my program. The only additions I’ve made here are the Drag Coefficient and a new class called Cuboid.

## Description of Algorithms and User Interface

### Menu Form



|  |
| --- |
| *TextBoxes*: tbUsername, tbPword, tbCreatePword, tbCreateUsername  *Buttons:* bLogin, bCreate |

The Menu form is the first form that should be loaded first. This Form creates the Database when it’s first launched. When the program is run a check will be made to see if the database already exists. If it does not exist, it will be created using the following DDL commands:

CREATE TABLE Accounts([Username] VARCHAR(50), [Pword] VARCHAR(50), PRIMARY KEY (Username))

CREATE TABLE Shapes([Username]VARCHAR(50),[ShapeName] VARCHAR(50),[ShapeType] CHAR(8), [Colour] VARCHAR(50), [Area] FLOAT, [Mass] FLOAT, [DragCo] FLOAT, PRIMARY KEY (ShapeName), FOREIGN KEY (Username) REFERENCES Accounts (Username))

CREATE TABLE Circle ([CircleID] AUTOINCREMENT, [ShapeName] VARCHAR(50), [Radius] FLOAT, PRIMARY KEY (CircleID), FOREIGN KEY (ShapeName) REFERENCES Shapes (ShapeName))

CREATE TABLE Cube([CubeID] AUTOINCREMENT, [ShapeName] VARCHAR(50), [Length] FLOAT, PRIMARY KEY (CubeID), FOREIGN KEY (ShapeName) REFERENCES Shapes (ShapeName))

CREATE TABLE Cuboid([CuboidID] AUTOINCREMENT, [ShapeName] VARCHAR(50), [Length] FLOAT,[Width] FLOAT, [Height] FLOAT, PRIMARY KEY (CuboidID), FOREIGN KEY (ShapeName) REFERENCES Shapes (ShapeName))

CREATE TABLE Cylinder([CylinderID] AUTOINCREMENT, [ShapeName] VARCHAR(50), [Length] FLOAT, [Radius] FLOAT, PRIMARY KEY (CylinderID), FOREIGN KEY (ShapeName) REFERENCES Shapes (ShapeName))

CREATE TABLE Pyramid([PyramidID] AUTOINCREMENT, [ShapeName] VARCHAR(50), [Length] FLOAT,[Height] FLOAT, PRIMARY KEY (PyramidID), FOREIGN KEY (ShapeName) REFERENCES Shapes (ShapeName))

CREATE TABLE LaunchValues([LaunchID] AUTOINCREMENT, [Username] VARCHAR(50),[ShapeName] VARCHAR(50), [Velocity] FLOAT, [Angle] FLOAT, [Gravity] FLOAT, [FluidDensity] FLOAT, [DragOn] BIT, [HitGround] BIT, [MaxHeight] FLOAT, [Range] FLOAT, [Time] FLOAT, PRIMARY KEY (LaunchID))

The user will be prompted to enter in their username and password. If they don’t have an Account yet they can create an account by entering their username and password into the textboxes that are above the Create Account button. Validate username and password to make sure they aren’t empty

IF Username = “” OR Password = “”

THEN SHOW MessageBox that says, “Please fill out both username and password.”

Also pass validation to make sure that both username and password are below 50 characters, and that username is at least 7 characters long...

IF Length of text in TextBoxUsername IS GREATER THAN 50 OR Length of text in TextBoxUsername IS LESS THAN 7

THEN SHOW MessageBox that says, “Make sure username is between 7 and 50 characters.”

AND CLEAR text in TextBoxUsername

And validation for password...

IF Length of text in TextBoxPassword IS GREATER THAN 50

THEN SHOW MessageBox that says, “Make sure password is smaller than 50 characters.”

AND CLEAR text in TextBoxUsername

When the Create Account button is clicked and both the username and password are valid, the programs check through the Accounts table, if there is a username that matches the input username the program should ask the user to pick a different username.

Search Table Accounts for Where Username = Text in TextboxUsername

IF Record is found...

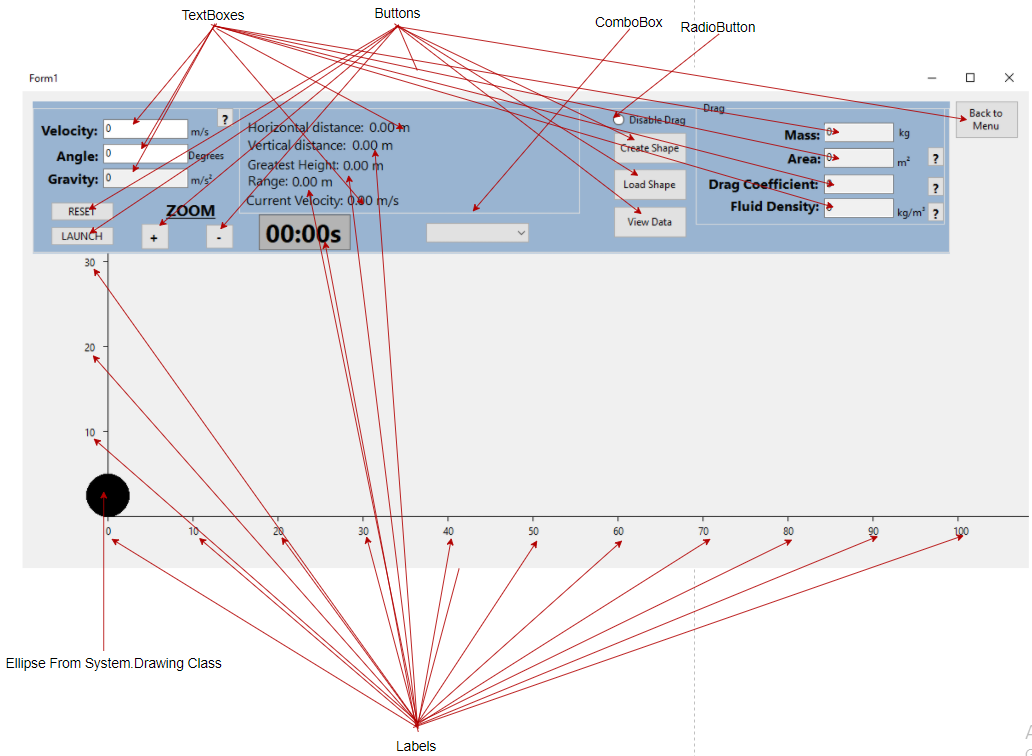
Then Show MessageBox that says, “Username already taken”

If the username is unique then the textboxes will be cleared, and the program will inform the user that their account has been created.

INSERT INTO Accounts (text in TextBoxUsername, text in TextBoxPassword)

The user can then use their login details to login to the Practice Form. If both username and password match the ones in the database, then the user can login.

### Practice Form (Main Form)



|  |
| --- |
| *TextBoxes:* tbVelocity, tbAngle, tbGravity (ReadOnly), tbMass (ReadOnly), tbArea (ReadOnly), tbDC (ReadOnly), tbFDensity (ReadOnly)  *Buttons:* btReset, btLaunch, btPause, ZoomIn, ZoomOut, btExit, btSaveLaunch  *Labels*: Xdistance, Ydistance, lGreatestHeight, lRange, Time, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, y1, y2, y3  *ComboBox*: cmbPlanets  *RadioButton* : rdDrag |

The Practice Form acts as a central hub to the rest of the forms. This is where the calculations are done. In this form the user has the option to launch objects based on their own inputs and the program will perform calculations on how far the object can travel, how high it was launched, and its velocity and displacement based on the time since launch. These inputs are...

• tbVelocity for entering an initial velocity.

• tbAngle for entering an initial angle.

• cmbPlanets can be used to select a planet or moon in our solar system, depending on the planet, the value of g and fluid density will change and will be displayed in tbGravity and tbFDensity respectively.

FOR CALCULATIONS WITHOUT DRAG

INSERT INTO LaunchValues ([Username], [ShapeName], [Velocity], [Angle], [Gravity], [FluidDensity], [DragOn], [HitGround], [MaxHeight], [Range], [Time]) VALUES('" +Menu.CurrentUser+ "','-DIMENSIONLESS-','" +tbVelocity.Text+ "','" +tbAngle.Text+ "','" +tbAcceleration.Text+ "','0','0','" +HitGround+ "','" + MaxHeight + "','" +range+ "','" +t+ "'

FOR CALCULATIONS WITH DRAG

INSERT INTO LaunchValues ([Username], [ShapeName], [Velocity], [Angle], [Gravity], [FluidDensity], [DragOn], [HitGround], [MaxHeight], [Range], [Time]) VALUES('" + Menu.CurrentUser+ "','" +ShapeLoad.SelectedShapeName+ "','" + tbVelocity.Text + "','" + tbAngle.Text + "','" + tbAcceleration.Text + "','" +tbFDensity.Text+ "','1','" + HitGround + "','" +MaxHeight+ "','" +range+ "','" +t+ "')

Values for Gravity, Angle and initial Velocity MUST be filled out to start calculations.

• btReset Will reset all the object to its original value and reset all values.

• btLaunch will start calculations and verify if all inputs are valid.

• ZoomIn and ZoomOut will change values in the x and y markers and the size of the shape being launched to create the effect of zooming in or out.

• btSaveLaunch cannot be seen on this diagram but the user can save values of a particular launch; this can be done once a launch ends by either the time running out or the object hitting the ground. If the user chooses to save a Launch, the properties of this launch will be stored in a database. These properties can later be loaded back into the Practice Form through the ValueLoader Form.

• Practice Form can be left by pressing the Exit button and returning to the Menu Form.

• The user has two main ways of launching the object; with or without drag, this option can be made by toggling rdDrag.

This form has information boxes which the user can click on if they aren’t sure on what a value might mean.

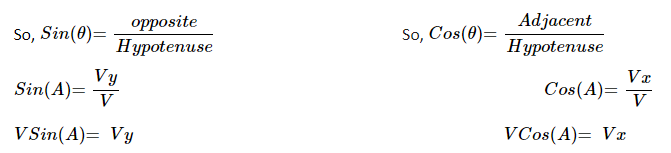
### Calculations

#### Calculations for initial Vy and Vx

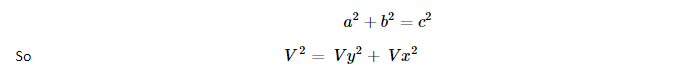
Most simple calculations of projectile motion rely on the fact that the horizontal and vertical components of velocity are independent of each other and can be treated separately. When the object being launched is dimensionless the only force acting on it is gravity as it doesn’t have a volume or mass.

Firstly, we must start with the user inputs. The user will have to input the Initial velocity and the initial angle that the object will travel at. To make this easier I will call Initial Velocity (V) and Angle (A). For the program to calculate the motion of the object we must split its Velocity (V) into horizontal (Vx) and vertical (Vy) components. To do we must view the Velocity as a right-angle triangle with the Opposite being Vy the Adjacent Vx and the Hypotenuse being V. Now that we know this, we can calculate Vy and Vx.

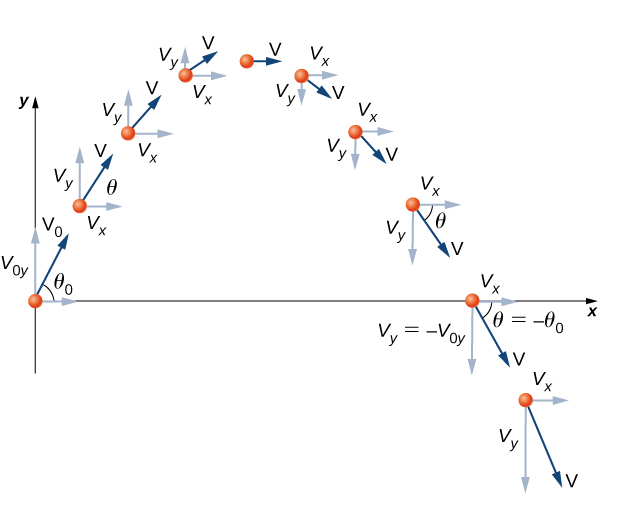
Note that Math.Cos and Math.Sin in C# work based on radians and NOT based on degrees. This doesn’t change any of the calculations it just means that angles will be converted to radians...



Now that we have split the velocity into its components, we can use this to predict the objects movement throughout time. Its horizontal velocity will remain constant as there are not outside forces acting horizontally however the ball is affected by gravity (g), so every second we must take away g from Vy then use the Pythagorean Theorem to recalculate V.

We can repeat these steps to get a parabola shape of coordinates for the basic trajectory of the object as seen below.

The user to have the option of picking between different planets in our solar system and for g to change accordingly, g will always have to be negative as it acts opposite to the initial vertical displacement.



#### Deriving Displacement from SUVAT equations

SUVAT equations also known as Equations of Motion are a series of equations that are used in in Kinematics and Mechanics to calculate a variety of variables of an object while its moving. SUVAT is an acronym for the variables and stands for...

S = Displacement

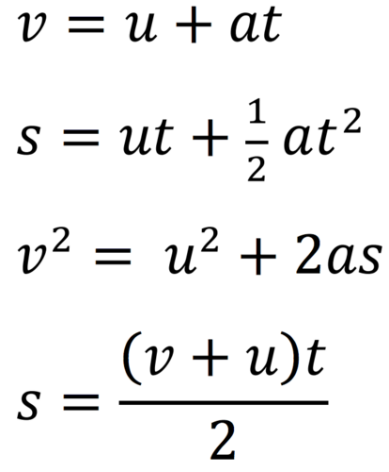
U = initial velocity

V= Final velocity

A = Acceleration

T = Time

And the equations are...



I will be using to derive the displacement of the object from its original point. To do this I will have to use the horizontal and the vertical components of the initial velocity individually and find a horizontal and vertical displacements. From those I will adjust the x and y coordinates of the object. I will need two variables Sy and Sx for vertical and horizontal displacement, then as the timer ticks and time in incremented, a new value of Sy and Sx are calculated depending on the time.

IN CODE

Velocity <- TextBoxVelocity Input

Angle <- TextBoxVelocity Input

Gravity <- TextBoxGravity

HorizontalVelocity <- Velocity \* Cos(Angle)

VerticalDisplacement <- Velocity \* Sin(Angle)

Time = 0

WHILE time LESS THAN 30

HorizontalDisplacement = HorizontalVelocity\*time

VerticalDisplacement = VerticalVelocity\*time + 0.5\*Gravity\*time²

ObjectPosition = (HorizontalDisplacement+orign, VerticallDisplacement + orign)

Time = Time + 0.1

END WHILE

#### Calculations for Drag / Air resistance

With drag considered Acceleration will have to be negative. This is because Velocity is speed with a direction, as Acceleration due to drag (Deceleration) will be negative as it acts opposite to Velocity. Furthermore, vertically there this also acceleration due to gravity acting on the ball along with drag so I will have to add the two to calculate the displacement.

When the user chooses to disable drag the horizontal component of Velocity will remain constant as there is no acceleration acting on it (no drag and acceleration due to gravity only acts vertically downwards).

Fd = drag force

ρ = fluid density

v = the speed of the object

Cd= Drag Coefficient

A = Area

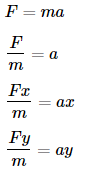
Drag establishes a relation between horizontal and vertical velocity that previously wasn’t mentioned, velocities higher than, 20 km/h, the drag of a moving body is proportional to the square of the velocity. Drag works directly against Velocity direction, this square dependency causes coupling of the horizontal and vertical forces that act on the projectile. The first thing to make clear is that this equation is used to calculate a FORCE not a velocity. As we know from Newton’s First Law if a force is applied there is a change in velocity and acceleration will occur. Again, we must split the Drag (D) up into its vertical and horizontal components (Dx and Dy) and as Drag and Velocity are proportional that means that

We can use this to find the horizontal and vertical Forces (F) on a stationary object:

W is weight or mg which I will talk about in my Mass section as I want to make it a parameter that the user can adjust.

This is where our original Drag force equation comes back in to calculate a value for D

And which can be rearranged to find vertical and horizontal Drag Forces. From there we can calculate the total forces acting on the object on both the horizontal and vertical space.



The important thing to note here is that the timer increments every 0.1 seconds, this means that a new value for velocity will be calculated every 0.1 seconds. For our calculations to work, we must assume that the deceleration produced by the drag force is constant for 0.1 seconds. With this assumption we can again use both horizontally and vertically and to calculate new values for velocity.

IN CODE

Velocity <- TextBoxVelocity Input

Angle <- TextBoxVelocity Input

Gravity <- TextBoxGravity

HorizontalVelocity <- Velocity \* Cos(Angle)

VerticalDisplacement <- Velocity \* Sin(Angle)

FluidDensity <- TextBoxFluidDensity Input

DragCoefficnent <- TextBoxDragCoefficnient Input

Area <- TextBoxArea Input

Mass <- TextBoxMass Input

Time = 0

HorizontalDisplacement = 0

VerticallDisplacement =0

WHILE Time IS LESS THAN 30

TotalDragForce = 0.5 \* FluidDensity \* Velocity \* Velocity \* DragCoefficient \* Area

HorizontalDragForce = - TotalDragForce \* (HorizotalVelocity/Velocity) VerticalDragForce = - TotalDragForce \* (VerticalVelocity/Velocity)

HorizontalDeceleration = HorizontalDragForce/Mass

VerticalDeceleration = (VerticalDragForce/Mass) +Gravity

HorizontalVelocity = HorizontalVelocity + HorizontalDeceleration\*0.1

VertialVelocity = VerticalVelocity + VerticalDeceleration\*0.1

HorizontalDisplacement = HorizontalDisplacement + HorizontalVelocity\*0.1+ 0.5\* HorizontalDeceleration \*0.1²

VerticalDisplacement = VerticalDisplacement + VerticalVelocity\*0.1+ 0.5\* VerticalDeceleration \*0.1²

Velocity = SquareRoot(HorizontalVelocity² + VerticalVelocity²)

Time = Time + 0.1

END WHILE

#### Calculations for Max Height and Range

I want the user to have values of Max Height and Range when saving data into the LaunchValues table. To calculate these values, we can again use SUVAT when the object is launched without drag.

Considering velocity vertically, when the greatest height is reached, the vertical velocity of the object will be 0 m/s. This is because at max height the ball is going from moving upwards to moving downwards, the vertical velocity is going from a positive value to a negative value. We can use the SUVAT equation to calculate Max Height.

Firstly, rearrange equation to equal s (displacement) .

V (final velocity) will be 0. so .

Now u (initial vertical velocity) will be negative, this isn’t a problem as a (gravity g) is also negative as it acts opposite to the initial vertical velocity. This results in the equation, where s is the max height reached.

For range we must do something a little more complicated. First, we must calculate the time when vertical displacement is 0. this means finding the time when the ball hits the ground. Resolving vertically only, we can use , if s (vertical displacement) is 0 and a is gravity (-g) then

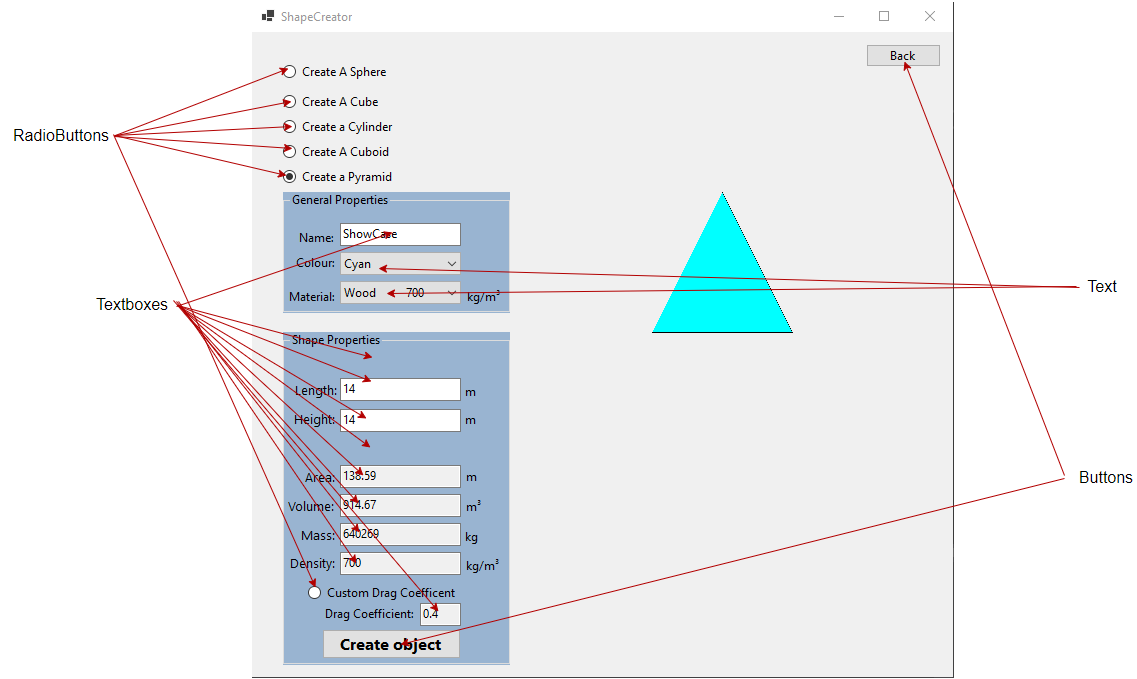
. From here we can rearrange for t.

, where t is the time when the object this the ground.

From here we can resolve values horizontally to find the distance travelled horizontally (range) by using , since there is no horizontal acceleration. where s is range and u is the initial horizontal displacement.

For calculations with drag, the initial velocities and acceleration are varied constantly due to the drag force so SUVAT cannot be used. This means that that to calculate the greatest height and range the program will compare previous greatest height and range and if the new values are larger than the previous ones then they will be replaced. This is a less accurate method as the values are only incremented every 0.1 seconds.

### ShapeCreator Form



|  |
| --- |
| *TextBoxes:* tbName, tbHeight, tbWidth, tbLength, tbRadius, btVolume (ReadOnly), tbMass (ReadOnly), tbArea (ReadOnly), tbDesnity (ReadOnly), tbDC  *Buttons:* btCreate, btExit,  *ComboBox*: cmbColour, cmbMat  *RadioButton* : rbSphere, rbCube, rbCuboid, rbCylinder, rbPyramid, rbCustom |

This form can only be accessed through Practice Form and can be left by clicking the Back button to return to Practice Form.

• tbName is used to create a name for the shape, this is mandatory and must be unique.

• tbHeight, tbWidth, tbLength, tbRadius are all used to adjust the size of the shape being draw.

• rbSphere, rbCube, rbCuboid, rbCylinder, rbPyramid are used to select what shape the user wants to create.

• rbCustom, this button is used to set the value of Drag Coefficient to ReadOnly or writable so that the user can enter a custom value for Drag coefficient.

• cmbColour, cmbMat are used to select the colour of the shape being created, and the material it's made from. Selecting a material will change its density and in turn change its mass.

In this form the user will have an option to select between a variety of shapes. The user can edit the size and dimension of the same which will alter its Volume. This is done by calling different classes which are associated with the shape being created. When values are input into textboxes to determine the dimensions of the shape, a class will be instantiated depending on which radiobutton is ticked. depending on these variables the mass will also be calculated.

IN CODE

IF tbName isn’t empty AND cmbColour isn’t empty AND cmbMat isn’t empty

THEN

IF rbSphere is Checked AND Value in tbRadius != 0

THEN

Sphere.Radius = VALUE IN tbRadius

Text in tbArea <- Sphere.GetArea()

Text in tbVolume <- Sphere.GetVolume()

END IF

ELSE IF rbCube is Checked AND Value in tbLength != 0

THEN

Cube.Lenth = VALUE IN tbLength

Text in tbArea <- Cube.GetArea()

Text in tbVolume <- Cube.GetVolume()

IF rbCylinder is Checked AND Value in tbRadius != 0 AND Value in tbHeight != 0

THEN

Cylinder.Radius = VALUE IN tbRadius

Cylinder.Height = VALUE IN tbHeight

Text in tbArea <- Cylinder.GetArea()

Text in tbVolume <- Cylinder.GetVolume()

END IF

IF rbPyramid Checked AND Value in tbLength != 0 AND Value in tbHeight != 0

THEN

Pyramid.Length = VALUE IN tbLength

Pyramid.Height = VALUE IN tbHeight

Text in tbArea <- Pyramid.GetArea()

Text in tbVolume <- Pyramid.GetVolume()

END IF

IF rbCuboid CHECKED AND Value in tbLength != 0 AND Value in tbHeight != 0 AND Value in tbWidth !=0

Cuboid.Length = VALUE IN tbLength

Cuboid.Height = VALUE IN tbHeight

Cuboid.Width = VALUE IN tbWidth

Text in tbArea <- Cuboid.GetArea()

Text in tbVolume <- Cuboid.GetVolume()

END IF

END IF

Shapes.GetMass()

Since every shape is different the way to calculate their Area and Volume are different however to calculate mass its always the same so that does not need to be altered at all.

Sphere

Sphere.GetArea()

{

}

Sphere.GetVolume()

{

}

Cube

Cube.GetArea()

{

}

Cube.GetVolume()

{

}

Cylinder

Cylinder.GetArea()

{

}

Cylinder.GetVolume()

{

}

Pyramid

Pyramid.GetArea()

{

}

Pyramid.GetVolume()

{

}

Cuboid

Cuboid.GetArea()

{  
   
}

Cuboid.GetVolume()

{

}

Once the user has created the shape they want, and all the values are filled out and validified, the user must fill out an appropriate name. If ‘Create Object’ button is clicked all the values will be inserted into the corresponding tables. Depending on the radiobutton that’s ticked, ShapeType in table Shapes will be different and the table that specific values are saved into are different also.

IN CODE

Search Table Shapes for Where Name = Text in tbName

IF Record is found...

Then Show MessageBox that says, “Shape name already taken”.

ELSE IF Text in tbName = “” OR Length of Text in tbName GREATER THAN 50

THEN Show MessageBox that says, “Please select an appropriate name for your shape”.

Validation of name

IF rbSphere IS CHECKED

INSERT INTO Shapes VALUES(CurrentUser’s Username ,ShapeName ,Circle, cmbColour.Text, tbArea.Text, tbMass.Text , tbDC.Text)

INSERT INTO Circle VALUES( ShapeName, tbRadius.Text)

ELSE IF rbCube IS CHECKED

INSERT INTO Shapes VALUES(CurrentUser’s Username ,ShapeName ,Cube, cmbColour.Text, tbArea.Text, tbMass.Text , tbDC.Text)

INSERT INTO Cube VALUES( ShapeName, tbLength.Text)

Else IF rbCylinder IS CHECKED

INSERT INTO Shapes VALUES(CurrentUser’s Username, ShapeName, Cylinder, cmbColour.Text, tbArea.Text, tbMass.Text , tbDC.Text)

INSERT INTO Cylinder VALUES(ShapeName, tbHeight.Text, tbRadius.Text)

ELSE IF rbPyramid IS CHECKED

INSERT INTO Shapes VALUES(CurrentUser’s Username ,ShapeName,Pyramid, cmbColour.Text, tbArea.Text, tbMass.Text , tbDC.Text)

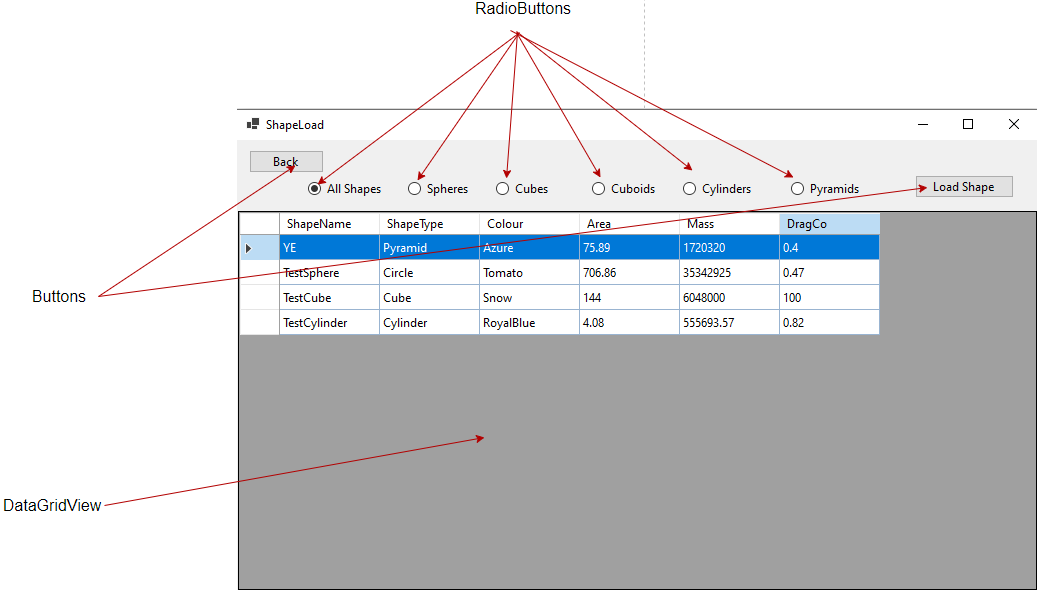
INSERT INTO Pyramid VALUES(ShapeName, tbLength.Text, tbHeight.Text)

ELSE IF rbCuboid IS CHECKED

INSERT INTO Shapes VALUES(CurrentUser’s Username ,ShapeName ,Cuboid, cmbColour.Text, tbArea.Text, tbMass.Text , tbDC.Text)

INSERT INTO Cuboid VALUES(ShapeName, tbLength.Text, tbWidth.Text , tbHeight.Text)

### ShapeLoad Form



|  |
| --- |
| *RadioButtons:* rbAll, rbSphere, rbCube, rbCuboid, rbCylinder, rbPyramid  *Buttons:* btLoad, btExit  *DataGridView:* dgwShape |

This form can only be accessed through Practice Form and can be left by clicking the Back button to return to Practice Form.

• The program starts with All Shapes being ticked at first, the user can then select between different shapes as it makes it easier to navigate the menu. If the user knows that they want to load a Cylinder, then they can filter out the rest of the shapes by clicking Cylinders.

• dgwShapes is a table that displays all the user’s created shapes. The user should not be able to add, remove or edit values in the DataGridView. If the user clicks the header of one of the rows, the row gets selected. Once selected a row is selected a row cannot be unselected however the selected row can change. Once selected, btLoad should appear in the top right corner.

• If btLoad is clicked ShapeLoaded in Practice Form should be set to true along with whatever shape was loaded (e.g. ShapeLoaded = True). the value in the first column should be selected (ShapeName is always first column in this table). Once the first row is selected then assign value to variable SelectedShapeName. The program should access the Shapes table and select all the values that are saved under the selected ShapeName. Most importantly ShapeType should be selected and read. All other values are assigned to variables in Practice Form, such as Colour, Mass and Area. Based on the ShapeType the program should access the table of that shape and set the IsShapeLoaded specified to that shape to true (e.g. CuboidLoaded = true). The program should load the shapes vertical and horizontal dimension so that it can be displayed accurately along the scale.

IN CODE

IF btLoad IS PRESSED

Practice.ShapeLoaded <- true

SelectedShapeName <- CurrentSelectedRow.Column[1]

SELECT Mass, Area, DragCo, ShapeType FROM Shapes WHERE ShapeName = SelectedShapeName AND Username = CurrentUsername

Practice.Mass <- Mass FROM Shapes

Practice.Area <- Area FROM Shapes

Practice.DragCoefficient <- DragCo FROM Shapes

IF ShapeType = “Circle”

IsCircleLoaded = true

SELECT Radius FROM Circle WHERE ShapeName= SelectedShapeName

Practice.Radius= Radius FROM Sphere

ELSE IF ShapeType = “Cube”

IsCubeLoaded = true

SELECT Length FROM Cube WHERE ShapeName= SelectedShapeName

Practice.Length = Length FROM Cube

ELSE IF ShapeType = “Cylinder”

IsCylinderLoaded= true

SELECT Radius, Height FROM Cylinder WHERE ShapeName= SelectedShapeName

Practice.Radius= Radius FROM Cylinder

Practice.Height = HeightFROM Cylinder

ELSE IF ShapeType = “Pyramid”

IsPyramidLoaded = true

SELECT Length, Height FROM Pyramid WHERE ShapeName= SelectedShapeName

Practice.Length = Length FROM Pyramid

Practice.Height = HeightFROM Pyramid

ELSE IF ShapeType = “Cuboid”

IsCuboidLoaded = true

SELECT Length, Height FROM Cuboid WHERE ShapeName= SelectedShapeName

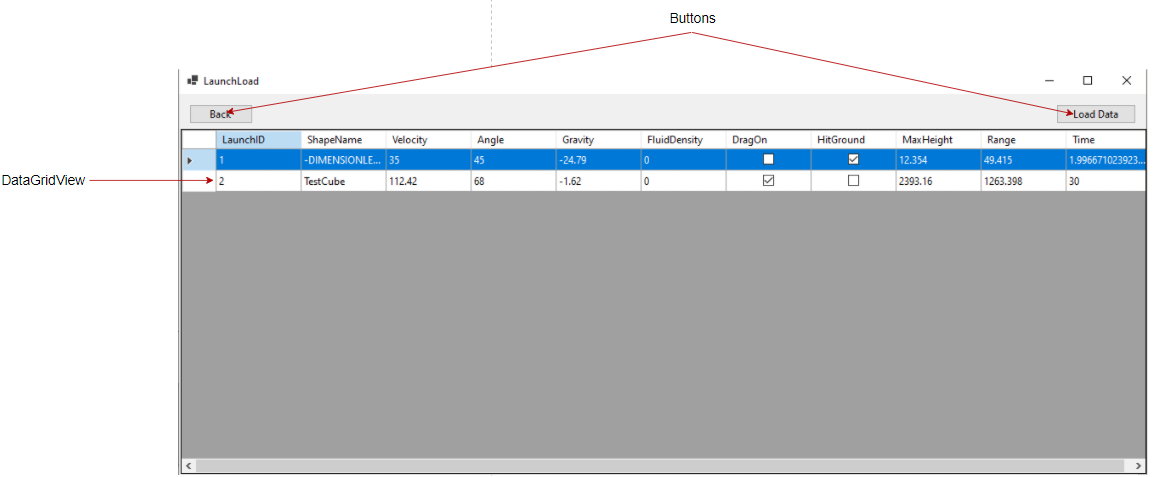
Practice.Length = Length FROM Cube

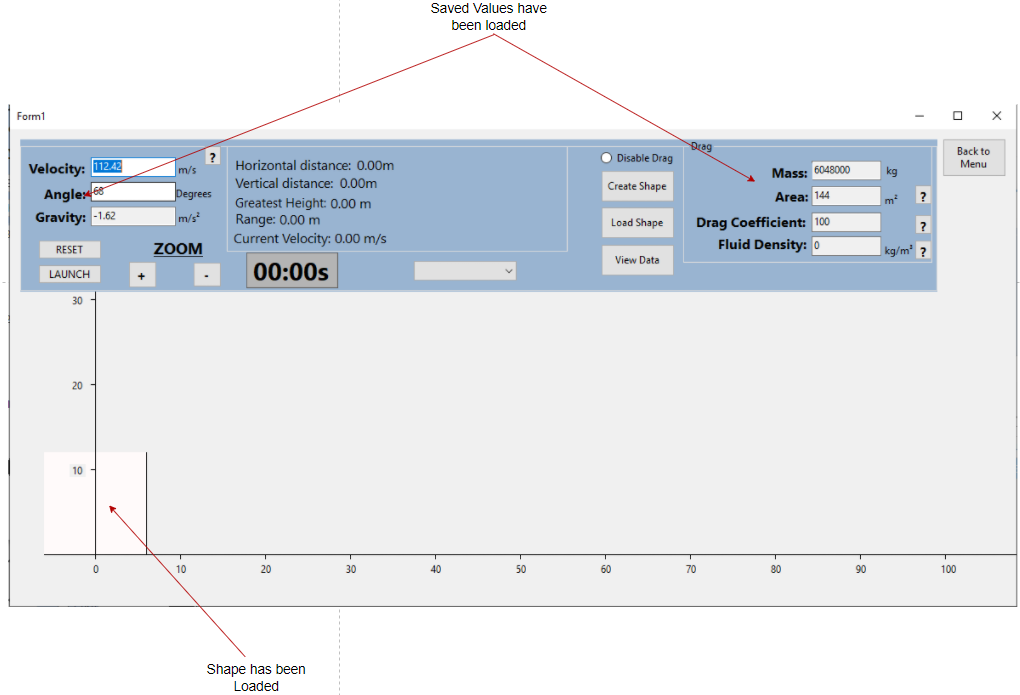
Practice.Height = HeightFROM Cube

END IF

END IF

### ValueLoader Form

dgwLaunchvalues is a table that displays all the user’s saved launch values. The user should not be able to add, remove or edit values in the DataGridView. If the user clicks the header of one of the rows, the row gets selected. Once selected a row is selected a row cannot be unselected however the selected row can change. Once selected, btLoad should appear in the top right corner.



This form can only be accessed through Practice Form and can be left by clicking the Back button to return to Practice Form.

If the user saved any values from Practice Form, they can all be seen here. Values save into table LaunchValues will be displayed through a DataGridView. The user can select a row of values and load it back into the Practice Form. This includes loading the shape which the calculations were performed with.

When btLoad is pressed program should select the first column of the selected row (LaunchID), use this value to

SELECT \* FROM LaunchValues WHERE LaunchID= SelectedLaunchID AND Username =CurrentUser

All the selected values stored under that LaunchID should be loaded back into Practice Form, as seen above. The same function should be used to Load the shape as the one descibed in ShapeLoad Form.

# **Technical Solution**

## List of Functions and classes

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Description | Form | Page No. |
| Menu\_Load | When the Menu is launched, the program checks if the file for the database exists, if it doesn’t then create the file and create all the tables along with it. Loads all components of the Menu. | Menu | 45 |
| bCreate\_Click | When the Create button is pressed, the program validated the text in CreateUsername and CreatePassword textboxes, makes sure they aren’t empty, too long or too short. The program then inserts the text into the Accounts table IF they aren’t already in there. | Menu | 46 |
| bLogin\_Click | When the Login Button is pressed, the program will take the text typed into the Username and Password textboxes and check for them in the Accounts table, if they are found the Menu closes and opens Practice Form and assigns CurrentUser to the Username, if the records aren’t found then tell user. | Menu | 44 |
| Form1\_Load | When the Practice Form is loaded, the Zoom Multiplier and scale are set to default value so the program is at its default zoom. The program checks if any shapes are loaded and enables drag accordingly. Sets textboxes to read only so values cannot be altered. Sets values to be stored inside textboxes to be displayed. Loads all the Planets into combobox | Practice | 49 |
| SetAllValues | This function just sets all the texts in the textboxes to the values in stored in the variables. Displays all values to textboxes. | Practice | 48 |
| bLaunch\_Click | When launch button is pressed, one of two things happen depending on whether it’s time is 0 or time is greater than 0. If time is 0 (initial launch) then the program validates the values in Acceleration, angle, velocity and Fluid Density textboxes so they aren’t empty. Validates velocity and angle so that they are appropriate values. If all values are valid then initial calculations are done to launch the object and timer is started. If time is greater than 0 all it does is start the timer and show the Pause Button. Also disables buttons to load values or textboxes to change any input values. Displays btSaveLaunch once the animation is complete. | Practice | 50 |
| timer1\_Tick | This function is called every 0.1 seconds, performs calculations discussed in Calculations section depending on if Drag is enabled or not and changed the X and Y coordinates of the shape depending on the zoom level and the calculated values. | Practice | 52 |
| Zoom | This function is called when ZoomIn or ZoomOut is clicked. Changed the text in marker textboxes to an appropriate value set by Zoom Multiplier. Applies a new value to the X and Y coordinates of the object so they are in appropriate locations. | Practice | 54 |
| Form1\_Paint | This function redraws the surface of the entire from when called. Usually happens when an object’s X and Y coordinates are changed or when Invalidate() is called. Depending on what type of shape is loaded the function draws that shape by its assigned values of size and position. Also draws Horizontal and vertical axis. | Practice | 56 |
| ZoomOut\_Click | Increases ZoomCounter (Maximum of 23) and removes the scale from Vertical and Horizontal values so that the Zoom function can set the new ones. Sets new scale and sets new zoom multiplier and calls Zoom at the end. | Practice | 57 |
| ZoomIN\_Click | Decreases ZoomCounter (Minimum of 0) and removes the scale from Vertical and Horizontal values so that the Zoom function can set the new ones. Sets new scale and sets new zoom multiplier and calls Zoom at the end. | Practice | 57 |
| Drag\_Click | Displays the groupbox storing drag values. If shape has an area, then set shapeloaded to true, if pressed again the (ticked) then disable drag and unload shape. | Practice | 58 |
| Pause\_Click | When pressed, pauses timer and animation, displays launch button so it can be pressed to start the animation again. | Practice | 58 |
| bReset\_Click | When pressed, resets all the displayed and internal values and re enables buttons and textboxes that were disabled by Launch so that new values can be entered. | Practice | 59 |
| btExit\_Click | Opens login Menu. | Practice | 60 |
| btCreator\_Click | Opens ShapeCreator Form. | Practice | 60 |
| btLoader\_Click | Opens ShapeLoad Form. | Practice | 62 |
| btLoadLaunch\_Click | Opens LaunchLoad Form. | Practice | 62 |
| cmbPlanets\_SelectedIndexChanged | Uses a switch case for when a new planet is selected from combobox. Depending on which planet is selected new values for Fluid Density and Gravity are selected and loaded into their respective textboxes for display. | Practice | 60 |
| Info Buttons | These are multiple buttons, but I wanted to group them all into one row as they all serve the same purpose. They display information about a given value or variable that the user may not be familiar with. For example, if the user doesn’t know what velocity is, they can click the info button to give them extra information to describe what velocity is. | Practice | 61/62 |
| DoubleValidation | This function is called whenever a textbox a key is pressed on a textbox for a character to be entered into it. This is a public static and is used across multiple forms. The function checks if the character entered is a digit or a decimal point. If not, then the character isn’t loaded into the textbox. If it’s a decimal point, then the program checks if it’s there is already a decimal point in the textbox or if the decimal is the first value to be entered. If it is then don’t enter it into the textbox. | Practice | 62 |
| Textboxes\_KeyPress | This applies to multiple textboxes in Practice, but they all serve the same function. When a key is pressed into the textbox, DoubleValidation is called. | Practice | 62 |
| btSaveLaunch\_Click | Once the animation is complete, this button is displayed. Calculates whether the ground was hit or not. Loads values for the given Launch into LaunchValues table. | Practice | 62 |
| ShapeCreator\_Load | When ShapeCreator is loaded, copy all the names of colours and load them into a combobox. Load all Materials into a combobox. | ShapeCreator | 65 |
| cmbColor\_SelectedIndexChanged | When a colour is selected from the combobox, Redraw the entire form so that the colour of the shape changes. | ShapeCreator | 64 |
| ShapeCreator\_Paint | Validates the text entered into Textboxes radius, Length, Height and Width. Based on which radiobutton is clicked and all the shape has dimensions assigned and a colour, it will draw the given shape. | ShapeCreator | 65 |
| ChangeMass | Uses Shapes class to calculate the mass of the shape. | ShapeCreator | 67 |
| cmbMat\_SelectedIndexChanged | Uses a switch case for when a new material is selected from combobox. Depending on which material is selected a new value for density is loaded into textbox for display. | ShapeCreator | 67 |
| btCreate\_Click | When create button is clicked, it validates the name of the shape by making sure its not already in the Shapes table or the Name textbox isn’t empty. If all values are present and NOT empty or 0, then the program will insert two records into two tables. One table will be inserted into Shapes table which will contain all the universal properties of a shape and the other table will be the selected shape the user wanted to create. | ShapeCreator | 68 |
| ClearAll | This function sets all the text in texboxes to be empty. | ShapeCreator | 70 |
| rbSphere\_Click | When clicked display all textboxes for dimensions for a sphere. | ShapeCreator | 71 |
| rdCube\_Click | When clicked display all textboxes for dimensions for a cube. | ShapeCreator | 71 |
| rbCuboid\_Click | When clicked display all textboxes for dimensions for a cuboid. | ShapeCreator | 72 |
| rbCylinder\_Click | When clicked display all textboxes for dimensions for a cylinder. | ShapeCreator | 73 |
| rbPyramid\_Click | When clicked display all textboxes for dimensions for a pyramid. | ShapeCreator | 73 |
| ValidScale | This function reads the number entered into a textbox and makes sure it's not above the limit which is assigned to it. If the number is greater than display messagebox saying so and clear the textbox. | ShapeCreator | 74 |
| tbRadius\_TextChanged | Depending on which shape is selected and which other values are filled out, the program will calculate the new Volume and Area of the object when radius is changed. | ShapeCreator | 74 |
| tbLength\_TextChanged | Depending on which shape is selected and which other values are filled out, the program will calculate the new Volume and Area of the object when length is changed. | ShapeCreator | 75 |
| tbHeight\_TextChanged | Depending on which shape is selected and which other values are filled out, the program will calculate the new Volume and Area of the object when height is changed. | ShapeCreator | 77 |
| tbWidth\_TextChanged | Will change Area and Volume of a Cuboid if textboxes Height and Length are filled out. | ShapeCreator | 78 |
| btExit\_Click | Loads Practice Form when clicked. | ShapeCreator | 79 |
| rbCustom\_Click | Allows user to input a custom value for Drag Coefficient. | ShapeCreator | 79 |
| Textboxes\_keyPress | This applies to multiple textboxes in ShapeCreator, but they all serve the same function. When a key is pressed into the textbox, DoubleValidation is called. | ShapeCreator | 79/80 |
| ShapeLoad | Sets rbAll to be checked and calls CheckButtons. | ShapeLoad | 80 |
| CheckButtons | Checks through all the radiobuttons. If a radiobutton is ticked, then select the records from Shapes table of the given shape. For example, is rbAll is checked all records are selected where the username matches that of the logged in user, but if rbPyramid is checked then only records which store values for pyramids are selected. The selected records are then loaded into dgwShapes so that the user can select them. | ShapeLoad | 81 |
| RadioButtons\_Click | This applies to multiple radiobuttons but they all do the same thing. When a radiobutton is clicked, call CheckButtons. | ShapeLoad | 82 |
| dgwShapes\_RowHeaderMouseClick | When the header of a row is selected, show btLoad so the user can load values in a selected row. | ShapeLoad | 82 |
| btLoad\_Click | When Load button is clicked, LoadSelectedShape is called, and the Practice Form is loaded. | ShapeLoad | 82 |
| LoadSelectedShape | This function is used to select values from the database and load those values into the Practice Form so that the user can perform calculations with those values. This was already described in detail in the Description of Algorithms and User Interface under ShapeLoad Form. This function is public static because it is called in LaunchLoad also. | ShapeLoad | 82/83 |
| btExit\_Click | Practice Form is loaded, | ShapeLoad | 85 |
| LaunchLoad | Loads components for LaunchLoad and selects values from LaunchValues table where the username is equal to the current user so that those values can be loaded into dgwLaunchValues. | LaunchLoad | 85 |
| btLoad\_Click | Selects launch values from table where the LaunchID matches the one selected by the user, calls LoadSelectedShape so that if a shape is involved in the calculations it is loaded. | LaunchLoad | 86 |
| btExit\_Click | Loads Practice Form when clicked. | LaunchLoad | 86 |
| dgwLaunchValues\_RowHeaderMouseClick | When the header of a row is selected, show btLoad so the user can load values in a selected row. | LaunchLoad | 86 |
| Shapes Class | This is the base class for all other classes, it is used to set the colour of the shapes and calculate their mass. | - | 87 |
| Sphere Class | This class inherits from Shapes, it is used to calculate the frontal surface area and volume of a Sphere. | - | 87 |
| Cube Class | This class inherits from Shapes, it is used to calculate the frontal surface area and volume of a Cube. | - | 88 |
| Cuboid Class | This class inherits from Shapes, it is used to calculate the frontal surface area and volume of a Cuboid. | - | 88 |
| Cylinder Class | This class inherits from Shapes, it is used to calculate the frontal surface area and volume of a Cylinder. | - | 89 |
| Pyramid Class | This class inherits from Shapes, it is used to calculate the frontal surface area and volume of a Pyramid. | - | 89 |

## Menu

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using System.Data.OleDb;  using System.IO;      namespace NEA  {  public partial class Menu : Form  {    public Menu()  {  InitializeComponent();    }  public static string CurrentUser; // Set as public static as it is used in other Forms.  private void bLogin\_Click(object sender, EventArgs e) // When login Button is clicked.  {  OleDbConnection Conn = new OleDbConnection(Program.connString);  Conn.Open(); // Opens Connection to the database.  OleDbCommand Cmd = new OleDbCommand(); //Create a database command object.  Cmd.Connection = Conn;  Cmd.CommandText = "SELECT \* FROM Accounts WHERE Username ='" + tbUsername.Text + "'"; // Selects Username and Password where username equals the entered text.  OleDbDataReader reader = Cmd.ExecuteReader(); //Runs the query & allows results to be read.  if (reader.HasRows) //if a record is found display details.  {  reader.Read(); //Read the first record found.  if (tbPassword.Text == reader["Pword"].ToString()) // If Password in database associated with selected username matches the password entered  {  // If both password and username mattch then logged in.  CurrentUser = reader["Username"].ToString(); // Set the CurrentUser to the Username of the person logged in.  Practice p = new Practice();  p.ShowDialog(); // Runs code for Practice Form.  this.Close(); // Closes this Form.  }  else  {  MessageBox.Show("Wrong password"); // Display that the password is wrong (but username exists in database).  }  }  else // If username isn't found.  {  MessageBox.Show("Username not found."); // Display that username doesn't exist in the database.  }  reader.Close(); // Closes the reader.  Conn.Close(); // Closes the connection to database.  }  private void Menu\_Load(object sender, EventArgs e)  {  if (File.Exists("Database.accdb") == false) // Run code if the database doesn't exits. This is done so that the database is only created once and if the file is deleted it remakes it.  {  ADOX.Catalog cat = new ADOX.Catalog();  cat.Create(Program.connString);  OleDbConnection Conn = new OleDbConnection(Program.connString); ////Use constant defined in Program.cs to create database.  Conn.Open(); // Open connection to database.  OleDbCommand Cmd = new OleDbCommand(); //Create a database command object.  Cmd.Connection = Conn;  Cmd.CommandText = "CREATE TABLE Accounts([Username] VARCHAR(50), [Pword] VARCHAR(50), PRIMARY KEY (Username))"; // Creates a table named Accounts.  Cmd.ExecuteNonQuery(); //Execute (non-query) SQL command.  Cmd.CommandText = "CREATE TABLE Shapes([Username]VARCHAR(50),[ShapeName] VARCHAR(50),[ShapeType] VARCHAR(8), [Colour] VARCHAR(50), [Area] FLOAT, [Mass] FLOAT, [DragCo] FLOAT, PRIMARY KEY (ShapeName), FOREIGN KEY (Username) REFERENCES Accounts (Username))"; // Creates table named Shapes.  Cmd.ExecuteNonQuery();  Cmd.CommandText = "CREATE TABLE Circle ([CircleID] AUTOINCREMENT, [ShapeName] VARCHAR(50), [Radius] FLOAT, PRIMARY KEY (CircleID), FOREIGN KEY (ShapeName) REFERENCES Shapes (ShapeName))"; // Creates table named Circle.  Cmd.ExecuteNonQuery();  Cmd.CommandText = "CREATE TABLE Cube([CubeID] AUTOINCREMENT, [ShapeName] VARCHAR(50), [Length] FLOAT, PRIMARY KEY (CubeID), FOREIGN KEY (ShapeName) REFERENCES Shapes (ShapeName))"; // Creates table named Cube.  Cmd.ExecuteNonQuery();  Cmd.CommandText = "CREATE TABLE Cuboid([CuboidID] AUTOINCREMENT, [ShapeName] VARCHAR(50), [Length] FLOAT,[Width] FLOAT, [Height] FLOAT, PRIMARY KEY (CuboidID), FOREIGN KEY (ShapeName) REFERENCES Shapes (ShapeName))"; // Creates table named Cuboid.  Cmd.ExecuteNonQuery();  Cmd.CommandText = "CREATE TABLE Cylinder([CylinderID] AUTOINCREMENT, [ShapeName] VARCHAR(50), [Length] FLOAT, [Radius] FLOAT, PRIMARY KEY (CylinderID), FOREIGN KEY (ShapeName) REFERENCES Shapes (ShapeName))"; // Creates table named Cylinder.  Cmd.ExecuteNonQuery();  Cmd.CommandText = "CREATE TABLE Pyramid([PyramidID] AUTOINCREMENT, [ShapeName] VARCHAR(50), [Length] FLOAT,[Height] FLOAT, PRIMARY KEY (PyramidID), FOREIGN KEY (ShapeName) REFERENCES Shapes (ShapeName))"; // Creates table named Pyramid.  Cmd.ExecuteNonQuery();  Cmd.CommandText = "CREATE TABLE LaunchValues([LaunchID] AUTOINCREMENT, [Username] VARCHAR(50),[ShapeName] VARCHAR(50), [Velocity] FLOAT, [Angle] FLOAT, [Gravity] FLOAT, [FluidDensity] FLOAT, [DragOn] BIT, [HitGround] BIT, [MaxHeight] FLOAT, [Range] FLOAT, [Time] FLOAT, PRIMARY KEY (LaunchID))"; // Creates table named LaunchValues.  Cmd.ExecuteNonQuery();  Conn.Close(); // Closes the connection to database.  }  }    private void bCreate\_Click(object sender, EventArgs e) // When Create button is pressed  {  if (tCreateUser.Text == "" || tbCreatePword.Text == "" ) // If textboxes are empty (so empty values aren't inserted into the database).  {  MessageBox.Show("Please fill out both username and password.");  }  else if(tCreateUser.Text.Length > 50 || tCreateUser.Text.Length < 7) // Username validation.  {  MessageBox.Show("Make sure username is between 7 and 50 characters.");  tCreateUser.Text = "";  }  else if(tbCreatePword.Text.Length > 50) // Password validation.  {  MessageBox.Show("Make sure your password isn't longer than 50 characters.");  tbCreatePword.Text = "";  }  else  {  OleDbConnection Conn = new OleDbConnection(Program.connString);  Conn.Open();  OleDbCommand Cmd = new OleDbCommand();  Cmd.Connection = Conn;  Cmd.CommandText = "SELECT \* FROM Accounts WHERE Username ='" + tCreateUser.Text + "'"; // Selects Username and Password where username equals the entered text.  OleDbDataReader reader = Cmd.ExecuteReader();  if (reader.HasRows) //if a record is found display details.  {  MessageBox.Show("Username already taken"); // If the database already contains the username Display the message.  tbCreatePword.Text = "";  tCreateUser.Text = "";  // Set textboxes to be empty.  }  else // If username is unique.  {  reader.Close(); // //Close the reader so another connection can be made.  Cmd.Connection = Conn;  Cmd.CommandText = "INSERT INTO Accounts VALUES('" + tCreateUser.Text + "','" + tbCreatePword.Text + "')"; // Stores text from textboxes into database.  Cmd.ExecuteNonQuery(); //Execute (non-query) SQL command.  MessageBox.Show("Account details saved"); // Lets the user know that the values were saved.  tbCreatePword.Text = "";  tCreateUser.Text = "";  // Clears textboxes.  }  Conn.Close(); // Closes Connection to database.  }    }  }  } |

## Practice

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using System.Threading;  using System.IO;  using System.Data.OleDb;  using ShapeLibrary;    namespace NEA  {  public partial class Practice : Form  {  //variables without drag  private double t = 0;  public static double Velocity;  public static double angle;  private double Vdisplacement;  private double Hdisplacement;  private int Vdisplay;  private int Hdisplay;  public static double AtoG;  private double HorizontalVelocity;  private double VerticalVelocity;  private double InitialVerticalVelocity;  private const double defaultradius =2.5; // The dimensionless object is represented by a circle so the radius is set to a default size so there is a visual representation (this radius can't be changed).    //drag variables  private double TotalDragForce;  private double HorizontalDragForce;  private double VerticalDragForce;  public static double fluidD;  private double DecelerationX;  private double DecelerationY;    // adjustment variables  private double scale = 10;  public static int size;  public static int size2;    //random variables  private bool isChecked = false;  private int ZoomMultiplier;  private double MaxHeight = 0;  private double range = 0;  private double finaltime;  private double ZoomCounter =0;    //Shape variables;  public static bool IsShapeLoaded;  public static bool IsCircleLoaded;  public static bool IsCubeLoaded;  public static bool IsCuboidLoaded;  public static bool IsCylinderLoaded;  public static bool IsPyramidLoaded;  public static Rectangle circle = new Rectangle(75, 450, 50, 50); // Creates a circle object which can be altered later with values.  public static Rectangle cube = new Rectangle(); // Creates a cube object which can be altered later with values.  public static Rectangle cuboid = new Rectangle(); // Creates a cuboid object which can be altered later with values.  public static Rectangle Cylinder = new Rectangle(); // Creates a Cylinder object which can be altered later with values.  public static Point[] TrianglePoints = new Point[3]; // Creates an array of 3 points which can draw a triangle    public Practice()  {  InitializeComponent();  }  private void SetAllValues() // When loading values from database in other forms this subroutine reads the values to the textboxes to display to the user.  {  tbAcceleration.Text = AtoG.ToString();  tbVelocity.Text = Velocity.ToString();  tbAngle.Text = angle.ToString();  tbMass.Text = Shapes.Mass.ToString();  tbArea.Text = Shapes.Area.ToString();  tbDC.Text = Shapes.DragCoefficient.ToString();  tbFDensity.Text = fluidD.ToString();  }  private void Form1\_Load(object sender, EventArgs e) // When form is loaded.  {  ZoomMultiplier = 2; // Default Zoom is set  scale = 10; // Default scale is set.  if (IsPyramidLoaded == true) // is a pyramid has been loaded  {  // This is done for the Pyramid only as DrawPolygon works based on an array of points.  // Loaded values size and size2 are used to set the starting points of the triangle.  size = Convert.ToInt32(Pyramid\_squrebase.base\_length \* scale);  size2 = Convert.ToInt32(Pyramid\_squrebase.height \* scale);  TrianglePoints[0].X = 100 - Convert.ToInt32(0.5 \* size);  TrianglePoints[0].Y = 500;    TrianglePoints[1].X = 100 + Convert.ToInt32(0.5 \* size);  TrianglePoints[1].Y = 500;    TrianglePoints[2].X = 100;  TrianglePoints[2].Y = 500 - size2;  }  // Default settings when the Form is loaded.  tbMass.ReadOnly = true;  tbArea.ReadOnly = true;  tbDC.ReadOnly = true;  tbFDensity.ReadOnly = true;  tbAcceleration.ReadOnly = true; // These values are all set to ReadOnly so they cannot be changed unless you load values from a database or values from the combobox.  cmbPlanets.Items.Add("Mercury");  cmbPlanets.Items.Add("Venus");  cmbPlanets.Items.Add("Earth");  cmbPlanets.Items.Add("Moon");  cmbPlanets.Items.Add("Mars");  cmbPlanets.Items.Add("Jupiter");  cmbPlanets.Items.Add("Saturn");  cmbPlanets.Items.Add("Uranus");  cmbPlanets.Items.Add("Neptune");  cmbPlanets.Items.Add("Pluto");  // Adds all of the Items to the combobox so that they can be used.  cmbPlanets.DropDownStyle = ComboBoxStyle.DropDownList; // ComboBoxStyle must be set to DropDownList so that the user edit it's text.  if(IsShapeLoaded== false) // If a shape isn't loaded from another Form...  {  rdDrag.Checked = false; // check radiobutton to be false so that the user can't see drag values (which they aren't using).  Zoom(); // Zoom is called to make sure the correct scale is set and the object is in the correct position.  }  if (IsShapeLoaded == true) // If a shape is loaded from another Form...  {  rdDrag.Checked = true; // Enable radiobutton drag so values can be seen  }  SetAllValues(); // Set all values to display in textbox.  Drag\_Click(sender, e); // Drag\_Click is called to ensure all the values are set correctly and displayed.  }  private void bLaunch\_Click(object sender, EventArgs e) // When the launch button is clicked.  {  if(t==0) // These commands only need to be executed the first time an object is launched.  {    if(tbAcceleration.Text == "" || tbAngle.Text == "" || tbVelocity.Text == "" || tbFDensity.Text == "") // If one of these values isn't filled in then dont launch object.  {  MessageBox.Show("Please fill out all available variables!");  }  else if (double.Parse(tbVelocity.Text) <= 0 || double.Parse(tbVelocity.Text) >= 299792458) // Check if the Velocity of the object is valid.  {  MessageBox.Show("Please input an an appropriate value for Velocity" +  "between 0 and 299792458 m/s (the speed of light)."); // Nothing can travel faster than the speed of light.  tbVelocity.Text = ""; // Empty textbox.  t = 0; // Set time back to 0;  timer1.Stop(); // If invalid reset timer.  }  else if (double.Parse(tbAngle.Text) > 90 || double.Parse(tbAngle.Text) < 0 ) // Validation for the Angle.  {  MessageBox.Show("Please input a value between 0 and 90 degrees for the angle");  tbVelocity.Text = "";  t = 0;  timer1.Stop();  }  else // If both inputs are valid.  {  btCreator.Enabled = false;  btLoader.Enabled = false;  bLaunch.Hide();  bPause.Show(); // Launch button is hidden and Pause is shown so that the user can stop the timer when they want.  Velocity = double.Parse(tbVelocity.Text); // Reads input text to variable so it can be used for calculations.  if(Velocity >= 5000000) // If velocity is above 5000000 Hdisplay and Vdisplay will be very large when zoomed in.  {  MessageBox.Show("Since your value for velocity is so large, the program will have to zoom out");  ZoomOut\_Click(sender, e);  ZoomOut\_Click(sender, e);  ZoomOut\_Click(sender, e);  ZoomOut\_Click(sender, e);  ZoomOut\_Click(sender, e);  ZoomOut\_Click(sender, e);  ZoomOut\_Click(sender, e);  ZoomOut\_Click(sender, e);  ZoomOut\_Click(sender, e);  // Zoom out 6 times so that H and V display values aren't too large for Int32.  ZoomIN.Enabled = false;  ZoomOut.Enabled = false; // Stops the the user from zooming in and out until they reset the program.  }  angle = double.Parse(tbAngle.Text);  angle = angle \* (Math.PI / 180); // Input angle is in degrees, Math.Sin and Math.Cos take values to be in radians, line converts from degree to radian.  HorizontalVelocity = Velocity \* Math.Cos(angle); // Calculates inital Horizontal Velocity.  VerticalVelocity = Velocity \* Math.Sin(angle); // Calculates inital Vertical Velocity.  InitialVerticalVelocity = VerticalVelocity; // One value is used as the inital velocity and the other to display resultant velocity, this is not needed for horizontal as it remains constant in SUVAT  if(IsShapeLoaded == false) // If the object has no drag...  {  MaxHeight = -(VerticalVelocity \* VerticalVelocity) / (2 \* AtoG); // When there is no drag basic SUVAT equations can be used to calculate MaxHeight.  lGreatestHeight.Text = Math.Round(MaxHeight, 3).ToString() + " m"; // Set text to display max height.    finaltime = (-2 \* VerticalVelocity) / AtoG; // Calculate time taken for ball to hit the ground.  range = HorizontalVelocity \* finaltime; // When there is no drag basic SUVAT equations can be used to calculate Range.  lRange.Text = Math.Round(range, 3).ToString() + " m"; // Set text to display range.  }    tbVelocity.ReadOnly = true;  tbAngle.ReadOnly = true;  cmbPlanets.Enabled = false;  rdDrag.Enabled = false;  // Set input fields to ReadOnly / Disabled so values can't be changed midflight.  timer1.Start(); //Timer starts.  }  }  else //If the timer was paused (time is greater than 0), just continue the timer and swap buttons to show Pause.  {  timer1.Start();  bLaunch.Hide();  bPause.Show();  }    }  private void timer1\_Tick(object sender, EventArgs e) // Do this every 0.1 seconds.  {  t = t + 0.1; // Time is incremented.  t = Math.Round(t, 2); // time is rounded to the decimal place.  if (rdDrag.Checked == true || Shapes.Area == 0) // If object is dimensionless...  {  Vdisplacement = (InitialVerticalVelocity \* t) + 0.5 \* AtoG \* (t \* t); // Perform basic SUVAT equation to calculate the Vertical Displacement.  Hdisplacement = HorizontalVelocity \* t; // Perform basic SUVAT equation to calculate the Horizontal Displacement.  VerticalVelocity = VerticalVelocity + (AtoG/10); // Gets new value for v as SUVAT uses inital velocity which never changes.  Vdisplay = Convert.ToInt32(Vdisplacement \* scale); // Round to an integer and multiply Vertical Displacement by the scale so it can be used to change the location of the object in the Form.  Hdisplay = Convert.ToInt32(Hdisplacement \* scale); // // Round to an integer and multiply Horizontal Displacement by the scale so it can be used to change the location of the object in the Form.  circle.X = 100 + Hdisplay - size; // Change Horizontal location of the object.  circle.Y = 500 - Vdisplay - 2 \* size; // Change Vertical location of the object.  }  else if (rdDrag.Checked == false && Shapes.Area != 0) // If object has dimensions...  {  TotalDragForce = 0.5 \* fluidD \* Velocity \* Velocity \* Shapes.DragCoefficient \* Shapes.Area; // Calculate the total Drag force on the object.  HorizontalDragForce = -TotalDragForce \* (HorizontalVelocity / Velocity); // Calculate Horizontal Drag force on the object.  VerticalDragForce = -TotalDragForce \* ((VerticalVelocity) / Velocity); // Calculate Vertical Drag Force on the object.  DecelerationX = (HorizontalDragForce) / Shapes.Mass; // Calculate Deceleration due to drag Horizontally.  DecelerationY = VerticalDragForce/ Shapes.Mass + AtoG; // // Calculate Deceleration due to drag and gravity Vertically.    HorizontalVelocity = HorizontalVelocity + DecelerationX \* 0.1; // Calculate new value for Horizontal Velocity.  VerticalVelocity = VerticalVelocity + DecelerationY \* 0.1; // // Calculate new value for Vertical Velocity.    Hdisplacement = Hdisplacement + HorizontalVelocity \* 0.1 + 0.5 \* DecelerationX \* (0.1 \* 0.1); // Calculate new value for Horizontal Displacement.  Vdisplacement = Vdisplacement + VerticalVelocity \* 0.1 + 0.5 \* DecelerationY \* (0.1 \* 0.1); // // Calculate new value for Vertical Displacement.    Vdisplay = Convert.ToInt32(Vdisplacement \* scale); // Round to an integer and multiply Vertical Displacement by the scale so it can be used to change the location of the object in the Form.  Hdisplay = Convert.ToInt32(Hdisplacement \* scale); // Round to an integer and multiply Horizontal Displacement by the scale so it can be used to change the location of the object in the Form.  if (IsShapeLoaded== true)  {  if (IsCircleLoaded == true) // If a circle is loaded...  {  circle.X = 100 + Hdisplay-size;  circle.Y = 500 - Vdisplay- 2\*size;  // Change location based on scale.  }  else if(IsCubeLoaded== true) // If a cube is loaded...  {  cube.X = 100 + Hdisplay - Convert.ToInt32(size\*0.5);  cube.Y = 500 - Vdisplay - size;  // Change location based on scale.  }  else if(IsCuboidLoaded == true) // If a cuboid is loaded...  {  cuboid.X = 100 + Hdisplay - Convert.ToInt32(size \* 0.5);  cuboid.Y = 500 - Vdisplay - size2;  // Change location based on scale.  }  else if(IsCylinderLoaded == true) // If a cylinder is loaded...  {  Cylinder.X = 100 + Hdisplay - Convert.ToInt32(size2 \* 0.5);  Cylinder.Y = 500 - Vdisplay - size \* 2;  // Change location based on scale.  }  else if(IsPyramidLoaded == true) // If a Pyramid is loaded...  {  TrianglePoints[0].X = 100 - Convert.ToInt32(0.5 \* size) + Hdisplay;  TrianglePoints[0].Y = 500 - Vdisplay;    TrianglePoints[1].X = 100 + Convert.ToInt32(0.5 \* size) + Hdisplay;  TrianglePoints[1].Y = 500 - Vdisplay;    TrianglePoints[2].X = 100 + Hdisplay;  TrianglePoints[2].Y = 500 - size2 - Vdisplay;  // Change location of points in the array to move the drawing of the triangle.  }  if(Vdisplacement >= MaxHeight) // If the current Vertical Displacement is greater than the previous Max Height reached...  {  MaxHeight = Vdisplacement; // set Vertical Displacement as new max height.  lGreatestHeight.Text = Math.Round(MaxHeight, 3).ToString() + " m"; // Display Max Height.  }  if(Hdisplacement >= range) // If the current Horizontal Displacement is greater than the previous Max Distance reached...  {  range = Hdisplacement; // set Horizontal Displacement as new range.  lRange.Text = Math.Round(range, 3).ToString() + " m"; // Display range.  }  }    }  Velocity = Math.Sqrt(HorizontalVelocity \* HorizontalVelocity + VerticalVelocity \* VerticalVelocity); // Velocity is calculated from Horizontal and Vertical components.  lCurrentVelocity.Text = Math.Round(Velocity, 3).ToString() + " m/s";  Xdistance.Text = Math.Round(Hdisplacement, 3).ToString() + " m"; // Display current Distance.  Ydistance.Text = Math.Round(Vdisplacement, 3).ToString() + " m"; // Display current Height.  Invalidate(); // Invalidate is called to redraw the Form (move object).  this.lTime.Text = t.ToString() + "s"; // Display current time.  if (t >= 30 || Vdisplacement <=0) // if the timer runs out (30 seconds) or object hits the ground...  {  timer1.Stop(); // Stop the timer.  bPause.Hide(); // Stop user from being able to pause (animation is over).  btSaveLaunch.Show(); // Display button so the user has the option to save values of the launch.  }  }      private void Zoom()  {  x1.Text = (5 \* ZoomMultiplier).ToString();  x2.Text = (10 \* ZoomMultiplier).ToString();  x3.Text = (15 \* ZoomMultiplier).ToString();  x4.Text = (20 \* ZoomMultiplier).ToString();  x5.Text = (25 \* ZoomMultiplier).ToString();  x6.Text = (30 \* ZoomMultiplier).ToString();  x7.Text = (35 \* ZoomMultiplier).ToString();  x8.Text = (40 \* ZoomMultiplier).ToString();  x9.Text = (45 \* ZoomMultiplier).ToString();  x10.Text = (50 \* ZoomMultiplier).ToString();    y1.Text = (5 \* ZoomMultiplier).ToString();  y2.Text = (10 \* ZoomMultiplier).ToString();  y3.Text = (15 \* ZoomMultiplier).ToString();  // Multiples the displayed distance markers by the new multiplier.  Hdisplay = Convert.ToInt32(Hdisplacement \* scale);  Vdisplay = Convert.ToInt32(Vdisplacement \* scale);  // Multiples the displayed displacement by the new scale so that the object is in the correct position when displayed.  if (rdDrag.Checked == false && Shapes.Area != 0) // if object has dimensions...  {  if (IsCircleLoaded == true)  {  size = Convert.ToInt32(Sphere.radius \* scale);  circle = new Rectangle(100 - size + Hdisplay, 500 - size \* 2 - Vdisplay, size \* 2, size \* 2);  }  else if (IsCubeLoaded == true)  {  size = Convert.ToInt32(Cube.length \* scale);  cube = new Rectangle(100 - Convert.ToInt32(size \* 0.5) + Hdisplay, 500 - size - Vdisplay, size, size);  }  else if(IsCuboidLoaded == true)  {  size = Convert.ToInt32(Cuboid.length\* scale);  size2 = Convert.ToInt32(Cuboid.height \* scale);  cuboid = new Rectangle(100 - Convert.ToInt32(size \* 0.5) + Hdisplay, 500 - size2 - Vdisplay, size, size2);  }  else if(IsCylinderLoaded == true)  {  size = Convert.ToInt32(cylinder.radius \* scale);  size2 = Convert.ToInt32(cylinder.height \* scale);  Cylinder = new Rectangle(100 - Convert.ToInt32(size2 \* 0.5) + Hdisplay, 500 - size\*2 - Vdisplay, size2, size\*2);  }  else if (IsPyramidLoaded == true)  {  size = Convert.ToInt32(Pyramid\_squrebase.base\_length \* scale);  size2 = Convert.ToInt32(Pyramid\_squrebase.height \* scale);  TrianglePoints[0].X = 100 - Convert.ToInt32(0.5 \* size) + Hdisplay;  TrianglePoints[0].Y = 500 - Vdisplay;    TrianglePoints[1].X = 100 + Convert.ToInt32(0.5 \* size) + Hdisplay;  TrianglePoints[1].Y = 500 - Vdisplay;    TrianglePoints[2].X = 100 + Hdisplay;  TrianglePoints[2].Y = 500 - size2 - Vdisplay;  }  //Changes the location of the object based on an new scale so that they match the displayed markers and changes the size of the displayed object giving an effect that the object is further / closer.  }  else  {  size = Convert.ToInt32(defaultradius \* scale);  circle = new Rectangle(100 - size + Hdisplay, 500 - size \* 2 - Vdisplay, size \* 2, size \* 2);  }  Invalidate(); // Invalidate redraws the entire Form so that the new values are displayed.  }      Pen BlackPen = new Pen(Color.Black); // Creates a black pen in which the outlines of objects can be drawn in.  Brush CustomColour = new SolidBrush(Shapes.colour); // Creates a brush which is used to fill in objects in the colour which is loaded from the database in ShapeLoad Form.  Brush BlackBrush = new SolidBrush(Color.Black); // Crates a black brush which is used to fill in objects in black.  private void Form1\_Paint(object sender, PaintEventArgs e)  {  Graphics g = e.Graphics; // Creates graphics g which can be used to draw objects.  if(IsShapeLoaded ==true)  {  if(IsCircleLoaded == true) // if a circle is loaded...  {  g.DrawEllipse(BlackPen, circle); // Create black outline with Black Pen in the shape of an ellipse.  g.FillEllipse(CustomColour, circle); // Fills in outline with colour from database in the shape of an ellipse.  }  else if (IsCubeLoaded == true) // if a Cube is loaded...  {  g.DrawRectangle(BlackPen, cube); // Create black outline with Black Pen in the shape of a Rectangle.  g.FillRectangle(CustomColour, cube); // Fills in outline with colour from database in the shape of a Rectangle.  }  else if(IsCuboidLoaded == true) // if a Cuboid is loaded...  {  g.DrawRectangle(BlackPen, cuboid);  g.FillRectangle(CustomColour, cuboid);  }  else if(IsCylinderLoaded == true) // if a Cylinder is loaded...  {  g.DrawRectangle(BlackPen, Cylinder);  g.FillRectangle(CustomColour, Cylinder);  }  else if(IsPyramidLoaded == true) // if a Pyramid is loaded...  {  g.DrawPolygon(BlackPen, TrianglePoints); // Create black outline with Black Pen in the shape of a Triangle based on the 3 points.  g.FillPolygon(CustomColour, TrianglePoints); // Fills in outline with colour from database in the shape of a Triangle based on the 3 points.  }  }  else if(IsShapeLoaded == false) // if shape is dimensionless...  {  g.DrawEllipse(BlackPen, circle); // Create an outline of a circle in black. Circle size is set to a default so size cannot be changed.  g.FillEllipse(BlackBrush, circle); // Fills in the outline in black.  }    g.DrawLine(BlackPen, 100, 500, 1200, 500); // draws horizontal axis.  g.DrawLine(BlackPen, 100, 500, 100, 50); // draws vertical axis.  int position = 100;  for (int i = 0; i <= 10; i++) // Creates 10 lines where markers are to make it look like a graph horizontally.  {  g.DrawLine(BlackPen, position, 500, position, 505); // Draw small vertical lines to indicate markers.  position = position + 100; // Increment markers every 100 pixels horizontally.  }  position = 500; // set position to 500 as thats the origin for the y axis.  for (int j = 0; j <= 3; j++) // Creates 3 lines where markers are to make it look like a graph vertically.  {  g.DrawLine(BlackPen, 100, position, 95, position); // Draws small horizontal lines to indicate markers.  position = position - 100; // Increment markers every 100 pixels vertically.  }  }    private void ZoomIN\_Click(object sender, EventArgs e)  {  if(ZoomCounter >=0) // ZoomCounter cannot be smaller than 0, if ZoomCounter is larger than 0...  {  ZoomCounter--; // Take away from counter.  Hdisplay = Convert.ToInt32(Hdisplacement / scale); // Divide Hdisplay by the old scale so an new one can be applied in Zoom subroutine.  Vdisplay = Convert.ToInt32(Vdisplacement / scale); // Divide Vdisplay by the old scale so an new one can be applied in Zoom subroutine.  if (scale >= 10) // Scale cannot be smaller than 10;  {  scale = 10;  }  else // Scale increases as ZoomCounter decreases (things get bigger as you zoom in).  {  scale = scale \* 2; // New scale is set.  }    if (ZoomMultiplier <= 2) // ZoomMultiplier cannot be smaller than 0.  {  ZoomMultiplier = 2;  }  else // Zoom multiplier decreases as ZoomCounter decreases (markers get smaller as you zoom in).  {  ZoomMultiplier = ZoomMultiplier / 2;  }  Zoom(); // Zoom gets called.  }  }    private void ZoomOut\_Click(object sender, EventArgs e)  {  if(ZoomCounter <= 23) // ZoomCounter cannot be larger than 23, if ZoomCounter is larger than 23 Int32 starts going into negatives and breaks scale.  {  ZoomCounter++; // Add to Counter;  Hdisplay = Convert.ToInt32(Hdisplacement / scale); // Divide Hdisplay by the old scale so an new one can be applied in Zoom subroutine.  Vdisplay = Convert.ToInt32(Vdisplacement / scale); // Divide Vdisplay by the old scale so an new one can be applied in Zoom subroutine.  scale = scale / 2; // New scale is set.  ZoomMultiplier = ZoomMultiplier \* 2; // New Multiplier is set (marker values get larger as you zoom out).  Zoom(); // Zoom gets called.  }  }  private void Drag\_Click(object sender, EventArgs e) // When radiobutton drag is clicked...  {  if (rdDrag.Checked && !isChecked) // If radiobutton isn't Checked (Drag is enabled)...  {  if(Shapes.Area!=0) // Shape has Dimensions...  {  IsShapeLoaded = true; // Shape is loaded.  rdDrag.Text = "Unload Shape";  }  rdDrag.Checked = false;  DragStuff.Show(); // Show groubox with all of the drag values.  }  else // If Drag is disabled.  {  IsShapeLoaded = false;  IsCubeLoaded = false;  IsCuboidLoaded = false;  IsCylinderLoaded = false;  IsPyramidLoaded = false;  rdDrag.Checked = true;  isChecked = false;  // Set everything to false as nothing is loaded.  DragStuff.Hide(); // Hide Drag stuff as its not needed.  Shapes.Mass = 0;  Shapes.Area = 0;  Shapes.DragCoefficient = 0;  bReset\_Click(sender, e); // Reset all of the values.  rdDrag.Text = "Show drag values.";  }  }    private void Drag\_CheckedChanged(object sender, EventArgs e)  {  isChecked = rdDrag.Checked;  }    private void Pause\_Click(object sender, EventArgs e) // When pause is clicked...  {  timer1.Stop(); // Timer Stops.  bPause.Hide(); // Pause is hidden so you launch can be seen.  bLaunch.Show(); // Launch is shown so user can continue animation.  }    private void bReset\_Click(object sender, EventArgs e) // When reset is clicked...  {  if(IsPyramidLoaded == true)  {  TrianglePoints[0].X = 100 - Convert.ToInt32(0.5 \* size);  TrianglePoints[0].Y = 500;    TrianglePoints[1].X = 100 + Convert.ToInt32(0.5 \* size);  TrianglePoints[1].Y = 500;    TrianglePoints[2].X = 100;  TrianglePoints[2].Y = 500 - size2;  // Sets the trinagles coordinates back to its origin.  }  timer1.Stop(); // Stops timer.  t = 0; // Resets time.  Xdistance.Text = "0.00 m";  Ydistance.Text = "0.00 m";  lGreatestHeight.Text = "0.00 m";  MaxHeight = 0;  range = 0;  lRange.Text = "0.00 m";  lTime.Text = "00:00s";  lCurrentVelocity.Text= "0.00 m";  Velocity = double.Parse(tbVelocity.Text);  angle = double.Parse(tbAngle.Text);  // Sets all displays back to 0 so no incorrect values are displayed to the user.  // Sets Velocity and angle back to their original values so that if the user wants to use the same values again they can.  TotalDragForce = 0;  HorizontalDragForce = 0;  VerticalDragForce = 0;  Vdisplay = 0;  Hdisplay = 0;  DecelerationX = 0;  DecelerationY = 0;  HorizontalVelocity = 0;  VerticalVelocity = 0;  Hdisplacement = 0;  Vdisplacement = 0;  // Sets all the values back to 0 so that they dont affect the next set of calculations.  bLaunch.Show(); // Shows Launch again so that object can be launched.  bPause.Hide();  rdDrag.Enabled = true; // Re-enables Drag radiobutton so that it can be pressed to on and off as it no longer affects calculations.  cmbPlanets.Enabled = true; // Re-enables Planets Combobox so that it can be used as it no longer affects calculations.  tbVelocity.ReadOnly = false; // Re-enables Velocity textbox so that it can be used as it no longer affects calculations.  tbAngle.ReadOnly = false; //// Re-enables Angle textbox so that it can be used as it no longer affects calculations.  ZoomIN.Enabled = true;  ZoomOut.Enabled = true;  btCreator.Enabled = true;  btLoader.Enabled = true;    // if speed was too large and zooms had to be disabled reset re-enables them.  btSaveLaunch.Hide(); // All values are reset so there is no use for a save button.  Zoom(); // Zoom is called to set the display of the object back to its origin.  }    private void btExit\_Click(object sender, EventArgs e)  {  Menu menu = new Menu();  menu.ShowDialog(); // Opens Menu.  this.Close(); // Closes Practice.  }    private void btCreator\_Click(object sender, EventArgs e)  {  ShapeCreator SC = new ShapeCreator();  SC.ShowDialog(); // Opens ShapeCreator.  this.Close(); // Closes Practice.  }    private void cmbPlanets\_SelectedIndexChanged(object sender, EventArgs e)  {  switch (cmbPlanets.SelectedIndex)  {  // Mercury  case 0:  AtoG = -3.7;  fluidD = 0;  break;  // Venus  case 1:  AtoG = -8.87;  fluidD = 67;  break;  // Earth  case 2:  AtoG = -9.81;  fluidD = 1.225;  break;  // Moon  case 3:  AtoG = -1.62;  fluidD = 0;  break;  // Mars  case 4:  AtoG = -3.721;  fluidD = 0.02;  break;  // Jupiter  case 5:  AtoG = -24.79;  fluidD = 0.16;  break;  // Saturn  case 6:  AtoG = -10.44;  fluidD = 0.19;  break;  // Uranus  case 7:  AtoG = -8.69;  fluidD = 0.42;  break;  // Neptune  case 8:  AtoG = -11;  fluidD = 0.45;  break;  // Pluto  case 9:  AtoG = -0.66;  fluidD = 0;  break;  // Sets values according to which case is selected.  }  tbAcceleration.Text = AtoG.ToString(); // Sets new Gravity value to display.  tbFDensity.Text = fluidD.ToString(); // Sets new Fluid Density value to display.  }    private void VelocityInfo\_Click(object sender, EventArgs e)  {  MessageBox.Show("Velocity is defined as the speed at which something is travelling at in a given direction. This is the inital speed at which the object will be fired at.");  }      private void AreaInfo\_Click(object sender, EventArgs e)  {  MessageBox.Show("Area refers to the frontal surface area of the object which is coming in contact with the air molecules and is perpendiculat to the direction of flow.");  }    private void DCInfo\_Click(object sender, EventArgs e)  {  MessageBox.Show("the drag coefficient is a dimensionless quantity that aerodynamicists use to model all of the complex dependencies of shape, inclination, and flow conditions on objects.");  }    private void FDInfo\_Click(object sender, EventArgs e)  {  MessageBox.Show("The object being launched is travelling in a fluid, such as air. This quantity refers to how close the particles in the given fluid are.");  }    private void btLoader\_Click(object sender, EventArgs e)  {  ShapeLoad Loader = new ShapeLoad();  Loader.ShowDialog(); // Opens ShapeLoad Form.  this.Close(); // Closes Practice.  }    public static void DoubleValidation(KeyPressEventArgs e, TextBox tb) // Public Static void as subroutine is used for other textboxes in different forms.  {  char EnteredCharacter = e.KeyChar;    if (EnteredCharacter == 46 && tb.Text.IndexOf('.') != -1 && tb.Text.Length>0) // If entered charater is a . and the first digit isn't a . and a . isn't already present.  {  e.Handled = true; // character can be displayed.  return;  }    if (!Char.IsDigit(EnteredCharacter) && EnteredCharacter != 8 && EnteredCharacter != 46) // If the entered character isn't a digit, a backspace or a . don't display.  {  e.Handled = true; // character can be displayed.  }  }  private void tbVelocity\_KeyPress(object sender, KeyPressEventArgs e)  {  DoubleValidation(e, tbVelocity); // Validation for tbVelocity.  }    private void tbAngle\_KeyPress(object sender, KeyPressEventArgs e)  {  DoubleValidation(e, tbAngle); // Validation for tbAngle.  }    private void btSaveLaunch\_Click(object sender, EventArgs e) // When SaveLaunch is pressed (SaveLaunch can only be seen if time runs out or object hits the ground)  {  MaxHeight =Math.Round(MaxHeight, 3); // Rounds the value of MaxHeight to 3 Decimal Places so its not too long when stored in the table.  range = Math.Round(range, 3); // Rounds the value of Range to 3 Decimal Places so its not too long when stored in the table.  int HitGround;  if(t==30 && Vdisplacement> 0) // If the timer is equal to 30 (time ran out) and the object is still above the ground.  {  HitGround = 0; // Ground isn't hit.  }  else // If timer doesn't equal to 30 and the object hit the ground then...  {  HitGround = 1; // Ground is hit.  if(IsShapeLoaded == false)  {  t = finaltime; // Final time gives a more accurate reading of the time.  }  }  OleDbConnection Conn = new OleDbConnection(Program.connString);  Conn.Open(); // Creates connection with database.  OleDbCommand Cmd = new OleDbCommand(); //Create a database command object.  Cmd.Connection = Conn;  if (IsShapeLoaded == false) // If shape is Dimensionless...  {  Cmd.CommandText = "INSERT INTO LaunchValues ([Username], [ShapeName], [Velocity], [Angle], [Gravity], [FluidDensity], [DragOn], [HitGround], [MaxHeight], [Range], [Time]) VALUES('" +Menu.CurrentUser+ "','-DIMENSIONLESS-','" +tbVelocity.Text+ "','" +tbAngle.Text+ "','" +tbAcceleration.Text+ "','0','0','" +HitGround+ "','" + MaxHeight + "','" +range+ "','" +Math.Round(t,3)+ "')";  Cmd.ExecuteNonQuery();  // Save all values into the database with the name -DIMENSIONLESS-.  }  else if (IsShapeLoaded == true) // If shape has Dimensions...  {  Cmd.CommandText = "INSERT INTO LaunchValues ([Username], [ShapeName], [Velocity], [Angle], [Gravity], [FluidDensity], [DragOn], [HitGround], [MaxHeight], [Range], [Time]) VALUES('" + Menu.CurrentUser+ "','" +ShapeLoad.SelectedShapeName+ "','" + tbVelocity.Text + "','" + tbAngle.Text + "','" + tbAcceleration.Text + "','" +tbFDensity.Text+ "','1','" + HitGround + "','" +MaxHeight+ "','" +range+ "','" +t+ "')";  Cmd.ExecuteNonQuery();  // Save all values into the database with the name of the object that was launched (links to the Shapes table which has all the details of the shape so it can be loaded again).  }  MessageBox.Show("Successfully saved Data");  btSaveLaunch.Hide();  }    private void btLoadLaunch\_Click(object sender, EventArgs e)  {  LaunchLoad Data = new LaunchLoad();  Data.ShowDialog(); // Opens LaunchLoad.  this.Close(); // Closes Practice.  }  }  } |

## ShapeCreator

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using ShapeLibrary;  using System.Data.OleDb;    namespace NEA  {  public partial class ShapeCreator : Form  {  // Varibales so that radiobuttons can be checked and unchecked.  private bool isCheckedSphere = false;  private bool isCheckedCube = false;  private bool isCheckedCuboid = false;  private bool isCheckedCylinder = false;  private bool isCheckedPyramid = false;  private bool isCheckedCustom = false;    public ShapeCreator()  {  InitializeComponent();  // Set all values to read only so they can't be altered by user.  tbArea.ReadOnly = true;  tbVolume.ReadOnly = true;  tbMass.ReadOnly = true;  tbDensity.ReadOnly = true;  tbDC.ReadOnly = true;    cmbColour.DropDownStyle = ComboBoxStyle.DropDownList;  cmbMat.DropDownStyle = ComboBoxStyle.DropDownList;  // Set comboboxes to DropDownList so text can't be altered.  }    private void cmbColor\_SelectedIndexChanged(object sender, EventArgs e)  {  // When user changes colour in combobox redraw shape with new colour.  Invalidate();  }    private void ShapeCreator\_Load(object sender, EventArgs e)  {  foreach (System.Reflection.PropertyInfo prop in typeof(Color).GetProperties()) // Loop for each colour.  {  if (prop.PropertyType.FullName == "System.Drawing.Color") // If type of Property is System.Drawing.Color (this is so that only colours which can be used is System.Drawing are selected).  {  cmbColour.Items.Add(prop.Name); // Adds the name of the colour to the combobox.  }  }  cmbMat.Items.Add("Aluminium 2700"); //  cmbMat.Items.Add("Barium 3780"); //  cmbMat.Items.Add("Silver 10500"); //  cmbMat.Items.Add("Steel 7820"); //  cmbMat.Items.Add("Gold 19300"); //  cmbMat.Items.Add("Copper 8960"); //  cmbMat.Items.Add("Lead 11343"); //  cmbMat.Items.Add("Glass 2500 "); //  cmbMat.Items.Add("Concrete 2400"); //  cmbMat.Items.Add("Wood 700"); //  cmbMat.Items.Add("Plastics 1,175"); //  cmbMat.Items.Add("Diamond 3500"); //  cmbMat.Items.Add("Iron 7870"); //  cmbMat.Items.Add("Brass 8600"); //  cmbMat.Items.Add("Granite 2700"); //  // Adds all the materials to the combobox.  }    private void ShapeCreator\_Paint(object sender, PaintEventArgs e)  {  Graphics g = e.Graphics; // Instantiates class so that Graphics can be drawn.  Pen myPen = new Pen(Color.Black); // Creates a black pen.  Shapes.colour = Color.FromName(cmbColour.Text); // Colour is set to the colour which is selected in the combobox.  Brush brush = new SolidBrush(Shapes.colour); // Brush is created using the colour from the combobox.  ValidScale(tbRadius,1000); // Checks if radius isn't too large.  ValidScale(tbLength,1000); // Checks if Length isn't too large.  ValidScale(tbHeight, 1000); // Checks if Height isn't too large.  ValidScale(tbWidth, 1000); // Checks if Width isn't too large.  if (rbSphere.Checked == true && tbRadius.Text != "" && cmbColour.Text != "") // If Create sphere is checked and a colour is selected and radius isn't 0 or empty.  {  Sphere.radius = double.Parse(tbRadius.Text); // Sets radius to the value in textbox.  int size = Convert.ToInt32(Sphere.radius \* 10); // Multiplies the radius by 10 so that displayed circle is large enough to be seen. Must be in Int as System.Draw works based on pixels and must be an integer.  g.DrawEllipse(myPen, 400 - size, 300 - size \* 2, size \* 2, size \* 2); // Draws a circle in Black Pen which has a size of 2 Radius (Diameter).  g.FillEllipse(brush, 400 - size, 300 - size \* 2, size \* 2, size \* 2); // Fills in the circle with the selected colour.  }  if(rbCube.Checked == true && tbLength.Text != "" && cmbColour.Text != "") // If Create cube is checked and a colour is selected and Length isn't 0 or empty.  {  Cube.length = double.Parse(tbLength.Text); // Sets length to the value in textbox.  int size = Convert.ToInt32(Cube.length \* 10); // Multiplies the Length by 10 so that displayed circle is large enough to be seen. Must be in Int as System.Draw works based on pixels and must be an integer.  g.DrawRectangle(myPen, 400 - size, 300 - size, size, size); // Draws a Rectangle in Black Pen which has an equal Length and Height at position 400, 300.  g.FillRectangle(brush, 400 - size, 300 - size, size, size); // Fills in the cube with the selected colour.  }  if(rbCylinder.Checked == true && tbHeight.Text != "" && tbRadius.Text != "" && cmbColour.Text != "") // If Create Cylinder is checked and a colour is selected and radius and Length aren't 0 or empty.  {  cylinder.height = double.Parse(tbHeight.Text); // Sets height to the value in textbox.  cylinder.radius = double.Parse(tbRadius.Text); // Sets radius to the value in textbox.  int HeightSize = Convert.ToInt32(cylinder.height \* 10);  int RadiusSize = Convert.ToInt32(cylinder.radius \* 10);  g.DrawRectangle(myPen, 400 - HeightSize, 300 - RadiusSize\*2, HeightSize, RadiusSize \* 2); // Draws a circle in Black Pen which has a height of 2 Radius (Diameter) and horizontal length Height.  g.FillRectangle(brush, 400 - HeightSize, 300 - RadiusSize \* 2, HeightSize, RadiusSize \* 2); // Fills in the rectangle with the selected colour.  }  if(rbPyramid.Checked && tbHeight.Text != "" && tbLength.Text != "" && cmbColour.Text != "")  {  Pyramid\_squrebase pyramid = new Pyramid\_squrebase(); // Instantiates class.  Pyramid\_squrebase.base\_length = double.Parse(tbLength.Text); // Sets length to the value in textbox.  Pyramid\_squrebase.height = double.Parse(tbHeight.Text); // Sets height to the value in textbox.  int BaseSize = Convert.ToInt32(Pyramid\_squrebase.base\_length \* 10);  int HeightSize = Convert.ToInt32(Pyramid\_squrebase.height \* 10);    Point[] pnt = new Point[3]; // Creates an array of 3 different points in the form.  pnt[0].X = 400;  pnt[0].Y = 300;  // First point is at (400,300) origin.  pnt[1].X = 400 + BaseSize;  pnt[1].Y = 300;  // Second point is on the same Y point but the X value is base Length from the origin  // These two points create a straight line that becomes the base of the triangle.  pnt[2].X = 400 + Convert.ToInt32(0.5 \* BaseSize);  pnt[2].Y = 300 - HeightSize;  // Third point is the top of the triangle.  g.DrawPolygon(myPen, pnt); // Creates a polygon (triangle) with 3 points.  g.FillPolygon(brush, pnt); // Fills in the Polygon with selected colour.  }  if(rbCuboid.Checked && tbHeight.Text != "" && tbLength.Text != "" && tbWidth.Text != "" && cmbColour.Text != "") // If Create Cuboid is checked and a colour is selected and height and Length aren't 0 or empty.  {  Cuboid.length = double.Parse(tbLength.Text);  Cuboid.height = double.Parse(tbHeight.Text);  // As the Cuboid is represented in 2D only 2 values are needed. Width cannot be seen  int HeightSize = Convert.ToInt32(Cuboid.height \* 10);  int LengthSize = Convert.ToInt32(Cuboid.length \* 10);    g.DrawRectangle(myPen, 400 - LengthSize, 300 - HeightSize, LengthSize, HeightSize); // Creates a rectangle with Height HeightSize and Base LengthSize From the Origin (400,300)  g.FillRectangle(brush, 400 - LengthSize, 300 - HeightSize, LengthSize, HeightSize); // Fills in the rectangle with the selected colour.  }  }  private void ChangeMass()  {    if (tbVolume.Text != "" && tbDensity.Text != "") // If Volume of Density are empty Mass cannot be calculated.  {  Shapes.Density = double.Parse(tbDensity.Text);  Shapes.Volume = double.Parse(tbVolume.Text);  Shapes.GetMass(); // Gets the Value for Mass From Class.  tbMass.Text = Math.Round(Shapes.Mass, 2).ToString(); // Mass is rounded to 2 Decimal places and set textbox to display the value.  }    }    private void cmbMat\_SelectedIndexChanged(object sender, EventArgs e)  {  switch (cmbMat.SelectedIndex)  {  case 0:  Shapes.Density = 2700;    break;  case 1:  Shapes.Density = 3780;  break;  case 2:  Shapes.Density = 10500;  break;  case 3:  Shapes.Density = 7820;  break;  case 4:  Shapes.Density = 19300;  break;  case 5:  Shapes.Density = 8960;  break;  case 6:  Shapes.Density = 11343;  break;  case 7:  Shapes.Density = 2500;  break;  case 8:  Shapes.Density = 2400;  break;  case 9:  Shapes.Density = 700;  break;  case 10:  Shapes.Density = 1175;  break;  case 11:  Shapes.Density = 3500;  break;  case 12:  Shapes.Density = 7870;  break;  case 13:  Shapes.Density = 8600;  break;  case 14:  Shapes.Density = 2700;  break;  }  tbDensity.Text = Shapes.Density.ToString(); // New desnity is set to be displayed.  ChangeMass(); // New mass is calculated if density is changed.  }    private void btCreate\_Click(object sender, EventArgs e)  {    OleDbConnection Conn = new OleDbConnection(Program.connString);  Conn.Open(); // Opens connection with database.  OleDbCommand Cmd = new OleDbCommand(); //Create a database command object.  Cmd.Connection = Conn;    string ShapeName = tbName.Text;  Cmd.CommandText = "SELECT ShapeName FROM Shapes WHERE ShapeName ='" + ShapeName + "'"; // Selects ShapeName from where ShapeName is the same as entered Name and username is the same.  OleDbDataReader reader = Cmd.ExecuteReader();  if(reader.HasRows) // If there is a record with entered name (If name is already taken under that username).  {  MessageBox.Show("Shape name already taken");  }  else if (ShapeName == "" || ShapeName.Length > 50) // If ShapeName is Empty of Longer than 50 Characters.  {  MessageBox.Show("Please enter an appropriate name for your shape.");    }  else if (tbArea.Text == "0" || tbVolume.Text == "0" || Convert.ToDouble(tbDC.Text) == 0) // If Area or Volume are 0 then one of the values (radius, length, height, width) must be empty.  {  MessageBox.Show("Please fill out all shape properties!");  }  else if (cmbColour.Text != "" && tbArea.Text != "" && tbMass.Text != "" && tbDC.Text != "") // Checks if all values are filled out.  {  // Inserts Values based on selected shape into the correct tables, since there is a forgein key there must be ShapeName in both Shapes table and another table.  // ShapeType is different based on different shapes.  reader.Close(); // Closes reader.  if (rbSphere.Checked == true) // If Create Sphere is checked...  {  Cmd.CommandText = "INSERT INTO Shapes VALUES('" +Menu.CurrentUser+ "','" + ShapeName + "','Circle','" + cmbColour.Text + "','" + tbArea.Text + "','" + tbMass.Text + "','" +tbDC.Text+ "')";  Cmd.ExecuteNonQuery();  Cmd.CommandText = "INSERT INTO Circle (ShapeName, Radius) VALUES('" + ShapeName + "','" + tbRadius.Text + "')";  Cmd.ExecuteNonQuery();  }  else if(rbCube.Checked) // If Create Cube is checked...  {  Cmd.CommandText = "INSERT INTO Shapes VALUES('" + Menu.CurrentUser + "','" + ShapeName + "','Cube','" + cmbColour.Text + "','" + tbArea.Text + "','" + tbMass.Text + "','" + tbDC.Text + "')";  Cmd.ExecuteNonQuery();  Cmd.CommandText = "INSERT INTO Cube (ShapeName,Length) VALUES('" + ShapeName + "','" + tbLength.Text + "')";  Cmd.ExecuteNonQuery();  }  else if(rbCuboid.Checked) // If Create Cuboid is checked...  {  Cmd.CommandText = "INSERT INTO Shapes VALUES('" + Menu.CurrentUser + "','" + ShapeName + "','Cuboid','" + cmbColour.Text + "','" + tbArea.Text + "','" + tbMass.Text + "','" + tbDC.Text + "')";  Cmd.ExecuteNonQuery();  Cmd.CommandText = "INSERT INTO Cuboid (ShapeName, Length, Width, Height) VALUES('" + ShapeName + "','" + tbLength.Text + "','" +tbWidth.Text+ "','" +tbHeight.Text+ "')";  Cmd.ExecuteNonQuery();  }  else if(rbCylinder.Checked) // If Create Cylinder is checked...  {  Cmd.CommandText = "INSERT INTO Shapes VALUES('" + Menu.CurrentUser + "','" + ShapeName + "','Cylinder','" + cmbColour.Text + "','" + tbArea.Text + "','" + tbMass.Text + "','" + tbDC.Text + "')";  Cmd.ExecuteNonQuery();  Cmd.CommandText = "INSERT INTO Cylinder (ShapeName, Length, Radius) VALUES('" + ShapeName + "','" + tbHeight.Text + "','" +tbRadius.Text+ "')";  Cmd.ExecuteNonQuery();  }  else if(rbPyramid.Checked) // If Create Pyramid is checked...  {  Cmd.CommandText = "INSERT INTO Shapes VALUES('" + Menu.CurrentUser + "','" + ShapeName + "','Pyramid','" + cmbColour.Text + "','" + tbArea.Text + "','" + tbMass.Text + "','" + tbDC.Text + "')";  Cmd.ExecuteNonQuery();  Cmd.CommandText = "INSERT INTO Pyramid (ShapeName, Length, Height) VALUES('" + ShapeName + "','" + tbLength.Text + "','" +tbHeight.Text+ "')";  Cmd.ExecuteNonQuery();  }  else // If any values are missing...  {  MessageBox.Show("Please fill out all shape properties!");  }  MessageBox.Show("Shape Successfully saved!");  }  Conn.Close(); // Closes connection to the Database.  }  private void ClearAll() // Clears all displayed values.  {  tbName.Text = "";  cmbColour.Text = "";  cmbMat.Text = "";  tbRadius.Text = "";  tbLength.Text = "";  tbWidth.Text = "";  tbHeight.Text = "";  tbArea.Text = "";  tbVolume.Text = "";  tbMass.Text = "";  tbDC.Text = "";  }  private void rbSphere\_CheckedChanged(object sender, EventArgs e)  {  isCheckedSphere = rbSphere.Checked;  lradius.Hide();  tbRadius.Hide();  mRadius.Hide();  }  private void rbSphere\_Click(object sender, EventArgs e)  {  if (rbSphere.Checked && !isCheckedSphere)  {  // if sphere not checked  rbSphere.Checked = false;    }  else  {  // if sphere checked  rbSphere.Checked = true;  isCheckedSphere = false;  lradius.Show();  tbRadius.Show();  mRadius.Show();  ClearAll();  tbDC.Text = "0.47"; // Sets display value for Drag Coefficient after everything was cleared.    }  }  private void rdCube\_CheckedChanged(object sender, EventArgs e)  {  isCheckedCube = rbCube.Checked;  lLength.Hide();  tbLength.Hide();  mLength.Hide();  }  private void rdCube\_Click(object sender, EventArgs e)  {  if (rbCube.Checked && !isCheckedCube)  {  // if cube not checked  rbCube.Checked = false;    }  else  {  // if cube checked  rbCube.Checked = true;  isCheckedCube = false;  lLength.Show();  tbLength.Show();  mLength.Show();  ClearAll();  tbDC.Text = "1.05";  }  }    private void rbCylinder\_CheckedChanged(object sender, EventArgs e)  {  isCheckedCylinder = rbCylinder.Checked;  lradius.Hide();  tbRadius.Hide();    lHeight.Hide();  tbHeight.Hide();    mHeight.Hide();  mRadius.Hide();  }    private void rbCylinder\_Click(object sender, EventArgs e)  {  if (rbCylinder.Checked && !isCheckedCylinder)  {  // if cylinder not checked  rbCylinder.Checked = false;    }  else  {  // if cylinder checked  rbCylinder.Checked = true;  isCheckedCylinder = false;  lradius.Show();  tbRadius.Show();    mHeight.Show();  mRadius.Show();  lHeight.Show();  tbHeight.Show();  ClearAll();  tbDC.Text = "0.82";  }  }    private void rbCuboid\_CheckedChanged(object sender, EventArgs e)  {  isCheckedCuboid = rbCuboid.Checked;  lLength.Hide();  tbLength.Hide();    lHeight.Hide();  tbHeight.Hide();    lWidth.Hide();  tbWidth.Hide();    mLength.Hide();  mHeight.Hide();  mWidth.Hide();  }    private void rbCuboid\_Click(object sender, EventArgs e)  {  if (rbCuboid.Checked && !isCheckedCuboid)  {  // if cuboid not checked  rbCuboid.Checked = false;    }  else  {  // if cuboid checked  rbCuboid.Checked = true;  isCheckedCuboid = false;  lLength.Show();  tbLength.Show();    lHeight.Show();  tbHeight.Show();    lWidth.Show();  tbWidth.Show();  mLength.Show();  mHeight.Show();  mWidth.Show();  ClearAll();  tbDC.Text = "2.05";  }  }    private void rbPyramid\_CheckedChanged(object sender, EventArgs e)  {  isCheckedPyramid = rbPyramid.Checked;  lHeight.Hide();  tbHeight.Hide();    lLength.Hide();  tbLength.Hide();    mHeight.Hide();  mLength.Hide();  }    private void rbPyramid\_Click(object sender, EventArgs e)  {  if (rbPyramid.Checked && !isCheckedPyramid)  {  // if Pyramid not checked  rbPyramid.Checked = false;    }  else  {  // if Pyramid checked  rbPyramid.Checked = true;  isCheckedPyramid = false;  lLength.Show();  tbLength.Show();  mHeight.Show();  mLength.Show();  lHeight.Show();  tbHeight.Show();  ClearAll();  tbDC.Text = "0.4";  }  }  private void ValidScale(TextBox tb, int MaxValue) // Makes sure the size of the object being drawn isn't too large.  {  double value;  if(tb.Text != "") // If the textbox isn't empty do nothing  {  value = double.Parse(tb.Text); // Set value to the number input into the textbox.  if (value >MaxValue)  {  MessageBox.Show("Please input a smaller value!");  tb.Text = "";  }  }    }  private void tbRadius\_TextChanged(object sender, EventArgs e)  {  if(rbSphere.Checked) // If radius is being used for a sphere...  {  Sphere sphere = new Sphere(); // Instantiates class.  if (tbRadius.Text == "") // If textbox is empty...  {  Sphere.radius = 0;  }  else  {  Sphere.radius = double.Parse(tbRadius.Text); // If textbox isn't empty read the value to class.  }  sphere.GetArea(); // Calculates the Surface area of a sphere based on radius.  tbArea.Text = Math.Round(Sphere.Area, 2).ToString(); // Displays calculated Area.  sphere.GetVolume(); // Calculates the Volume of a sphere based on radius.  tbVolume.Text = Math.Round(Sphere.Volume, 2).ToString(); // Displays calculated Volume.  }  if(rbCylinder.Checked && tbHeight.Text != "") // If radius is being used for a Cylinder and Height isn't empty...  {  cylinder cylinder = new cylinder(); // Instantiates class.  if (tbRadius.Text == "") // If textbox is empty...  {  cylinder.radius = 0;  }  else  {  cylinder.radius = double.Parse(tbRadius.Text); // If textbox isn't empty read the value to class.  }  cylinder.height = double.Parse(tbHeight.Text);  cylinder.GetArea(); // Calculates the Surface area of a cylinder based on radius.  tbArea.Text = Math.Round(cylinder.Area, 2).ToString(); // Displays calculated Area.  cylinder.GetVolume(); // Calculates the Volume of a cylinder based on radius and Height.  tbVolume.Text = Math.Round(cylinder.Volume, 2).ToString(); // Displays calculated Volume.  }  ChangeMass(); // New volume means different mass so ChangeMass is called to calculate that.  Invalidate(); // Draws different size object based on values.  }    private void tbLength\_TextChanged(object sender, EventArgs e)  {  if (rbCube.Checked) // If length is being used for a cube...  {  Cube cube = new Cube(); // Instantiates class.  if (tbLength.Text == "") // If textbox is empty...  {  Cube.length = 0;  }  else  {  Cube.length = double.Parse(tbLength.Text); // If textbox isn't empty read the value to class.  }  cube.GetArea(); // Calculates the Surface area of a cube based on length.  tbArea.Text = Math.Round(Cube.Area, 2).ToString(); // Displays calculated Area.  cube.GetVolume(); // Calculates the Volume of a cube based on length.  tbVolume.Text = Math.Round(Cube.Volume, 2).ToString(); // Displays calculated Volume.  }  if (rbCuboid.Checked && tbHeight.Text != "" && tbWidth.Text != "") // If length is being used for a Cuboid and other values aren't empty...  {  Cuboid cuboid = new Cuboid(); // Instantiates class.  if (tbLength.Text == "") // If textbox is empty...  {  Cuboid.length = 0;  }  else  {  Cuboid.length = double.Parse(tbLength.Text); // If textbox isn't empty read the value to class.  }  Cuboid.height = double.Parse(tbHeight.Text);  Cuboid.width = double.Parse(tbWidth.Text);  // As the other values aren't empty they are read to the class so that Area and Volume can be Calculated.  cuboid.GetArea(); // Calculates the Surface area of a cuboid based on length, width and height.  tbArea.Text = Math.Round(Cuboid.Area, 2).ToString(); // Displays calculated Area.  cuboid.GetVolume(); // Calculates the Volume of a cuboid based on length, width and height.  tbVolume.Text = Math.Round(Cuboid.Volume, 2).ToString(); // Displays calculated Volume.  }  if (rbPyramid.Checked && tbHeight.Text != "") // If length is being used for a Pyramid and Height isn't empty...  {  Pyramid\_squrebase pyramid = new Pyramid\_squrebase(); // Instantiates class.  if (tbLength.Text == "") // If textbox is empty...  {  Pyramid\_squrebase.base\_length = 0;  }  else  {  Pyramid\_squrebase.base\_length = double.Parse(tbLength.Text); // If textbox isn't empty read the value to class.  }  Pyramid\_squrebase.height = double.Parse(tbHeight.Text);  // As the other value isn't empty it is read to the class so that Area and Volume can be Calculated.  pyramid.GetArea(); // Calculates the Surface area of a Pyramid based on length and height.  tbArea.Text = Math.Round(Pyramid\_squrebase.Area, 2).ToString(); // Displays calculated Area.  pyramid.GetVolume(); // Calculates the Volume of a Pyramid based on length and height.  tbVolume.Text = Math.Round(Pyramid\_squrebase.Volume, 2).ToString(); // Displays calculated Volume.  }  ChangeMass(); // New volume means different mass so ChangeMass is called to calculate that.  Invalidate(); // Draws different size object based on values.  }    private void tbHeight\_TextChanged(object sender, EventArgs e)  {  if (rbCylinder.Checked && tbRadius.Text != "") // If Height is being used for a Cylinder and radius isn't empty...  {  cylinder cylinder = new cylinder(); // Instantiates class.  if (tbHeight.Text == "") // If textbox is empty...  {  cylinder.height = 0;  }  else  {  cylinder.height = double.Parse(tbHeight.Text); // If textbox isn't empty read the value to class.  }  cylinder.radius = double.Parse(tbRadius.Text);  // As the other value isn't empty it is read to the class so that Area and Volume can be Calculated.  cylinder.GetArea(); // Calculates the Surface area of a Cylinder based on radius and height.  tbArea.Text = Math.Round(cylinder.Area, 2).ToString(); // Displays calculated Area.  cylinder.GetVolume(); // Calculates the Volume of a cylinder based on radius and Height.  tbVolume.Text = Math.Round(cylinder.Volume, 2).ToString(); // Displays calculated Volume.  }  if (rbCuboid.Checked && tbLength.Text != "" && tbWidth.Text != "") // If height is being used for a Cuboid and other values aren't empty...  {  Cuboid cuboid = new Cuboid(); // Instantiates class.  if (tbHeight.Text == "") // If textbox is empty...  {  Cuboid.height = 0;  }  else  {  Cuboid.height = double.Parse(tbHeight.Text); // If textbox isn't empty read the value to class.  }  Cuboid.length = double.Parse(tbLength.Text);  Cuboid.width = double.Parse(tbWidth.Text);  // As the other values aren't empty they are read to the class so that Area and Volume can be Calculated.  cuboid.GetArea(); // Calculates the Surface area of a cuboid based on length, width and height.  tbArea.Text = Math.Round(Cuboid.Area, 2).ToString(); // Displays calculated Area.  cuboid.GetVolume(); // Calculates the Volume of a cuboid based on length, width and height.  tbVolume.Text = Math.Round(Cuboid.Volume, 2).ToString(); // Displays calculated Volume.  }  if(rbPyramid.Checked && tbLength.Text != "")  {  Pyramid\_squrebase pyramid = new Pyramid\_squrebase(); // Instantiates class.  if (tbHeight.Text == "") // If textbox is empty...  {  Pyramid\_squrebase.height = 0;  }  else  {  Pyramid\_squrebase.height = double.Parse(tbHeight.Text); // If textbox isn't empty read the value to class.  }  Pyramid\_squrebase.base\_length = double.Parse(tbLength.Text); // As the other value isn't empty it is read to the class so that Area and Volume can be Calculated.  pyramid.GetArea(); // Calculates the Surface area of a Pyramid based on length and height.  tbArea.Text = Math.Round(Pyramid\_squrebase.Area, 2).ToString(); // Displays calculated Area.  pyramid.GetVolume(); // Calculates the Volume of a Pyramid based on length and height.  tbVolume.Text = Math.Round(Pyramid\_squrebase.Volume, 2).ToString(); // Displays calculated Volume.  }  ChangeMass(); // New volume means different mass so ChangeMass is called to calculate that.  Invalidate(); // Draws different size object based on values.  }    private void tbWidth\_TextChanged(object sender, EventArgs e)  {  if (rbCuboid.Checked && tbLength.Text != "" && tbHeight.Text != "")  {  Cuboid cuboid = new Cuboid(); // Instantiates class.  if (tbWidth.Text == "") // If textbox is empty...  {  Cuboid.width = 0;  }  else  {  Cuboid.width = double.Parse(tbWidth.Text); // If textbox isn't empty read the value to class.  }  Cuboid.height = double.Parse(tbHeight.Text);  Cuboid.length = double.Parse(tbLength.Text);  // As the other values aren't empty they are read to the class so that Area and Volume can be Calculated.  cuboid.GetArea(); // Calculates the Surface area of a cuboid based on length, width and height.  tbArea.Text = Math.Round(Cuboid.Area, 2).ToString(); // Displays calculated Area.  cuboid.GetVolume(); // Calculates the Volume of a cuboid based on length, width and height.  tbVolume.Text = Math.Round(Cuboid.Volume, 2).ToString(); // Displays calculated Volume.  }  ChangeMass(); // New volume means different mass so ChangeMass is called to calculate that.  // Invalidate isn't called as width cannot be seen in a 2D representation.  }    private void btExit\_Click(object sender, EventArgs e)  {  Practice practice = new Practice(); // Instantiates class for Practie.  practice.ShowDialog(); // Opens Practice.  this.Close(); // Closes ShapeCreator.  }  private void rbCustom\_Click(object sender, EventArgs e)  {    if (rbCustom.Checked && !isCheckedCustom)  {  rbCustom.Checked = false;  tbDC.ReadOnly = true; // If custom DC is checked allow user to write to textbox.  }  else  {  rbCustom.Checked = true;  isCheckedCustom = false;  tbDC.ReadOnly = false; // If custom DC isn't checked don't allow user to write to textbox.  }  }  private void rbCustom\_CheckedChanged(object sender, EventArgs e)  {  isCheckedCustom = rbCustom.Checked;  }    private void tbRadius\_KeyPress(object sender, KeyPressEventArgs e)  {  Practice.DoubleValidation(e, tbRadius); // Validation for tbRadius.  }    private void tbLength\_KeyPress(object sender, KeyPressEventArgs e)  {  Practice.DoubleValidation(e, tbLength); // Validation for tbLength.  }    private void tbWidth\_KeyPress(object sender, KeyPressEventArgs e)  {  Practice.DoubleValidation(e, tbWidth); // Validation for tbWidth.  }    private void tbHeight\_KeyPress(object sender, KeyPressEventArgs e)  {  Practice.DoubleValidation(e, tbHeight); // Validation for tbHeight.  }    private void tbDC\_KeyPress(object sender, KeyPressEventArgs e)  {  Practice.DoubleValidation(e, tbDC); // Validation for tbDC.  }    private void tbDC\_TextChanged(object sender, EventArgs e)  {  if(rbCustom.Checked && tbDC.Text != "")  {  ValidScale(tbDC,100); // Checks if Drag Coefficient is valid.  }  }  }  } |

## ShapeLoad

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using System.Data.OleDb;  using ShapeLibrary;    namespace NEA  {  public partial class ShapeLoad : Form  {    public static string SelectedShapeName = ""; // Sets SelectedShapeName to empty as when form is opened no Shape has been Selected.  public ShapeLoad()  {  InitializeComponent();  rbAll.Checked = true; // When Form is first opened Check rbAll so all Shapes stored on that account are seen.  CheckButtons();  }  private void CheckButtons()  {    OleDbConnection Conn = new OleDbConnection(Program.connString);  Conn.Open(); // Opens connection with Database.  OleDbCommand Cmd = new OleDbCommand(); //Create a database command object.  Cmd.Connection = Conn;  if (rbAll.Checked) // If all Shapes are checked select all Shape Properties stored with associated the username.  {  Cmd.CommandText = "SELECT ShapeName, ShapeType, Colour, Area, Mass, DragCo FROM Shapes WHERE Username= '" + Menu.CurrentUser + "'";  }  else if (rbSphere.Checked) // If Sphere is checked select all Shape Properties stored with associated the username where ShapeType is Circle.  {  Cmd.CommandText = "SELECT ShapeName, ShapeType, Colour, Area, Mass, DragCo FROM Shapes WHERE Username= '" + Menu.CurrentUser + "' AND ShapeType = 'Circle'";  }  else if (rbCubes.Checked) // If Cubes is checked select all Shape Properties stored with associated the username where ShapeType is Cube.  {  Cmd.CommandText = "SELECT ShapeName, ShapeType, Colour, Area, Mass, DragCo FROM Shapes WHERE Username= '" + Menu.CurrentUser + "' AND ShapeType = 'Cube'";  }  else if (rbCuboid.Checked) // If Cuboid is checked select all Shape Properties stored with associated the username where ShapeType is Cuboid.  {  Cmd.CommandText = "SELECT ShapeName, ShapeType, Colour, Area, Mass, DragCo FROM Shapes WHERE Username= '" + Menu.CurrentUser + "' AND ShapeType = 'Cuboid'";  }  else if (rbCylinder.Checked) // If Cylinder is checked select all Shape Properties stored with associated the username where ShapeType is Cylinder.  {  Cmd.CommandText = "SELECT ShapeName, ShapeType, Colour, Area, Mass, DragCo FROM Shapes WHERE Username= '" + Menu.CurrentUser + "' AND ShapeType = 'Cylinder'";  }  else if (rbPyramid.Checked) // If Pyramid is checked select all Shape Properties stored with associated the username where ShapeType is Pyramid.  {  Cmd.CommandText = "SELECT ShapeName, ShapeType, Colour, Area, Mass, DragCo FROM Shapes WHERE Username= '" + Menu.CurrentUser + "' AND ShapeType = 'Pyramid'";  }  OleDbDataAdapter da = new OleDbDataAdapter(Cmd);  DataTable table = new DataTable();  da.Fill(table); // Stores selected data into table.  dgwShapes.DataSource = table; // Displays selected data in DataGridView object.  Conn.Close(); // Closes Connection to database.  }  private void rbAll\_Click(object sender, EventArgs e)  {  CheckButtons(); // When a radiobutton is checked it unchecks any other radiobuttons so only one can be selected. Calls CheckButtons to display appropriate data.  }    private void rbSphere\_Click(object sender, EventArgs e)  {  CheckButtons(); // When a radiobutton is checked it unchecks any other radiobuttons so only one can be selected. Calls CheckButtons to display appropriate data.  }    private void rbCubes\_Click(object sender, EventArgs e)  {  CheckButtons(); // When a radiobutton is checked it unchecks any other radiobuttons so only one can be selected. Calls CheckButtons to display appropriate data.  }    private void rbCuboid\_Click(object sender, EventArgs e)  {  CheckButtons(); // When a radiobutton is checked it unchecks any other radiobuttons so only one can be selected. Calls CheckButtons to display appropriate data.  }    private void rbCylinder\_Click(object sender, EventArgs e)  {  CheckButtons(); // When a radiobutton is checked it unchecks any other radiobuttons so only one can be selected. Calls CheckButtons to display appropriate data.  }    private void rbPyramid\_Click(object sender, EventArgs e)  {  CheckButtons(); // When a radiobutton is checked it unchecks any other radiobuttons so only one can be selected. Calls CheckButtons to display appropriate data.  }    private void dgwShapes\_RowHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)  {  btLoad.Show(); // When the Header of a row is clicked the button to load the values stored in that row is shown.  }    private void btLoad\_Click(object sender, EventArgs e)  {  LoadSelectedShape(dgwShapes, 0); // Loads Selected Shape based on ShapeName (in Column 0) in DataGridView Shapes.  Practice p = new Practice();  p.ShowDialog(); // Open Practice.  this.Close(); // Closes ShapeLoad once shape has been loaded.  }    public static void LoadSelectedShape(DataGridView DGW, int Column)  {  Practice.IsShapeLoaded = false;  Practice.IsCircleLoaded = false;  Practice.IsCubeLoaded = false;  Practice.IsCuboidLoaded = false;  Practice.IsCylinderLoaded = false;  Practice.IsShapeLoaded = true;  // Sets all values to false as a completely new shape is being loaded, specific shape will be set to true once its been determined which shape is being loaded.  // IsShapeLoaded is set to true as a shape is being loaded.  // Only one type of shape can be loaded at one time so the rest must be set to false.  SelectedShapeName = DGW.CurrentRow.Cells[Column].Value.ToString(); // Sets the name of the selected shape, which Column name is in depends on which DataGridView its being loaded from.  OleDbConnection Conn = new OleDbConnection(Program.connString);  Conn.Open(); // Opens connection with Database.  OleDbCommand Cmd = new OleDbCommand(); //Create a database command object.  Cmd.Connection = Conn;  Cmd.CommandText = "SELECT Mass, Area, DragCo, Colour, ShapeType FROM Shapes WHERE ShapeName='" + SelectedShapeName + "' AND Username= '" + Menu.CurrentUser + "'";  // Selects Mass, Area, DragCo, Colour, ShapeType of the selected Shape stored under the Current User's Username.  OleDbDataReader reader = Cmd.ExecuteReader(); //Runs the query & allows results to be read.  reader.Read(); //Read the record found.  Shapes.Mass = Convert.ToDouble(reader["Mass"]);  Shapes.Area = Convert.ToDouble(reader["Area"]);  Shapes.DragCoefficient = Convert.ToDouble(reader["DragCo"]);  Shapes.colour = Color.FromName(reader["Colour"].ToString());  // Sets basic shape values from database so they can be displayed in Launch  if (reader["ShapeType"].ToString() == "Circle") // If a circle is loaded...  {  Practice.IsCircleLoaded = true;  reader.Close(); // Closes previously opened reader so another query can be ran.  Cmd.CommandText = "SELECT Radius FROM Circle WHERE ShapeName='" + SelectedShapeName + "'";  reader = Cmd.ExecuteReader(); //Runs the query & allows results to be read.  reader.Read(); //Read the record found.  Sphere.radius = Convert.ToDouble(reader["Radius"]); // Sets the value of radius so that it can be used to draw the Shape in Practice Form.  Practice.size = Convert.ToInt32(Sphere.radius \* 10);  Practice.circle = new Rectangle(100 - Practice.size, 500 - Practice.size \* 2, Practice.size \* 2, Practice.size \* 2); // Sets new values for Drawing object in Practice Forms which is drawn when Pratice is opened.  }  else if (reader["ShapeType"].ToString() == "Cube") // If a Cube is loaded...  {  Practice.IsCubeLoaded = true;  reader.Close(); // Closes previously opened reader so another query can be ran.  Cmd.CommandText = "SELECT Length FROM Cube WHERE ShapeName='" + SelectedShapeName + "'";  reader = Cmd.ExecuteReader(); //Runs the query & allows results to be read.  reader.Read(); //Read the record found.  Cube.length = Convert.ToDouble(reader["Length"]); // Sets the value of length so that it can be used to draw the Shape in Practice Form.  Practice.size = Convert.ToInt32(Cube.length \* 10);  Practice.cube = new Rectangle(100 - Convert.ToInt32(Practice.size \* 0.5), 500 - Practice.size, Practice.size, Practice.size); // Sets new values for Drawing object in Practice Forms which is drawn when Pratice is opened.  }  else if (reader["ShapeType"].ToString() == "Cuboid") // If a circle is loaded...  {  Practice.IsCuboidLoaded = true;  reader.Close(); // Closes previously opened reader so another query can be ran.  Cmd.CommandText = "SELECT Length, Height FROM Cuboid WHERE ShapeName='" + SelectedShapeName + "'";  reader = Cmd.ExecuteReader(); //Runs the query & allows results to be read.  reader.Read(); //Read the record found.  Cuboid.length = Convert.ToDouble(reader["Length"]); // Sets the value of length so that it can be used to draw the Shape in Practice Form.  Practice.size = Convert.ToInt32(Cuboid.length \* 10);  Cuboid.height = Convert.ToDouble(reader["Height"]); // Sets the value of height so that it can be used to draw the Shape in Practice Form.  Practice.size2 = Convert.ToInt32(Cuboid.height \* 10);  Practice.cuboid = new Rectangle(100 - Convert.ToInt32(Practice.size \* 0.5), 500 - Practice.size2, Practice.size, Practice.size2); // Sets new values for Drawing object in Practice Forms which is drawn when Pratice is opened.  }  else if (reader["ShapeType"].ToString() == "Cylinder") // If a Cylinder is loaded...  {  Practice.IsCylinderLoaded = true;  reader.Close(); // Closes previously opened reader so another query can be ran.  Cmd.CommandText = "SELECT Radius, Length FROM Cylinder WHERE ShapeName='" + SelectedShapeName + "'";  reader = Cmd.ExecuteReader(); //Runs the query & allows results to be read.  reader.Read(); //Read the record found.  cylinder.radius = Convert.ToDouble(reader["Radius"]); // Sets the value of radius so that it can be used to draw the Shape in Practice Form.  Practice.size = Convert.ToInt32(cylinder.radius \* 10);  cylinder.height = Convert.ToDouble(reader["Length"]); // Sets the value of length so that it can be used to draw the Shape in Practice Form.  Practice.size2 = Convert.ToInt32(cylinder.height \* 10);  Practice.Cylinder = new Rectangle(100 - Convert.ToInt32(Practice.size2 \* 0.5), 500 - Practice.size \* 2, Practice.size2, Practice.size \* 2); // Sets new values for Drawing object in Practice Forms which is drawn when Pratice is opened.  }  else if (reader["ShapeType"].ToString() == "Pyramid") // If a Pyramid is loaded...  {  Practice.IsPyramidLoaded = true;  reader.Close(); // Closes previously opened reader so another query can be ran.  Cmd.CommandText = "SELECT Length, Height FROM Pyramid WHERE ShapeName='" + SelectedShapeName + "'";  reader = Cmd.ExecuteReader(); //Runs the query & allows results to be read.  reader.Read(); //Read the record found.  Pyramid\_squrebase.base\_length = Convert.ToDouble(reader["Length"]); // Sets the value of length so that it can be used to draw the Shape in Practice Form.  Practice.size = Convert.ToInt32(Pyramid\_squrebase.height \* 10);  Pyramid\_squrebase.height = Convert.ToDouble(reader["Height"]); // Sets the value of height so that it can be used to draw the Shape in Practice Form.  Practice.size2 = Convert.ToInt32(Pyramid\_squrebase.height \* 10);  // Polygon works based on a set of points in an array, if Pyramid is loaded then the points are set when Practice Form is Opened.  }  Conn.Close(); // Closes Connection to database.    }  private void btExit\_Click(object sender, EventArgs e)  {  Practice practice = new Practice();  practice.ShowDialog(); // Opens Practice.  this.Close(); // Closes ShapeLoad.  }  }  } |

## LaunchLoad

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using System.Data.OleDb;    namespace NEA  {  public partial class LaunchLoad : Form  {  public LaunchLoad()  {  InitializeComponent();    OleDbConnection Conn = new OleDbConnection(Program.connString);  Conn.Open(); // Opens Connection to the database.  OleDbCommand Cmd = new OleDbCommand(); //Create a database command object.  Cmd.Connection = Conn;  Cmd.CommandText = "SELECT LaunchID, ShapeName, Velocity, Angle, Gravity, FluidDensity, DragOn, HitGround, MaxHeight, Range, Time FROM LaunchValues WHERE Username= '" + Menu.CurrentUser + "'"; // Selects values where Username matches the logged in user.  OleDbDataAdapter values = new OleDbDataAdapter(Cmd);  DataTable table = new DataTable();  values.Fill(table); // fills the table with the values stored with the logged in user.  dgwLaunchValues.DataSource = table;  Conn.Close(); // Closes connection to the database.  }    private void btExit\_Click(object sender, EventArgs e)  {  Practice practice = new Practice();  practice.ShowDialog(); // Opens Practice.  this.Close(); //Closes LaunchLoad.  }    private void btLoad\_Click(object sender, EventArgs e)  {    int SelectedLaunchID = Convert.ToInt32(dgwLaunchValues.CurrentRow.Cells[0].Value); // The value stored in the selected rows first culumn (the LaunchID) is set to SelectedLaunchID.    OleDbConnection Conn = new OleDbConnection(Program.connString);  Conn.Open(); // Opens Connection to the database.  OleDbCommand Cmd = new OleDbCommand(); //Create a database command object.  Cmd.Connection = Conn;  Cmd.CommandText = "SELECT \* FROM LaunchValues WHERE LaunchID=" +SelectedLaunchID+ " AND Username = '" +Menu.CurrentUser+ "'"; // Selects all the values from the appropriate LaunchID.  OleDbDataReader reader = Cmd.ExecuteReader();  reader.Read(); //Runs the query & allows results to be read.  Practice.Velocity = Convert.ToDouble(reader["Velocity"]); // Sets the read value's "Velocity" to the Velocity in Practice Form.  Practice.angle = Convert.ToDouble(reader["Angle"]); // Sets the read value's "Angle" to the Angle in Practice Form.  Practice.AtoG = Convert.ToDouble(reader["Gravity"]); // Sets the read value's "Gravity" to the AtoG in Practice Form.  Practice.fluidD = Convert.ToDouble(reader["FluidDensity"]); // Sets the read value's "FluidDensity" to the fluidD in Practice Form.  if (Convert.ToInt32(reader["DragOn"]) == 0) // If Drag is off (shape is dimensionless)...  {  Practice.IsShapeLoaded = false; // Set shape loaded to false (no shape is loaded).  }  else // If object has dimensions.  {  Conn.Close(); // Close previusly opened connection.  ShapeLoad.LoadSelectedShape(dgwLaunchValues,1); // Call LoadSelectedShape subroutine to read the value in the second row (ShapeName) and Load the Values associated wiht that shape.  }  Practice practice = new Practice();  practice.ShowDialog(); // Opens Practice.  this.Close(); // Closes LaunchLoad.  }    private void dgwLaunchValues\_RowHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)  {  btLoad.Show(); // When the Header of a row is clicked the button to load the values stored in that row is shown.  }  }  } |

## Classes

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Drawing;    namespace ShapeLibrary  {  public abstract class Shapes  {  public static double Mass { get; set; }  public static double Volume { get; set; }  public static double Density { get; set; }  public static double Area { get; set; }  public static Color colour { get; set; }  public static double DragCoefficient {get; set;}    // set as abstract so that they have to be overriden in subclass as all shapes have different calculations for volume.  public abstract void GetVolume();  public abstract void GetArea();  public static void GetMass()  {  Mass = Density \* Volume;  }    }  } |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Drawing;    namespace ShapeLibrary  {  public class Sphere : Shapes  {  public static double radius { get; set; }  private double multiplier = 1.3333333333333333;    public override void GetVolume()  {  Volume = multiplier\*Math.PI \* radius \* radius \* radius;  }  public override void GetArea()  {  Area = Math.PI \* radius \* radius;  }  }  } |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Drawing;    namespace ShapeLibrary  {  public class Cube : Shapes  {  public static double length { get; set; }  public override void GetVolume()  {  Volume = length \* length \* length;  }  public override void GetArea()  {  Area = length \* length;  }    }  } |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Drawing;    namespace ShapeLibrary  {  public class Cuboid : Shapes  {  public static double length { get; set; }  public static double width { get; set; }  public static double height { get; set; }  public override void GetVolume()  {  Volume = length \* width \* height;  }  public override void GetArea()  {  Area = height \* width;  }  }  } |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Drawing;    namespace ShapeLibrary  {  public class cylinder : Shapes  {  public static double radius { get; set; }  public static double height { get; set; }  public override void GetVolume()  {  Volume = Math.PI \* radius \* radius \* height;  }  public override void GetArea()  {  Area = Math.PI \* radius \* radius;  }  }  } |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Drawing;    namespace ShapeLibrary  {  public class Pyramid\_squrebase : Shapes  {  public static double base\_length { get; set; }  public static double height { get; set; }  public override void GetVolume()  {  Volume = (0.3333333333333333) \* base\_length \* base\_length \* height;  }  public override void GetArea()  {  Area = 0.5 \* base\_length \* Math.Sqrt((base\_length \* base\_length) +(height \* height));  }  }  } |

# **Testing**

## Testing Plan

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# **Evaluation**

## Evaluation against objectives

## Overall assessment of the project

## User feedback

## Analysis of User feedback

## Possible extensions and improvements