Mechanics Simulator 2014

Project Log

Matthew Arnold

Barton Peveril Sixth Form College

Centre Number: 58231

Table of Contents

Analysis **5**
Background to the problem 5
Description of the current system 5
Identification of the prospective user 6
Identification of user needs and acceptable limitations 6
User Needs 6
Acceptable limitations 6
Potential Solutions 7
Justification of chosen solution 8
Analysis Data Dictionary 8
Data Flow Diagrams 10
Data Volumes 10
Objects 11
Objectives 11
General Objectives 11
Specific Objectives 11
Evidence of use of appropriate analysis techniques 12
Design **13**
Overall System Design 13
Description of Modular Structure of the System 13
Definition of Data Requirements 17
File Organisation and Description of Record Structure 17
Validation Required 18
Identification of Storage Media 19
Identification of Suitable Algorithms for Data Transformation 19
Encryption and Decryption 19
Timers 20
Dragging the Main Window Around 20
Projectile Motion Simulation 21
Class Definitions 23
Buttons 23
Menus 24

Text Boxes 25
Screens 26
Screen Manager 34
User Interface Design 35
Main Menu 35
Simulation 36
Test Mode 39
My Progress 41
Settings 43
Security of Data 43
System Security 43
Overall Test Strategy 43
System Testing 44
Test Series 1 44
Test Series 2 46
Test Series 3 48
Test Series 4 50
Sliders 52
Test Series 5 52
Test Series 6 54
Test Series 7 56
Test Series 8 63
Test Series 9 64
Test Series 10 68
Test Series 11 76
System Maintenance 79
System Overview 79
Graphics 80
Dragging the Main Window Around 81
Managing Screens 81
Updating, Handling Input and Keeping a List of Screens 82
Drawing Screens 83
Screen Transitions 83
Debug Screen 84

Encryption and Decryption 85
Encryption 85
Decryption 86
Timers 88
Projectile Motion Simulation 90
Code 93
Forms 93
Classes 102
ScreenManager 102
BaseScreen 104
Debug 104
Settings 107
Title 108
SimulationButton 109
TestButton 112
MyProgressButton 114
SimulationMenu 116
ProjectileMotion 119
ProjectileMotionSimulation 124
ResolvingForces 129
ResolvingForcesSimulation 134
ForcesOnSlopes 137
ForcesOnSlopesSimulation 142
TestMenu 146
ProjectileMotionTest 149
ResolvingForcesTest 155
ForcesOnSlopesTest 159
TestReport 163
MyProgressReport 166
UserSelection 171
TestUserSelection 173
MyProgressUserSelection 174
BaseButton 175
PictureButton 176
TextButton 177
BaseMenu 181
AlignLeftMenu 181
AlignCentreMenu 183

NumberBox 184
WritingBox 187
User Manual 190
Appraisal 190
Completion of Project Objectives 190
General Objectives 190
Specific Objectives 193
Evidence of Authenticated User Feedback 197
Analysis of User Feedback 198
Possible Extensions 199
Appendices 200
Appendix 1 – User Guide 200

Analysis

Background to the problem

Mechanics could be made better and more interesting by using interactive simulations. A system will be devised to help teach students Mechanics (from a Maths or Physics course), and help the students revise it. It will allow users to actually see what happens in Mechanics situations, rather than having to imagine it using calculated numbers.

Description of the current system

At the moment teaching mechanics and, more importantly, revising it is heavily based on doing previous years' exam paper questions. Some of these questions may use a simple diagram (such as the example below) to try to illustrate the problem. However, not all questions use these diagrams and the diagrams only show one moment in time, such as the starting condition of the situation.

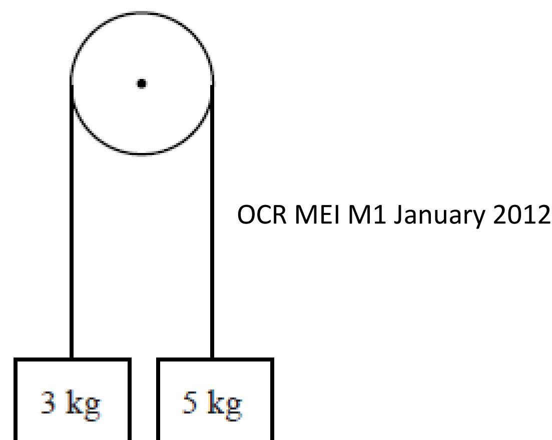


Fig. 1

This is what students have to deal with in an actual exam, but for learning the topics for the first time, it may be difficult to understand what actually happens after the initial condition. The only indication of what happens is from calculated numbers.

According to my End User, other current learning techniques include drawing posters in groups about the most difficult topics, which could involve still pictures of a Mechanics situation. As well as this, he asks students to answer questions from text books, which often don't have diagrams. He felt that some students would find it difficult and therefore take longer to understand the concepts behind the questions because of the lack of visual illustration. He wants a system to address this issue.

Identification of the prospective user

The end user for my system is Tristan White, a Maths Teacher at my college. He taught me a Mechanics module as part of my Maths AS course. He is an acceptable end user because he understands the theory of the subject and can easily communicate ideas with me without too much explanation.

Identification of user needs and acceptable limitations

User Needs

Following an interview with my end user, I have found some features that he would like to be included in the system:

- For the simulations, there should be a pre-set situation in which values could be altered, rather than having an open space with a toolbox of components and the ability to create custom situations. For learning, students would benefit better from being given a framework to play around with and learn from. My end user pointed out that it would probably take a whole lesson for a student to create a model useful enough to learn with from scratch about something they haven't learned yet.
- There should be simulations covering various different topics in mechanics:
    - Vectors
    - Kinematics
    - Newton's Laws
    - Forces at Angles
    - Projectiles

  However, my end user suggested that the system should focus more on teaching the projectiles topic, as well as a "stuff on slopes" model, since these areas are ones which he thinks students struggle the most on.

- I asked my end user who would be ultimately using the system, teachers or students and whether a program would therefore need a password to be entered to use. He said that the system could be used by either students or teachers. It would be mainly used by teachers in demonstrations for lessons, but then if there was time in a computer room, the students could experiment with it. He said that, for this reason, a password would not be necessary.

Acceptable limitations

- For teachers and students, a basic understanding of Mechanics can be assumed. It shouldn't be too technical, so that learning students can still understand it but teachers may not need the system to explain technical details that they would be expected to know if they are teaching the subject. For example, abbreviating the word gravity to the letter 'g' could be seen as acceptable.
- The Mechanics course that I will be targeting the system at is based on modelling motion and forces in real life. However, these models are obviously not completely realistic and, especially at this relatively low level of the subject, some significant assumptions are made by the course itself. It would therefore be acceptable, and probably necessary, to comply with these assumptions in the simulations. The course's assumptions include:

    - Allowing any effect of air resistance to be ignored, resulting in simulated falling objects to continue accelerating indefinitely.
    - Gravity remains constant regardless of height.
- The Mechanics course only models two-dimensional motion, and so it would be an unnecessary extension to create a system which deals with 3-D objects. It may also not be feasible try to create a program which renders in 3-D because of my time constraint. Trying this would require significantly more time to complete, which I do not have.

Potential Solutions

A very simple solution could be one that does not use a computer at all. By drawing a Mechanics situation on paper at multiple points in time, a kind of slideshow is made with a slight simulation aspect. The disadvantage of this solution would be that drawing lots of pictures is very time consuming. Also, the different frames may not be drawn in a way that makes them accurate to compare. For example, an object may be drawn smaller in one than in another.

I could make a VB.NET Windows Forms Application for desktop computers or laptops, which simulates Mechanics situations on the screen. There could be a separate simulation for each type of exam paper question situation, such as one for Projectile Motion and one for Masses on a Slope. The user could change the initial constants and variables, and then press a play button to see what happens. The simulations would run smoothly and be in colour, like a video. They could be paused at any point. There would also be a 'test mode' in which the user is presented with an initial situation and is asked to calculate the answer to a question about the situation. Once they enter their answer, the simulation runs to show them if they are correct. Their progress would then be saved in a text file, and the user could view their overall progress for each simulation.

Another potential solution could be to make an Android Mobile simulation app. Much like the VB.NET application, there would be smooth, colour simulations, except the user would be able to run them on their mobile. This portability is good and allows for revision or learning anywhere, such as on the bus home, or during breaks between lessons. However, it would be impractical to use the app within a lesson, since it would encourage the use of mobile phones during lessons. Also, it would make using it for teacher demonstrations difficult as it couldn't easily be projected onto an interactive whiteboard like a PC program could be. Another drawback could be that, due to the fact that most mobile devices have quite small screens, it may be difficult to make enough sense of the simulation, and there wouldn't be much room for a meaningful simulation.

I could create a database of Mechanics past exam paper questions, where questions could be searched for by their topic, relative difficulty, length or year or release. There could also be a feature which generates a 'mock exam paper' for a particular topic or difficulty. This would definitely make using past papers to revise much more efficient, but there is no aspect of simulation, and this would make the system very difficult to learn things for the first time from.

Justification of chosen solution

I have decided to make the VB.NET Windows Forms Application. As well as the benefits I have already mentioned, a bigger screen would allow for more detailed and meaningful simulations which can be run with one click. Also, the completion of this program would be very time flexible due to the ability to split it into different simulations. I can start by working on the most important simulations, for topics which students struggle most on. If I manage to complete those with plenty of time remaining, I could extend the program by adding simulations for more topics. This means that I can make the most efficient use of my time when creating the program, and will hopefully end up with a finished set of simulations.

Analysis Data Dictionary

The following table lists the fields that will be used to store data in a text file about each user's progress in the test mode part of the program. There will be a separate text file for each user of the program:

Field Name	Description	Data Type	Field Size	Example
Category	The simulation that the question asked was about.	String	Any	Projectile Motion
Score	The score (as a percentage) that the user got for the question.	Integer	3 figures	67
TimeScored	The date and time that the user got this score. This could be used to chronologically order multiple test scores, to see progress over time.	Date	DD/MM/YYYY HH:MM	06/10/2013 21:07

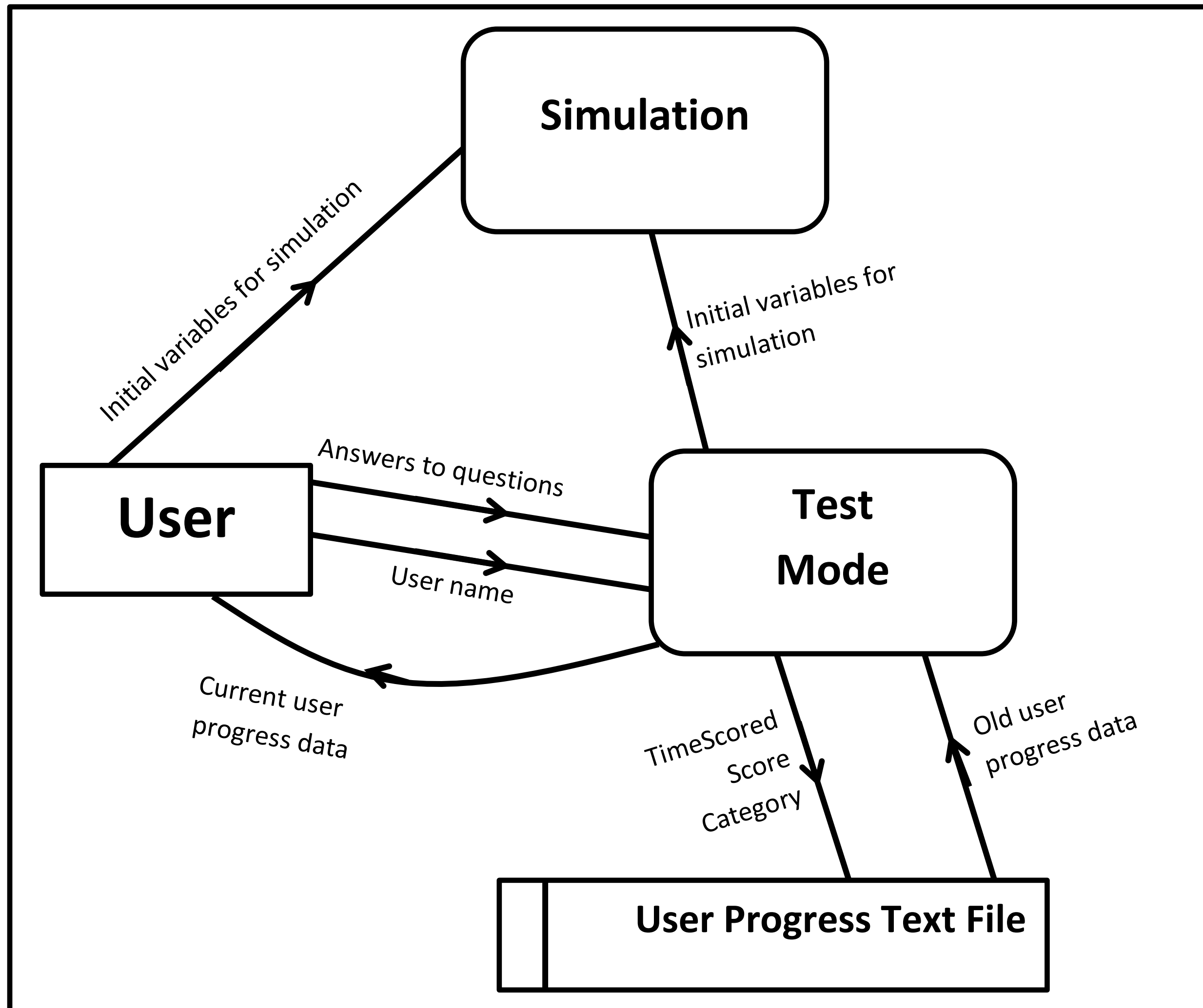
The next table lists the most important variables that will apply to most moving objects in a simulation. These variables are ones which the user could change or set before running the simulation, but would be updated by calculation as the simulations run:

Variable Name	Description	Data Type	Variable Size	Example
XA	The horizontal component of the acceleration of the object (Pixels per Tick per Tick).	Double	8 bytes	0.0
YA	The vertical component of the acceleration of the object (Pixels per Tick per Tick).	Double	8 bytes	2.125
XV	The horizontal component of the velocity of the object (Pixels per Tick).	Double	8 bytes	20.0
YV	The vertical component of the velocity of the object (Pixels per Tick).	Double	8 bytes	56.7878
XS	The horizontal component of the displacement of the object from the top-left corner of the simulation (Pixels).	Double	8 bytes	0.125
YS	The vertical component of the displacement of the object from the top-left corner of the simulation	Double	8 bytes	125.65

	(Pixels).			
Mass	The mass of the object (Kilograms). This is only applicable for objects in simulations where resolving forces is required.	Integer	4 bytes	200
XF	The horizontal component of the resultant force acting on the object (Newtons). This is only applicable for objects in simulations where resolving forces is required.	Double	8 bytes	400.6
YF	The vertical component of the resultant force acting on the object (Newtons). This is only applicable for objects in simulations where resolving forces is required.	Double	8 bytes	459.0
Angle	The angle, in Radians ($0 \leq angle < 2\pi$, 0 being directly upwards), of the motion. This is only applicable for objects which are projectiles.	Double	8 bytes	3.14159

Data Flow Diagrams

Since there is no real existing system in place at the moment for the learning of Mechanics, I cannot create a Data Flow Diagram for it. However, the Data Flow Diagram below shows my proposed system:



The user could either choose to just look at the simulations, in which case they would enter/alter the initial variables for the simulation before watching it, or use the test mode. In this case the initial variables are set up automatically and questions are asked about the simulation before it is run. After the user inputs their answers to the questions, the simulation runs. The user's score on each test is saved in a text file, and the user's overall progress so far can be displayed to them. Each new user of the program on a machine will be assigned a new Progress Text File.

Data Volumes

Since I am going to create a program for desktop PCs and laptops, I won't need to worry about not having enough memory space. However, it is still useful to think about how I could minimise the space used.

Because I will be using VB.NET to write the program, the most logical way to integrate sound would be to add the sound files to the project resources. The only sound file type supported by this method is the .wav (uncompressed) format, which uses a relatively huge amount of space. If I decided to use sound in my program, I should try to minimise the duration of any sounds to be played. These sound files, at a 24-bit sample resolution, will take up approximately 4GB for one hour.

Image files with large resolutions could also potentially become a storage problem and I should consider an image's format and size before including it in my project.

Objects

There are four different types of objects that I will need to create for my program:

- Simulations – The most important part of the program. These will be the actual animated situations that the user would be able to play around with
- Tests – These could use the simulations to generate exam-style questions
- Menu screens – The user will need to be able to navigate around the program. This will include the title screen, as well as the menu for possible simulations and tests
- Tools – I will create classes for buttons and lists so that I can customise the look and behaviour of them, rather than using the Windows Forms controls

Objectives

General Objectives

1. Create a VB.NET Windows Forms Application which could be used to help to teach students Mechanics principles for the first time.
2. The program should also act as an effective revision tool for students.
3. There should be at least one simulation about projectile motion.
4. There should be at least one simulation about resolving forces.
5. There should be at least one simulation about resolving forces at angles ("Stuff on slopes").
6. There should be a graphics system in place which ensures that the simulations run smoothly without any flickering or 'lag' on an average machine.
7. As well as the simulations the program should include a test mode, in which the user is asked an exam-style question based on the starting condition of a prepared situation before seeing a simulation that reveals the answer.

Specific Objectives

1. In the test mode the questions asked should have a total mark and the user's answers should be marked as a percentage.
2. Each time a user answers a question in test mode, the score, date/time of answering and question category should be saved in a text file.
3. Each user of the program on a machine should have their own progress text file assigned to them. If a new user uses the program, a new text file should be created.
4. A user should be able to view their progress over time with the test mode for any particular question category. This could be displayed as a graph.

5. When the test mode or 'My Progress' is selected, a list of existing users should be displayed. If the user has used the program on that machine before, they can select their name from the list. If they are not on the list, there will be a text box for them to create a new user name, thus creating a new progress text file.
6. The simulations should be visually pre-set, but users should be able to input/alter starting variables and constants before running the simulation.
7. Simulations should be able to be paused at any time.
8. There should be keyboard bindings to the simulation play, pause and reset functions. For example, the user could press the space bar to pause the simulation. This would make those functions easier to use, and gives an alternative to clicking with the mouse.
9. I will need to be able to use traditional SI units for quantities, such as "metres per second" for velocity, rather than "pixels per tick". For this reason, I will create a method for converting between the pixel and metre forms.
10. On the menu for selecting simulations, there should be an image previewing each simulation. This would make the program look more interesting, as well as giving the user a taster of each simulation before needing to run them.
11. Each user's progress data string should be encrypted before being written to file, to prevent users from cheating by altering their scores.

Evidence of use of appropriate analysis techniques

To gather more information about what my end user wanted from the new system, I interviewed him. The transcript is as follows:

Q - What are your current methods for teaching Mechanics?
A - One might be to get students to work in pairs to create posters on the more difficult topics like projectiles. Also, I get them to answer questions from the text book. The limitation or lack of actual diagrams involved in these make it difficult for some students to learn.
Q - Are there any topics within Mechanics which students find the hardest and perhaps deserve more focus in the program?
A - Projectiles would be one topic. Also if possible a 'stuff on slopes' model would be nice.
Q - The program will use simulations. Do you think it would be better to have a series of pre-set situations, in which the user could change the numbers (i.e. change the mass, angle or friction), or be able to make a unique situation using, for example, a toolbox of masses and wires?
A - A pre-set situation would be more useful. Many students would take the whole lesson to get anything approaching useful as a model.
Q - Would the program be used by only teachers (and shown to the class using the projector) or be available for students to use as well? This also implies the question: Would the program need some kind of password protection so only teachers could access it?
A - It would be used in teacher led demonstrations, but then if there were time in a computer room where the students could play around with it. A password would not be necessary.
Q - Do you have any resources or documents which you think may help me to make the program?
A - I would suggest looking at Section B past paper questions to get an idea of the difficulty of questions that would need to be tackled.

Design

Overall System Design

There will be three main sections to my program:

- Simulation
- Test
- My Progress

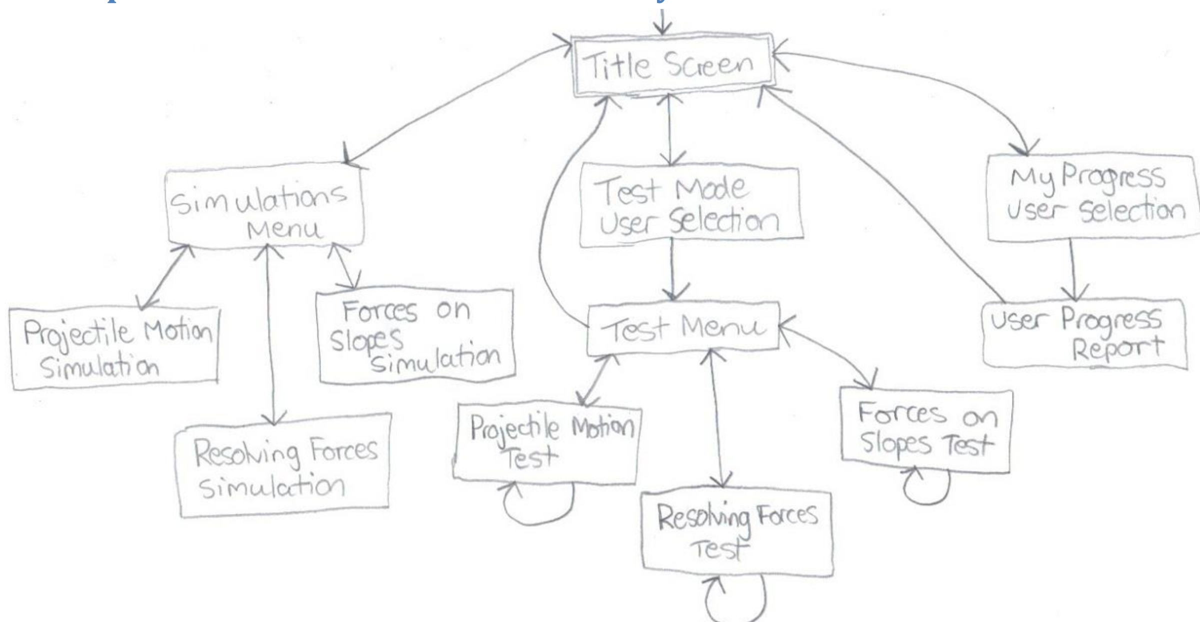
The first, Simulation, will be for playing around with the various simulations in the program. All of the important variables and constants could be set before pressing play and watching the simulation run. This section would be good for learning, and useful for teacher demonstrations.

The second, Test, is for examining the knowledge learned from lessons, or from the simulation section. An exam-style question will be asked about the simulation category that the user chooses. After inputting the answers, the simulation runs to show the user if they are correct. The user's score is saved in their progress file.

The third and final section, My Progress, will be a way for a user to see their progress over time in the Test section. This information could help them decide which topics they need to revise more on, and which topics they are doing really well in.

I will not be using the conventional graphics system for a Windows Forms Application. This means that I will be using very few in-built objects, such as buttons and text boxes, and won't be using the Windows Forms Designer much. Instead, I will be using the Graphics.CreateGraphics methods, and drawing to a Bitmap in memory, which will be drawn to the user's screen frame-by-frame.

Description of Modular Structure of the System

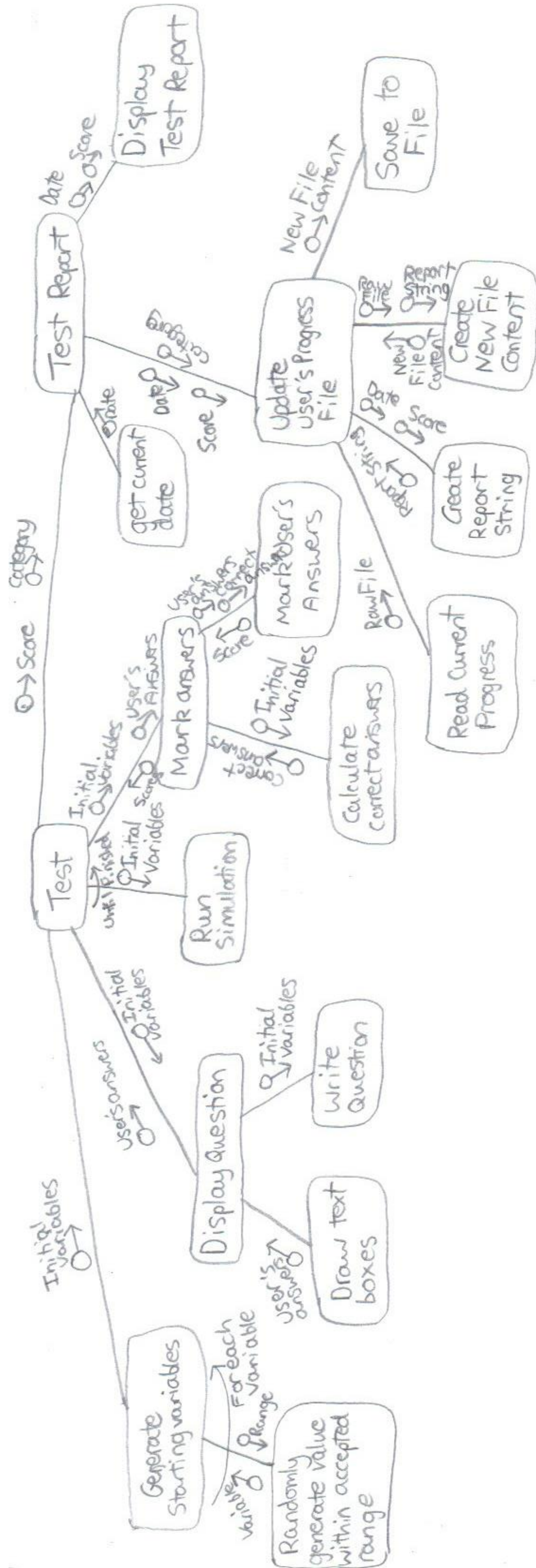


The diagram above shows the screen navigation layout for the program. Each box represents one screen, or distinct view to the user. The arrows show the possible paths between screens, for example, by clicking buttons. The first screen shown to the user, obviously, is the title screen.

This diagram highlights how the user selection process works. When the user wishes to look at the Test or My Progress sections, they first must pass a user selection screen, to select or create their user name. Once past the user selection screens, the user reaches the actual section they were looking for. It is not possible to get directly back to user selection. Any 'Back' button would go back to the title screen.

In the tests for any category, there is a loop. This shows that after completing a test, the user will have the option to reload that same test category without having to go back to the menu and select it again.

It is important to note that in both the file reading and writing processes, the encryption and decryption of the text file's contents would also take place. (See the Identification of Suitable Algorithms section)



The diagram to the left shows the structure of a test, from generating the initial variables, to getting the user's answers, to running and displaying the simulation, marking the user's answers and displaying the test report.

Read User Progress File

Display Data

Display Average Score

Display worst and Best categories

Find Best and worst category

Find average score per each category

Calculate average score

Get average

Add up scores

Split into Reports

Split Reports Into Fields

Split Raw File into Each Report

Read Raw File as String

The diagram to the left shows the structure of the process which reads a user's text file. This is what will be used for the My Progress section and it shows how some of the statistics about a particular user's data are calculated.

Definition of Data Requirements

The table below shows the main controls and variables which will be important to the program as a whole.

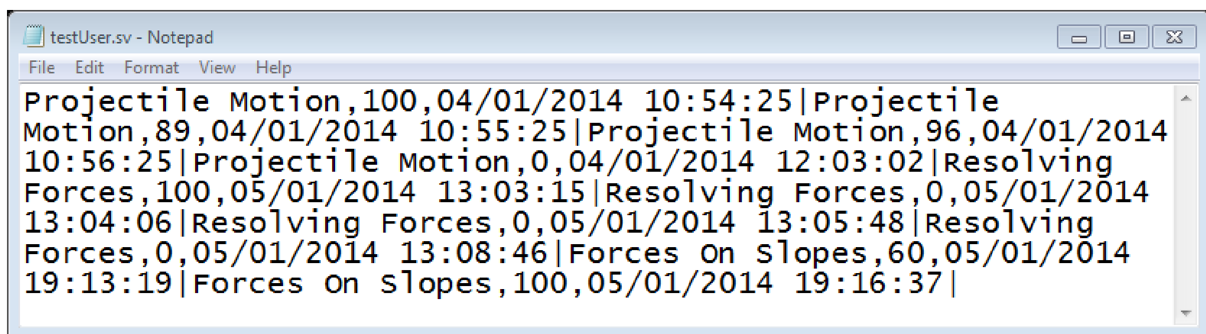
Variable Name	Description	Data Type	Example
BMP	The image in memory to which everything is drawn to	Bitmap	-
Display	The only visible control on the main form. Display's image is set to BMP every tick of MainTimer	PictureBox	
MainTimer	The timer with a minimum time interval which makes the Screen Manager update and draw all enabled screens	Timer	
ScreenManager	(See the Class Definitions section)	ScreenManager	
KeysDown	Saves the ASCII values for all keys which are pressed. The Form's KeyDown event will add the pressed key to this list. This list is cleared at the end of every MainTimer tick	List(Of Integer)	65 ("a")
KeysUp	Saves the ASCII values for all keys which are released. The Form's KeyDown event will add the released key to this list. This list is cleared at the end of every MainTimer tick	List(Of Integer)	112 (F1)
MouseButtonsDown	Saves the location and button value whenever a mouse button is pressed. This list is cleared at the end of every MainTimer tick	List(Of MouseButonInfo)	
MouseButonsUp	Saves the location and button value whenever a mouse button is released. This list is cleared at the end of every MainTimer tick	List(Of MouseButonInfo)	
ProgramPause	Used to indicate whether the whole program needs to be paused. If this holds true, the Screen Manager won't be used every tick of MainTimer	Boolean	True
CurrentUser	Holds the name of the current user logged into the program	String	TestUser69

File Organisation and Description of Record Structure

I will be using text files to save each user's progress in the test section of the program. The user text files will be saved in the user's document folder. This is a location which is easily found if files needed to be removed or accessed. As shown in the data dictionary, there are three variables that I will save for each test report:

Field Name	Description	Data Type	Field Size	Example
Category	The simulation that the question asked was about.	String	Any	Projectile Motion
Score	The score (as a percentage) that the user got for the question.	Integer	3 figures	67
TimeScored	The date and time that the user got this score. This could be used to chronologically order multiple test scores, to see progress over time.	Date	DD/MM/YYYY HH:MM	06/10/2013 21:07

Within each recorded test report I will separate the three variables by commas (","). Each Record will have a "|" symbol at the end to show the data reading algorithm where the end of each record is. Below is an example of a user's text file.



Projectile Motion,100,04/01/2014 10:54:25|Projectile
Motion,89,04/01/2014 10:55:25|Projectile Motion,96,04/01/2014
10:56:25|Projectile Motion,0,04/01/2014 12:03:02|Resolving
Forces,100,05/01/2014 13:03:15|Resolving Forces,0,05/01/2014
13:04:06|Resolving Forces,0,05/01/2014 13:05:48|Resolving
Forces,0,05/01/2014 13:08:46|Forces On Slopes,60,05/01/2014
19:13:19|Forces On Slopes,100,05/01/2014 19:16:37|

The general structure of each record is "Category,Score,TimeScored|" The first report in the example shows a test completed on the 4th of January 2014 at 10:54 (and 25 seconds) about Projectile Motion. The score for that test was 100%.

Validation Required

Because of the graphics system I will be using, I will need to create my own "Text Box" classes. I can take advantage of this and create one for writing using letters and one for entering numbers.

The number text box would be used when entering initial conditions for simulations, as well as answering test questions. These will only accept inputs which are numbers (from across the top of the keyboard, or from the NumPad) and a point or full stop to act as a decimal point. Also, only one decimal point should be allowed at one time and the decimal point won't be able to be entered first.

The writing text box will only need to be used when typing in a new user name. It is important that the usernames are created such that they don't use any characters which will break the file system. Therefore, I will only allow numbers 0-9 and upper or lowercase letters. Also, there will be a 10 character length limit when creating a user name. This is so that names which could be really long don't take up too much space. For example, when displaying the list of current users to login with, names may overlap if they are too long.

There will need to be validation for the initial variables entered into the simulations, to make sure that impossible or unrealistic situations are avoided, for example, when entering the friction for various simulations. In the mechanics model that I will use, there are limits to what the frictional force can be: It cannot be negative (since that would mean that it doesn't oppose motion) and there is an upper limit so that it doesn't cause the objects to move in the opposite direction to that which they should.

Identification of Storage Media

My program will be intended to be stored and used on a PC or laptop with average performance. I predict that the program will take up a tiny amount of space, since I have decided not to use sound. The program itself shouldn't end up being larger than 1 Megabyte. Using images that are not too large or high quality will help to reduce the size of the executable program file. The user progress text files will take up even less space, since they will only contain text. A progress file with the number of test reports in the example from the previous section won't be larger than a few Kilobytes. The small size and the fact that the program will be used on PCs means that minimizing storage space will not be a problem at all.

Potential distribution of the program would either be by downloading from the internet, or from a USB flash drive. Due to the very small size of the program, using a single-write distribution medium, such as a read-only CD, would be a waste of storage space. USB sticks are good because they are re-writable and internet downloading is feasible because of the small file size that would need to be downloaded.

Identification of Suitable Algorithms for Data Transformation

Encryption and Decryption

I will encrypt the contents of each user's progress text file, to protect them from tampering.

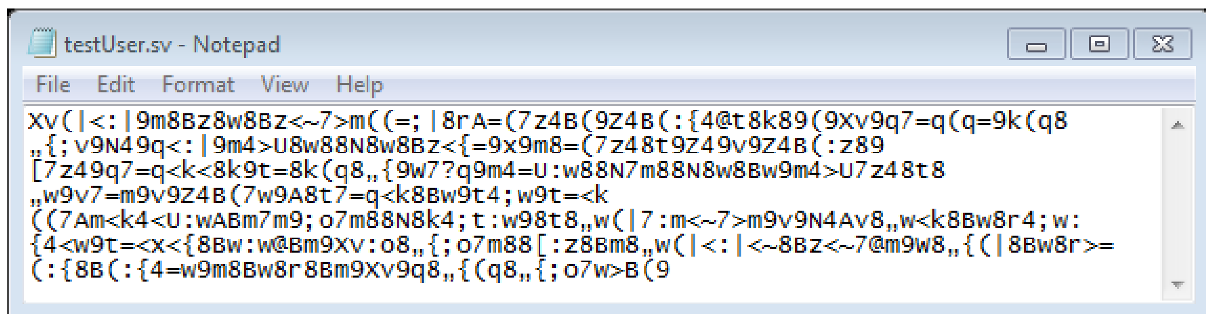
The algorithm for encryption will be:

1. Generate a random integer between 1 and 4. Call this NumOfLoops
2. For NumOfLoops times(steps 3-6):
3. Move all characters 2 ASCII codes up
4. Reverse the order of the characters in the string
5. Split the string so it has ALL of the evenly indexed characters followed by the oddly indexed characters. For example, "helloworld" would be turned into "elwrdhlool"
6. Reverse the string again
7. Put 2* NumOfLoops onto the beginning of the string
8. Repeat one iteration of steps 3 to 6

I will also need an algorithm for decryption, which will effectively be the reverse of the encryption one:

1. Reverse the string
2. Split the string into two halves. For odd length strings, first half is shorter.

3. Reconstruct the full string, by taking a character from the second half, then the first half, then the second half etc.
4. Reverse the string again
5. Move all characters 2 ASCII codes down
6. Take the first character from the string. Divide this by two, this is the NumOfLoops generated at encryption
7. For NumOfLoops times repeat steps 1 to 5



The image above shows one possible user save file after it has been encrypted using my algorithm.

Timers

There will be one Windows Forms Timer for my program called MainTimer, which will be always enabled and will have the minimum time interval. This will effectively be an infinite loop for my program. At every tick of this timer, the ScreenManager (talked about in the Class Definitions section) will update and draw all currently enabled screens. This means that I won't be able to use Windows Forms Timers easily for other things that need timing, such as the simulations, so will need to make my own timers within the Update procedures of the screen classes. I plan to use Date variables to make these timers work. The algorithm for a timer will be:

1. When the screen is instantiated, save the current time into a variable, TimerTime
2. In the screen's Update procedure, where the timer is needed:
3. If (CurrentTime - TimerTime) is greater than the intended timer interval:
4. TimerTime ← CurrentTime
5. Code to be carried out each tick of the timer

Dragging the Main Window Around

The Main program Window will have a fixed size. Normally to drag a window around, the user clicks and drags on the window's title bar. However, I intend to not have a Window Title Bar for my program, and instead have a border going all the way around. The user should be able to click and drag anywhere on the border to move the program window. The algorithm for this process is:

1. If a mouse button is pressed while the mouse cursor is hovered over the program border, save the X and Y distance of the mouse cursor position from the top-left corner of the window

2. If the mouse is moved while the mouse button is still held down, update the window's position on the screen
    a. The window's X coordinate should become the mouse cursor's X coordinate translated to the left by the X distance saved
    b. The window's Y coordinate should become the mouse cursor's Y coordinate translated up by the Y distance saved

Projectile Motion Simulation

One of the Simulations which I plan on creating is for the category of Projectile Motion. A hand-drawn design for this Simulation can be found on page 37. The Simulation involves a ball being fired from a cannon through a gap in a wall and the main purpose of it will be to update the position of the ball based on how long the Simulation has been running for.

The core algorithm will be inside a Timer (Timers are explained on the previous page). The algorithm will need to gradually increase the time variable of the Simulation, and calculate where the ball should be at that time. It will also need to work out if the ball should collide with the wall or the bounds of the Simulation.

The underpinning part of the theory of Mechanics for this Simulation is the SUVAT equation:

$$s = ut + \frac{1}{2}at^2$$

Where s = Displacement of a particle, u = Initial velocity of the particle, t = Elapsed time, a = acceleration of the particle.

Since the displacement (s) of the particle can be seen as the position relative to the particle's initial position, the equation can be re-written as:

$$p - p_0 = ut + \frac{1}{2}at^2$$

$$p = p_0 + ut + \frac{1}{2}at^2$$

Where p_0 = Initial position of the particle, p = Current position of the particle

The Projectile Motion algorithm will use this equation to separately calculate the new X and Y coordinates of the ball (the Simulation assumes that the ball is a particle). For the calculation of the horizontal, X- coordinate, there is never any horizontal acceleration. This shortens the equation for finding the X-coordinate:

$$p_x = p_{x0} + u_x t$$

Where the *x* subscript denotes a horizontal component.

For the calculation of the vertical, Y coordinate, there is a constant downwards acceleration dues to gravity. This, for the Mechanics topic I am basing my project on, is given as $9.8ms^{-2}$. Since upwards will be considered positive, a value of -9.8 would need to be used:

$$p_y = p_{y0} + u_y t + \frac{1}{2}(-9.8)t^2$$

$$p_y = p_{y0} + u_y t - \frac{1}{2} \times 9.8t^2$$

Where the *y* subscript denotes a vertical component.

The core algorithm for the Projectile Motion Simulation is:

1. Every 25 milliseconds (using a timer, defined on page 20):
2. Calculate, in metres and using the equations below, the expected position of the ball as if no collision were to happen
   a. $p_x = p_{x0} + u_x t$
   b. $p_y = p_{y0} + u_y t - 0.5 * 9.8t^2$
3. Check horizontal collisions and in the case of a collision, update velocities and positions appropriately. If no collision is found, update the ball's X coordinate with the expected X coordinate
   a. Check if the ball would have collided with the left edge of the screen
   b. Check if the ball would have collided with the wall
   c. Check if the ball would have collided with the right edge of the screen, past the wall
4. Check vertical collisions and in the case of a collision, update velocities and positions appropriately. If no collision is found, update the ball's Y coordinate with the expected Y coordinate
   a. Check if the ball reaches the top edge of the screen. In this case, the displayed ball would not move up any further, but the theoretical one would
   b. Check if the ball collides with the ground

5. Increase the elapsed time of the Simulation by 1 microsecond ($1x10^{-6}$ seconds)
6. Repeat steps 2 to 5 10,000 times

Approximately every 25 milliseconds of real time, the program simulates 0.01 seconds of the ball's motion.

Class Definitions

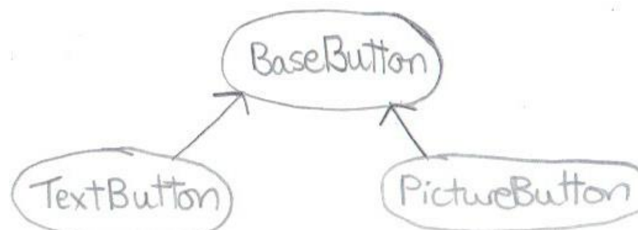
In the following section of this document, the class diagrams follow a general format:

- Data types of variables, functions and class names are blue
- When a class overrides one of its inherited methods, the method name is green
- Base classes are in red
- It can be assumed that all attributes and methods are public unless stated as private or protected
- It can be assumed that all public or protected attributes and methods from a subclass's base class are inherited

Buttons

Buttons will be used in my program when an input of a single click is needed, for example for proceeding or going back a screen. Each button can be in one of three states:

- Default: The mouse cursor is outside of the button
- MouseHover: The mouse cursor is inside the button, but the left mouse button is not pressed.
- MouseDown: The mouse cursor is inside the button and the left mouse button is pressed.



Class BaseButton	
Attributes:	**Methods:**
Location : Point	Function Clicked : String
Size : Size	DrawDefault
MouseHover : Boolean	DrawMouseHover
MouseDown : Boolean	DrawMouseDown
	Draw

The MouseHover and MouseDown Boolean variables are used to determine which state the button is in at any time.

If the user released the mouse button whilst keeping the cursor in the button, the button's Clicked function would return "Clicked". Otherwise, this function would return the state name ("MouseDown" or "Hover") or "" if the button is in the default state.

The Draw procedure would choose which one of the three other drawing procedures to call based on which state the button is in.

Class TextButton (Inherits BaseButton)	
Attributes:	**Methods:**
Text: String	New
Private BorderThickness : Integer	New
Private Colours : Color	DrawDefault
Private TextFont : Font	DrawMouseHover
	DrawMouseDown

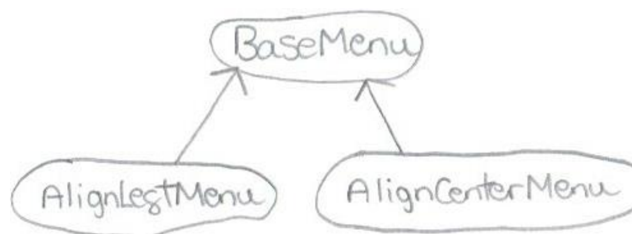
The Colours attribute represents all of the different colours associated with different parts of the button. These are the border, background and text colours for the three different states of each button.

The TextButton class has two possible overloading New procedures. This is because they both have different parameters. One of them allows for complete customisation of the various states' colours, and the other asks for the program section and uses the pre-set colours for the different program sections.

Class PictureButton (Inherits BaseButton)	
Attributes:	**Methods:**
Private DefaultImage : Image	New
Private MouseHoverImage : Image	DrawDefault
Private MouseDownImage : Image	DrawMouseHover
	DrawMouseDown

Menus

I plan to use menus on the title screen and on both user selection screens. They will allow for easier implementation of single click inputs where there are a lot (or a list) of related options. They will be much easier to use than having to create a separate TextButton for each option.



Class BaseMenu	
Attributes:	**Methods:**
Font : Font	New
Location : Point	AddOption
OptionY : Integer	

Options : List(Of String) Colours : Color DropShadow : Boolean	

Class AlignLeftMenu (Inherits BaseMenu)	
Attributes:	**Methods:** Function Update : String Draw

Class AlignCentreMenu (Inherits BaseMenu)	
Attributes:	**Methods:** Function Update : String Draw

The difference between the two subclasses is probably obvious from their names: how they are drawn. The inherited Location variable will either represent the point of the top left corner of the menu, or the top centre point on the menu.

Text Boxes

As explained in the validation section, I will have two different types of text boxes: one for writing, and one for numbers.

Class NumberBox	
Attributes: Text : String Location : Point Private Size : Size Private BorderThickness : Integer Private MaxChars : Integer Private Font : Font Focused : Boolean ReachedMaxChars : Boolean Private DefaultBorderColour : Color Private FocusedBorderColour : Color	**Methods:** New Function HandleInput: String Draw

Class WritingBox	
Attributes:	**Methods:**
Text : String	New
Location : Point	Function HandleInput : String
Private Size : Size	Draw
Private BorderThickness : Integer	
Private MaxChars : Integer	
Private Font : Font	
Focused : Boolean	
ReachedMaxChars : Boolean	
Private DefaultBorderColour : Color	
Private FocusedBorderColour : Color	

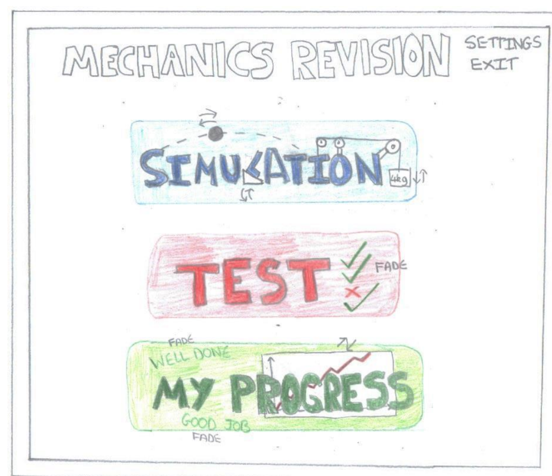
In both classes, the HandleInput function will perform validation for allowed inputs for the respective text box type and will return "Entered" if the user has pressed the enter key.

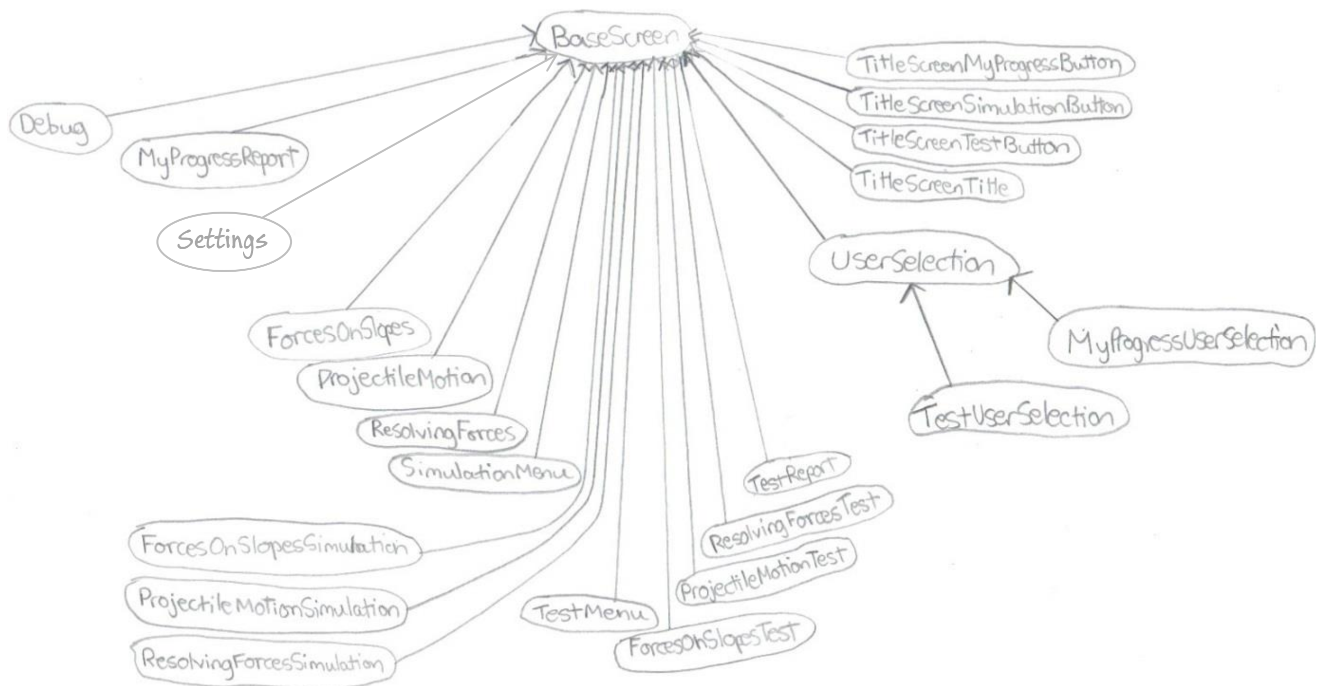
Screens

For my program, a screen is not necessarily the whole view, but a distinct part of it which deserves its own separate procedures for drawing, updating and handling input and therefore deserves its own class.

For example, on the title screen (the design of which is in the drawing below) there will be four screens being enabled at once:

- One for the title and menu on the right
- One for the Simulation Button
- One for the Test Button
- One for the My Progress Button





Class BaseScreen	
Attributes:	**Methods:**
Name : String	HandleInput
State : ScreenState	Update
Location : Point	Draw
	Unload

ScreenState is a custom enumeration with the possible options of Active, Hidden, NoInput, Sleep, ShutDown. These screen states are explained in the Screen Manager section.

Class Debug (Inherits BaseScreen)	
Attributes:	**Methods:**
ScreenLists : List(Of String)	New
Output : String	HandleInput
Private fpsCounter : Integer	Update
Private fpsTimer : Date	Draw
Private fpsText : String	

The point of the debug screen is for me to be able to see various data about the program while it runs, such as the fps (frames per second) and the current enabled screens. It will help with debugging as I will be able to see useful information without having to insert breakpoints into the code. The ScreenLists strings will each contain a list of the current screens which are in a certain state. The debug screen will always be enabled.

Class MyProgressReport (Inherits BaseScreen)	
Attributes:	**Methods:**
Private TestReports : List(Of TestReportInfo)	New
Private AverageScore : Integer	Private Function GetAverageScore : Integer
Private FirstTestDate : Date	Private SetGraphPoints
Private RecentTestDate : String	HandleInput
Private CornerMenu : AlignLeftMenu	Draw
Private GraphButtons : List(Of TextButton)	
Private GraphPoints( ) : Point	
Private CurrentCategory : String	

In the My Progress Report, TestReportInfo is a record representing one test report. This list is populated when the class reads the current user's progress text file and processes the data.

Structure TestReportInfo
Attributes:
Title : String
Score : Integer
CompletionDate : Date

The SetGraphPoints procedure will see which graph category is currently selected and look through the list of Test Reports and create an array of points to be plotted on the graph.

Class Settings (Inherits BaseScreen)	
Attributes:	**Methods:**
Private PreviousScreens : List(Of BaseScreen)	New
Private BackButton : TextButton	HandleInput
Private BorderColourSelector : TextButton	Draw
Private EnableDebugToggling : TextButton	

The PreviousScreens list will hold a list of all of the enabled screens before navigating to the settings screen. This means that when the back button is pressed, the program knows which screens to load up again.

Class ForcesOnSlopes (Inherits BaseScreen)	
Attributes:	**Methods:**
Private MenuButton : TextButton	New
Private SettingsButton : TextButton	HandleInput
Private PlayButton : TextButton	Private GetValuesFromSim
Private PauseButton : TextButton	Update
Private StopButton: TextButton	Draw
Private VariableBoxes : List(Of NumberBox)	
Private Simulation : ForcesOnSlopesSimulation	

The variable boxes list will contain all of the number boxes that the user will use to input or see the variables of the simulation. Examples of these for the Forces On Slopes simulation would be Mass, Friction and Slope Angle.

The GetValuesFromSim procedure will update the variable number boxes to display their correct current respective information from the simulation.

Class ProjectileMotion (Inherits BaseScreen)	
Attributes:	**Methods:**
Private MenuButton : TextButton	New
Private SettingsButton : TextButton	HandleInput
Private PlayButton : TextButton	Private GetValuesFromSim
Private PauseButton : TextButton	Update
Private StopButton: TextButton	Draw
Private VariableBoxes : List(Of NumberBox)	
Private Simulation : ProjectileMotionSimulation	

Class ResolvingForces (Inherits BaseScreen)	
Attributes:	**Methods:**
Private MenuButton : TextButton	New
Private SettingsButton : TextButton	HandleInput
Private PlayButton : TextButton	Private GetValuesFromSim
Private PauseButton : TextButton	Update
Private StopButton: TextButton	Draw
Private VariableBoxes : List(Of NumberBox)	
Private Simulation : ResolvingForcesSimulation	

Class SimulationMenu (Inherits BaseScreen)	
Attributes:	**Methods:**
Private Simulations(2) : SimulationInfo	New
Private MainMenuButton : TextButton	HandleInput
Private SettingsButton : TextButton	Draw

In the simulation menu, SimulationInfo is a structure which contains information about each simulation in the list.

Structure SimulationInfo
Attributes:
Title : String
Description : String
LaunchButton : TextButton
Location : Point
Enabled : Boolean

Class ForcesOnSlopesSimulation (Inherits BaseScreen)	
Attributes:	**Methods:**
Finished : Boolean	New
Scale : Double	ResetVariables
g : Double	SetTestVariables
T : Double	Function Metres : Double
TTimer : Date	Function Pixels : Double
Enabled : Boolean	Update
SimulationVariables : List(Of Single)	Draw

Class ProjectileMotionSimulation (Inherits BaseScreen)	
Attributes:	**Methods:**
Finished : Boolean	New
Scale : Double	ResetVariables
g : Double	SetTestVariables
T : Double	Function Metres : Double
TTimer : Date	Function Pixels : Double
Enabled : Boolean	Update
SimulationVariables : List(Of Single)	Draw

Class ResolvingForcesSimulation (Inherits BaseScreen)	
Attributes:	**Methods:**
Finished : Boolean	New
Scale : Double	ResetVariables
g : Double	SetTestVariables
T : Double	Function Metres : Double
TTimer : Date	Function Pixels : Double
Enabled : Boolean	Update
SimulationVariables : List(Of Single)	Draw

All of the simulation screens will not take up the whole view, because there needs to be room for the variable input screens to be drawn.

The SimulationVariables list would contain all of the appropriate variables for the simulation. This could include things like mass, velocity and angle. Each simulation would obviously need different variables.

The ResetVariables procedure will set all simulation variables to preset values which indicate the start of the simulation. This procedure would be called when the user clicks the stop button. The SetTestVariables procedure would only be used when the simulation is being used for the test mode. This would take the randomly generated initial variables from the test screen as parameters and update the simulation's variables.

Class TestMenu (Inherits BaseScreen)	
Attributes:	**Methods:**
Private MainMenuButton : TextButton	New
Private SettingsButton : TextButton	HandleInput
Private RandomTestButton : TextButton	Draw
Private Tests( ) : TestInfo	

TestInfo will be a structure, similar to the SimulationInfo structure for the simulation menu. It contains data about one test to choose from in the list on the menu.

Structure TestInfo
Attributes:
Title : String
AverageScore : Integer
TestButton : TextButton
Location : Point
Enabled : Boolean

Class ForcesOnSlopesTest (Inherits BaseScreen)	
Attributes:	**Methods:**
Private MenuButton : TextButton	New
Private SettingsButton : TextButton	HandleInput
Private AnswerBoxes : List(Of NumberBox)	Update
Private MarkButton : TextButton	Draw
Private CorrectAnswers( ) : Decimal	
Private Simulation : ForcesOnSlopesSimulation	
Private Report : TestReport	
Private InitialVariables ( ) : Single	

Class ProjectileMotionTest (Inherits BaseScreen)	
Attributes:	**Methods:**
Private MenuButton : TextButton	New
Private SettingsButton : TextButton	HandleInput
Private AnswerBoxes : List(Of NumberBox)	Update
Private MarkButtons : List(Of TextButton)	Draw
Private CorrectAnswers( ) : Decimal	
Private Simulation : ProjectileMotionSimulation	
Private Report : TestReport	
Private InitialVariables ( ) : Single	

Class ResolvingForcesTest (Inherits BaseScreen)	
Attributes:	**Methods:**
Private MenuButton : TextButton	New
Private SettingsButton : TextButton	HandleInput
Private AnswerBoxes : List(Of NumberBox)	Update
Private MarkButton : TextButton	Draw
Private CorrectAnswers( ) : Decimal	
Private Simulation : ResolvingForcesSimulation	
Private Report : TestReport	
Private InitialVariables ( ) : Single	

In the tests, the InitialVariables array will be a list of all of the starting variables needed for the test's simulation. These will be semi-randomly generated in the New procedure. The Correct answers to the question will also be calculated in the New procedure.

Class TestReport (Inherits BaseScreen)	
Attributes:	**Methods:**
Private AnotherTestButton : TextButton	New
Private CompletionDate : Date	Update
Private TestName : String	Draw
Private Parts : List(Of TestQuestionPart)	
Private TotalAchieved : Integer	
Private TotalOutOf : Integer	

The Test Report screen will be part of each Test screen and will be enabled once the test has been completed. It will show information about the test, including the score for each part, to total score as a percentage and the time of completion. The New procedure will append the test report in the correct format to the user's progress text file.

Structure TestQuestionPart
Attributes:
ScoreAchieved : Integer
ScoreOutOf : Integer

Class Title (Inherits BaseScreen)	
Attributes:	**Methods:**
Private CornerMenu : AlignLeftMenu	New
	HandleInput
	Draw

The TitleScreenTitle screen is responsible for the top quarter of the title screen, including the title and the settings/exit menu in the top right corner.

Class MyProgressButton (Inherits BaseScreen)	
Attributes:	**Methods:**
Private MouseHover : Boolean	New
Private AniTimer : Date	Update
Private AniCount : Integer	HandleInput
Private GraphCoverSrcRect : Rectangle	Draw
Private GraphCoverX : Integer	
Private WellDoneAlpha : Integer	
Private GoodJobAlpha : Integer	

Class SimulationButton (Inherits BaseScreen)	
Attributes:	**Methods:**
Private MouseHover : Boolean	New
Private AniTimer : Date	Update
Private AniCount : Integer	HandleInput
Private LTriangle(2) : Point	Draw
Private IRect : Rectangle	
Private MassRect : Rectangle	
Private ProjectileRect : Rectangle	

Class TestButton (Inherits BaseScreen)	
Attributes:	**Methods:**
Private MouseHover : Boolean	New
Private AniTimer : Date	Update
Private AniCount : Integer	HandleInput
Private TickPoints(1,2) : Point	Draw
Private CrossPoints(3) : Point	
Private TickAlpha(1) : Integer	
Private CrossAlpha : Integer	

The three classes above are for the other quarters of the title screen. They will be for the three large animated buttons that navigate to the three sections of the program: Simulation, Test and My Progress. Although all three of them look different, they will work in the same way. They each have the integer variable AniCount. This will start at 0. If the mouse hovers over a button, its AniCount will gradually increase up to a limit of 100. If the mouse cursor is not in a button its AniCount will decrease down to a minimum of 0. The AniCount variable represents how far along a button is in its animation, as a percentage. This will give the effect of the animation playing in reverse while the cursor is not on the button.

Class UserSelection (Inherits BaseScreen)	
Attributes: Protected MenuButton : TextButton Protected NewUserBox : WritingBox Protected CreateUerButton : TextButton Protected UserLists : List(Of AlignLeftMenu) Protected Users : List(Of String) Protected UserAlreadyExistsError : Date Protected SectionColour : Color	**Methods:** Protected RefreshExistingUserLists HandleInput Protected Advance Draw

Since the Test and My Progress user selection screens will be identical apart from their colour and where they navigate to, I will create a User Selection base class. The RefreshExistingUserLists procedure will look in the directory where all user text files are and add the name of the file (without the ".sv" file extension) to the Users list.

The Advance procedure holds the code to unload the selection screen and load the next screen. This procedure needs to be overridden by each of the subclasses because it will be different for both: the TestUserSelection screen will navigate towards the Test menu and the MyProgressUserSelection screen will navigate towards the My Progress report.

Class MyProgressUserSelection (Inherits UserSelection)	
Attributes:	**Methods:** New Advance

Class TestUserSelection (Inherits UserSelection)	
Attributes:	**Methods:** New Advance

Screen Manager

Class ScreenManager	
Attributes: Private Screens : List(Of BaseScreen) Private NewScreens : List(Of BaseScreen) Private DebugScreen : Debug	**Methods:** New Update Draw SetDebugOutputMessage AddScreen SetScreenState UnloadScreen

The screen manager will be instantiated as soon as the program starts running. Its purpose will be to hold a list of all of the enabled screens and to call their HandleInput, Update, and Draw procedures depending on which state they are in. The different possible screen states are:

State	Call HandleInput?	Call Update?	Call Draw?	Other Info
Active	Yes	Yes	Yes	Default State
Hidden	Yes	Yes	No	
NoInput	No	Yes	Yes	
Sleep	No	No	No	
ShutDown	No	No	No	Removed when Screen Manager next updates

The Update procedure will remove any screens which are in the ShutDown state, add any screens that are in the NewScreens list, and call the HandleInput and Update procedures for all applicable screens. The Draw procedure will call the Draw procedure in all screens in the Active or NoInput states.

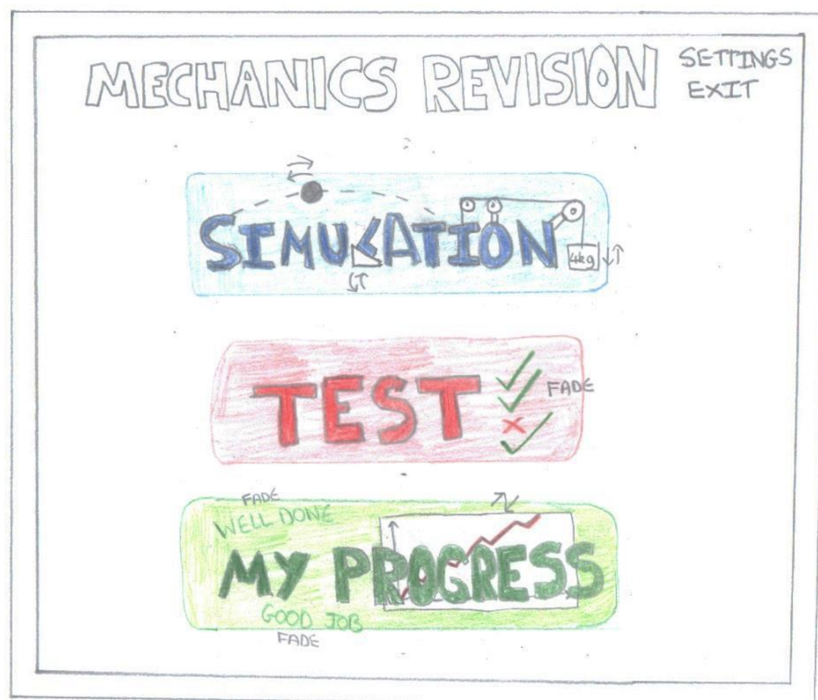
The AddScreen procedure will add a new instance of a screen to the NewScreens list. The purpose of the NewScreens list is so that new screens can be added before the Update and HandleInput loop starts.

The UnloadScreen procedure will set a screen's state to ShutDown.

User Interface Design

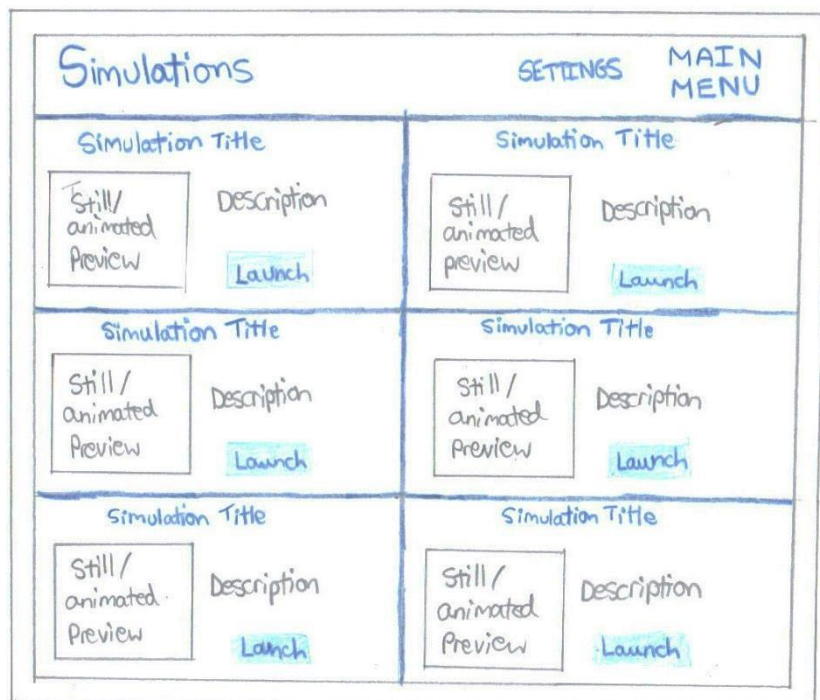
Throughout the program, I plan on keeping a consistent colour scheme. Any screens to do with the simulation section would be dominated by blue, Test with red and My Progress with green. I think that this will be important because it will separate the different sections, and will make the program as a whole look more professional. In my design drawings, I have not included all of the colours that I intend on using in the program, but the colours that I feel will be important.

Main Menu



The Main Menu will have three main buttons, one to go to each of the three main sections. I think that it would be good if the buttons had some kind of animation when the mouse rolled over them. For example, on the My Progress button, the graph would be an empty set of axes by default, but the line would appear to be drawn as the mouse rolled over it. Also, on the Simulation button, various parts would move, such as the letter 'I' being raised and the 'L' being inclined. A ball will fly across the top left of the simulation button as well. I could use the first half of a sine oscillation for one arc, or use a |sine| oscillation for the ball to bounce once half-way. I think that this design feature would make this screen more interesting to my target group (Mechanics students and teachers), since they would probably be interested in things that move.

Simulation



Above is the simulations menu screen. It will be shown after the user clicks the simulation button on the main menu. I think that it is very important to have an image to preview the simulation as well as the title and description of each simulation, because it gives the user a much better idea of what it's going to be like before they run it. I also think that having pictures for each simulation would make this screen look more interesting, and encourage users to try them out.

Projectile Motion
SETTINGS
MENU

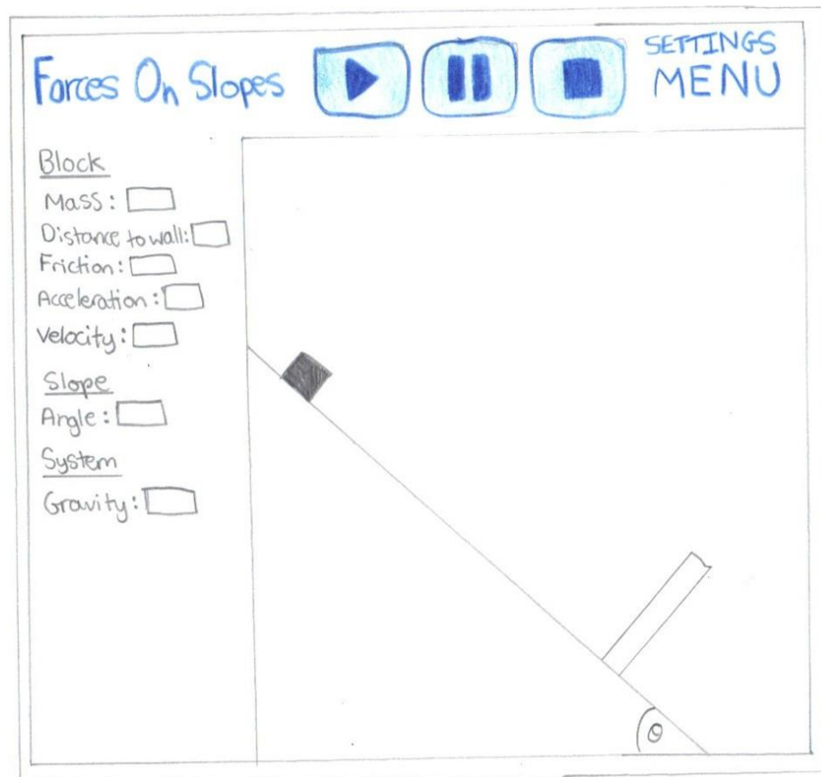
Ball
Position
X: Y:
Velocity
X: Y:
Speed:
Force
X: Y:
Resultant:
Angle of Motion:
Cannon
Firing Angle:
Firing Velocity
X: Y:
Speed:
Wall Gap
W_1:
W_2:

W_2
x
W_1
h
θ
d

Resolving Forces
SETTINGS
MENU

m_1
mass:
friction:
position
X:
m_2
mass:
position
Y:
System
gravity:

m_1
_ kg
m_2
_ kg



Above are three example screens for actual simulations: Projectile Motion, Resolving Forces and Forces on Slopes. These would be accessed from the Simulations menu screen. Each simulation that I make will have the same general design:

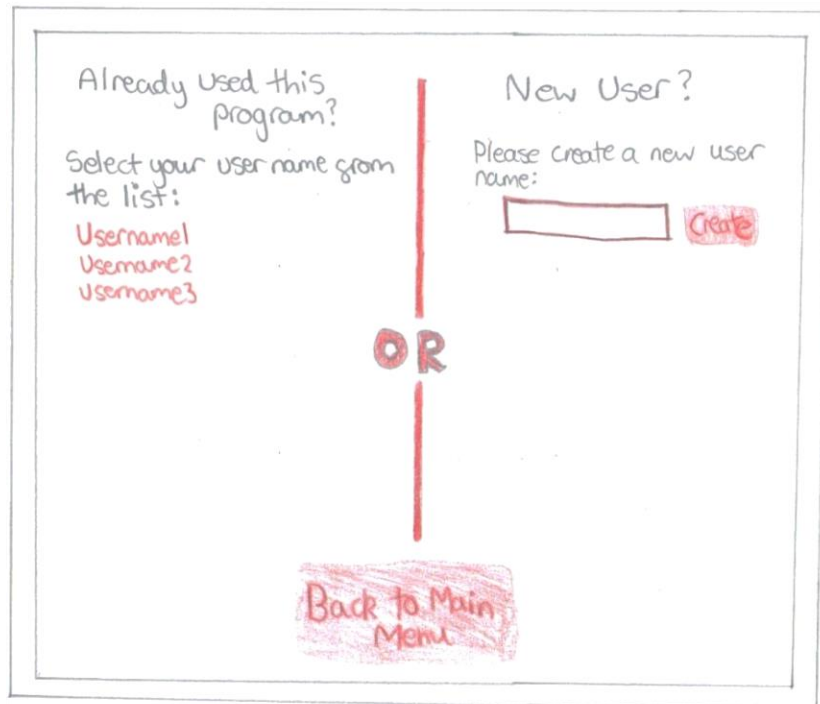
The strip along the top would have the title of the simulation on the left, then play, pause and stop buttons and settings and menu buttons on the right. The simulation running control buttons will be quite big so that they are obvious and easy to click. The play and pause buttons are self-explanatory and the stop button would pause the simulation, then reset it to its initial condition.

Down the left side of the screen will be where the user can view and change variables or constants about the simulation before running it. Sliders will be used for some variables' input to make it easier and as a form of validation. It would make sure users don't enter ridiculous values that could potentially cause the simulation to crash, or not be useful. An example of where a slider could be appropriate is in the projectile motion simulation. There needs to be a gap in the wall, and users will be able to change the Y-position of the edges of the gap. A slider could ensure that the gap doesn't ever have a negative width.

The rest of the screen, bounded by the blue lines, will be for the simulation itself. An important feature of the simulations will be arrows to show directions of appropriate vector quantities, such as forces on objects or velocities of objects. For the projectile motion simulation, I may also add an 'ant-trail' path to show the trajectory behind the projectile.

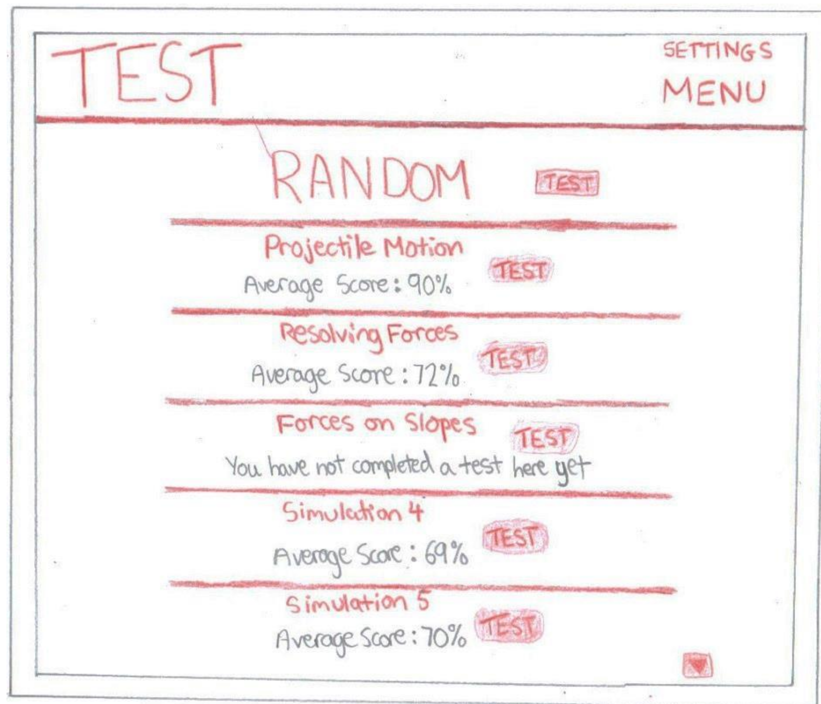
Test Mode

My choice of red for the Test Mode section was due to its connotation of seriousness and importance. If the user is taking a 'test', they are being serious and no longer playing around.

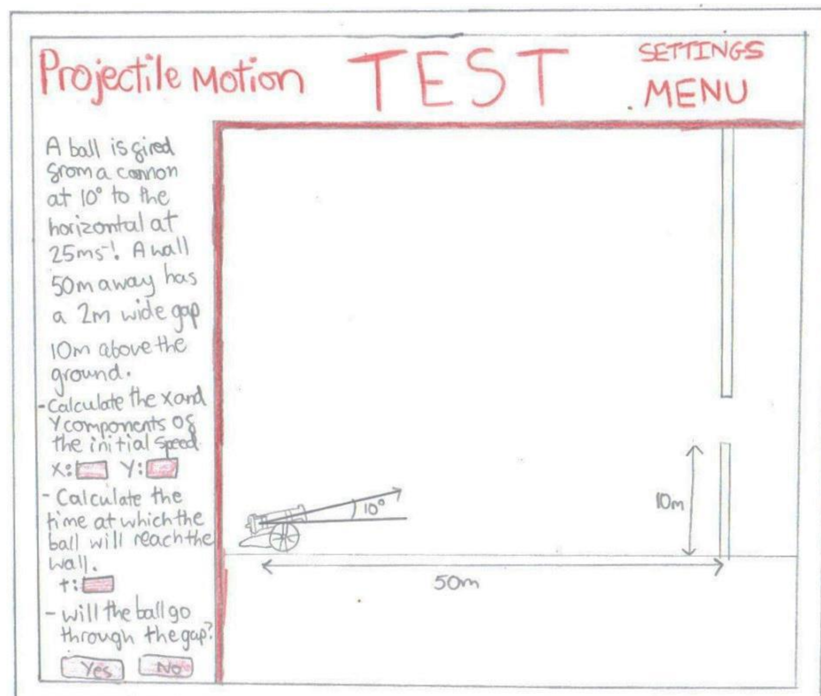


Above is the user selection screen for the Test Mode. This will be shown when the user clicks on the Test button on the main menu, and before advancing to the Test Mode menu. Since the test mode will ultimately save data to the user's progress file, it needs to know which user is taking the test so it knows which file to write to.

The new username text box will need to have some sort of validation. Because the user's username will be used for the name of their progress file, they shouldn't be allowed to input characters that would not be appropriate in a filename (e.g. \ /:*?"<>|.). Also, there should be a check to make sure that a user cannot create a new username that is exactly the same as one which already exists.



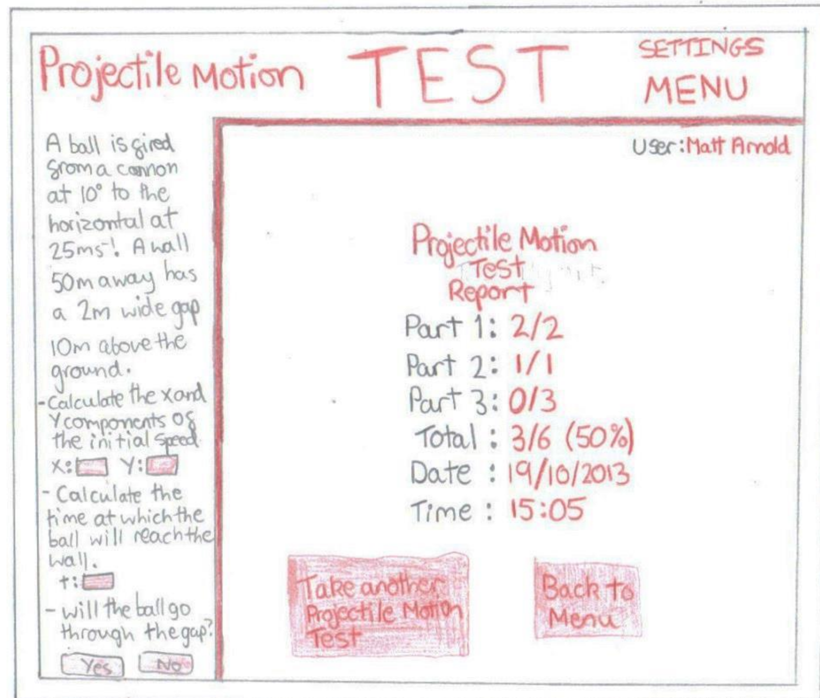
Above is the Test Mode menu, which will be seen after the user is selected. I think that it will be useful to show the user's average score for each category. This means that the user will be able to see which categories they are doing worst on, and maybe encourage them to try more tests on those. There would also be an option for the user to be tested on a random category.



Above is an example of the first phase of a test for Projectile Motion. The design of a test is very similar to that of a simulation. The actual simulation window, the title and the settings and menu buttons are in exactly the same places. Instead of the play, pause and stop buttons, there is a big

piece of text saying 'TEST'. Instead of the variables and constants, there is the test question itself. The only other design difference is obviously the colour, red instead of blue.

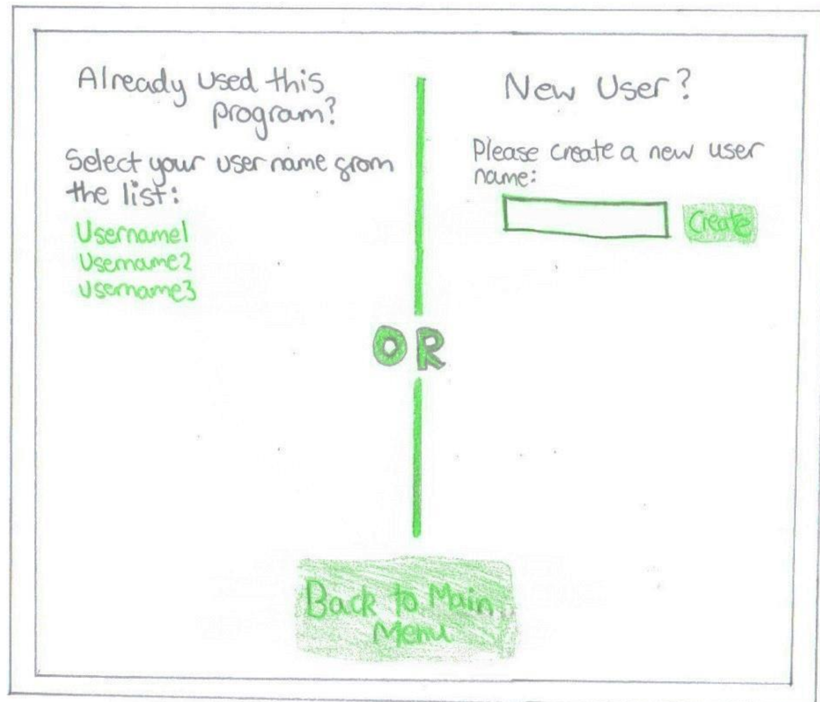
The first phase is the answering phase, which shows the 'first frame' of the simulation to illustrate the initial conditions.



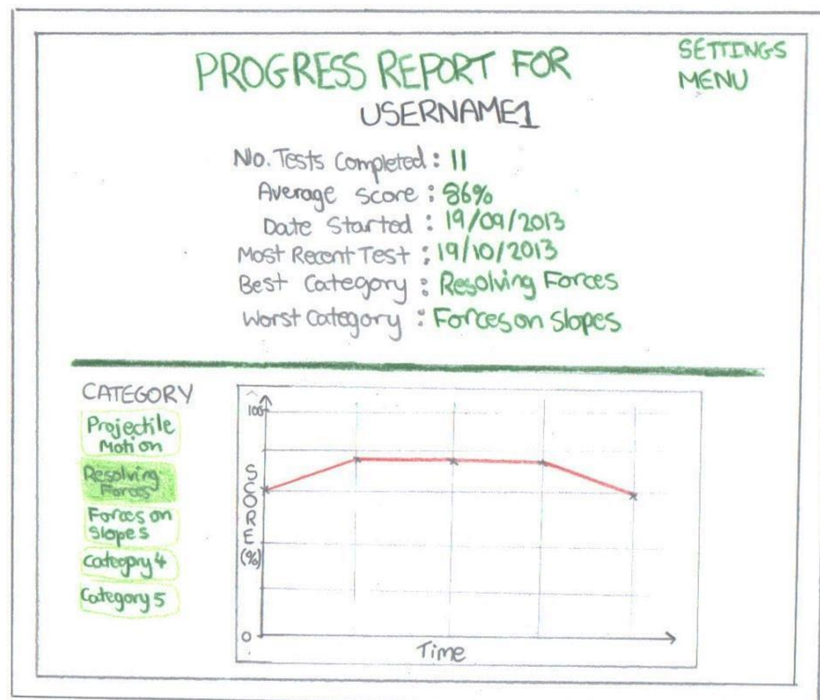
Above is the second phase of a test. After the user answers the question, and after the simulation finishes running to show the user what happens, the simulation part of the screen will be replaced with the report. I have decided to do this and not have an entirely new screen because the user will be able to see their answers to the questions as well as their mark to each part. At this point, the program also saves the user's test result in their progress file.

My Progress

I decided to use green for the My Progress section because of its association with positivity, and friendliness. Since the user would be getting feedback, I think it is a good idea to present it to them in a positive and encouraging way.



The My Progress section will also need to find out the current user of the program, since it will be reading their file and processing the data in it. It will be exactly the same as the Test Mode user selection screen except for the colour (green instead of red). For both of the user selection screens, the list of existing users on the left will be sorted by how many tests they have done, in a descending order. This means that the most frequent users should be shown at the top, and, on average, users will therefore be able to find themselves more easily.



Above is user progress report. The top half of this screen shows overall user data and the graph on the bottom half will be specific for each test/simulation category. The user would select one of the categories to the left of the graph and a graph would be drawn specifically for the category. At the moment, I plan on drawing a line graph with the data, but if this proves too difficult to do using my graphics system, I could draw a bar chart instead, which would be easier. The x-axis on the graph will not be an accurate, to-scale representation of time, but rather the number of the test taken. The points will however be represented in chronological order.

Settings

My hand-drawn designs of the various user interface features for my program include a button directing to a 'settings' screen. I will make a settings screen for any user options that I may wish to include in the program when creating it.

Security of Data

The data stored by my program will be the user progress data in the various user progress text files. Although the information about the users won't be particularly personal to them, it is still a good idea to have some security measures in place. I will encrypt the user progress data, to stop people 'cheating' and pretending that they have progressed further than they have. If the data isn't encrypted, people may be able to work out what the data means, and could easily change bits of it to their benefit. When it's encrypted, if they decide to change something (to see what happens) the decryption algorithm shouldn't work, and the program would know that the data is corrupted. I will make my own algorithm for encryption/decryption.

System Security

The program itself will not have a great need to be really secure, as it is intended as a tool for revision that anyone could use. It will not just be the teacher using the program for lesson demonstrations, but also students, if they want to learn or test themselves in their own time. For this reason, a whole program security system (such as a password to open the program) would not be needed.

Overall Test Strategy

My general testing method will be black-box testing. I will test the program using strategically chosen inputs and check that it responds correctly (for example, give errors at the right time). If the program crashes unexpectedly, I would use a white-box testing method, by following through the code at certain points using break-points to see where the error occurs. If this still doesn't seem to help me understand the problem, I will do a dry run on paper of the particular piece of code.

The important sub-systems that will need testing are:

- Making sure that the test mode text file data is saved in a format that can be read correctly, and that the encryption/decryption of the files works correctly.
- 'Extreme' values for the simulation initial conditions
- Validation for creating a new user
- Saving into the user's text file after a test

System Testing

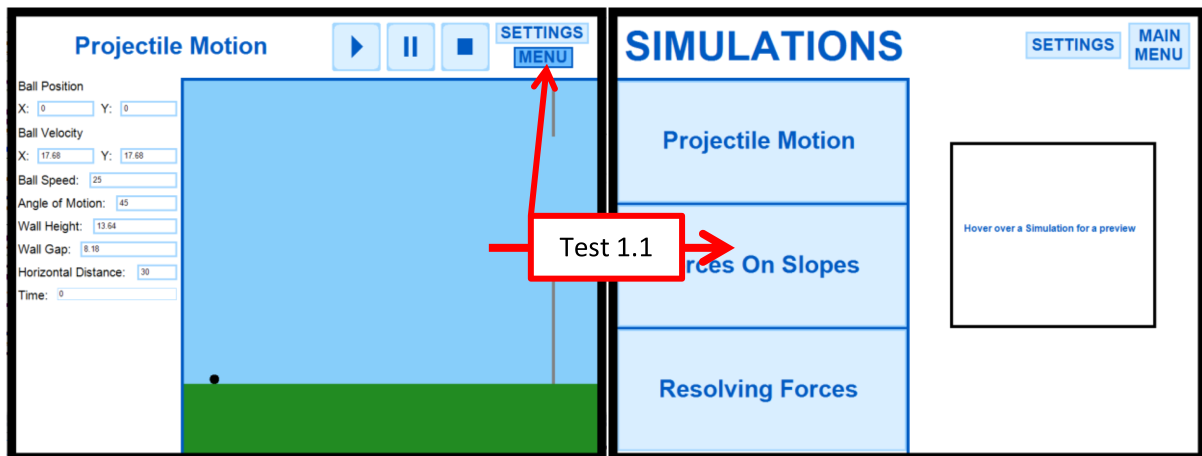
Test Series	Purpose/Description
1	Navigation between screens of the program
2	Validation of Input in the Projectile Motion simulation
3	Validation of Input in the Resolving Forces simulation
4	Validation of Input in the Forces On Slopes simulation
5	Writing to a User's Data string
6	Interpreting a User's Data String
7	Encryption and Decryption
8	File Reading and Writing
9	Creating a new User name
10	Test Mode
11	Program Settings

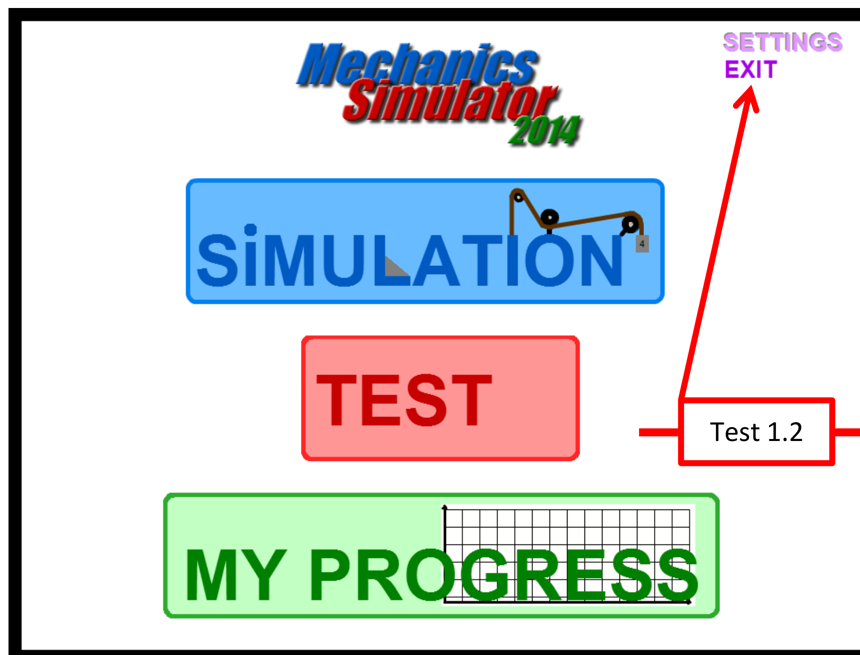
Test Series 1

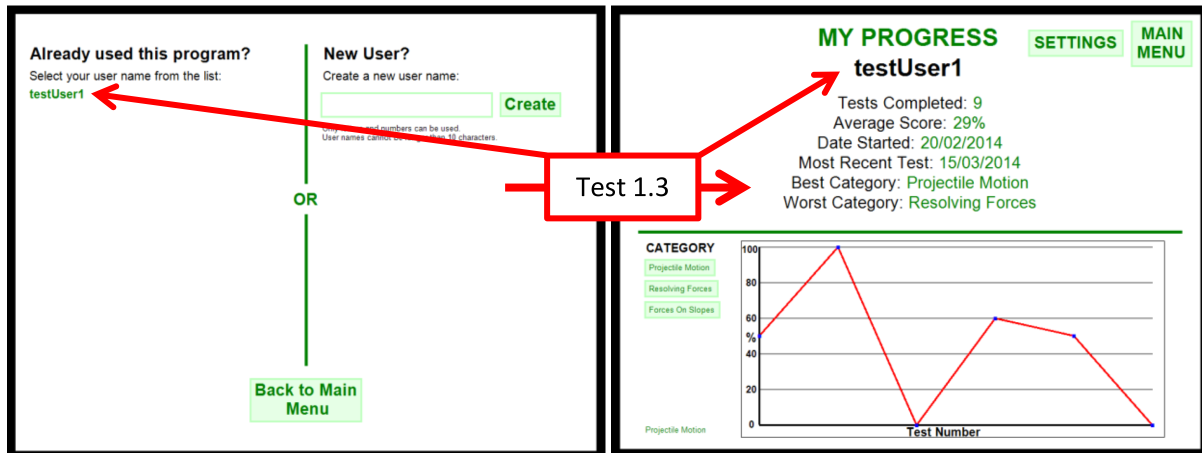
Test Series and Number	Purpose/ Description	Test Data and Type	Expected Result	Actual Result
1.1	Buttons should be able to be used to go between screens	**Typical**: Clicking on the 'Menu' button in the Projectile Motion simulation	Screen change to the simulation menu	Screen change to the simulation menu
1.2	The user should be able to exit the program from the title screen	**Typical**: Clicking 'Exit' on the title screen	Program terminates	Program terminates
1.3	Selecting a user from a user selection screen should advance to the next screen	**Typical**: Click on a user name from the 'My Progress' user selection screen	Screen change to progress report for the selected user	Screen change to progress report for the selected user
1.4	Completing a test should cause the	**Typical**: Completing a test for Projectile	Simulation screen changes to test report screen	Simulation screen changes to test report screen

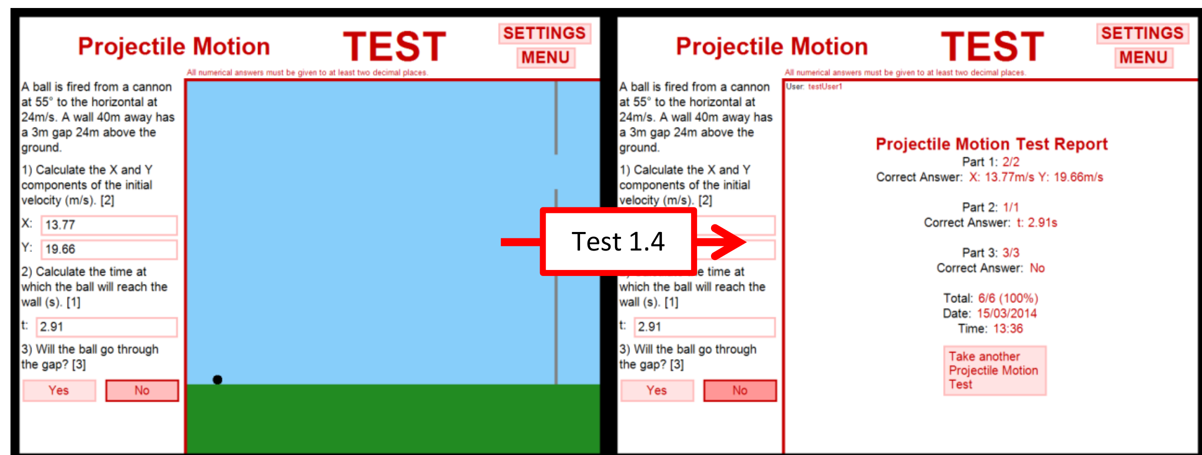
	simulation screen to be replaced with the test report screen	Motion and waiting for the simulation to finish		

The purpose of this Test Series was to test the types of navigational features of the program which took the user between screens.









Test Series 2

Test Series and Number	Purpose/ Description	Test Data and Type	Expected Result	Actual Result
2.1	Horizontal Distance	**Boundary**: Attempt to enter 0m for the horizontal distance	Unknown	Program crashes due to dividing by zero when trying to calculate the pixels-to-metres scale factor from the horizontal distance
2.2	Horizontal Distance after code changed	**Boundary**: Attempt to enter 0m for the horizontal distance	No crash	No crash, and the value for the horizontal distance changes back to what it was before (30m)
2.3	Changing the firing angle should update the components of the initial	**Typical**: Change the angle to 20 degrees	Velocity X and Y components update so that X = 25cos(20) = 23.49 and Y = 25sin(20) = 8.55	Velocity X and Y components update so that X = 25cos(20) = 23.49 and Y = 25sin(20) = 8.55

	velocity			

The purpose of this test series was to look at the Projectile Motion simulation (from the simulation section) and test different values for some of the available initial conditions.

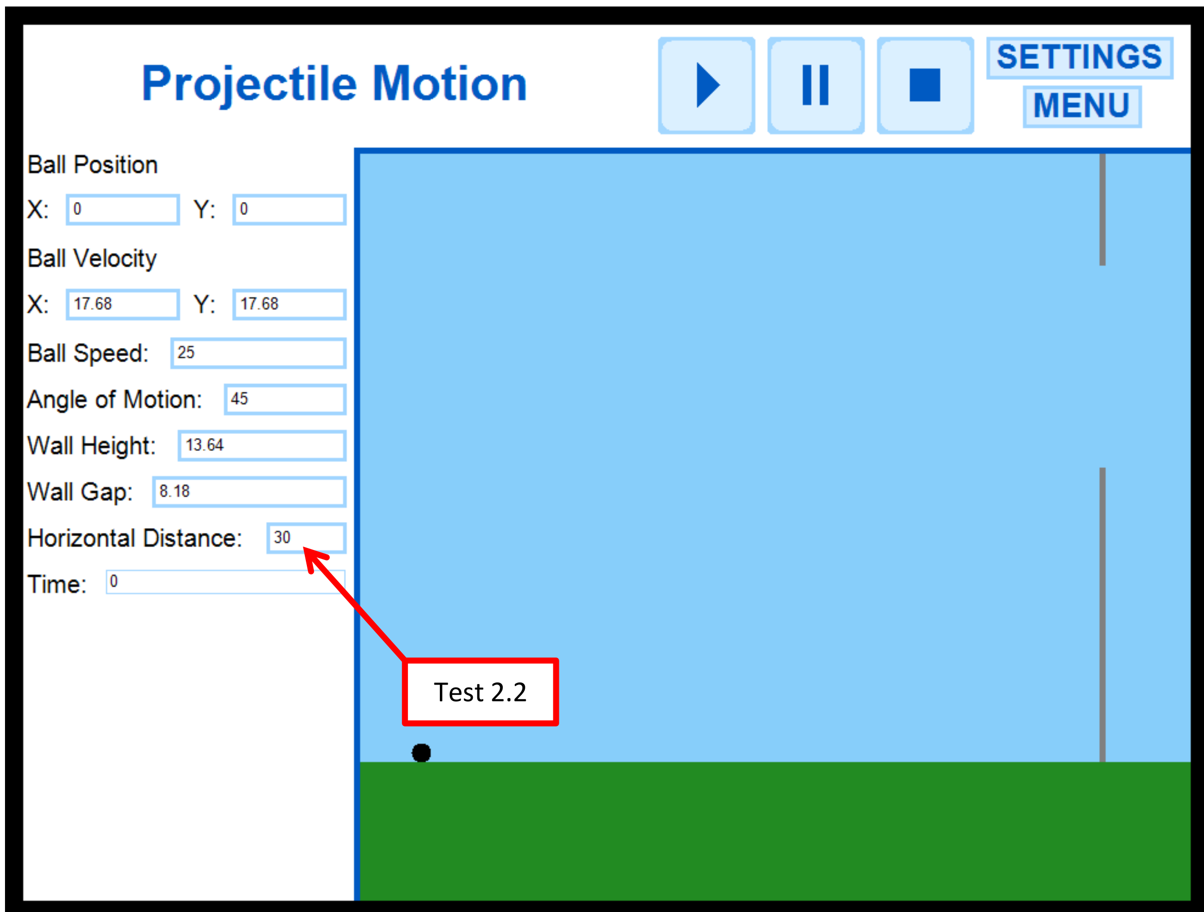
```
ElseIf XDistanceBox.HandleInput() = "Entered" And XDistanceBox.Text <> "" Then
    Simulation.Scale = 550 / CDec(XDistanceBox.Text)
Else
    ChangeOccured = False
```

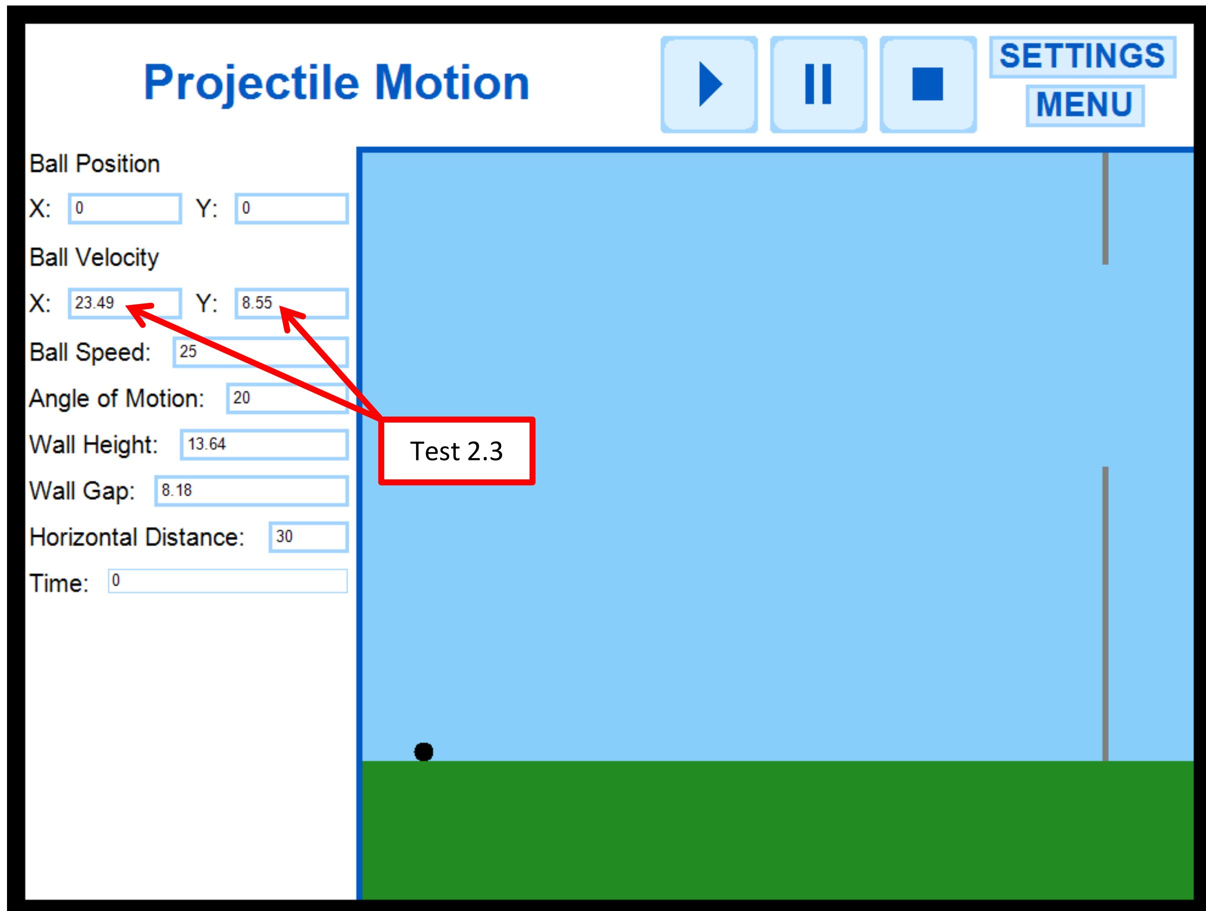
DivideByZeroException was unhandled

Attempted to divide by zero.

Test 2.1

I have now changed the code at this point to only handle the horizontal distance number box when a value greater than 0 is entered.



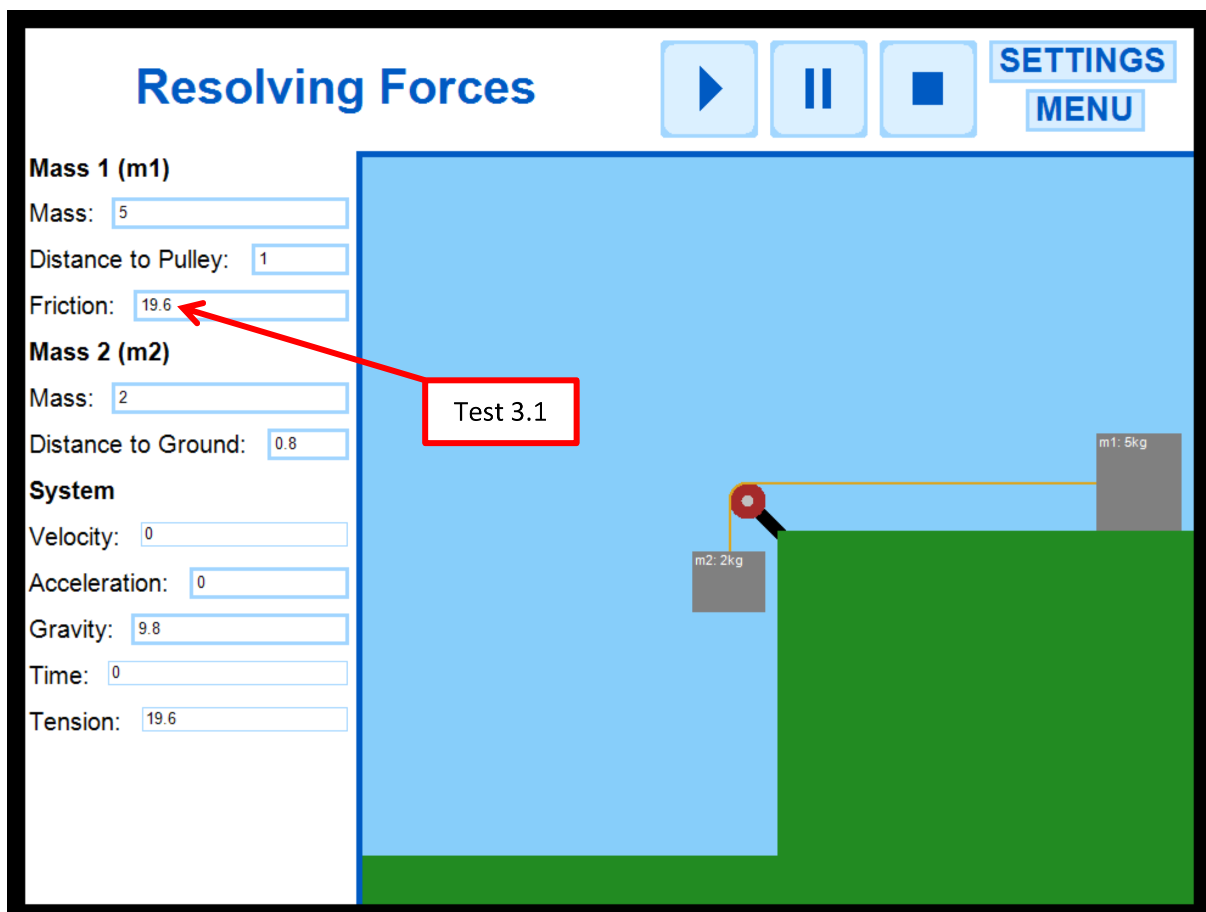


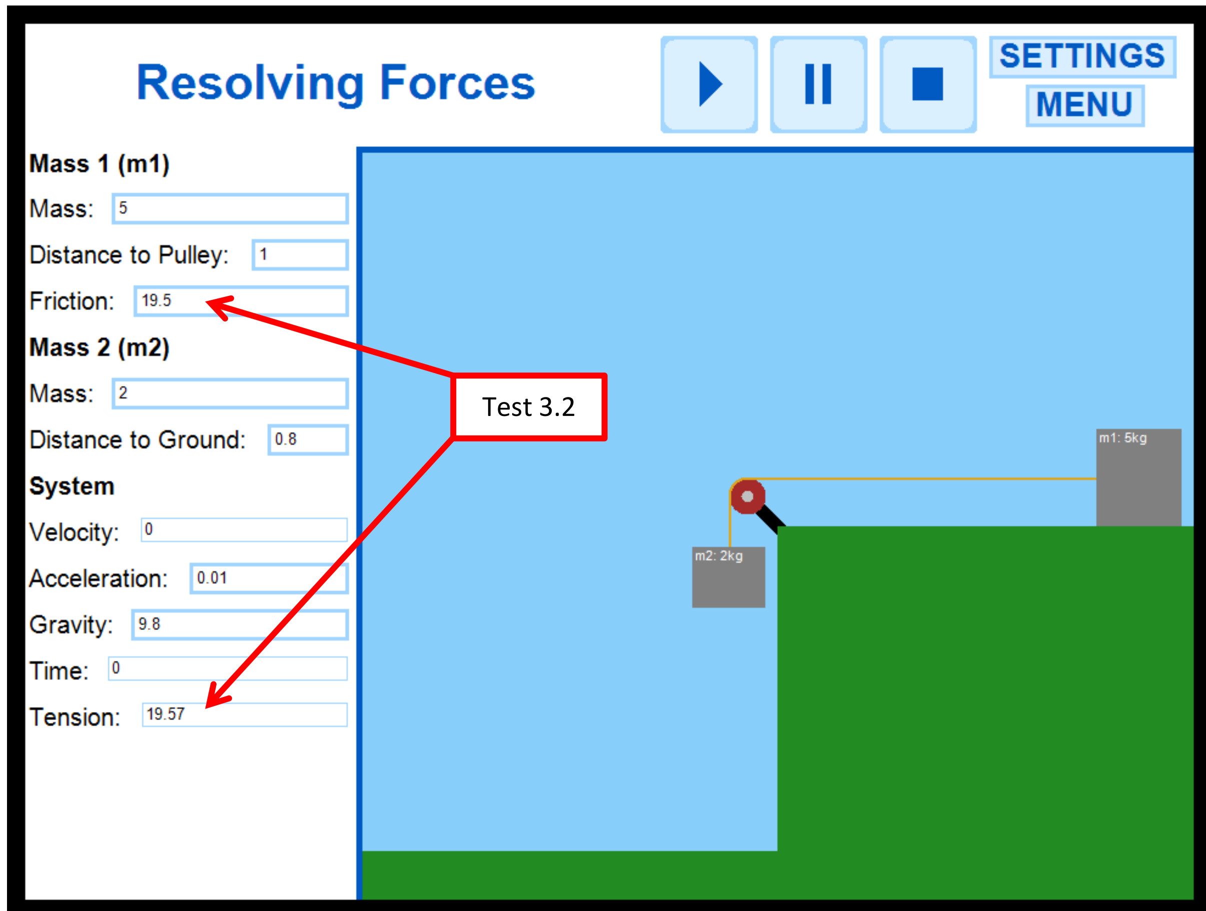
Test Series 3

Test Series and Number	Purpose/ Description	Test Data and Type	Expected Result	Actual Result
3.1	The friction of m1 must be less than or equal to the m2 mass multiplied by gravity (9.8ms^{-2})	**Boundary:** m2 mass = 2kg. Attempt to enter 19.7N for friction of m1	No crash, and the friction is set to 2 x 9.8 = 19.6N	No crash, and the friction is set to 2 x 9.8 = 19.6N
3.2	The friction of m1 must be less than or equal to the m2 mass multiplied by gravity (9.8ms^{-2})	**Boundary:** m2 mass = 2kg. Attempt to enter 19.6N for friction of m1	No crash, 19.6N accepted and acceleration updated to 0ms^{-2}	No crash, 19.6N accepted and acceleration updated to 0ms^{-2}
3.3	The friction of m1 must	**Boundary:** m2 mass =	No crash, 19.5N accepted and acceleration and	No crash, 19.5N accepted and acceleration and

	be less than or equal to the m2 mass multiplied by gravity (9.8ms^{-2})	2kg. Attempt to enter 19.5N for friction of m1	tension updated	tension updated. No screenshot because there is no change to the screen
3.4	Vertical Distance	**Boundary**: Attempt to enter 0m for the vertical distance from m2 to ground	No crash	No crash and the value for the vertical distance changes back to what it was before (0.8m). No screenshot because there is no change to the screen

The purpose of this test series was to look at the Resolving Forces simulation (from the simulation section) and test different values for some of the available initial conditions.



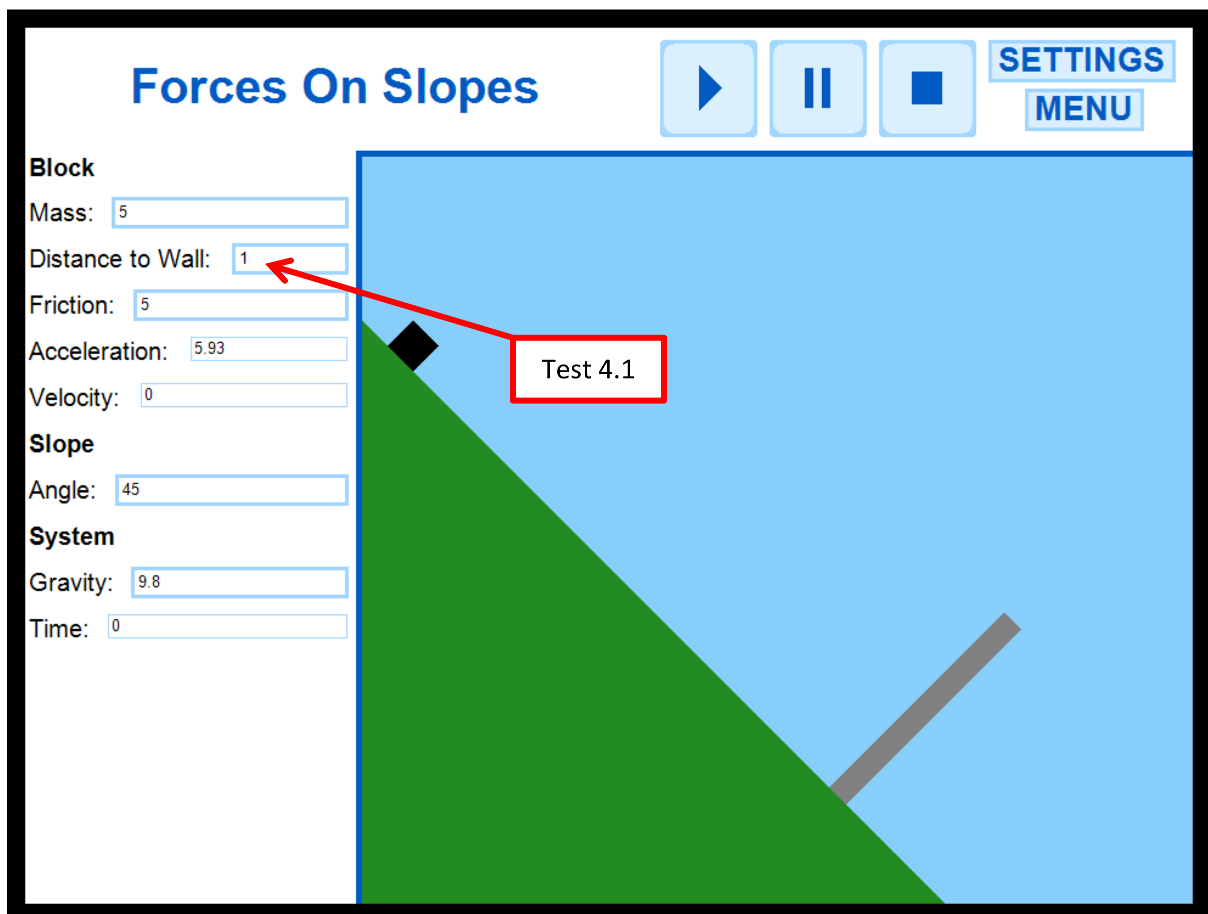


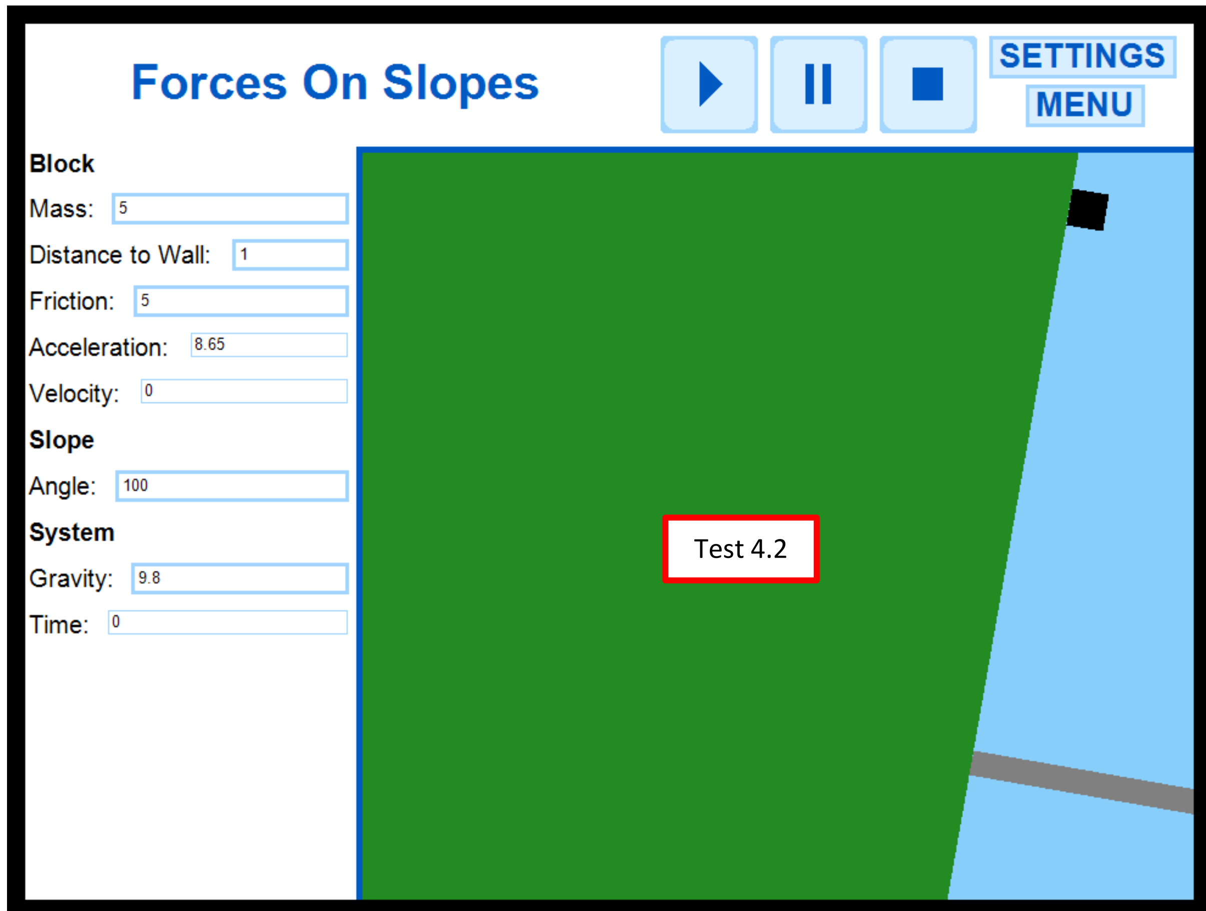
Test Series 4

Test Series and Number	Purpose/ Description	Test Data and Type	Expected Result	Actual Result
4.1	Distance to Wall	**Boundary**: Attempt to enter 0m for the distance from the mass to the wall	No crash	No crash and the value for the distance changes back to what it was before (1m).
4.2	Slope Angle	**Erroneous**: Attempt to enter an angle greater than 90° (100°)	Unknown	No crash, but the resulting simulation looks silly, and the block still sticks to the slope when the play button is pressed. This is a physically impossible situation, and therefore one which needs to be prevented. I changed the code to only accept angles between 0° and 90°

4.3	Slope Angle after change in code	**Erroneous**: Attempt to enter an angle greater than 90° (100°)	No change. Angle should go back to what was before the attempt to change it	Angle returns to what it was before (45°). No screenshot because there was no change

The purpose of this test series was to look at the Forces On Slopes simulation (from the simulation section) and test different values for some of the available initial conditions.





Sliders

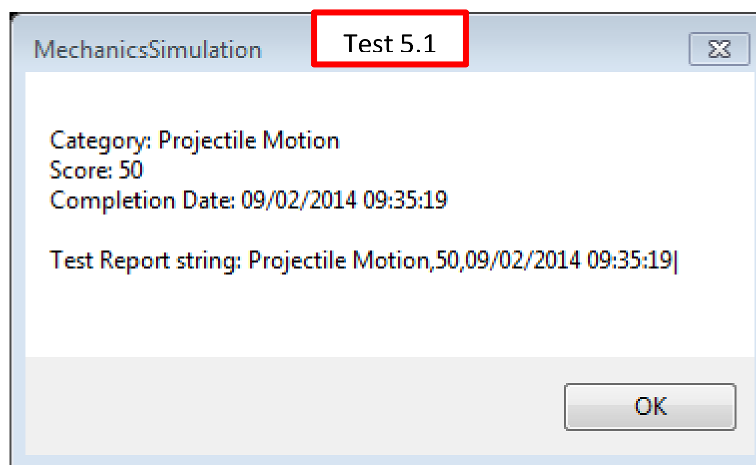
In my original design for all of the Simulations, I had intended to create sliders as a form of input which could also act as an effective form of validation for some variables (such as distances). However, in the creation of my program, I decided against this for two reasons. The first was that the validation methods of my text boxes are sufficient. The second is that it would be difficult to implement a slider which was small enough to fit on the screen, as well as accurate enough for easily inputting values up to two decimal places. If I created sliders as accurate as I wanted, they would have needed to be too long to fit on the screen.

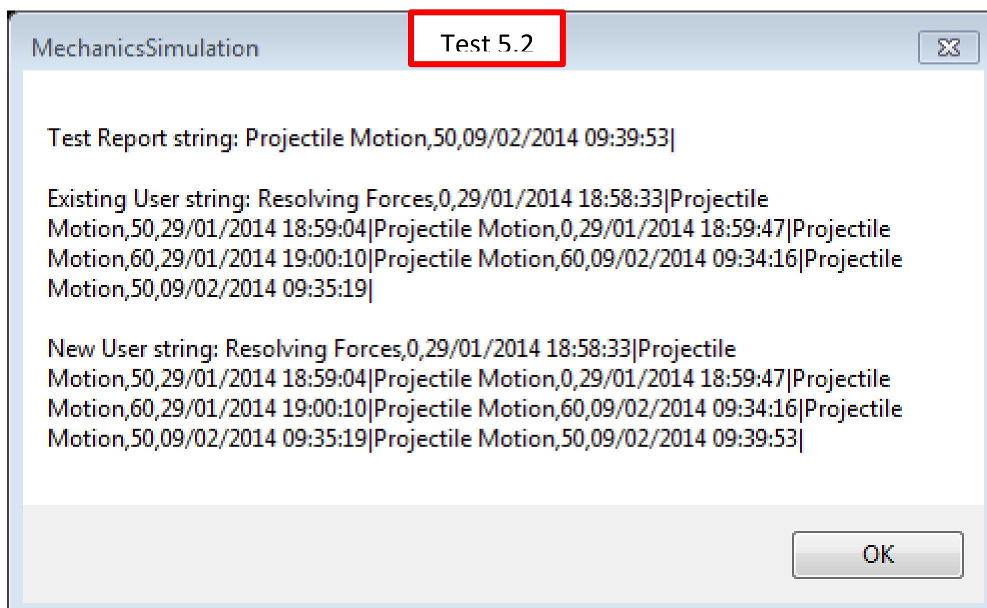
Test Series 5

Test Series and Number	Purpose/ Description	Test Data and Type	Expected Result	Actual Result
5.1	To put the three fields of the test report together in the correct format	**Typical**: A test report for projectile motion, with a score of 50%	Category,Score,TimeScored|	Category,Score,TimeScored|

5.2	To append the small string created to the end of the User's existing data string	**Typical**: A typical User date string and the result from test 5.1	A user data string in the correct form	A user data string in the correct form

This Test Series looked at the Test section of the program. Its purpose was to make sure that the program could successfully append a test report to the user's existing progress data string. For this test, I made the program show the contents of variables using message boxes, since the processing would, in reality, be done without any output visible to the user.

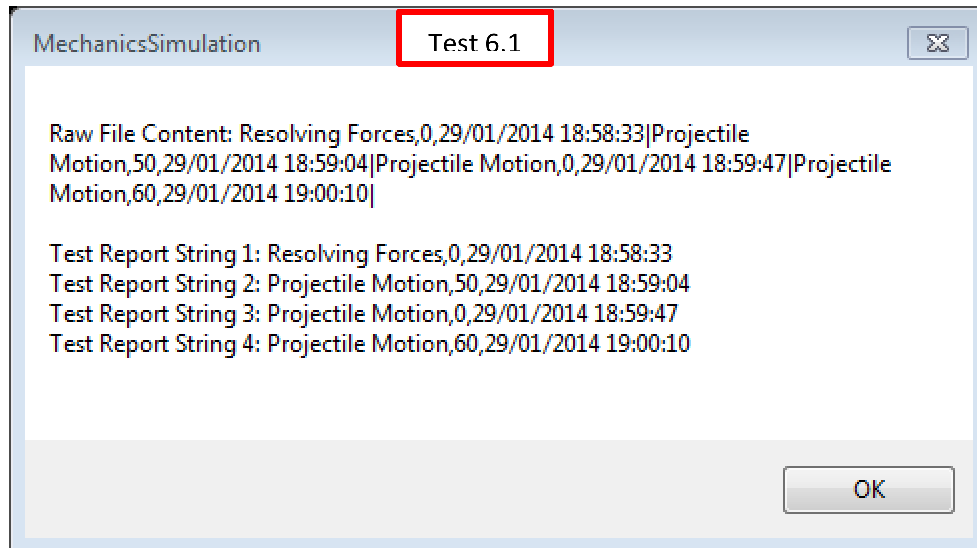




Test Series 6

Test Series and Number	Purpose/ Description	Test Data and Type	Expected Result	Actual Result
6.1	To split the full string into individual test reports	**Typical**: A typical user's test data string for 4 tests, saved in chronological order with more than one category	A list of separate test report data strings	A list of separate test report data strings
6.2	To split the individual test report data into Category, Score and Completion Date	**Typical**: The result from test 6.1	For each item in the list of test Reports, a list of the three fields	For each item in the list of test Reports, a list of the three fields
6.3	To analyse the User's past test reports to find the dates of the first and last tests	**Typical**: A typical user's text file	The correct starting and most recent test dates shown on the My Progress screen	The correct starting and most recent test dates shown on the My Progress screen
6.4	To analyse the User's past test reports to find the average score	**Typical**: A typical user's text file	The correct average score across all tests shown on the My Progress screen	The correct average score across all tests shown on the My Progress screen
6.5	To analyse the User's past test reports to find the best and worst categories by finding the average score for each category.	**Typical**: A typical user's text file	The correct best and worst categories shown on the My Progress screen	The correct best and worst categories shown on the My Progress screen

This Test Series looked at the My Progress section of the program, and how a User's data string (saved in their progress text file) is understood by it.

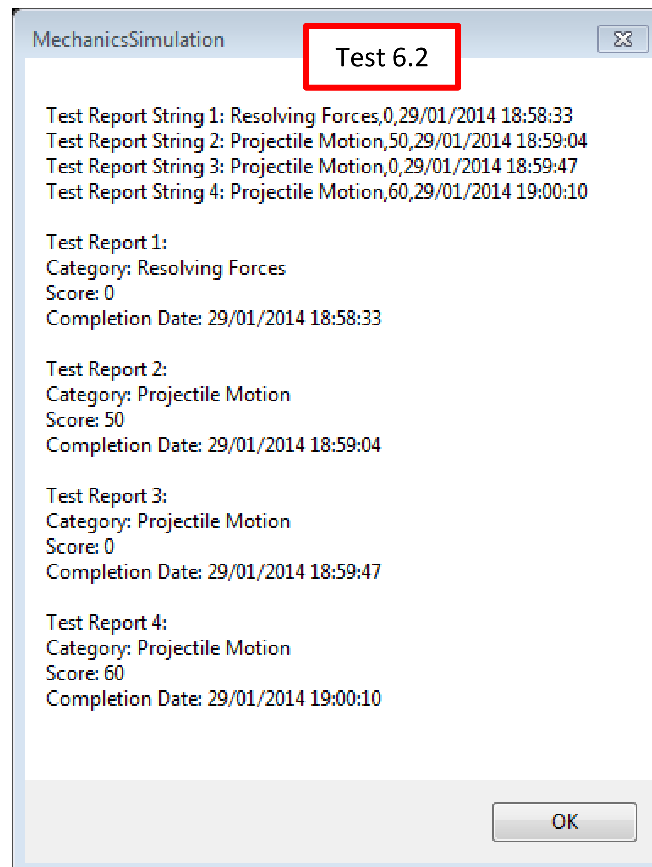

MechanicsSimulation
Test 6.1

Raw File Content: Resolving Forces,0,29/01/2014 18:58:33|Projectile Motion,50,29/01/2014 18:59:04|Projectile Motion,0,29/01/2014 18:59:47|Projectile Motion,60,29/01/2014 19:00:10|

Test Report String 1: Resolving Forces,0,29/01/2014 18:58:33
Test Report String 2: Projectile Motion,50,29/01/2014 18:59:04
Test Report String 3: Projectile Motion,0,29/01/2014 18:59:47
Test Report String 4: Projectile Motion,60,29/01/2014 19:00:10

OK

For tests 6.1 and 6.2, I temporarily added code to make the program output the values needed for testing purposes. This message box wouldn't normally show.


MechanicsSimulation
Test 6.2

Test Report String 1: Resolving Forces,0,29/01/2014 18:58:33
Test Report String 2: Projectile Motion,50,29/01/2014 18:59:04
Test Report String 3: Projectile Motion,0,29/01/2014 18:59:47
Test Report String 4: Projectile Motion,60,29/01/2014 19:00:10

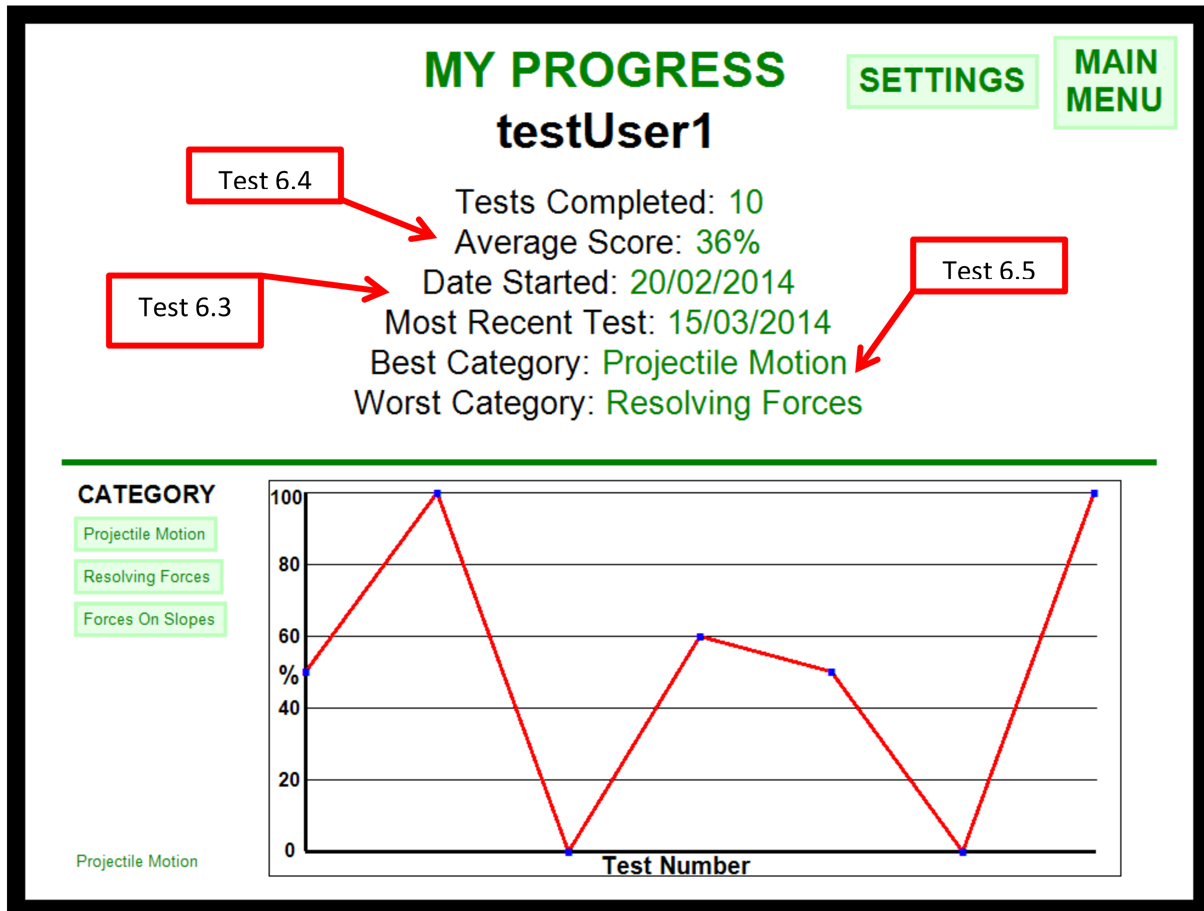
Test Report 1:
Category: Resolving Forces
Score: 0
Completion Date: 29/01/2014 18:58:33

Test Report 2:
Category: Projectile Motion
Score: 50
Completion Date: 29/01/2014 18:59:04

Test Report 3:
Category: Projectile Motion
Score: 0
Completion Date: 29/01/2014 18:59:47

Test Report 4:
Category: Projectile Motion
Score: 60
Completion Date: 29/01/2014 19:00:10

OK



Test Series 7

Test Series and Number	Purpose/ Description	Test Data and Type	Expected Result	Actual Result
7.1	The Encrypt procedure should correctly encrypt a string by the algorithm shown in the Design section	**Typical**: "Hello"	"Nrk6ur"	"Nrk6ur"
7.2	The Decrypt procedure should correctly decrypt an encrypted string by the algorithm shown in	**Typical**: "Nrk6ur"	"Hello"	"Hello"

	the Design section			
7.3	To see if both procedures can work together to encrypt and decrypt a typical user string (longer than just the word "Hello")	**Typical**: A typical user's data string	Return to the exact string before the process started	Return to the exact string before the process started
7.4	To see how the program handles trying to decrypt a user's data string which has been tampered with	**Erroneous**: A decrypted user data string with a single "a" added on the end	An error message, followed by the program continuing to run (i.e. not crashing)	An error message, followed by no crash. The program continued as if the user's data file had been empty (like a new user). A subsequent test would overwrite the existing data, thus clearing the correcting the error.

This Test Series looked at Encryption and Decryption. Its purpose was to make sure that each step in the encryption and decryption procedures correctly manipulated the strings in the way that they were supposed to, and to see how they handled a cypher text which had been tampered with.

Test 7.1

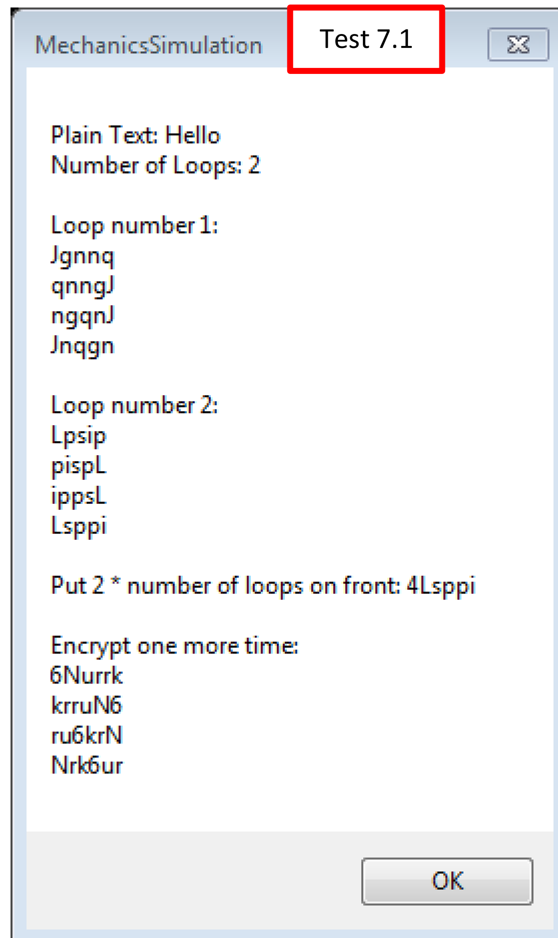
EncryptString("Hello")

NumOfLoops = Random {1,2,3,4} = 2

	String
For 2 times	Hello
1. Move chars 2 ASCII codes up	Jgnnq
2. Reverse	qnngJ
3. Split into evenly indexed, then oddly indexed	ngqnJ
4. Reverse	Jnqgn
Second time	
1.	Lpsip
2.	pispL
3.	ippsL
4.	Lsppi
Put 2* NumOfLoops onto beginning	4Lsppi
Encrypt one more time	
1.	6Nurrk
2.	krruN6
3.	ru6krN
4.	Nrk6ur

EncryptString("Hello") = "Nrk6ur"

For test 7.1, I did a dry run for the encryption algorithm on paper, so that I could make sure each step of the algorithm in the program worked correctly.

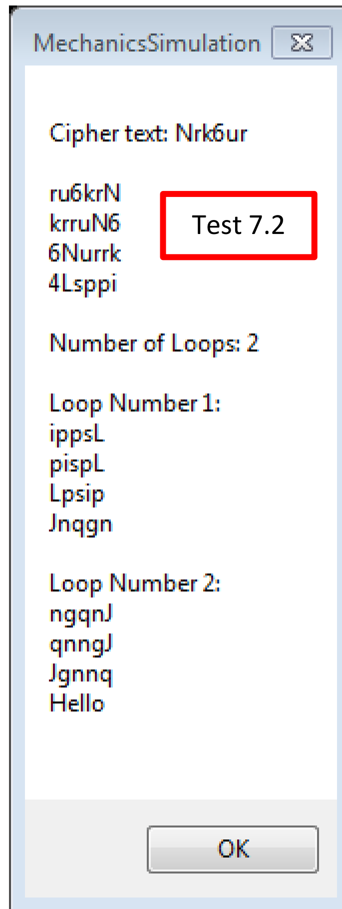

MechanicsSimulation
Test 7.1
Plain Text: Hello
Number of Loops: 2
Loop number 1:
Jgnnq
qnngJ
ngqnJ
Jnqgn
Loop number 2:
Lpsip
pispL
ippsL
Lsppi
Put 2 * number of loops on front: 4Lsppi
Encrypt one more time:
6Nurrk
krruN6
ru6krN
Nrk6ur
OK

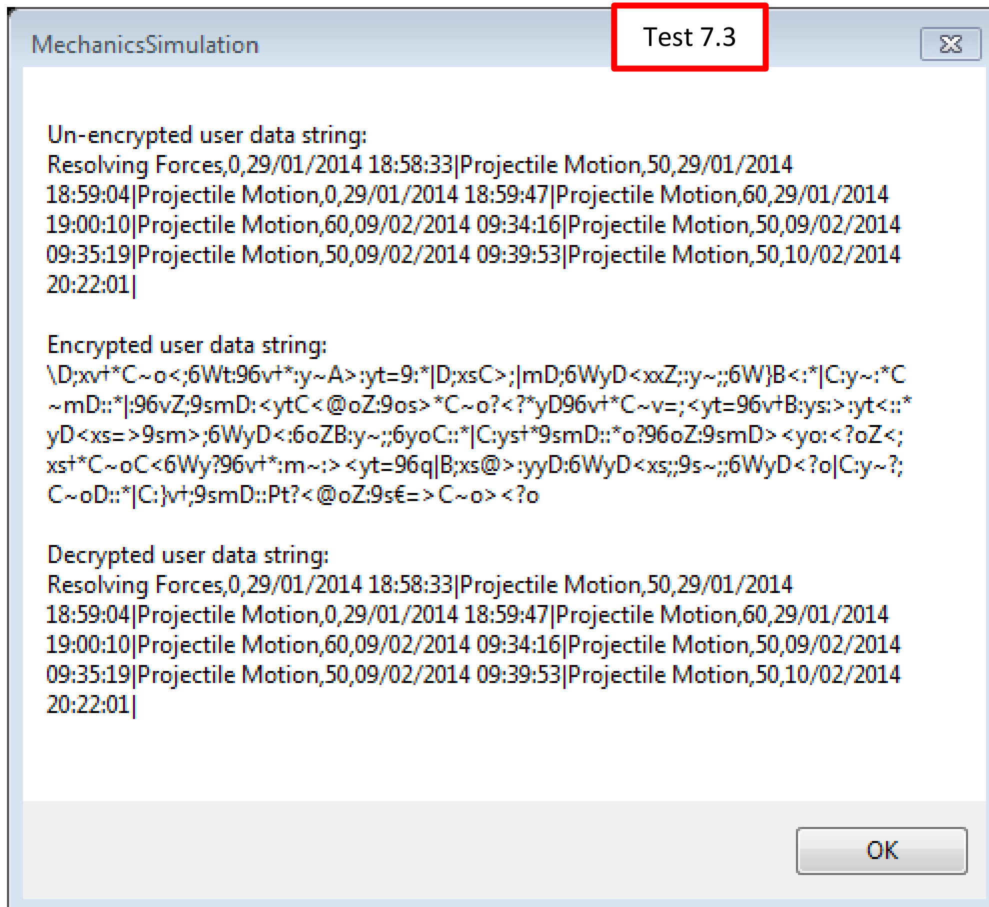
DecryptString("NrR6ur") **Test 7.2**

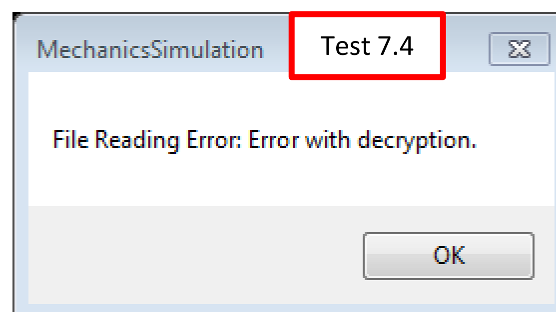
Step	String
	NrR6ur
1. Reverse	ru6krN
2. Split into two halves	[ru6] [krN]
3. Reconstruct String, with char from second, then first etc.	krruN6
4. Reverse	6Nurrk
5. Move chars 2 ASCII codes down	4Lsppi
Divide first char by 2. This is NumOfLoops	Lsppi
For 2 times	
1	ippsL
2	[ip] [psL]
3	pispL
4	Lpsip
5	Jnqgan
second time	
1	nggnJ
2	[ng] [qnJ]
3	qnngJ
4	Jgnnq
5	Hello

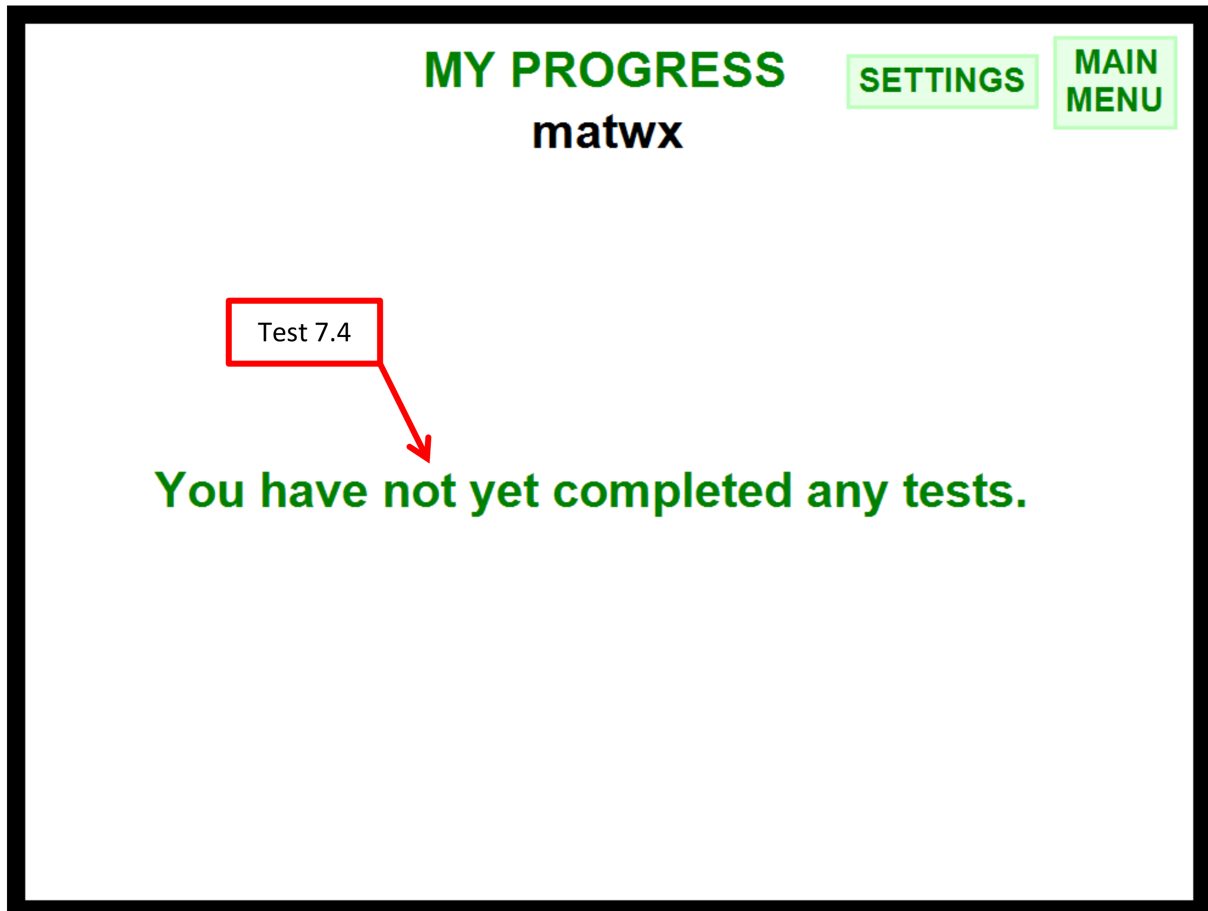
DecryptSting("Nrk6ur") = "Hello"

As with test 7.1, I did a dry run for the decryption algorithm.


MechanicsSimulation
Cipher text: Nrk6ur
ru6krN
krruN6
6Nurrk
4Lsppi
Test 7.2
Number of Loops: 2
Loop Number 1:
ippsL
pispL
Lpsip
Jnqgn
Loop Number 2:
ngqnJ
qnngJ
Jgnnq
Hello
OK


MechanicsSimulation
Test 7.3
Un-encrypted user data string:
Resolving Forces,0,29/01/2014 18:58:33|Projectile Motion,50,29/01/2014 18:59:04|Projectile Motion,0,29/01/2014 18:59:47|Projectile Motion,60,29/01/2014 19:00:10|Projectile Motion,60,09/02/2014 09:34:16|Projectile Motion,50,09/02/2014 09:35:19|Projectile Motion,50,09/02/2014 09:39:53|Projectile Motion,50,10/02/2014 20:22:01|
Encrypted user data string:
\D;xv†*C~o<;6Wt:96v†*:y~A>:yt=9:*|D;xsC>;|mD;6WyD<xxZ;:y~;;6W}B<:*|C:y~:*C~mD::*|:96vZ;9smD:<ytC<@oZ:9os>*C~o?<?*yD96v†*C~v=;<yt=96v†B:ys:>:yt<::*yD<xs=>9sm>;6WyD<:6oZB:y~;;6yoC::*|C:ys†*9smD::*o?96oZ:9smD><yo:<?oZ<;xs†*C~oC<6Wy?96v†*:m~:><yt=96q|B;xs@>:yyD:6WyD<xs;;9s~;;6WyD<?o|C:y~?;C~oD::*|C:}v†;9smD::Pt?<@oZ:9s€=>C~o><?o
Decrypted user data string:
Resolving Forces,0,29/01/2014 18:58:33|Projectile Motion,50,29/01/2014 18:59:04|Projectile Motion,0,29/01/2014 18:59:47|Projectile Motion,60,29/01/2014 19:00:10|Projectile Motion,60,09/02/2014 09:34:16|Projectile Motion,50,09/02/2014 09:35:19|Projectile Motion,50,09/02/2014 09:39:53|Projectile Motion,50,10/02/2014 20:22:01|
OK

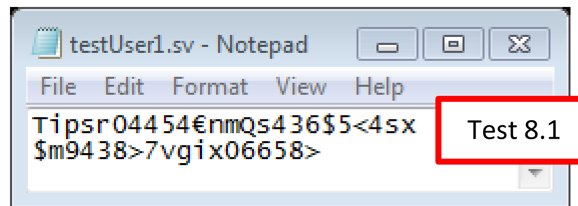

MechanicsSimulation
Test 7.4
File Reading Error: Error with decryption.
OK

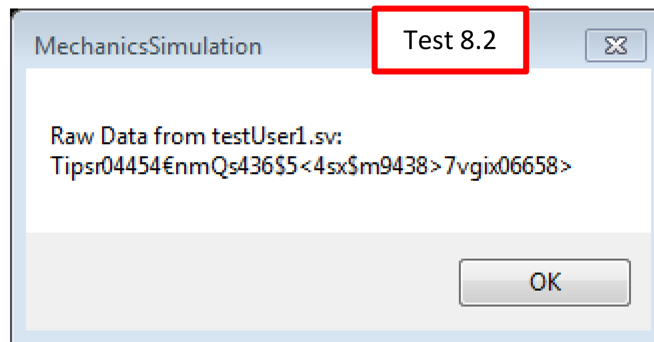


Test Series 8

Test Series and Number	Purpose/ Description	Test Data and Type	Expected Result	Actual Result
8.1	The program should be able to write the user's encrypted data to their file	**Typical**: A new user completes a test	The user's text file written to	The user's text file written to
8.2	The program should be able to read the user's encrypted data from their file	**Typical**: The user's file from test 8.1	The whole of the data and nothing else is read from the text file	The whole of the data and nothing else is read from the text file

This Test Series was to check that the program could correctly read and write from the user save files.



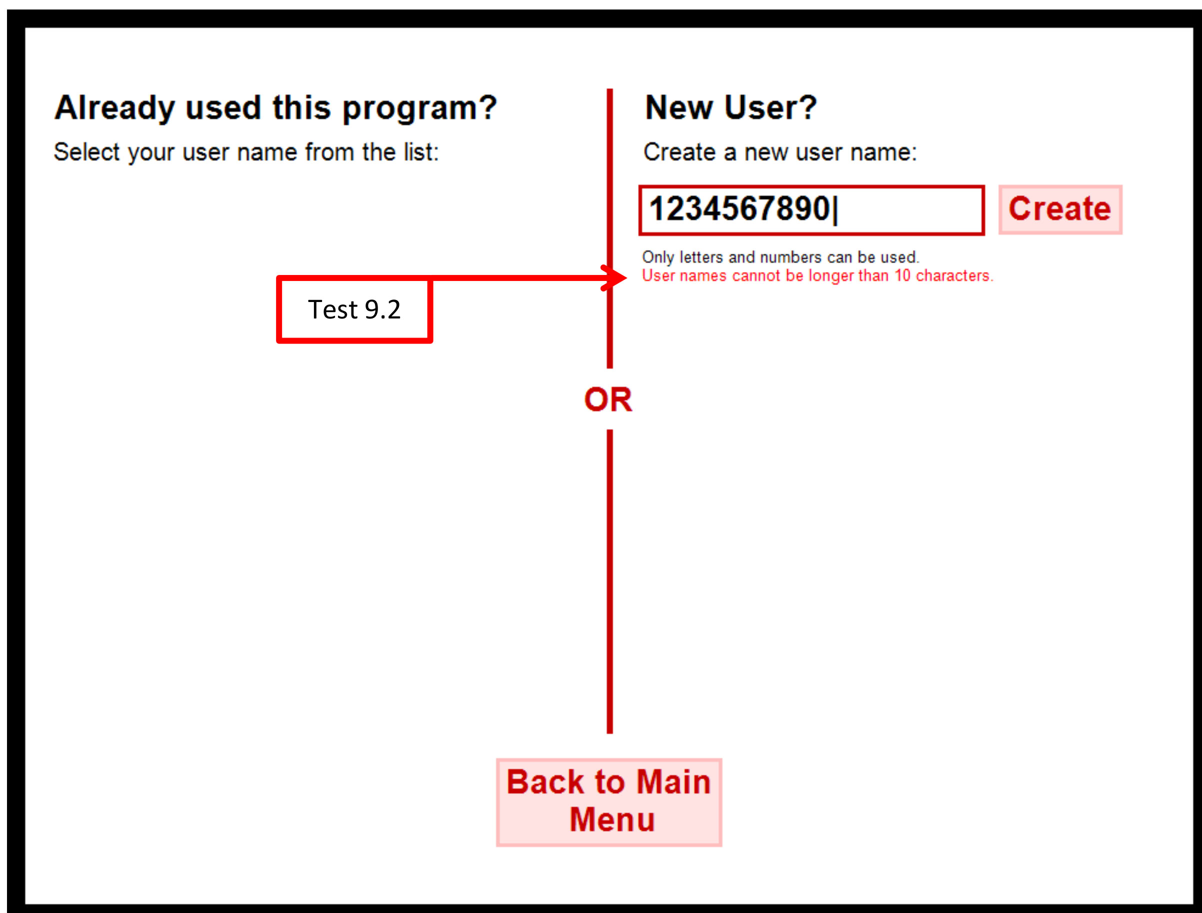


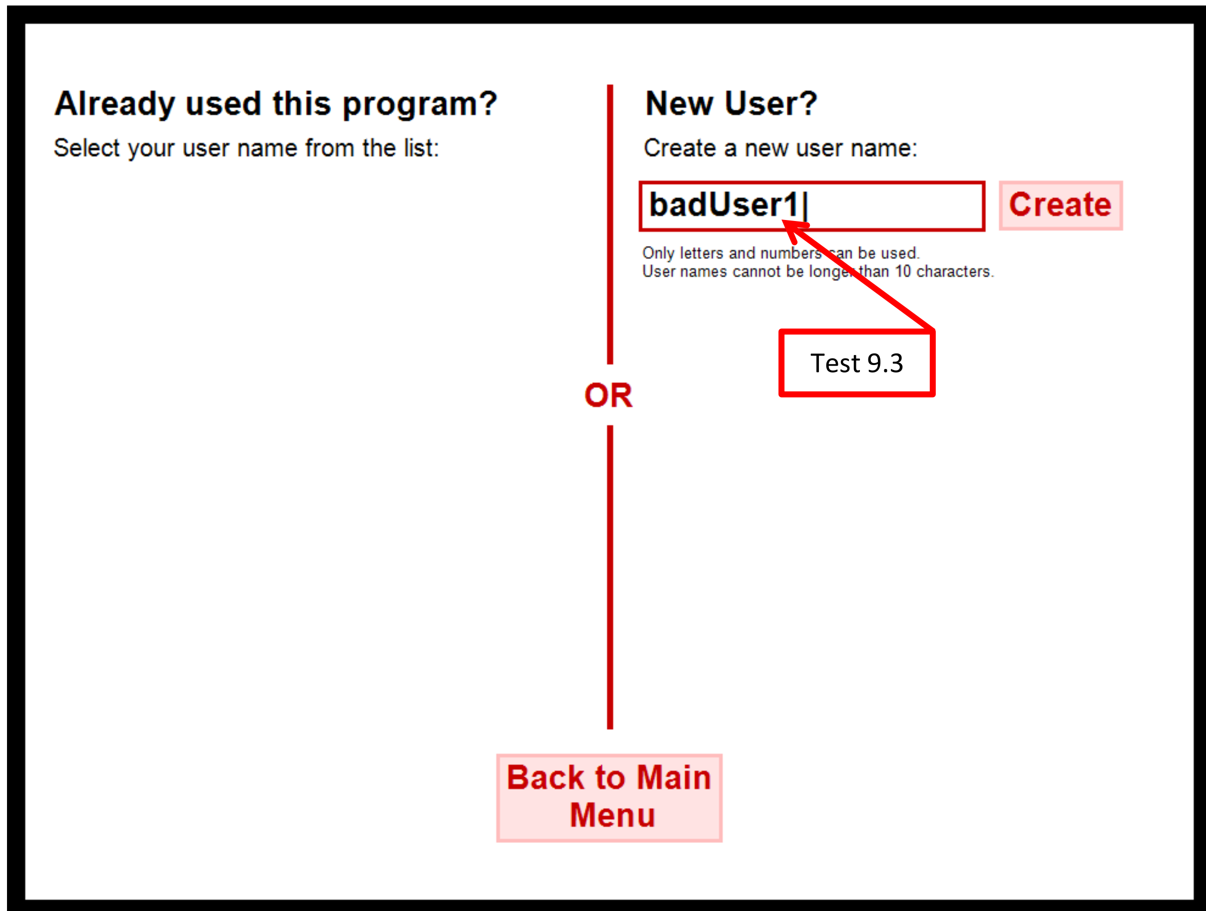
Test Series 9

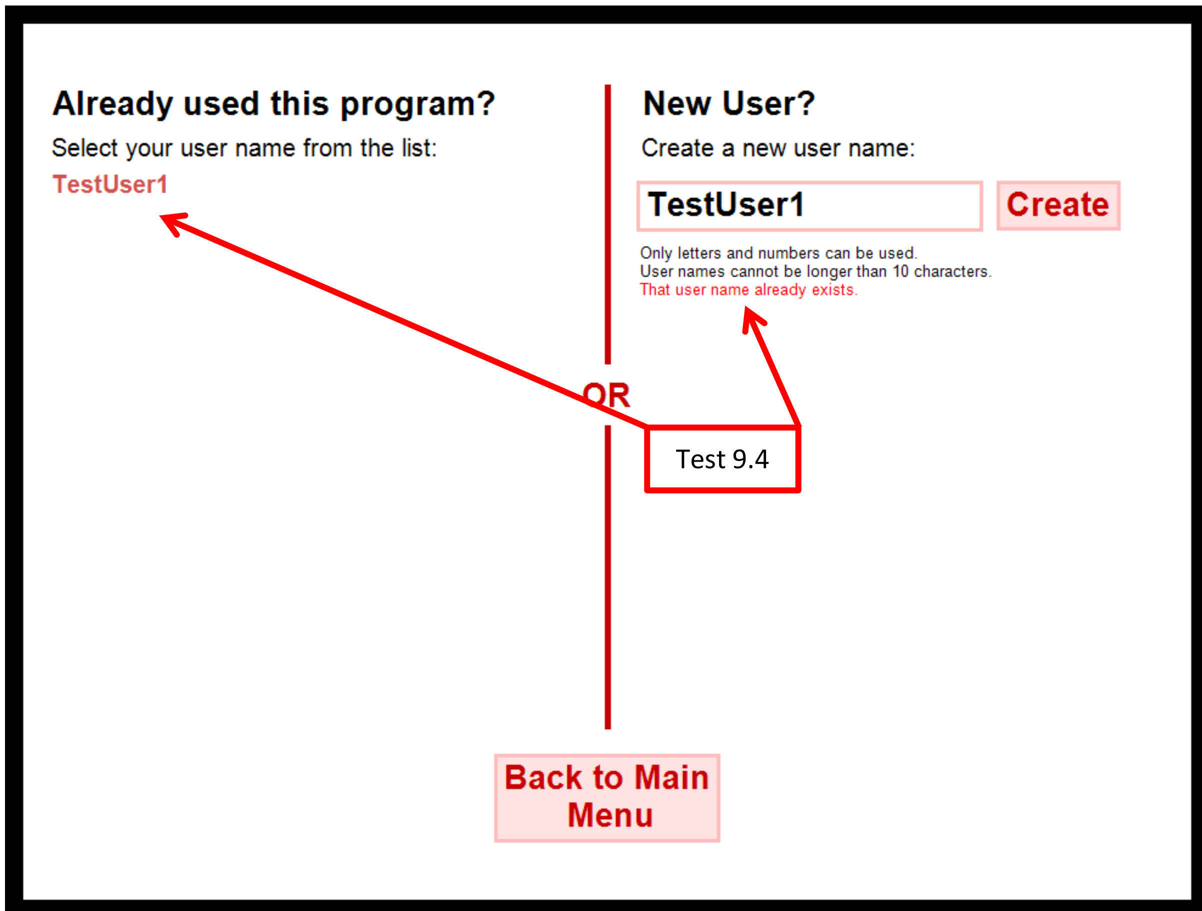
Test Series and Number	Purpose/ Description	Test Data and Type	Expected Result	Actual Result
9.1	The user name should not be blank (of zero length)	**Erroneous**: Attempt to enter a blank username	No user created	No user created, and text box becomes unfocused
9.2	The user name should not be longer than 10 characters	**Erroneous**: Attempt to enter a username longer than 10 characters: 12345678901	Any character after 10 doesn't show up and the 10 character limit is highlighted	Any character after 10 doesn't show up and the 10 character limit is highlighted
9.3	The user name should not contain characters other than lower and uppercase letters or numbers	**Erroneous**: Attempt to enter a username with invalid characters: bad_User_!	Any invalid character that is tried to be entered doesn't show up in the text box	Only the valid characters show up. The "!" is read as "1"
9.4	The username should not be identical to one which already exists	**Erroneous**: Attempt to enter two identical usernames: TestUser1	The first user is created normally, but the second doesn't work	The first user is created, but the second user causes an error message
9.5	For a valid user name, the text file should be created with the path "C:\Users\%pcUser%\Documents\	**Typical**: Attempt to enter a valid username: TestUser1	Empty text file created at the correct location	Empty text file created at the correct location

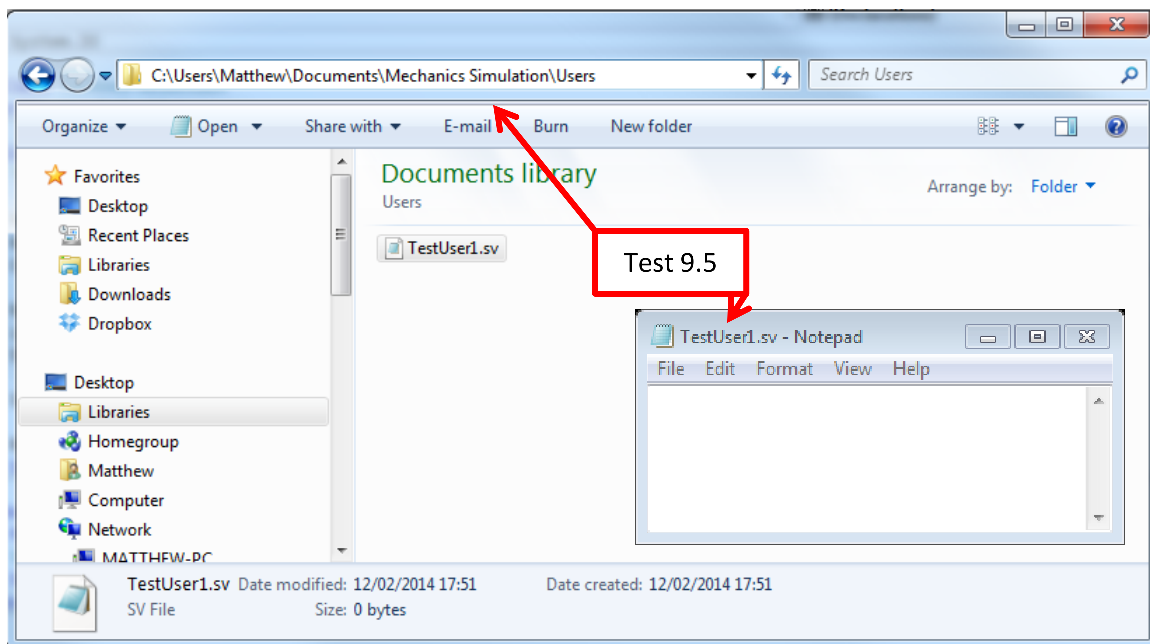
	Mechanics Simulation\Users\%newUsername %.sv”			
9.6	After a new user is created, the username should be visible on the user list to login immediately	**Typical**: Attempt to enter a valid username: TestUser1	Username visible on the left user list instantly	Username visible on the left user list instantly

This test series was for testing all of the processes associated with creating a new user profile for the program. This includes validation for the user name, as well as creating the text file for that user. For testing, only the test mode user selection screen was used, but all of the tests would apply to the My Progress user selection screen, as it is the same apart from the colour.




Already used this program?
Select your user name from the list:
New User?
Create a new user name:
badUser1
Create
Only letters and numbers can be used.
User names cannot be longer than 10 characters.
Test 9.3
OR
Back to Main Menu


Already used this program?
Select your user name from the list:
TestUser1
New User?
Create a new user name:
TestUser1
Create
Only letters and numbers can be used.
User names cannot be longer than 10 characters.
That user name already exists.
OR
Test 9.4
Back to Main Menu


C:\Users\Matthew\Documents\Mechanics Simulation\Users
Search Users
Organize
Open
Share with
E-mail
Burn
New folder
Favorites
Desktop
Recent Places
Libraries
Downloads
Dropbox
Desktop
Libraries
Homegroup
Matthew
Computer
Network
Documents library
Users
Arrange by: Folder
TestUser1.sv
Test 9.5
TestUser1.sv - Notepad
File Edit Format View Help
TestUser1.sv Date modified: 12/02/2014 17:51 Date created: 12/02/2014 17:51
SV File Size: 0 bytes

Already used this program?
Select your user name from the list:
TestUser1

Test 9.6

OR

New User?
Create a new user name:

Create

Only letters and numbers can be used.
User names cannot be longer than 10 characters.

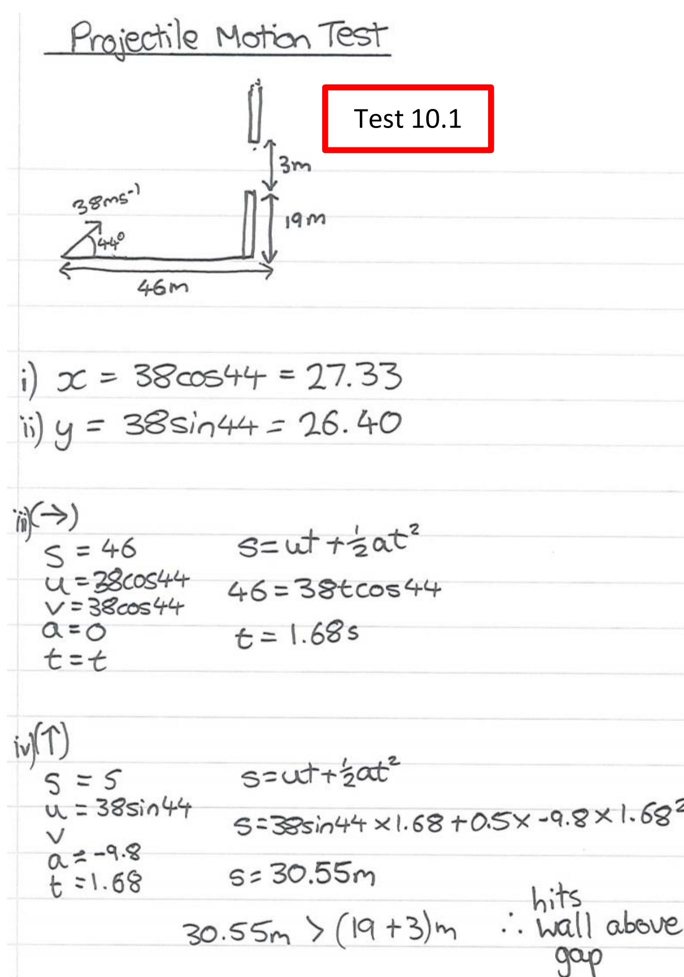
Back to Main Menu

Test Series 10

Test Series and Number	Purpose/ Description	Test Data and Type	Expected Result	Actual Result
10.1	To check that the answers for a Projectile Motion test are calculated correctly	**Typical**: Answers correct to at least 2 decimal places are input	Program recognises correct answers and marks the test with 100%	Program recognises correct answers and marks the test with 100%
10.2	To check that the answers for a Resolving Forces test are calculated correctly	**Typical**: Answers correct to at least 2 decimal places are input	Program recognises correct answers and marks the test with 100%	Program recognises correct answers and marks the test with 100%
10.3	To check that the answers for a Forces on Slopes test are calculated correctly	**Typical**: Answers correct to at least 2 decimal places are input	Program recognises correct answers and marks the test with 100%	Program recognises correct answers and marks the test with 100%

10.4	Inputs entered into the answer boxes should not contain more than one decimal point	**Erroneous**: Attempt to enter a value with more than one decimal place 69.4.5	The second attempt at keying in a dot should be ignored	69.45
10.5	Inputs entered into the answer boxes should not start with a decimal point	**Erroneous**: Attempt to enter a value starting with a decimal place: .498	The attempt at entering the dot first should be ignored	498
10.6	Answers should not be given to any less than two decimal places	**Erroneous**: Attempt to complete the test with one answer only given to one decimal place	Test should not be completed, warning at top of screen should flash blue and incorrect box's border should flash	Test not completed, warning at top of screen flashes blue and incorrect box's border flashes

The purpose of this test series was to look at the test section of the program. I tested each category of test once, and tested the validation of the answer boxes.



Projectile Motion TEST

SETTINGS
MENU

All numerical answers must be given to at least two decimal places.

A ball is fired from a cannon at 44° to the horizontal at 38m/s. A wall 46m away has a 3m gap 19m above the ground.

1) Calculate the X and Y components of the initial velocity (m/s). [2]

X: 27.33

Y: 26.40

2) Calculate the time at which the ball will reach the wall (s). [1]

t: 1.68

3) Will the ball go through the gap? [3]

Yes | No

User: matwx

Projectile Motion Test Report

Part 1: 2/2
Correct Answer: X: 27.33m/s Y: 26.4m/s

Part 2: 1/1
Correct Answer: t: 1.68s

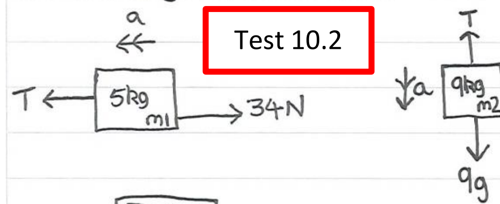
Part 3: 3/3
Correct Answer: No

Total: 6/6 (100%)
Date: 16/03/2014
Time: 10:09

Take another Projectile Motion Test

Test 10.1

Resolving Forces Test



F=ma

$m_1: (\leftarrow)\quad T-34=5a \quad (1)$

$m_2: (\downarrow)\quad 9\times9.8-T=9a$

$88.2-T=9a \quad (2)$

$(1)+(2):\quad 88.2-34=14a$

$54.2=14a$

$a=3.87\,ms^{-2}$

$(1)\quad T=5a+34$

$T=5\times3.87+34$

$T=53.36\,N$

$(\downarrow)\quad s=2 \qquad s=ut+\frac{1}{2}at^2$

$u=0 \qquad 2=0.5\times3.87\,t^2$

v

$a=3.87 \qquad t^2=1.03$

$t=t \qquad t=1.02s$

Resolving Forces TEST

SETTINGS

MENU

All numerical answers must be given to at least two decimal places.

Two masses are connected by a light, inextensible string over a smooth pulley. m1 has a mass of 5kg and is on a horizontal surface with constant friction of 34N. m2 has a mass of 9kg and is 2m above the ground. The system is released from rest.

1) Calculate the tension in the string (N). [2]

T: 53.36

2) Calculate the acceleration of the system (m/s²). [2]

a: 3.87

3) Calculate the time taken for m2 to reach the ground (s). [2]

t: 1.02

Mark

User: matwx

Resolving Forces Test Report

Part 1: 2/2
Correct Answer: T: 53.36N

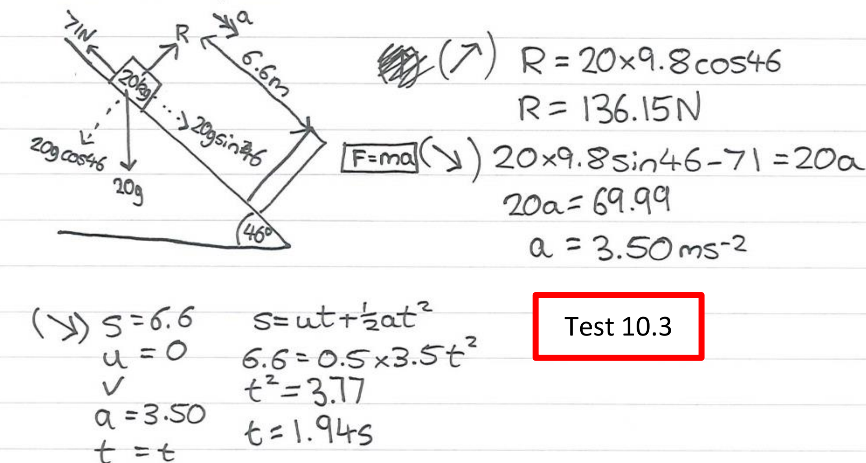
Part 2: 2/2
Correct Answer: a: 3.87m/s²

Part 3: 2/2
Correct Answer: t: 1.02s

Total: 6/6 (100%)
Date: 16/03/2014
Time: 10:16

Take another Resolving Forces Test

Test 10.2



Forces On Slopes TEST

SETTINGS

MENU

All numerical answers must be given to at least two decimal places.

A block of mass 20kg is released from rest on a slope at an angle of 46° to the horizontal, with a constant friction of 71N. A wall perpendicular to the slope is 6.6m away from the block.

1) Calculate the normal reaction force on the block (N). [1]

R: 136.15

2) Calculate the acceleration of the block (m/s^2). [2]

a: 3.50

3) Calculate the time taken for the mass to hit the wall (s). [2]

t: 1.94

Mark

User: matwx

Forces On Slopes Test Report

Part 1: 1/1
Correct Answer: R: 136.15N

Part 2: 2/2
Correct Answer: a: $3.5m/s^2$

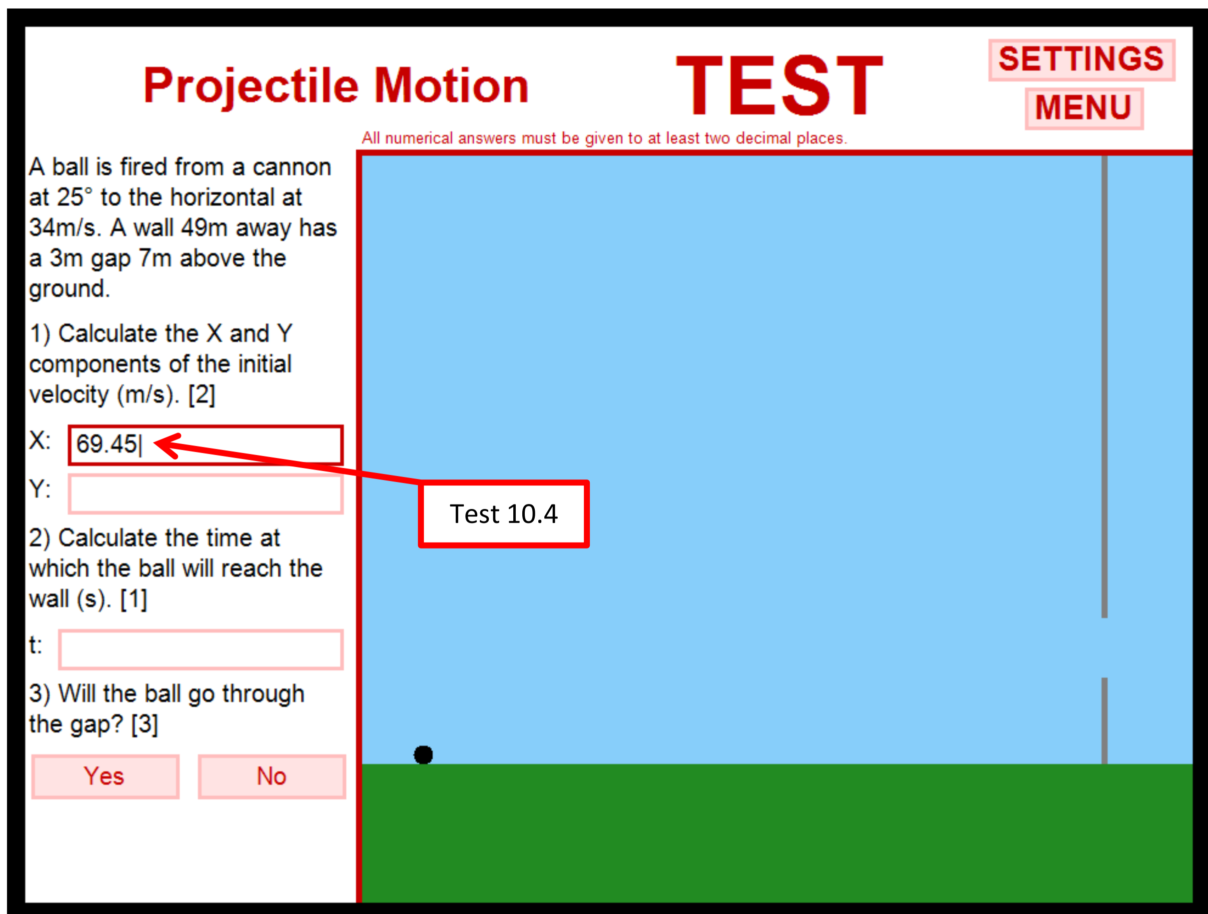
Part 3: 2/2
Correct Answer: t: 1.94s

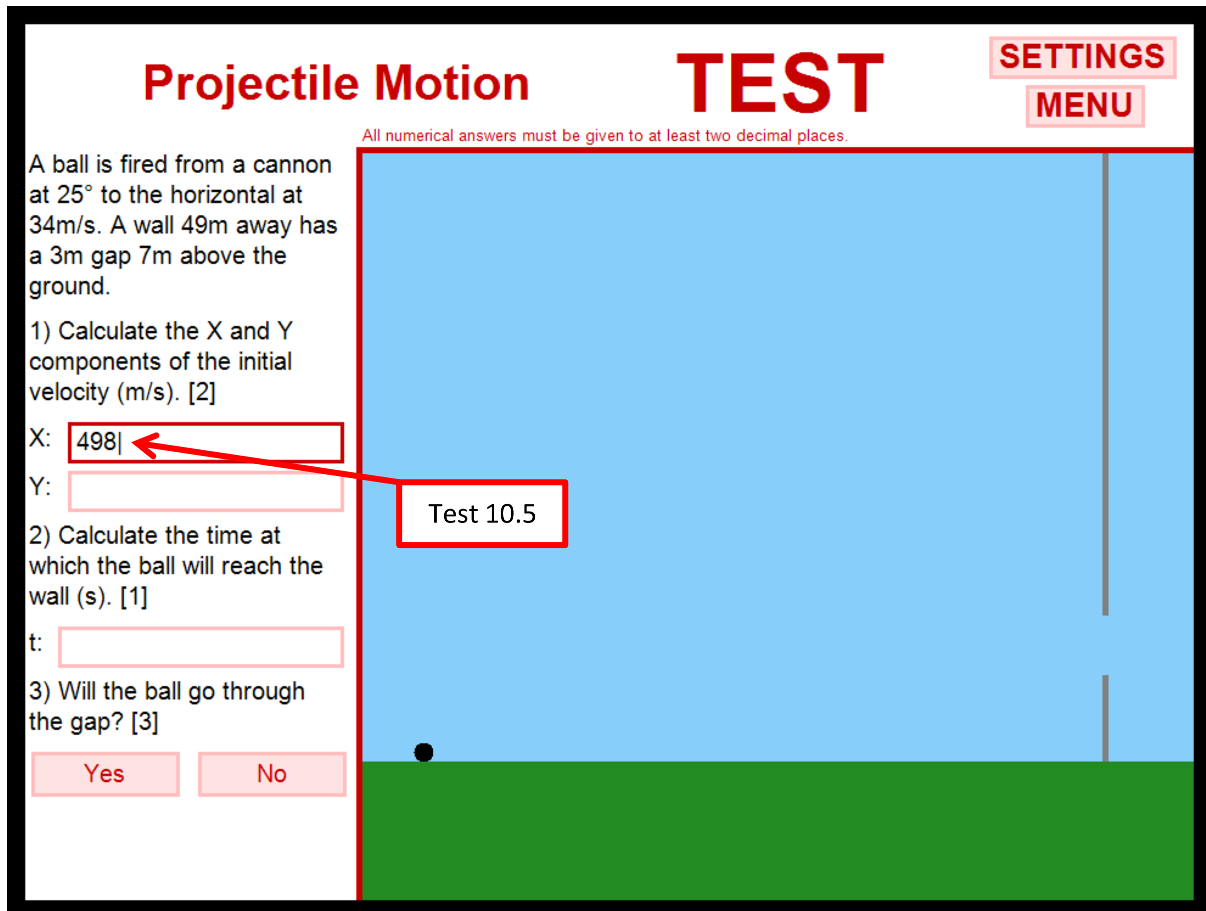
Total: 5/5 (100%)
Date: 16/03/2014
Time: 10:24

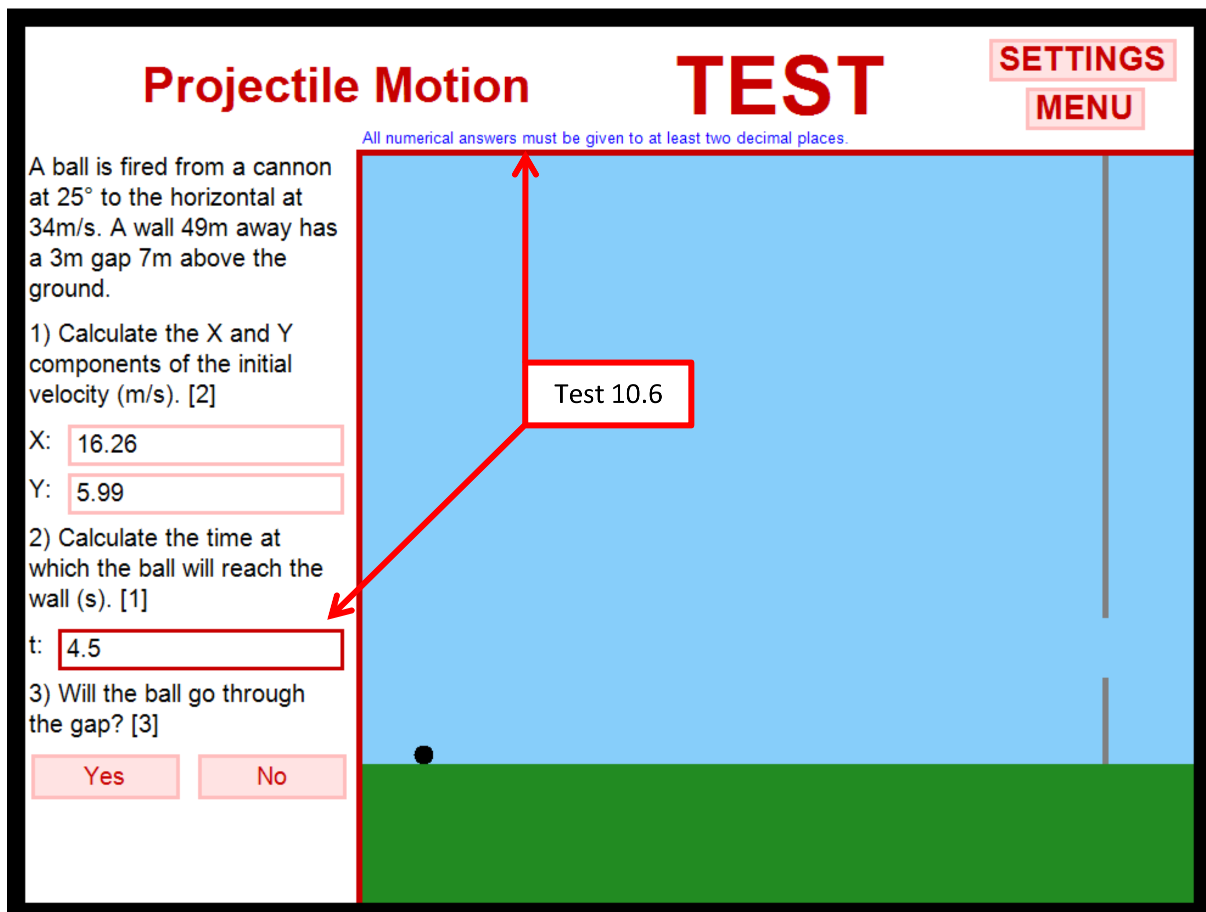
Take another Forces On Slopes Test

Test 10.3

For tests 10.1, 10.2 and 10.3, I worked through a problem from each category on paper and ensured that my answers were correct using the correct methods. The program marked all of my answers as correct, meaning that it must have calculated the answers correctly too.


Projectile Motion
TEST
SETTINGS
MENU
All numerical answers must be given to at least two decimal places.
A ball is fired from a cannon at 25° to the horizontal at 34m/s. A wall 49m away has a 3m gap 7m above the ground.
1) Calculate the X and Y components of the initial velocity (m/s). [2]
X: 69.45
Y:
2) Calculate the time at which the ball will reach the wall (s). [1]
t:
3) Will the ball go through the gap? [3]
Yes
No
Test 10.4


Projectile Motion
TEST
SETTINGS
MENU
All numerical answers must be given to at least two decimal places.
A ball is fired from a cannon at 25° to the horizontal at 34m/s. A wall 49m away has a 3m gap 7m above the ground.
1) Calculate the X and Y components of the initial velocity (m/s). [2]
X: 498
Y:
2) Calculate the time at which the ball will reach the wall (s). [1]
t:
3) Will the ball go through the gap? [3]
Yes
No
Test 10.5

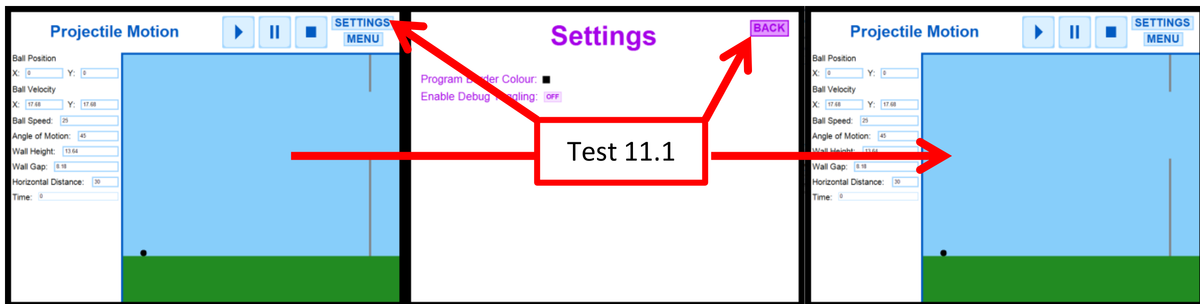


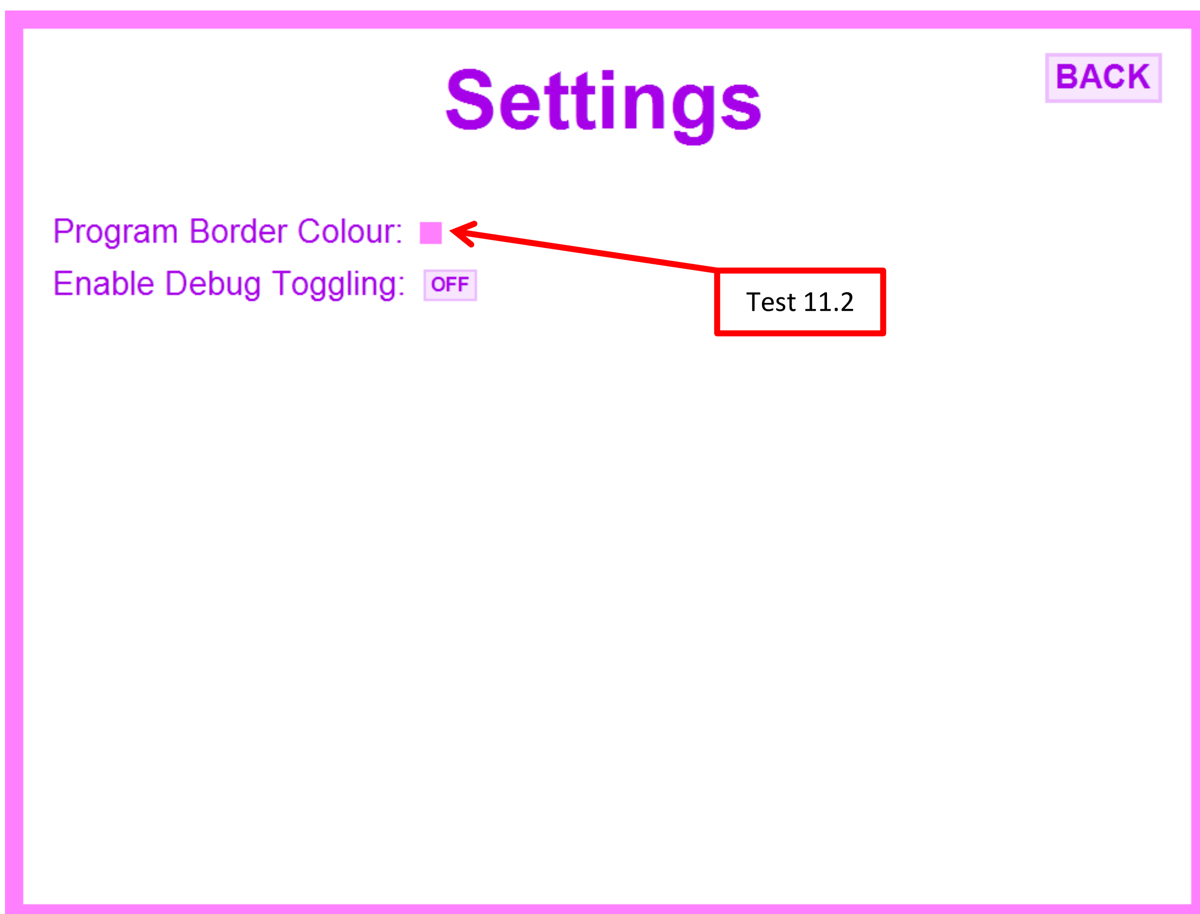
Test Series 11

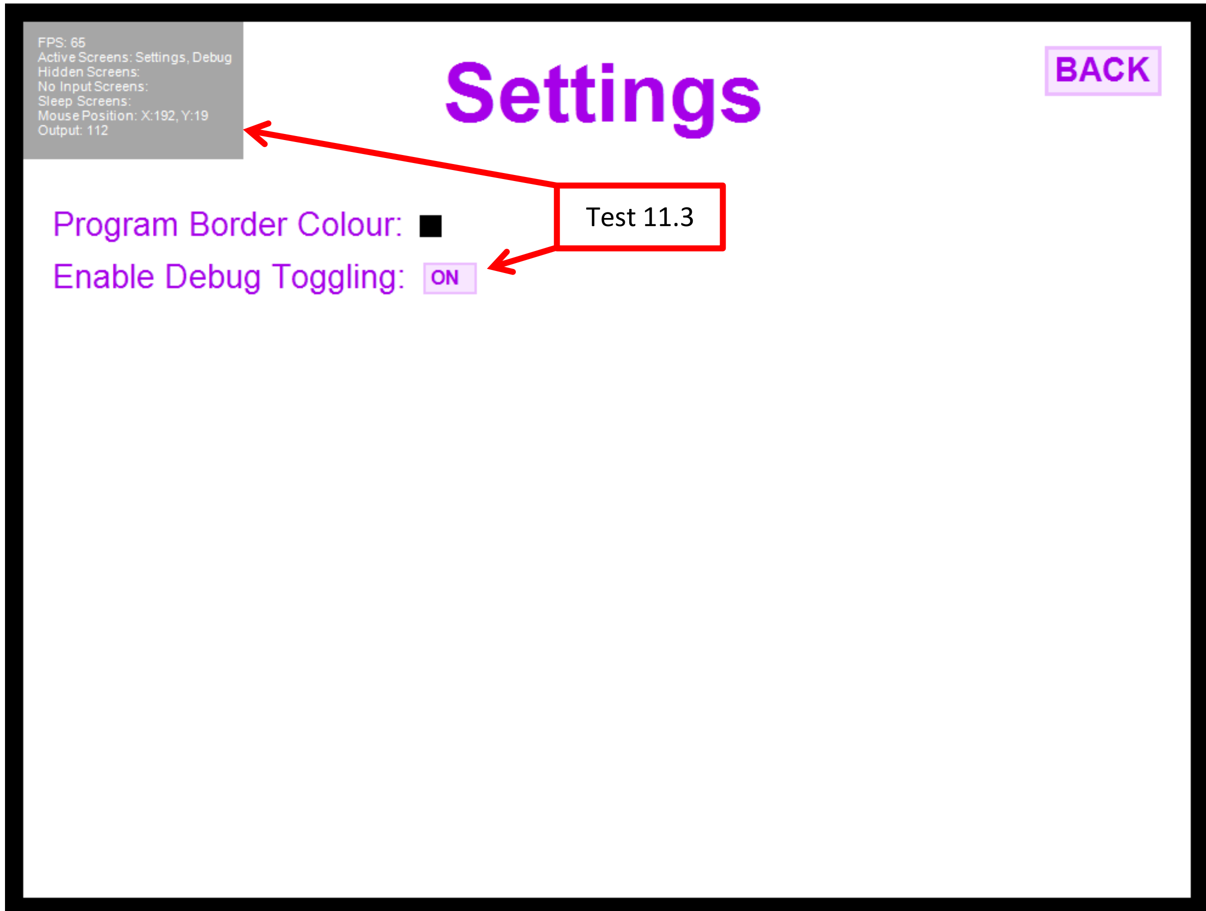
Test Series and Number	Purpose/ Description	Test Data and Type	Expected Result	Actual Result
11.1	The back button on the settings screen should return the user to which ever screen(s) they came from	**Typical**: Attempt to access settings screen from Projectile Motion Simulation, then click back button on settings screen	User is returned to the Projectile Motion Simulation	User is returned to the Projectile Motion Simulation
11.2	The Program Border Colour Selector should be able to be used to change the border colour of the main window	**Typical**: Attempt to click the colour selector and select a pink colour	Main window border and colour selector turns pink (black by default)	Main window border and colour selector turns pink
11.3	The Enable Debug Toggling setting should work	**Typical**: Press the F1 key, then click the button so that it says on, then press the F1 key again	Debug Screen only appears when the button is toggled to 'on'	Debug Screen only appears when the button is toggled to 'on'
11.4	If the debug screen	**Typical**: When the	Debug Screen disappears	Debug Screen

	is visible when the button is toggled to ‘off’, the debug screen should disappear	debug screen is visible, toggle the button to ‘off’		disappears (no screenshot needed)

This purpose of this test series was to make sure that some program settings on the settings screen worked at intended.





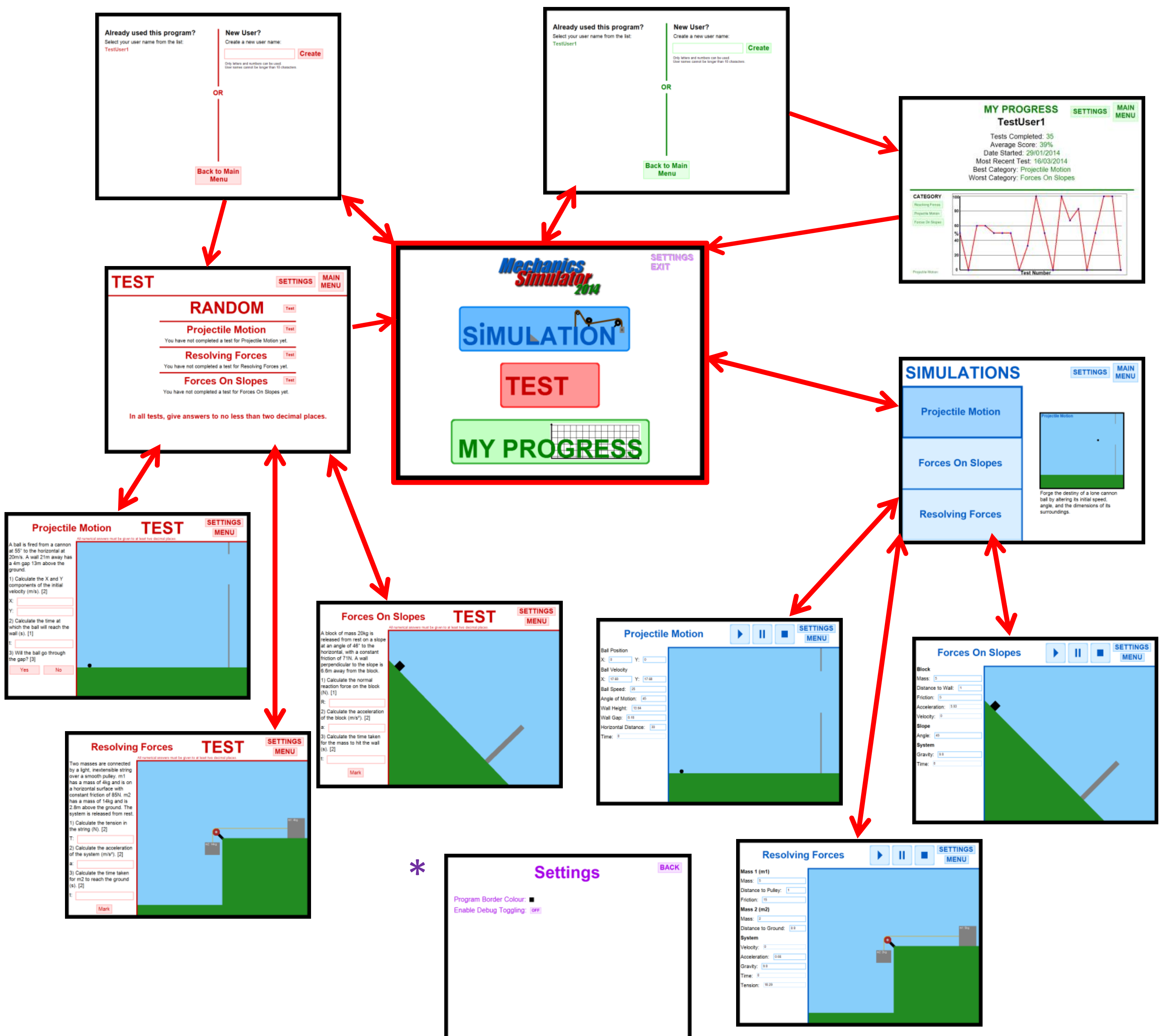

FPS: 65
Active Screens: Settings, Debug
Hidden Screens:
No Input Screens:
Sleep Screens:
Mouse Position: X:192, Y:19
Output: 112
Settings
BACK
Test 11.3
Program Border Colour:
Enable Debug Toggling:
ON

System Maintenance

System Overview

I have created my program as a Windows Forms Application using the VB.NET programming language. I used the Microsoft Visual Studio 2012 program to design and implement my program.

The diagram of images below shows the fundamental navigation between various screens in my program. This could be compared to my screen navigation design in the design section (page 13). Since the design stage, I have added a Settings menu*, which can be accessed from every screen except from the User Selection screens.



Graphics

My program does not use the conventional 'forms design view' approach to drawing graphics to the screen. This is because I found this method caused lag or flickering when there are objects being regularly moved around on the screen, which is essential for my Simulations. Instead I use a different drawing approach.

The Main Form has a PictureBox object called Display and a Bitmap called BMP, each with a width of 960px and a height of 720px. Display can be seen by the user on the screen, but BMP is in memory only and therefore cannot be seen. There is an object called GFX, which is of the Graphics type. This is an in-built object type which contains loads of methods for different ways of drawing graphics. I use this object to draw graphics to BMP, and every tick of the MainTimer, the image of Display is updated with BMP.

The algorithm for handling all graphics drawing for my program is below:

1. Each time the MainTimer ticks, do the following
2. If the program is paused (if ProgramPause is True) then do nothing, else do the following
3. Allow the Screen Manager to Update and Handle the Input of all screens
4. Clear the Bitmap by filling it with white
5. Allow the Screen Manager to Draw all of the appropriate screens to the Bitmap
6. Update Display's Image with the Bitmap

Below is the code in the Main form for the MainTimer tick event:

```
    Private Sub MainTimer_Tick(sender As System.Object, e As System.EventArgs) Handles
MainTimer.Tick
        If ProgramPause = False Then
            'UPDATE SCREENS
            ScreenManager.Update()

            'DRAW
            GFX.Clear(Color.White)

            ScreenManager.Draw()

            Display.Image = BMP
        End If
    End Sub
```

I think that this method of drawing is effective because the user's view isn't updated until all of the drawing for a cycle has finished. The conventional Windows Forms graphics method of moving around pre-designed objects from design view updates the user's view each time an object is moved. If an object is moved very frequently, or if multiple objects are being moved at once, this is likely to cause flickering.

If I want to draw something to the screen from anywhere in the program, I need to call the appropriate method in the GFX object on the main form. For example, to draw text onto the screen, I would write the code

```
Main.GFX.DrawString(Text, Font, New SolidBrush(Colour), Location)
```

Where Text is the Text to draw, Font is the Font to draw the text in, Colour is the colour I want the text to be and Location is a point containing the number of pixels along and down from the top-left corner of the Bitmap that I want the drawing to start. The important part of the code above is the "Main.GFX.", which is followed by and in-built drawing method.

Dragging the Main Window Around

The program window has a fixed size of 990x750 pixels. One of the features is the ability to drag the main window around the screen. In the Main form, there is a Point variable called DragFormPos which saves the mouse location before the window is dragged around. The algorithm for Dragging the Main Window as created in the Design process is below:

1. If a mouse button is pressed while the mouse cursor is hovered over the program border, save the X and Y distance of the mouse cursor position from the top-left corner of the window
2. If the mouse is moved while the mouse button is still held down, update the window's position on the screen
    a. The window's X coordinate should become the mouse cursor's X coordinate translated to the left by the X distance saved
    b. The window's Y coordinate should become the mouse cursor's Y coordinate translated up by the Y distance saved

The code in the Main form which executes this code is:

```
    Private Sub Form_MouseDown(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
        DragFormPos = New Point(e.X, e.Y)
    End Sub

    Private Sub Form_MouseMove(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseMove
        If Not DragFormPos = Nothing Then
            SetDesktopLocation(Windows.Forms.Form.MousePosition.X - DragFormPos.X,
Windows.Forms.Form.MousePosition.Y - DragFormPos.Y)
        End If
    End Sub

    Private Sub Form_MouseUp(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseUp
        DragFormPos = Nothing
    End Sub
```

This code sets the DragFormPos location variable to "Nothing" when a mouse button is released. This is the program's way of knowing if the mouse button is still held down when the mouse moves.

Managing Screens

As introduced in the design section (Page 34), the screen manager is an object instantiated when the program is executed whose purpose is to handle all of the program's screens neatly and efficiently. A screen, for my program, is a part of the user's view. Some examples of screens include the

Simulation button on the Title Screen, the Projectile Motion screen, the Projectile Motion Test and the Test Report.

The ScreenManager class has two lists: Screens and NewScreens. The Screens list holds all of the currently enabled screens, and the NewScreens list holds all of the screens which have been added by other parts of the program during the current cycle. These new screens are added to the Screens list in the next cycle.

Updating, Handling Input and Keeping a List of Screens

The algorithm for Updating and Handling Input for all of the currently enabled screens is as follows:

1. Add all screens in the NewScreens list to the main Screens list
2. Clear the NewScreens list
3. Look at a screen
4. If it is in the ShutDown screen state, remove it from the list
5. If it is in the Active or Hidden screen state, allow it to Handle Input
6. If it is not in the Sleep screen state, allow it to Update
7. Repeat steps 3 to 6 for each screen in the main Screens list

The code below shows most of the class's Update procedure:

```
    Public Sub Update()
        ' GENERATE LIST OF DEAD SCREENS FOR REMOVAL
        Dim RemoveScreens As New List(Of BaseScreen)

        For Each FoundScreen As BaseScreen In Screens
            If FoundScreen.State = ScreenState.ShutDown Then
                RemoveScreens.Add(FoundScreen)
            End If
        Next

        ' REMOVE DEAD SCREENS
        For Each FoundScreen As BaseScreen In RemoveScreens
            Screens.Remove(FoundScreen)
        Next

        ' ADD NEW SCREENS TO MAIN LIST FROM THE NEW SCREENS LIST
        For Each FoundScreen As BaseScreen In NewScreens
            Screens.Add(FoundScreen)
        Next
        NewScreens.Clear()

        ' CALL INPUT AND UPDATE PROCEDURES FOR APPLICABLE SCREENS
        For Each FoundScreen As BaseScreen In Screens
            If FoundScreen.State <> ScreenState.Sleep Then
                If Main.Focused And (FoundScreen.State = ScreenState.Active Or
FoundScreen.State = ScreenState.Hidden) Then
                    FoundScreen.HandleInput()
                End If
                FoundScreen.Update()
            End If
        Next
    End Sub
```

This procedure is run every tick of the main game timer. Its basic purpose is to call the HandleInput and Update procedures of all of the current screens. However, it also controls the addition and removal of screens.

The procedure first creates a list and populates it with all of the screens from the Screens list which are in the ShutDown state. It then cycles through all of the screens in this RemoveScreens list and removes their counterparts from the main Screens list. Screens from the NewScreens list are then added to the Screens list. At first, it may seem pointless to have so many different lists just for handling one set of screens. However, if there was only one list of screens, the procedure would have to eventually remove one of them whilst cycling through all of them. This would often cause errors because the program would expect there to be more screens in the list than there actually turn out to be. The NewScreens list is cleared after is has been used. This prevents 'new' screens from becoming 'old' and being added at every cycle.

After removing dead screens and adding new ones, the procedure finally calls the HandleInput and Update procedures of the screens in the Screens list, providing that they are in the appropriate states. The Screen Manager part of the Design section explains the different possible screen states (Page 35).

Drawing Screens

Another role of the Screen Manager as well at Handling Input and Updating screens is to draw the correct screens. The simple algorithm for drawing screens is below:

1. Look at a screen
2. If it is in the Active or the NoInput screen state, allow it to Draw
3. Repeat steps 1 and 2 for each screen in the main Screens list

The Active and NoInput screen states are the only ones which should allow drawing of the screen. See page 80 in this section for an explanation of how drawing works. The code in the Screen Manager class which carries out this algorithm is shown below:

```
    Public Sub Draw()
        ' CALL DRAW PROCEDURE FOR APPLICABLE SCREENS
        For Each FoundScreen As BaseScreen In Screens
            If FoundScreen.State = ScreenState.Active Or FoundScreen.State =
ScreenState.NoInput Then
                FoundScreen.Draw()
            End If
        Next
    End Sub
```

Screen Transitions

If there needs to be a screen transition anywhere in the program, the Screen Manager's AddScreen and UnloadScreen procedures need to be used.

The class's AddScreen procedure is what should be called anywhere else in the program if a new screen needs to be enabled:

```
Public Shared Sub AddScreen(ByVal screen As BaseScreen)
    NewScreens.Add(screen)
End Sub
```

This procedure simply adds the new screen to the NewScreen list.

The UnloadScreen procedure should be called when an enabled screen needs to be removed:

```
Public Shared Sub UnloadScreen(ByVal screen As String)
    'SET THE DESIRED SCREEN'S STATE TO SHUTDOWN
    For Each FoundScreen As BaseScreen In Screens
        If FoundScreen.Name = screen Then
            FoundScreen.Unload()
            Exit For
        End If
    Next
End Sub
```

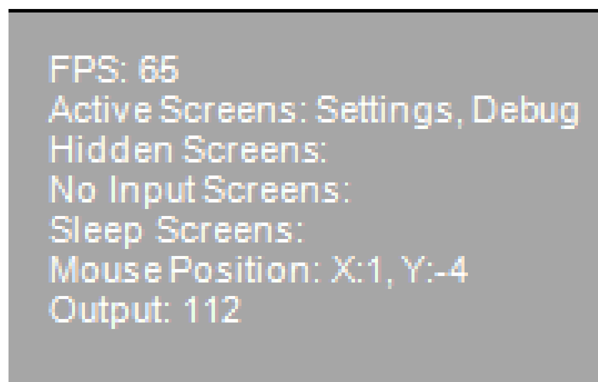
A screen's Unload procedure simply sets that screens state to ShutDown.

The following example shows how these two procedures should be used in the program to perform a screen transition. The example is going from the Simulation Menu back to the Title Screen:

```
ScreenManager.UnloadScreen("SimulationMenu")
ScreenManager.AddScreen(New Title)
ScreenManager.AddScreen(New SimulationButton)
ScreenManager.AddScreen(New TestButton)
ScreenManager.AddScreen(New MyProgressButton)
```

Debug Screen

The debug screen, pictured below, is a screen purely designed for the development of the program. It can be accessed through the settings screen by clicking the button next to "Enable debug toggling" The debug screen can now be toggled on or off by pressing the F1 key (you do not have to be viewing the settings screen to view the debug screen).



It shows important behind-the-scenes information about the program while it is running, such as the number of frames per second (FPS), the states of all of the currently enabled screens, and the mouse position relative to the top-left corner of the main display window. The "Output" part of the debug screen can show anything. By default, it shows the KeyValue associated with the most recent key pressed, which is useful for knowing which keys to allow when handling keyboard input. However, the debug Output can be updated with any string by calling the

Main.ScreenManager.SetDebugOutputMessage() procedure anywhere in the code. This procedure has one parameter, which is the string that you want to display.

Encryption and Decryption

Encryption

One of the algorithms defined in my design section was the encryption function. It is used to encrypt a user's progress text file data before actually saving it to the text file. As in the design section, my encryption algorithm is:

1. Generate a random integer between 1 and 4. Call this NumOfLoops
2. For NumOfLoops times(steps 3-6):
3. Move all characters 2 ASCII codes up
4. Reverse the order of the characters in the string
5. Split the string so it has ALL of the evenly indexed characters followed by the oddly indexed characters. For example, "helloworld" would be turned into "elwrdhlool"
6. Reverse the string again
7. Put 2* NumOfLoops onto the beginning of the string
8. Repeat one iteration of steps 3 to 6

The code below shows the EncryptString function. The capitalised green comments show the steps of the algorithm.

```
Public Function EncryptString(ByVal PlainText As String) As String
    Dim asciied, reversed, split, reversed2 As String
    Dim NumOfLoops As Integer = Rand.Next(1, 4 + 1)

    If PlainText = "" Then
        Return ""
    End If

    Try
        For times = 1 To NumOfLoops

            asciied = ""
            reversed = ""
            split = ""
            reversed2 = ""

            'MOVE ALL CHARACTERS 2 ASCII CODES UP
            For i = 1 To Len(PlainText)
                asciied = asciied & Chr(Asc(Mid(PlainText, i, 1)) + 2)
            Next

            'REVERSE THE ORDER OF THE CHARACTERS IN THE STRING
            reversed = StrReverse(asciied)

            'SPLIT THE STRING SO IT AS ALL OF THE EVENLY INDEXED CHARACTERS....
            For i = 2 To Len(reversed) Step 2
                split = split & Mid(reversed, i, 1)
            Next
```

```
            '....FOLLOWED BY THE ODDLY INDEXED CHARACTERS
            For i = 1 To Len(reversed) Step 2
                split = split & Mid(reversed, i, 1)
            Next

            'REVERSE THE STRING AGAIN
            reversed2 = StrReverse(split)

            PlainText = reversed2
        Next

        'PUT 2 * NumOfLoops ONTO THE BEGINNING OF THE STRING
        PlainText = 2 * NumOfLoops & reversed2

        'REPEAT ONE ITERATION OF THE ENCRYPTION
        asciied = ""
        reversed = ""
        split = ""
        reversed2 = ""

        For i = 1 To Len(PlainText)
            asciied = asciied & Chr(Asc(Mid(PlainText, i, 1)) + 2)
        Next

        reversed = StrReverse(asciied)

        For i = 2 To Len(reversed) Step 2
            split = split & Mid(reversed, i, 1)
        Next

        For i = 1 To Len(reversed) Step 2
            split = split & Mid(reversed, i, 1)
        Next

        reversed2 = StrReverse(split)

        Return reversed2
    Catch ex As Exception
        MessageBox("File Writing Error: Error with encryption.")
        Return ""
    End Try

End Function
```

Decryption

Paired with the encryption function is its opposite which is essential for it to be useful: the decryption function. Again, as in the design section, the algorithm for decryption is:

1. Reverse the string
2. Split the string into two halves. For odd length strings, first half is shorter.
3. Reconstruct the full string, by taking a character from the second half, then the first half, then the second half etc.
4. Reverse the string again
5. Move all characters 2 ASCII codes down
6. Take the first character from the string. Divide this by two, this is the NumOfLoops generated at encryption

7. For NumOfLoops times repeat steps 1 to 5

The code below shows the DecryptString function. The capitalised green comments show the steps of the algorithm.

```
    Public Function DecryptString(ByVal CipherText As String) As String
        Dim reversed, evenlySplit, oddlySplit, finalFused, reversed2, asciied As
String
        Dim NumOfLoops As Integer

        If CipherText = "" Then
            Return ""
        End If

        Try
            reversed = ""
            evenlySplit = ""
            oddlySplit = ""
            finalFused = ""
            reversed2 = ""
            asciied = ""

            'REVERSE THE STRING
            reversed = StrReverse(CipherText)

            'SPLIT THE STRING INTO TWO HALVES
            For i = 1 To Int(Len(reversed) / 2)
                evenlySplit &= Mid(reversed, i, 1)
            Next
            For i = Int(Len(reversed) / 2) + 1 To Len(reversed)
                oddlySplit &= Mid(reversed, i, 1)
            Next

            'RECONSTRUCT THE STRING, BY TAKING A CHARACTER FROM THE SECOND HALF,
            'THEN THE FIRST HALF, THEN THE SECOND HALF ETC.
            For i = 1 To Len(evenlySplit) + Len(oddlySplit)
                finalFused &= Mid(oddlySplit, i, 1) & Mid(evenlySplit, i, 1)
            Next

            'REVERSE THE STRING AGAIN
            reversed2 = StrReverse(finalFused)

            'MOVE ALL CHARACTERS 2 ASCII CODES DOWN
            For i = 1 To Len(reversed2)
                asciied &= Chr(Asc(Mid(reversed2, i, 1)) - 2)
            Next

            CipherText = asciied

            'TAKE THE FIRST CHARACTER AND DIVIDE THIS BY 2. THIS IS THE NumOfLoops
            'GENERATED AT ENCRYPTION
            NumOfLoops = Mid(CipherText, 1, 1)
            NumOfLoops = NumOfLoops / 2
            CipherText = CipherText.Substring(1, Len(CipherText) - 1)


            For times = 1 To NumOfLoops
                'REPEAT THE DECRYPTION
                reversed = ""
                evenlySplit = ""
```

```
                oddlySplit = ""
                finalFused = ""
                reversed2 = ""
                asciied = ""

                reversed = StrReverse(CipherText)

                For i = 1 To Int(Len(reversed) / 2)
                    evenlySplit &= Mid(reversed, i, 1)
                Next
                For i = Int(Len(reversed) / 2) + 1 To Len(reversed)
                    oddlySplit &= Mid(reversed, i, 1)
                Next

                For i = 1 To Len(evenlySplit) + Len(oddlySplit)
                    finalFused &= Mid(oddlySplit, i, 1) & Mid(evenlySplit, i, 1)
                Next

                reversed2 = StrReverse(finalFused)

                For i = 1 To Len(reversed2)
                    asciied &= Chr(Asc(Mid(reversed2, i, 1)) - 2)
                Next

                CipherText = asciied

            Next

            Return asciied
        Catch ex As Exception
            MessageBox("File Reading Error: Error with decryption.")
            Return ""
        End Try

    End Function
```

Timers

Another algorithm which I identified in the design section was one for timers within the program. I needed to create my own timers, for holding code that needs to be repeated for an unknown number of times and shouldn't be repeated as quickly as possible. An example of where a timer is used is the code for updating each of the simulations. If I simply put this code into the simulation screen's Update procedure, the code would repeat at the fastest possible rate, which is not helpful for viewing the simulation.

Below is the algorithm created during the design process for a Timer:

1. When the screen is instantiated, save the current time into a variable, TimerTime
2. In the screen's Update procedure, where the timer is needed:
3. If (CurrentTime - TimerTime) is greater than the intended timer interval:
4. TimerTime ← CurrentTime
5. Code to be carried out each tick of the timer

The code below shows the Update procedure of the Resolving Forces Simulation:

```
Public Overrides Sub Update()
    Dim NewM1X, NewM2Y As Single

    If Enabled = True Then
        If (Now - TTimer).TotalMilliseconds > 25 Then
            TTimer = Now

            'Every 25 milliseconds (ish)

            'Gradually increase the time variable
            'Calculate the expected position as if no collision happens, then
            'see if there should be a collision
            For i = 1 To 10000
                NewM1X = xDist - 0.5 * Acceleration * T ^ 2
                NewM2Y = 0.5 * Acceleration * T ^ 2
                Velocity = Acceleration * T

                If Velocity > 0 Then
                    If NewM2Y >= yDist Then
                        'm2 reaches floor, so stop
                        Velocity = 0
                        Acceleration = 0
                        m2Y = yDist
                        m1X = xDist * 0.2
                        Finished = True
                    Else
                        'no collision, so continue as usual
                        m1X = NewM1X
                        m2Y = NewM2Y
                    End If
                End If

                Tmicros += 1
                T = Tmicros / 1000000
            Next
        End If
    End If
End Sub
```

The code below shows the part of this procedure which is purely the ‘timer’ part:

```
If (Now - TTimer).TotalMilliseconds > 25 Then
    TTimer = Now
    'Every 25 milliseconds (ish)
End If
```

TTimer is a date variable which is essential for this timer to work. The Now function returns the current time as a date variable. The time difference between the saved time in TTimer and the current time is evaluated. If the total number of milliseconds between the two dates is greater than a specified amount (in this case, 25ms) the timer performs a tick. The code to be run every ‘tick’ goes inside the If statement. When a tick is performed, the TTimer variable is reset to the current time. This restarts the process, and another tick would be performed at least 25ms later.

Projectile Motion Simulation

I also created an algorithm in the design section for the main process of the Projectile Motion Simulation. This algorithm is repeated below and is for updating the position of the ball over time, taking into account collisions.

1. Every 25 milliseconds (using a timer, defined on page 20):
2. Calculate, in metres and using the equations below, the expected position of the ball as if no collision were to happen
   a. $p_x = p_{x0} + u_x t$
   b. $p_y = p_{y0} + u_y t - 0.5 * 9.8t^2$
3. Check horizontal collisions and in the case of a collision, update velocities and positions appropriately. If no collision is found, update the ball's X coordinate with the expected X coordinate
   a. Check if the ball would have collided with the left edge of the screen
   b. Check if the ball would have collided with the wall
   c. Check if the ball would have collided with the right edge of the screen, past the wall
4. Check vertical collisions and in the case of a collision, update velocities and positions appropriately. If no collision is found, update the ball's Y coordinate with the expected Y coordinate
   a. Check if the ball reaches the top edge of the screen. In this case, the displayed ball would not move up any further, but the theoretical one would
   b. Check if the ball collides with the ground
5. Increase the elapsed time of the Simulation by 1 microsecond ($1x10^{-6}$ seconds)
6. Repeat steps 2 to 5 10,000 times

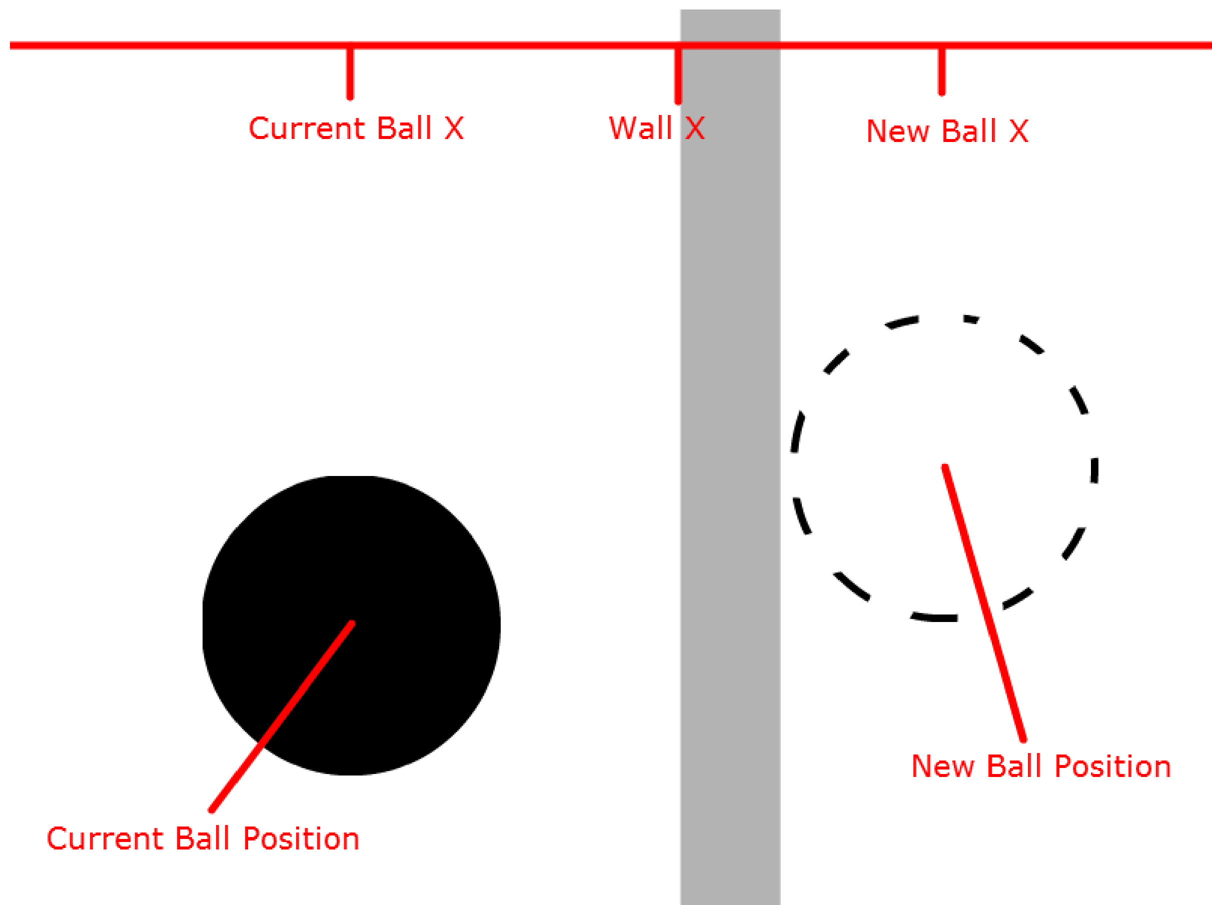
The code below shows the Update procedure of the Projectile Motion Simulation:

```
    Public Overrides Sub Update()
        If Enabled = True Then
            If (Now - TTimer).TotalMilliseconds > 25 Then
                TTimer = Now

                Dim NewBallX, NewBallY As Double

                'Every 25 milliseconds (ish)

                'Gradually increase the time variable
                'Calculate the expected position as if no collision happens, then
                'see if there should be a collision
                For i = 1 To 10000
                    NewBallX = InitialBallS.X + Pixels(FiringV.X * T)
                    NewBallY = InitialBallS.Y + Pixels(FiringV.Y * T + 0.5 * g * T ^
2)

                    'Update ball's velocity
                    BallV.Y = FiringV.Y + g * T

                    If Abs(BallV.X) > 0 Then
```

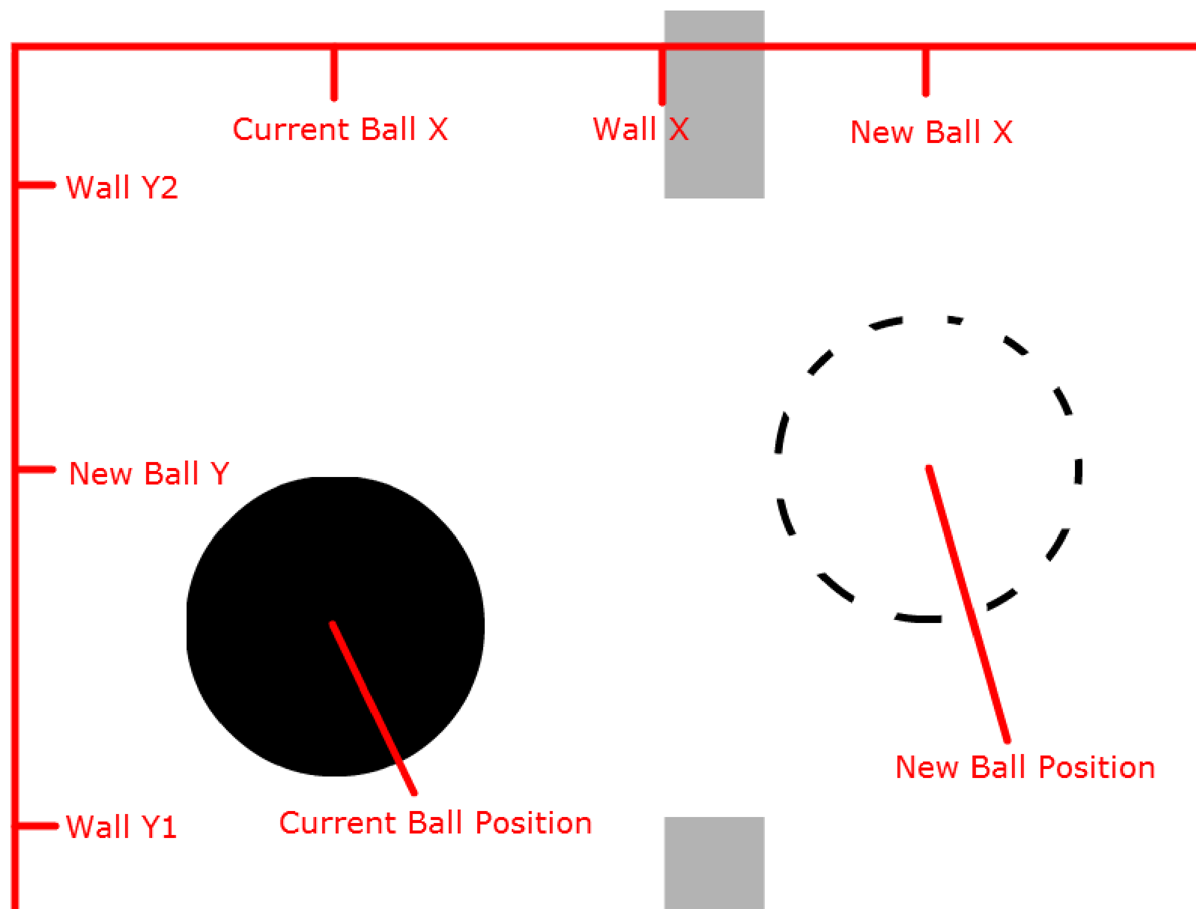
```
                        If NewBallX < 0 Or (BallS.X <= WallX And NewBallX >= WallX And
(GroundY - NewBallY <= WallY1 Or GroundY - NewBallY >= WallY2)) Or NewBallX >
Size.Width - BallRadius Then
                            If NewBallX < 0 Then
                                'Ball reaches left edge
                                BallS.X = 0
                                BallV.X = 0
                            ElseIf NewBallX >= Size.Width - BallRadius Then
                                'Ball has gone through wall and reaches right edge
                                BallS.X = Size.Width - BallRadius
                                BallV.X = 0
                            ElseIf NewBallX > WallX Then
                                'Ball hits wall
                                BallS.X = WallX
                                BallV.X = 0
                            End If
                        Else
                            'No special cases, free space ahead
                            BallS.X = NewBallX
                        End If
                    End If

                    If Abs(BallV.Y) > 0 Then
                        If NewBallY < 0 Or NewBallY > GroundY - 2 * BallRadius Then
                            If NewBallY > GroundY - 2 * BallRadius Then
                                'Ball reaches top edge
                                BallOutOfTop = True
                                AmountOutOfTop = Round(Metres(NewBallY - (GroundY - 2
* BallRadius)), 2)
                                BallS.Y = GroundY - 2 * BallRadius
                                BallV.Y = 0
                            ElseIf NewBallY < 0 Then
                                'Ball Reaches ground
                                BallS.Y = 0
                                BallV.Y = 0
                                BallV.X = 0
                                Finished = True
                            End If
                        Else
                            'No special cases, free space ahead
                            BallS.Y = NewBallY
                            BallOutOfTop = False
                        End If
                    End If

                    'Increase time by 1ms
                    Tmicros += 1
                    T = Tmicros / 1000000
                Next
            End If
            '0.01s of simulation has passed
        End If
    End Sub
```

The outer IF statement is to check if the Simulation is enabled. If the user has paused the Simulation, this would be set to false. The IF inside that can be recognised as the 25 millisecond timer. The equations for calculating the new position of the ball can be found just inside the FOR loop. The rest of the code inside this loop checks for collisions with the ball's environment, before finally incrementing the time (T) variable by 1 millionth of a second.

The diagram below illustrates simply how the program checks for a collision with the wall. The current ball is solid black, and the theoretical new ball is white with a dashed black border. The wall is shown by the vertical grey bar. In this situation, the program needs to work out that the ball has collided with the wall. The program can't just check to see if the new ball X coordinate is more than the wall's X coordinate, because the ball could have started beyond the wall in the first place. Therefore, the program also checks that the current ball X coordinate is less than the wall's. This would mean that, in one microsecond the ball starts to the left of the wall, and wants to finish to the right of it.



The second diagram (pictured below) builds on the collision system by adding in the factor of the gap in the wall. This means that both horizontal and vertical positions need to be evaluated. The same idea for the X coordinates is still the same, but this time, the program checks that the new ball Y coordinate is between the two Y coordinates for the gap in the wall before letting the ball through the gap. If the new ball's Y coordinate is above Wall Y2, or below Wall Y1, then the ball should collide with the wall and not get through.



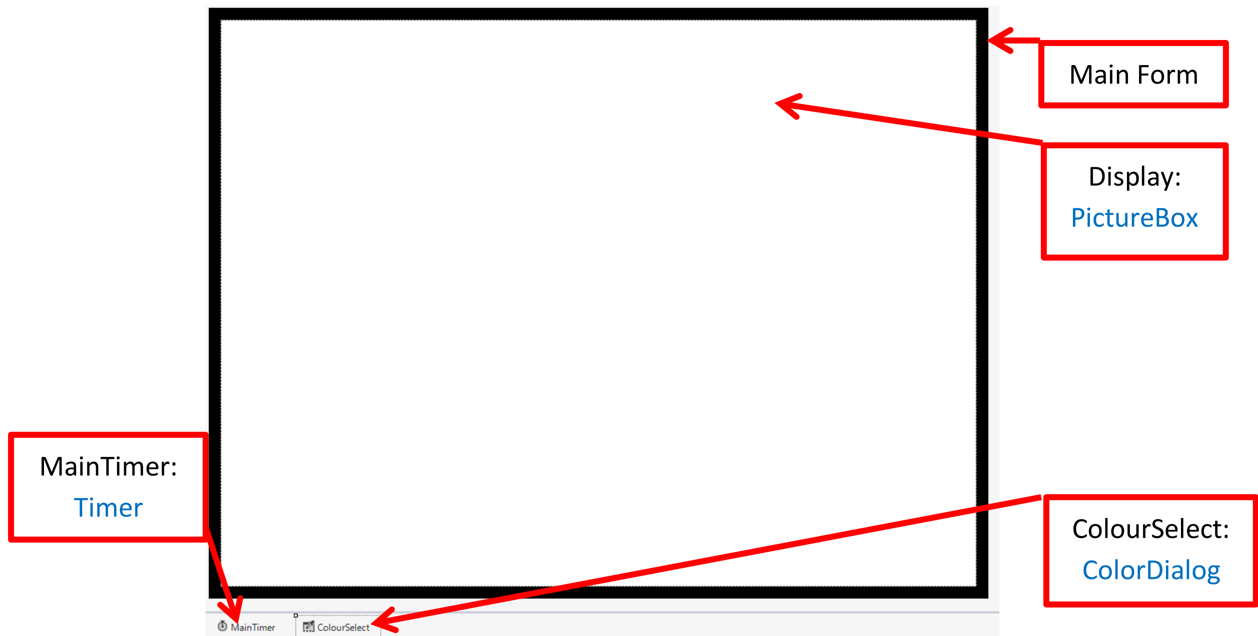
The FOR loop repeats 10,000 times. This means that the simulation time increases by 0.000001 seconds, 10,000 times (or 0.01 seconds) every tick of the timer (about 25 milliseconds of real time). The reason for calculating the ball's position at such small time intervals is to increase the precision of the Simulation. By trying out different numbers of loops and different time increments, I think that the values I have decided on maximise precision of the calculated values (more than enough for giving values to 2 decimal places, which is what is needed), whilst not slowing the program down noticeably due to too many calculations for the machine.

Code

The following section contains all of the code for my program, split into the various modules of classes and forms. There is a description of each of these before the code associated with it.

Forms

Although my program is a Windows Forms Application, is consists of only one form. Since the modular structure of my program is only classes, and my graphics system is drawing to an image at run-time, I have had no need to design separate forms for each screen. The Main form consists of a PictureBox object called Display, onto which the display of the program is drawn to each cycle, and a timer object called MainTimer, which has a minimal interval and acts as the main clock of the program. There is also a ColorDialog object, which is used for changing the main border colour in the Settings Screen.



Above is what my Main form looks like in design view. It looks quite boring because most of the space is taken up by the main PictureBox, Display. What looks like the black border around Display is actually the form background itself.

Below is the code for the Main form:

```
Imports System.IO

Public Class Main
    Public BMP As New Bitmap(960, 720)
    Public GFX As Graphics = Graphics.FromImage(BMP)

    Public Rand As New Random

    Public ScreenManager As New ScreenManager
    Public KeysDown As New List(Of Integer)
    Public KeysUp As New List(Of Integer)
    Public MouseButtonsDown As New List(Of MouseButtonInfo)
    Public MouseButtonsUp As New List(Of MouseButtonInfo)

    Public ProgramPause As Boolean = False
    Public DebugToggling As Boolean = False

    Private DragFormPos As Point

    Public CurrentUser As String = ""

    Public Structure MouseButtonInfo
        Dim Button As System.Windows.Forms.MouseButtons
        Dim Location As Point
    End Structure

    'FONTS
    Public Shared Georgia_32 As New Font("Georgia", 32)
    Public Shared Georgia_20 As New Font("Georgia", 20)
    Public Shared Georgia_20_Bold As New Font("Georgia", 20, FontStyle.Bold)
    Public Shared Arial_8 As New Font("Arial", 8)
```

```
    Public Shared Arial_10 As New Font("Arial", 10)
    Public Shared Arial_12_Bold As New Font("Arial", 12, FontStyle.Bold)
    Public Shared Arial_15 As New Font("Arial", 15)
    Public Shared Arial_15_Bold As New Font("Arial", 15, FontStyle.Bold)
    Public Shared Arial_20 As New Font("Arial", 20)
    Public Shared Arial_20_Bold As New Font("Arial", 20, FontStyle.Bold)
    Public Shared Arial_30_Bold As New Font("Arial", 30, FontStyle.Bold)
    Public Shared Arial_50_Bold As New Font("Arial", 50, FontStyle.Bold)
    Public Shared Impact_18 As New Font("Impact", 18)
    Public Shared Impact_32 As New Font("Impact", 32)

    Public Sub SelectColour(ByRef Button As TextButton)
        'Opens a colour selector and changes the colours of a button
        ProgramPause = True
        ColourSelect.ShowDialog()
        ProgramPause = False
        Button.DefaultBackColour = ColourSelect.Color
        Button.HoverBackColour = ColourSelect.Color
        Button.MouseDownBackColour = ColourSelect.Color
        Button.MouseDownBorderColour = ColourSelect.Color
        Button.HoverBorderColour = ColourSelect.Color
    End Sub

    Public Function AutoFitText(ByVal X As Integer, ByVal Y As Integer, ByVal MaxWidth
As Integer, ByVal Font As Font, ByVal Text As String, Optional ByVal Display As
Boolean = True)
        Dim Words As New List(Of String)
        Dim Lines As New List(Of String)
        Dim Current As String = ""
        Dim LineHeight As Integer = GFX.MeasureString("W", Font).Height

        'Split into words
        For i = 0 To Text.Length - 1
            If Text(i) = " " Then
                Words.Add(Current)
                Current = ""
            ElseIf i = Text.Length - 1 Then
                Current &= Text(i)
                Words.Add(Current)
                Current = ""
            Else
                Current &= Text(i)
            End If
        Next

        Current = ""
        'Split into lines
        For Each Word In Words
            If GFX.MeasureString(Current & Word, Font).Width <= MaxWidth Then
                Current &= Word & " "
            Else
                Lines.Add(Current)
                Current = Word & " "
            End If
        Next
        Lines.Add(Current)

        If Display = True Then
            'Draw Lines
            For i = 0 To Lines.Count - 1
                GFX.DrawString(Lines(i), Font, Brushes.Black, X, Y + i * LineHeight)
```

```
        Next
    End If

    Return Y + LineHeight * (Lines.Count + 0.5)
End Function

Public Sub MessageBox(ByVal Text As String)
    ProgramPause = True
    MsgBox(Text)
    ProgramPause = False
End Sub

Public Function EncryptString(ByVal PlainText As String) As String
    Dim asciied, reversed, split, reversed2 As String
    Dim NumOfLoops As Integer = Rand.Next(1, 4 + 1)

    If PlainText = "" Then
        Return ""
    End If

    Try
        For times = 1 To NumOfLoops

            asciied = ""
            reversed = ""
            split = ""
            reversed2 = ""

            'MOVE ALL CHARACTERS 2 ASCII CODES UP
            For i = 1 To Len(PlainText)
                asciied = asciied & Chr(Asc(Mid(PlainText, i, 1)) + 2)
            Next

            'REVERSE THE ORDER OF THE CHARACTERS IN THE STRING
            reversed = StrReverse(asciied)

            'SPLIT THE STRING SO IT AS ALL OF THE EVENLY INDEXED CHARACTERS....
            For i = 2 To Len(reversed) Step 2
                split = split & Mid(reversed, i, 1)
            Next

            '....FOLLOWED BY THE ODDLY INDEXED CHARACTERS
            For i = 1 To Len(reversed) Step 2
                split = split & Mid(reversed, i, 1)
            Next

            'REVERSE THE STRING AGAIN
            reversed2 = StrReverse(split)

            PlainText = reversed2
        Next

        'PUT 2 * NumOfLoops ONTO THE BEGINNING OF THE STRING
        PlainText = 2 * NumOfLoops & reversed2

        'REPEAT ONE ITERATION OF THE ENCRYPTION
        asciied = ""
        reversed = ""
        split = ""
        reversed2 = ""
```

```
            For i = 1 To Len(PlainText)
                asciied = asciied & Chr(Asc(Mid(PlainText, i, 1)) + 2)
            Next

            reversed = StrReverse(asciied)

            For i = 2 To Len(reversed) Step 2
                split = split & Mid(reversed, i, 1)
            Next

            For i = 1 To Len(reversed) Step 2
                split = split & Mid(reversed, i, 1)
            Next

            reversed2 = StrReverse(split)

            Return reversed2
        Catch ex As Exception
            MessageBox("File Writing Error: Error with encryption.")
            Return ""
        End Try
    End Function
    Public Function DecryptString(ByVal CipherText As String) As String
        Dim reversed, evenlySplit, oddlySplit, finalFused, reversed2, asciied As
String
        Dim NumOfLoops As Integer

        If CipherText = "" Then
            Return ""
        End If

        Try
            reversed = ""
            evenlySplit = ""
            oddlySplit = ""
            finalFused = ""
            reversed2 = ""
            asciied = ""

            'REVERSE THE STRING
            reversed = StrReverse(CipherText)

            'SPLIT THE STRING INTO TWO HALVES
            For i = 1 To Int(Len(reversed) / 2)
                evenlySplit &= Mid(reversed, i, 1)
            Next
            For i = Int(Len(reversed) / 2) + 1 To Len(reversed)
                oddlySplit &= Mid(reversed, i, 1)
            Next

            'RECONSTRUCT THE STRING, BY TAKING A CHARACTER FROM THE SECOND HALF,
            'THEN THE FIRST HALF, THEN THE SECOND HALF ETC.
            For i = 1 To Len(evenlySplit) + Len(oddlySplit)
                finalFused &= Mid(oddlySplit, i, 1) & Mid(evenlySplit, i, 1)
            Next

            'REVERSE THE STRING AGAIN
            reversed2 = StrReverse(finalFused)

            'MOVE ALL CHARACTERS 2 ASCII CODES DOWN
            For i = 1 To Len(reversed2)
```

```
                asciied &= Chr(Asc(Mid(reversed2, i, 1)) - 2)
            Next

            CipherText = asciied

            'TAKE THE FIRST CHARACTER AND DIVIDE THIS BY 2. THIS IS THE NumOfLoops
            'GENERATED AT ENCRYPTION
            NumOfLoops = Mid(CipherText, 1, 1)
            NumOfLoops = NumOfLoops / 2
            CipherText = CipherText.Substring(1, Len(CipherText) - 1)


            For times = 1 To NumOfLoops
                'REPEAT THE DECRYPTION
                reversed = ""
                evenlySplit = ""
                oddlySplit = ""
                finalFused = ""
                reversed2 = ""
                asciied = ""

                reversed = StrReverse(CipherText)

                For i = 1 To Int(Len(reversed) / 2)
                    evenlySplit &= Mid(reversed, i, 1)
                Next
                For i = Int(Len(reversed) / 2) + 1 To Len(reversed)
                    oddlySplit &= Mid(reversed, i, 1)
                Next

                For i = 1 To Len(evenlySplit) + Len(oddlySplit)
                    finalFused &= Mid(oddlySplit, i, 1) & Mid(evenlySplit, i, 1)
                Next

                reversed2 = StrReverse(finalFused)

                For i = 1 To Len(reversed2)
                    asciied &= Chr(Asc(Mid(reversed2, i, 1)) - 2)
                Next

                CipherText = asciied

            Next

            Return asciied
        Catch ex As Exception
            MessageBox("File Reading Error: Error with decryption.")
            Return ""
        End Try
    End Function

    Public Function Deg(ByVal Rad As Single) As Single
        Deg = Rad
        Deg *= 180
        Deg /= Math.PI
    End Function
    Public Function Rad(ByVal Deg As Single) As Single
        Rad = Deg
        Rad *= Math.PI
        Rad /= 180
    End Function
```

```
    Private Sub Main_MouseDown(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Display.MouseDown
        Dim MBD As New MouseButtonInfo
        MBD.Button = e.Button
        MBD.Location = New Point(e.Location.X, e.Location.Y)

        MouseButtonsDown.Add(MBD)
    End Sub
    Private Sub Main_MouseUp(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Display.MouseUp
        Dim MBD As New MouseButtonInfo
        MBD.Button = e.Button
        MBD.Location = New Point(e.Location.X, e.Location.Y)

        MouseButtonsUp.Add(MBD)
    End Sub

    Private Sub Main_KeyDown(sender As Object, e As System.Windows.Forms.KeyEventArgs)
Handles Me.KeyDown
        KeysDown.Add(e.KeyValue)
    End Sub
    Private Sub Main_KeyUp(sender As Object, e As System.Windows.Forms.KeyEventArgs)
Handles Me.KeyUp
        KeysUp.Add(e.KeyValue)

        ScreenManager.SetDebugOutputMessage(e.KeyValue)
    End Sub

    Private Sub Main_Load(sender As System.Object, e As System.EventArgs) Handles
MyBase.Load
        If Not Directory.Exists(My.Computer.FileSystem.SpecialDirectories.MyDocuments
& "\Mechanics Simulation\Users") Then

Directory.CreateDirectory(My.Computer.FileSystem.SpecialDirectories.MyDocuments &
"\Mechanics Simulation\Users")
        End If
        Environment.CurrentDirectory =
My.Computer.FileSystem.SpecialDirectories.MyDocuments & "\Mechanics Simulation\Users"

        DragFormPos = Nothing

        'STARTING SCREENS
        ScreenManager.AddScreen(New Title)
        ScreenManager.AddScreen(New SimulationButton)
        ScreenManager.AddScreen(New TestButton)
        ScreenManager.AddScreen(New MyProgressButton)
    End Sub

    Private Sub MainTimer_Tick(sender As System.Object, e As System.EventArgs) Handles
MainTimer.Tick
        If ProgramPause = False Then
            'UPDATE SCREENS
            ScreenManager.Update()

            'DRAW
            GFX.Clear(Color.White)

            ScreenManager.Draw()

            Display.Image = BMP
```

```
        End If
    End Sub

    Private Sub Form_MouseDown(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
        Focus()
        DragFormPos = New Point(e.X, e.Y)
    End Sub
    Private Sub Form_MouseMove(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseMove
        If Not DragFormPos = Nothing Then
            SetDesktopLocation(Windows.Forms.Form.MousePosition.X - DragFormPos.X,
Windows.Forms.Form.MousePosition.Y - DragFormPos.Y)
        End If
    End Sub
    Private Sub Form_MouseUp(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseUp
        DragFormPos = Nothing
    End Sub
End Class
```

Each procedure, function and variable in the Main form is listed below, along with a brief description of each one.

Main Form Variables		
Name	**Type**	**Description**
BMP	Bitmap	The image to which all drawing is done to
GFX	Graphics	The object which handles the drawing to BMP, which has many drawing methods, such as DrawLine or DrawString
Display	PictureBox	The only visible control on the main form. Display's image is set to BMP every tick of MainTimer
MainTimer	Timer	The timer with a minimum time interval which makes the Screen Manager update and draw all enabled screens
Rand	Random	Used for generating random integers
ScreenManager	ScreenManager	Object which manages all screens in the program
KeysDown	List(Of Integer)	Saves the ASCII values for all keys which are pressed. The Form's KeyDown event will add the pressed key to this list. This list is cleared at the end of every MainTimer tick
KeysUp	List(Of Integer)	Saves the ASCII values for all keys which are released. The Form's KeyDown event will add the released key to this list. This list is cleared at the end of every MainTimer tick
MouseButtonsDown	List(Of MouseButtonInfo)	Saves the location and button value whenever a mouse button is pressed. This list is cleared at the end of every MainTimer tick
MouseButtonsUp	List(Of MouseButtonInfo)	Saves the location and button value whenever a mouse button is released. This list is cleared at the end of every MainTimer tick

ProgramPause	Boolean	Used to indicate whether the whole program needs to be paused. If this holds true, the Screen Manager won't be used every tick of MainTimer
DebugToggling	Boolean	Whether or not the Debug screen can be toggle on or off
DragFormPos	Point	Used in the process of dragging the Main Window Around (See page 81)
CurrentUser	String	Saves the User Name of the currently logged in User
Fonts*	Font	*I have defined 14 different fonts for my program, and these can be seen in the code for the Main form above. The fonts may differ in size, style (i.e. Regular, or Bold) and Font Name (e.g. Georgia, Arial, Impact)

Main Form Procedures and Functions		
Name	**Returning Variables**	**Description**
SelectColour	None	Opens a colour selector and changes the colours of a button
AutoFitText	Y-value of the bottom of the drawn block	Splits a string of text into lines so that it can be fit into a given width
MessageBox	None	Pauses the program, then calls the standard MsgBox procedure
EncryptString	Encrypted String	Encrypts a string
DecryptString	Decrypted String	Decrypts a string
Deg	Angle in Degrees	Converts radians into degrees
Rad	Angle in Radians	Converts degrees into radians
Main_MouseDown	None	Event which triggers when a mouse button is pressed down
Main_MouseUp	None	Event which triggers when a mouse button is released
Main_KeyDown	None	Event which triggers when a key is pressed down
Main_KeyUp	None	Event which triggers when a key is released
Main_Load	None	Event which triggers when the program starts running. This instantiates the Screen Manager
MainTimer_Tick	None	Event which triggers each main cycle of the program
Form_MouseDown	None	Event which triggers when a mouse button is held down on the border of the form
Form_MouseMove	None	Event which triggers when the mouse moves
Form_MouseUp	None	Event which triggers when a mouse button is released on the border of the form

Classes

In the design section, I have explained how classes were to be used as the main form of modular structure in my program, including the main attributes and methods for each one and how they relate to each other by inheritance, and almost all of the code in my project comprises of classes which I have created. I have implemented all the classes that I designed mostly how I planned, so there is no need to provide a method/attribute list for the classes. See page 23 onwards in the design section for the lists of methods and attributes for my classes.

The classes in my program either have the purpose of a Screen, or a Tool (with the exception of the ScreenManager). Screens include all of the separate views of the program, and Tools include components of screens, such as buttons, Text Boxes and Menus.

ScreenManager

This class is probably the most important for the program to work. Simply, it handles all of the currently enabled screens and allowing the right ones to Update, Handle Input and Draw to the User's screen. All purposes of this class are explained in detail in the Design section (on page 34) and in the System Maintenance section (on page 81 onwards). At the top of the Screen Manager code file is the ScreenState enumeration which is used by all screens.

```
Public Enum ScreenState
    Active 'Draws and accepts input
    Hidden 'Doesn't draw but accepts input
    NoInput 'Draws but doesn't accept input
    Sleep 'Doesn't draw and doesn't accept input and doesnt update
    ShutDown 'Will be removed on next cycle
End Enum

Public Class ScreenManager
    Private Shared Screens As New List(Of BaseScreen)
    Private Shared NewScreens As New List(Of BaseScreen)
    Private DebugScreen As New Debug

    Public Sub New()
        DebugScreen.Output = "Output: "
        AddScreen(DebugScreen)
    End Sub

    Public Sub Update()
        DebugScreen.ActiveScreens = "Active Screens: "
        DebugScreen.HiddenScreens = "Hidden Screens: "
        DebugScreen.NoInputScreens = "No Input Screens: "
        DebugScreen.SleepScreens = "Sleep Screens: "

        ' GENERATE LIST OF DEAD SCREENS FOR REMOVAL
        Dim RemoveScreens As New List(Of BaseScreen)

        For Each FoundScreen As BaseScreen In Screens
            If FoundScreen.State = ScreenState.ShutDown Then
                RemoveScreens.Add(FoundScreen)
            Else
                'Add the names of alive screens to debug info lists
                Select Case FoundScreen.State
                    Case ScreenState.Active
                        DebugScreen.ActiveScreens &= FoundScreen.Name & ", "
                    Case ScreenState.Hidden
```

```
                        DebugScreen.HiddenScreens &= FoundScreen.Name & ", "
                    Case ScreenState.NoInput
                        DebugScreen.NoInputScreens &= FoundScreen.Name & ", "
                    Case ScreenState.Sleep
                        DebugScreen.SleepScreens &= FoundScreen.Name & ", "
                End Select
            End If
        Next

        ' REMOVE DEAD SCREENS
        For Each FoundScreen As BaseScreen In RemoveScreens
            Screens.Remove(FoundScreen)
        Next

        ' ADD NEW SCREENS TO MAIN LIST FROM THE NEW SCREENS LIST
        For Each FoundScreen As BaseScreen In NewScreens
            Screens.Add(FoundScreen)
        Next
        NewScreens.Clear()

        ' RESET DEBUG SCREEN TO END (TOP) OF LIST
        Screens.Remove(DebugScreen)
        Screens.Add(DebugScreen)

        ' CALL INPUT AND UPDATE PROCEDURES FOR APPLICABLE SCREENS
        For Each FoundScreen As BaseScreen In Screens
            If FoundScreen.State <> ScreenState.Sleep Then
                If Main.Focused And (FoundScreen.State = ScreenState.Active Or
FoundScreen.State = ScreenState.Hidden) Then
                    FoundScreen.HandleInput()
                End If
                FoundScreen.Update()
            End If
        Next

        ' CLEAR MOUSE AND KEYBOARD INPUT LISTS
        Main.KeysDown.Clear()
        Main.KeysUp.Clear()
        Main.MouseButtonsDown.Clear()
        Main.MouseButtonsUp.Clear()
    End Sub

    Public Sub Draw()
        ' CALL DRAW PROCEDURE FOR APPLICABLE SCREENS
        For Each FoundScreen As BaseScreen In Screens
            If FoundScreen.State = ScreenState.Active Or FoundScreen.State =
ScreenState.NoInput Then
                FoundScreen.Draw()
            End If
        Next
    End Sub

    Public Sub SetDebugOutputMessage(ByVal message As String)
        'Sets one of the fields shown by the Debug screen to a value
        DebugScreen.Output = "Output: " & message
    End Sub

    Public Shared Sub AddScreen(ByVal screen As BaseScreen)
        NewScreens.Add(screen)
    End Sub
```

```
    Public Shared Sub SetScreenState(ByVal screen As String, ByVal state As
ScreenState)
        For Each FoundScreen As BaseScreen In Screens
            If FoundScreen.Name = screen Then
                FoundScreen.State = state
                Exit For
            End If
        Next
    End Sub

    Public Shared Sub UnloadScreen(ByVal screen As String)
        'SET THE DESIRED SCREEN'S STATE TO SHUTDOWN
        For Each FoundScreen As BaseScreen In Screens
            If FoundScreen.Name = screen Then
                FoundScreen.Unload()
                Exit For
            End If
        Next
    End Sub
End Class
```

BaseScreen

This is the Parent Class which all screens inherit and is essential so that the Screen Manager can reference all screens by the same type.

```
Public Class BaseScreen
    Public Name As String = ""
    Public State As ScreenState = ScreenState.Active
    Public Location As Point

    Public Overridable Sub HandleInput()
        'Instructions for the screen taking in and processing user input
    End Sub

    Public Overridable Sub Update()
        'Instructions for updating screen variables
    End Sub

    Public Overridable Sub Draw()
        'Instructions for drawing screen contents
    End Sub

    Public Overridable Sub Unload()
        State = ScreenState.ShutDown
    End Sub
End Class
```

Debug

The Debug screen is an unusual one in that it is always enabled, just hidden by default. It is purely designed for making the development of screens easier, since it displays important information about the currently enabled screens. Instructions for how to use it can be found on page 84.

FPS: 65
Active Screens: Settings, Debug
Hidden Screens:
No Input Screens:
Sleep Screens:
Mouse Position: X:1, Y:-4
Output: 112

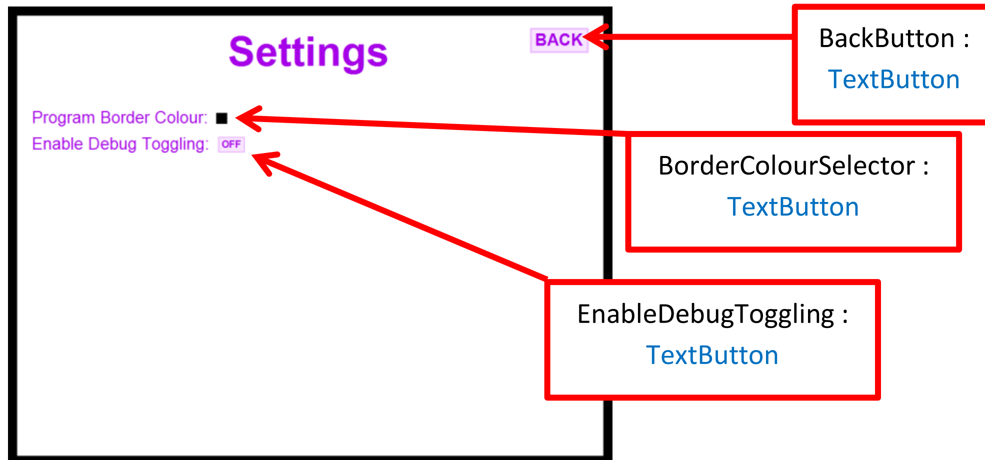
```
Public Class Debug
    Inherits BaseScreen

    Public ActiveScreens As String = ""
    Public HiddenScreens As String = ""
    Public NoInputScreens As String = ""
    Public SleepScreens As String = ""
    Public Output As String = ""
    Public MouseLocation As New Point(0, 0)

    Private fpsCounter As Integer
    Private fpsTimer As Date
    Private fpsText As String = ""

    Private BGRect As Rectangle

    Public Sub New()
        'This screen is hidden by default
        Name = "Debug"
        State = ScreenState.Hidden
    End Sub

    Public Overrides Sub HandleInput()
        'The F1 key toggles the visibility of the screen
        If Main.KeysDown.Contains(112) And Main.DebugToggling Then
            If State = ScreenState.Active Then
                State = ScreenState.Hidden
            ElseIf State = ScreenState.Hidden Then
                State = ScreenState.Active
            End If
        ElseIf Main.DebugToggling = False Then
            State = ScreenState.Hidden
        End If
    End Sub

    Public Overrides Sub Update()
        'Remove the final comma at the end of all applicable data strings
        If ActiveScreens.Length > 16 Then
            ActiveScreens = ActiveScreens.Substring(0, ActiveScreens.Length - 2)
        End If
        If HiddenScreens.Length > 16 Then
            HiddenScreens = HiddenScreens.Substring(0, HiddenScreens.Length - 2)
        End If
        If NoInputScreens.Length > 18 Then
            NoInputScreens = NoInputScreens.Substring(0, NoInputScreens.Length - 2)
        End If
        If SleepScreens.Length > 15 Then
```

```
            SleepScreens = SleepScreens.Substring(0, SleepScreens.Length - 2)
        End If
        If SleepScreens.Length > 15 Then
            SleepScreens = SleepScreens.Substring(0, SleepScreens.Length - 2)
        End If

        'Update the size of the screen's background using the data
        Dim txtWidth As Integer = 0
        Dim txtHeight As Integer = 0

        If Main.GFX.MeasureString(ActiveScreens, Main.Arial_8).Width > txtWidth Then
            txtWidth = Main.GFX.MeasureString(ActiveScreens, Main.Arial_8).Width
        End If
        If Main.GFX.MeasureString(HiddenScreens, Main.Arial_8).Width > txtWidth Then
            txtWidth = Main.GFX.MeasureString(HiddenScreens, Main.Arial_8).Width
        End If
        If Main.GFX.MeasureString(NoInputScreens, Main.Arial_8).Width > txtWidth Then
            txtWidth = Main.GFX.MeasureString(NoInputScreens, Main.Arial_8).Width
        End If
        If Main.GFX.MeasureString(SleepScreens, Main.Arial_8).Width > txtWidth Then
            txtWidth = Main.GFX.MeasureString(SleepScreens, Main.Arial_8).Width
        End If
        If Main.GFX.MeasureString(Output, Main.Arial_8).Width > txtWidth Then
            txtWidth = Main.GFX.MeasureString(Output, Main.Arial_8).Width
        End If
        If Main.GFX.MeasureString("Mouse Position: X:0000, Y:0000",
Main.Arial_8).Width > txtWidth Then
            txtWidth = Main.GFX.MeasureString("Mouse Position: X:0000, Y:0000",
Main.Arial_8).Width
        End If
        txtHeight = Main.GFX.MeasureString(ActiveScreens, Main.Arial_8).Height * 7
        BGRect = New Rectangle(0, 0, txtWidth + 20, txtHeight + 20)

        'Timer for updating the FPS counter
        If (Now - fpsTimer).TotalMilliseconds > 1000 Then
            fpsTimer = Now
            fpsText = "FPS: " & fpsCounter
            fpsCounter = 1
        Else
            fpsCounter += 1
        End If
    End Sub

    Public Overrides Sub Draw()
        MouseLocation = New Point(Windows.Forms.Form.MousePosition.X - Main.Left - 15,
Windows.Forms.Form.MousePosition.Y - Main.Top - 15)

        Main.GFX.FillRectangle(New SolidBrush(Color.FromArgb(35 / 100 * 255, 0, 0,
0)), BGRect)
        Main.GFX.DrawString(fpsText, Main.Arial_8, Brushes.White, New Point(10, 10))
        Main.GFX.DrawString(ActiveScreens, Main.Arial_8, Brushes.White, New Point(10,
22))
        Main.GFX.DrawString(HiddenScreens, Main.Arial_8, Brushes.White, New Point(10,
34))
        Main.GFX.DrawString(NoInputScreens, Main.Arial_8, Brushes.White, New Point(10,
46))
        Main.GFX.DrawString(SleepScreens, Main.Arial_8, Brushes.White, New Point(10,
58))
        Main.GFX.DrawString("Mouse Position: X:" & MouseLocation.X & ", Y:" &
MouseLocation.Y, Main.Arial_8, Brushes.White, New Point(10, 70))
        Main.GFX.DrawString(Output, Main.Arial_8, Brushes.White, New Point(10, 82))
```

```
    End Sub
End Class
```

Settings

This screen holds the program settings and can be accessed from almost all screens. It saves copies of the previous screens, so that it knows which screens to load again when the Back button is pressed.




```
Public Class Settings
    Inherits BaseScreen

    Private PreviousScreens As New List(Of BaseScreen)

    Private BackButton As New TextButton("BACK", Main.Arial_20_Bold,
ProgramSection.Other, New Point(840, 20), -1, -1, 3, 1)
    Private BorderColourSelector, EnableDebugToggling As TextButton

    Public Sub New(ByVal InputPreviousScreen() As BaseScreen)
        Dim TempY As Integer = 150

        Name = "Settings"
        Location = New Point(0, 0)
        State = ScreenState.Active

        'Save the previous screens, so the back button knows where to point to
        'Previous screens is an array because there may have been more than one
        'enabled screen before going to the settings menu (e.g on the Title Screen)
        For Each Screen In InputPreviousScreen
            PreviousScreens.Add(Screen)
        Next

        'Set up buttons
        BorderColourSelector = New TextButton("", Main.Arial_10, Main.BackColor,
Main.BackColor, Color.White, Main.BackColor, Main.BackColor, Main.BackColor,
Main.BackColor, Main.BackColor, Main.BackColor, New
Point(Main.GFX.MeasureString("Program Border Colour: ", Main.Arial_20).Width + 25,
TempY + 8), 20, 20, 1)
        TempY += Main.GFX.MeasureString("Program Border Colour: ",
Main.Arial_20).Height + 10
        EnableDebugToggling = New TextButton("OFF", Main.Arial_12_Bold,
ProgramSection.Other, New Point(Main.GFX.MeasureString("Enable Debug Toggling: ",
Main.Arial_20).Width + 25, TempY + 5), -1, -1, 2, 1)
        If Main.DebugToggling Then EnableDebugToggling.Text = "ON"
```

```
    End Sub

    Public Overrides Sub HandleInput()
        If BackButton.Clicked = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            'Reload all previous screens which were saved
            For Each Screen In PreviousScreens
                ScreenManager.AddScreen(Screen)
            Next
        End If

        If BorderColourSelector.Clicked() = "Clicked" Then
            Main.SelectColour(BorderColourSelector)
            Main.BackColor = BorderColourSelector.DefaultBackColour
        End If
        If EnableDebugToggling.Clicked = "Clicked" Then
            If Main.DebugToggling Then
                Main.DebugToggling = False
                EnableDebugToggling.Text = "OFF"
            Else
                Main.DebugToggling = True
                EnableDebugToggling.Text = "ON"
            End If
        End If
    End Sub

    Public Overrides Sub Draw()
        Dim TempY As Integer = 150
        'Title
        Main.GFX.DrawString("Settings", Main.Arial_50_Bold, New
SolidBrush(Color.FromArgb(166, 0, 232)), 480 - Main.GFX.MeasureString("Settings",
Main.Arial_50_Bold).Width \ 2, 20)
        BackButton.Draw()

        'Border Colour
        Main.GFX.DrawString("Program Border Colour: ", Main.Arial_20, New
SolidBrush(Color.FromArgb(166, 0, 232)), 20, TempY)
        TempY += Main.GFX.MeasureString("Program Border Colour: ",
Main.Arial_20).Height + 10
        BorderColourSelector.Draw()
        'Debug Toggling
        Main.GFX.DrawString("Enable Debug Toggling: ", Main.Arial_20, New
SolidBrush(Color.FromArgb(166, 0, 232)), 20, TempY)
        TempY += Main.GFX.MeasureString("Enable Debug Toggling: ",
Main.Arial_20).Height + 10
        EnableDebugToggling.Draw()
    End Sub
End Class
```

Title

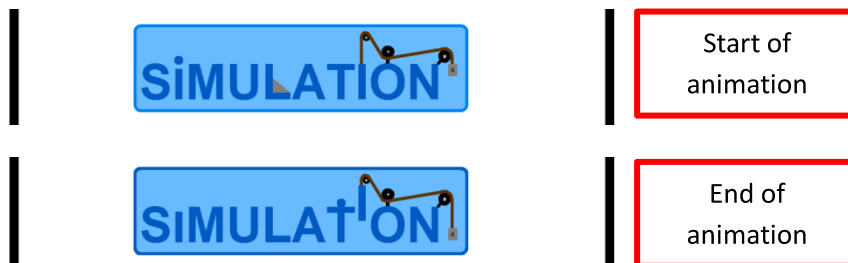
This screen is the top quarter of the Title Screen, and is responsible for the Settings/Exit menu as well as the main logo.

```
Public Class Title
```

```
    Inherits BaseScreen

    Private Size As New Point(960, 180)

    Private CornerMenu As New AlignLeftMenu(New Point(800, 10), Main.Arial_20_Bold,
Color.FromArgb(226, 153, 255), Color.FromArgb(166, 0, 232), True)

    Public Sub New()
        Name = "TitleScreenTitle"
        State = ScreenState.Active
        Location = New Point(0, 0)

        CornerMenu.AddOption("SETTINGS")
        CornerMenu.AddOption("EXIT")
    End Sub

    Public Overrides Sub HandleInput()
        Select Case CornerMenu.Update()
            Case "SETTINGS"
                ScreenManager.UnloadScreen("TitleScreenTitle")
                ScreenManager.UnloadScreen("SimulationButton")
                ScreenManager.UnloadScreen("TestButton")
                ScreenManager.UnloadScreen("MyProgressButton")
                ScreenManager.AddScreen(New Settings({New Title, New TestButton, New
SimulationButton, New MyProgressButton}))
            Case "EXIT"
                End
        End Select
    End Sub

    Public Overrides Sub Draw()
        'Title
        Main.GFX.DrawImage(My.Resources.logo, 307, 15)
        'Menu
        CornerMenu.Draw()
    End Sub
End Class
```

SimulationButton

This screen is the second quarter of the Title Screen, and is the big animated Simulation Button which, when clicked, navigates to the Simulations Menu. The animation advances when the mouse cursor is hovered over the button, and goes in reverse when the mouse cursor is not over it. The two images below show the beginning and end of the animation of this button.




```
Imports System.Math

Public Class SimulationButton
    Inherits BaseScreen
```

```
    Private MouseHover As Boolean = False

    Private AniTimer As Date
    Private AniCount As Integer = 0

    Private Size As New Point(548, 180)

    Private LPoints(5), LTriangle(2) As Point
    Private IRect As Rectangle
    Private MassRect As Rectangle
    Private ProjectileRect As Rectangle

    Public Sub New()
        Name = "SimulationButton"
        State = ScreenState.Active
        Location = New Point(188, 180)

        'INITIALISE STARTING MOVING PART SETTINGS
        'L Shape
        LPoints(0) = New Point(445, 292)
        LPoints(1) = New Point(445, 303)
        LPoints(2) = New Point(404, 303)
        LPoints(3) = New Point(404, 245)
        LPoints(4) = New Point(416, 245)
        LPoints(5) = New Point(416, 292)

        'L Triangle
        LTriangle(0) = New Point(445, 292)
        LTriangle(1) = New Point(416, 292)
        LTriangle(2) = New Point(416, 268)

        'I Shape
        IRect = New Rectangle(556, 245, 12, 58)

        'Mass
        MassRect = New Rectangle(703, 245, 15, 20)

        'Projectile
        ProjectileRect = New Rectangle(254, 231, 15, 15)
    End Sub

    Public Overrides Sub Update()
        If (Now - AniTimer).TotalMilliseconds > 25 Then
            AniTimer = Now

            'every 25 milliseconds (ish)
            'if the mouse is on the button, advance the animation
            'else make the animation go backwards

            If MouseHover = True Then
                AniCount += 2
            Else
                AniCount -= 2
            End If

            If AniCount >= 101 Then
                AniCount = 100
            ElseIf AniCount <= -1 Then
                AniCount = 0
            Else
                'DRAW STUFF IN THE BUTTON BASED ON THE PERCENTAGE
```

```
                Dim AniP As Single = AniCount / 100
                'L Triangle
                LTriangle(2).Y = 268 + 23 * AniP
                'I Shape
                IRect.Y = 245 - 35 * AniP
                'Mass
                MassRect.Y = 245 + 35 * AniP
                'Projectile
                ProjectileRect.Y = 231 - Abs(Sin(2 * PI * AniP)) * 41
                ProjectileRect.X = 254 + 266 * AniP
            End If
        End If
    End Sub

    Public Overrides Sub HandleInput()
        'CHECK IF MOUSE IS IN BUTTON
        If Windows.Forms.Form.MousePosition.X - Main.Left - 15 >= Location.X And
Windows.Forms.Form.MousePosition.X - Main.Left - 15 <= Location.X + Size.X And
Windows.Forms.Form.MousePosition.Y - Main.Top - 15 > Location.Y And
Windows.Forms.Form.MousePosition.Y - Main.Top - 15 <= Location.Y + 144 Then
            'Check for left mouse click on button
            For Each Click In Main.MouseButtonsUp
                If Click.Button = MouseButtons.Left Then
                    ScreenManager.UnloadScreen("SimulationButton")
                    ScreenManager.UnloadScreen("TestButton")
                    ScreenManager.UnloadScreen("MyProgressButton")
                    ScreenManager.UnloadScreen("TitleScreenTitle")

                    ScreenManager.AddScreen(New SimulationMenu)
                End If
            Next

            MouseHover = True
        Else
            MouseHover = False
        End If
    End Sub

    Public Overrides Sub Draw()
        'DRAW BUTTON
        'StaticImage
        Main.GFX.DrawImage(My.Resources.SimulationButton, Location)
        If MouseHover = True Then
            Main.GFX.DrawImage(My.Resources.SimulationButtonHoverBorder, Location)
        End If
        'L Shape
        Main.GFX.FillPolygon(New SolidBrush(Color.FromArgb(0, 90, 194)), LPoints)
        'L Triangle
        Main.GFX.FillPolygon(Brushes.Gray, LTriangle)
        'I Shape
        Main.GFX.FillRectangle(New SolidBrush(Color.FromArgb(0, 90, 194)), IRect)
        'Mass
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(92, 50, 3)), 5), 709,
233, 709, MassRect.Y + 5)
        Main.GFX.FillRectangle(Brushes.Gray, MassRect)
        Main.GFX.DrawString("4", Main.Arial_8, Brushes.Black, MassRect.X + 2,
MassRect.Y + 3)
        'Projectile
        Main.GFX.FillEllipse(New SolidBrush(Color.FromArgb(0, 90, 194)),
ProjectileRect)
    End Sub
```

```
End Class
```

TestButton

This screen is the third quarter of the Title Screen, and is the big animated Test Button which, when clicked, navigates to the Test User Selection. The animation advances when the mouse cursor is hovered over the button, and goes in reverse when the mouse cursor is not over it. The two images below show the beginning and end of the animation of this button.

```
Imports System.Math

Public Class TestButton
    Inherits BaseScreen

    Private MouseHover As Boolean = False

    Private AniTimer As Date
    Private AniCount As Integer = 0

    Private Size As New Point(319, 180)

    Private Tick1Points(2), Tick2Points(2), Cross1Points(3) As Point
    Private Tick1Alpha, Tick2Alpha, Cross1Alpha As Integer

    Public Sub New()
        Name = "TestButton"
        State = ScreenState.Active
        Location = New Point(320, 360)

        'INITIALISE STARTING MOVING PART SETTINGS
        'Tick 1
        Tick1Points(0) = New Point(570, 400)
        Tick1Points(1) = New Point(580, 410)
        Tick1Points(2) = New Point(600, 370)
        Tick1Alpha = 0
        'Tick 2
        Tick2Points(0) = New Point(570, 430)
        Tick2Points(1) = New Point(580, 440)
        Tick2Points(2) = New Point(600, 400)
        Tick2Alpha = 0
        'Cross 1
        Cross1Points(0) = New Point(570, 450)
        Cross1Points(1) = New Point(600, 480)
        Cross1Points(2) = New Point(570, 480)
        Cross1Points(3) = New Point(600, 450)
        Cross1Alpha = 0
    End Sub

    Public Overrides Sub Update()
        If (Now - AniTimer).TotalMilliseconds > 25 Then
```

```
            AniTimer = Now
            'every 25 milliseconds (ish)
            'if the mouse is on the button, advance the animation
            'else make the animation go backwards

            If MouseHover = True Then
                AniCount += 2
            Else
                AniCount -= 2
            End If

            If AniCount >= 101 Then
                AniCount = 100
            ElseIf AniCount <= -1 Then
                AniCount = 0
            Else
                'DRAW STUFF IN THE BUTTON BASED ON THE PERCENTAGE
                Dim AniP As Single = AniCount / 100

                Tick1Alpha = 255 * AniP ^ 1
                Tick2Alpha = 255 * AniP ^ 2
                Cross1Alpha = 255 * AniP ^ 4
            End If
        End If
    End Sub

    Public Overrides Sub HandleInput()
        'CHECK IF MOUSE IS IN BUTTON
        If Windows.Forms.Form.MousePosition.X - Main.Left - 15 >= Location.X And
Windows.Forms.Form.MousePosition.X - Main.Left - 15 <= Location.X + Size.X And
Windows.Forms.Form.MousePosition.Y - Main.Top - 15 > Location.Y And
Windows.Forms.Form.MousePosition.Y - Main.Top - 15 <= Location.Y + 144 Then
            'Check for left mouse click on button
            For Each Click In Main.MouseButtonsUp
                If Click.Button = MouseButtons.Left Then
                    ScreenManager.UnloadScreen("SimulationButton")
                    ScreenManager.UnloadScreen("TestButton")
                    ScreenManager.UnloadScreen("MyProgressButton")
                    ScreenManager.UnloadScreen("TitleScreenTitle")

                    'Load Test screens
                    ScreenManager.AddScreen(New TestUserSelection)
                End If
            Next

            MouseHover = True
        Else
            MouseHover = False
        End If
    End Sub

    Public Overrides Sub Draw()
        'DRAW BUTTON
        'StaticImage
        Main.GFX.DrawImage(My.Resources.TestButton, Location)
        If MouseHover = True Then
            Main.GFX.DrawImage(My.Resources.TestButtonHoverBorder, Location)
        End If
        'Tick1
        Main.GFX.DrawLines(New Pen(New SolidBrush(Color.FromArgb(Tick1Alpha,
Color.ForestGreen)), 3), Tick1Points)
```

```
        'Tick2
        Main.GFX.DrawLines(New Pen(New SolidBrush(Color.FromArgb(Tick2Alpha,
Color.ForestGreen)), 3), Tick2Points)
        'Cross1
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(Cross1Alpha,
Color.DarkRed)), 3), Cross1Points(0), Cross1Points(1))
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(Cross1Alpha,
Color.DarkRed)), 3), Cross1Points(2), Cross1Points(3))
    End Sub
End Class
```

MyProgressButton

This screen is the fourth quarter of the Title Screen, and is the big animated My Progress Button which, when clicked, navigates to the My Progress Selection. The animation advances when the mouse cursor is hovered over the button, and goes in reverse when the mouse cursor is not over it. The two images below show the beginning and end of the animation of this button.

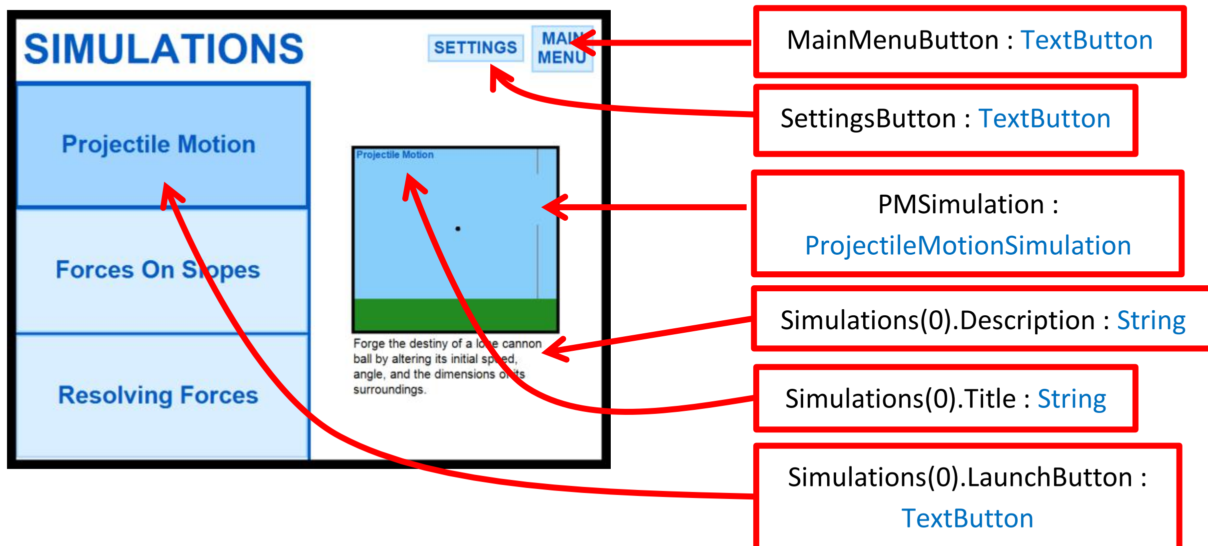
```
Public Class MyProgressButton
    Inherits BaseScreen

    Private MouseHover As Boolean = False

    Private AniTimer As Date
    Private AniCount As Integer = 0

    Private Size As New Point(637, 180)

    Private GraphCoverSrcRect As Rectangle
    Private GraphCoverX As Integer
    Private WellDoneAlpha, GoodJobAlpha As Integer

    Public Sub New()
        Name = "MyProgressButton"
        State = ScreenState.Active
        Location = New Point(162, 540)

        'INITIALISE STARTING MOVING PART SETTINGS
        'Graph Cover
        GraphCoverX = 480
        GraphCoverSrcRect = New Rectangle(0, 0, 291, 117)
        'Texts
        GoodJobAlpha = 0
        WellDoneAlpha = 0
    End Sub

    Public Overrides Sub Update()
        If (Now - AniTimer).TotalMilliseconds > 25 Then
            AniTimer = Now
```

```
            'every 25 milliseconds (ish)
            'if the mouse is on the button, advance the animation
            'else make the animation go backwards

            If MouseHover = True Then
                AniCount += 2
            Else
                AniCount -= 2
            End If

            If AniCount >= 101 Then
                AniCount = 100
            ElseIf AniCount <= -1 Then
                AniCount = 0
            Else
                'DRAW STUFF IN THE BUTTON BASED ON THE PERCENTAGE
                Dim AniP As Single = AniCount / 100
                'Graph Cover
                GraphCoverX = 480 + 291 * AniP
                GraphCoverSrcRect.X = 291 * AniP
                'Texts
                GoodJobAlpha = 255 * AniP
                WellDoneAlpha = 255 * AniP ^ 4
            End If
        End If
    End Sub

    Public Overrides Sub HandleInput()
        'CHECK IF MOUSE IS IN BUTTON
        If Windows.Forms.Form.MousePosition.X - Main.Left - 15 >= Location.X And
Windows.Forms.Form.MousePosition.X - Main.Left - 15 <= Location.X + Size.X And
Windows.Forms.Form.MousePosition.Y - Main.Top - 15 > Location.Y And
Windows.Forms.Form.MousePosition.Y - Main.Top - 15 <= Location.Y + 144 Then
            'Check for left mouse click on button
            For Each Click In Main.MouseButtonsUp
                If Click.Button = MouseButtons.Left Then
                    ScreenManager.UnloadScreen("SimulationButton")
                    ScreenManager.UnloadScreen("TestButton")
                    ScreenManager.UnloadScreen("MyProgressButton")
                    ScreenManager.UnloadScreen("TitleScreenTitle")

                    'Load My Progress screens
                    ScreenManager.AddScreen(New MyProgressUserSelection)
                End If
            Next

            MouseHover = True
        Else
            MouseHover = False
        End If
    End Sub

    Public Overrides Sub Draw()
        'DRAW BUTTON
        'StaticImage
        Main.GFX.DrawImage(My.Resources.MyProgressButton, Location)
        If MouseHover = True Then
            Main.GFX.DrawImage(My.Resources.MyProgressButtonHoverBorder, Location)
        End If
        'Well Done
```

```
        Main.GFX.DrawString("WELL DONE", Main.Arial_12_Bold, New
SolidBrush(Color.FromArgb(WellDoneAlpha, 0, 128, 0)), 190, 580)
        'Good Job
        Main.GFX.DrawString("GOOD JOB", Main.Georgia_20_Bold, New
SolidBrush(Color.FromArgb(GoodJobAlpha, 0, 128, 0)), 305, 560)
        'Graph Cover
        Main.GFX.DrawImage(My.Resources.MyProgressButtonGraphCover, GraphCoverX, 553,
GraphCoverSrcRect, GraphicsUnit.Pixel)
        'Static Text
        Main.GFX.DrawImage(My.Resources.MyProgressButtonText, Location)
    End Sub
End Class
```

SimulationMenu

This screen presents each of the three Simulations. There is a big button, a description and an animated preview for each one. The animated preview is just a half-size version of the simulation which constantly repeats itself. The information about a Simulation is shown when the mouse cursor hovers over the corresponding button. Clicking the button navigates to that Simulation.




```
Public Class SimulationMenu
    Inherits BaseScreen

    Private Const SimulationInfoWidth As Integer = 480
    Private Const SimulationInfoHeight As Double = 720 * 2 / 7

    Private Structure SimulationInfo
        Dim Title As String
        Dim Description As String
        Dim LaunchButton As TextButton
        Dim Location As Point
        Dim Enabled As Boolean
    End Structure

    Private Simulations(2) As SimulationInfo

    Private MainMenuButton As New TextButton(" MAIN" & vbNewLine & "MENU",
Main.Arial_20_Bold, ProgramSection.Simulation, New Point(845, 10), -1, -1, 3)
    Private SettingsButton As New TextButton("SETTINGS", Main.Arial_20_Bold,
ProgramSection.Simulation, New Point(675, 25), -1, -1, 3)
```

```
    'Bitmap for drawing simulation previews to
    Private BMP As New Bitmap(683 \ 2, 614 \ 2)

    Private FOSSimulation As New ForcesOnSlopesSimulation(SimulationMode.Simulation)
    Private RFSimulation As New ResolvingForcesSimulation(SimulationMode.Simulation)
    Private PMSimulation As New ProjectileMotionSimulation(SimulationMode.Simulation)

    Private VisibleSimulation As String = ""

    Public Sub New()
        Name = "SimulationMenu"
        State = ScreenState.Active
        Location = New Point(0, 0)

        Simulations(0).Title = "Projectile Motion"
        Simulations(2).Title = "Resolving Forces"
        Simulations(1).Title = "Forces On Slopes"

        'For each simulation, Set up the button, location and description
        For y = 0 To 2
            Simulations(y).Location = New Point(0, (720 \ 7) + (720 * 2 \ 7) * y)
            Select Case Simulations(y).Title
                Case "Projectile Motion"
                    Simulations(y).Description = "Forge the destiny of a lone cannon
ball by altering its initial speed, angle, and the dimensions of its surroundings."
                Case "Resolving Forces"
                    Simulations(y).Description = "Model a mass pulling another mass
around a smooth pulley by a light, inextensible string."
                Case "Forces On Slopes"
                    Simulations(y).Description = "Change the angle of the slope, the
mass of the block, and even the gravity."
                Case Else
                    Simulations(y).Description = ""
            End Select
            Simulations(y).LaunchButton = New TextButton(Simulations(y).Title,
Main.Arial_30_Bold, ProgramSection.Simulation, New Point(Simulations(y).Location.X,
Simulations(y).Location.Y + 4), SimulationInfoWidth - 2, SimulationInfoHeight - 5, 3)
            If Simulations(y).Title <> "" Then
                Simulations(y).Enabled = True
            Else
                Simulations(y).Enabled = False
            End If
        Next

        PMSimulation.Enabled = True
        FOSSimulation.Enabled = True
        RFSimulation.Enabled = True
    End Sub

    Public Overrides Sub Update()
        'Find out which button is being hovered over, then set that category as the
visible preview simulation.
        VisibleSimulation = ""
        For Each Simulation In Simulations
            If Simulation.Enabled = True Then
                If Simulation.LaunchButton.MouseHover = True Then
                    VisibleSimulation = Simulation.Title
                End If
            End If
        Next
```

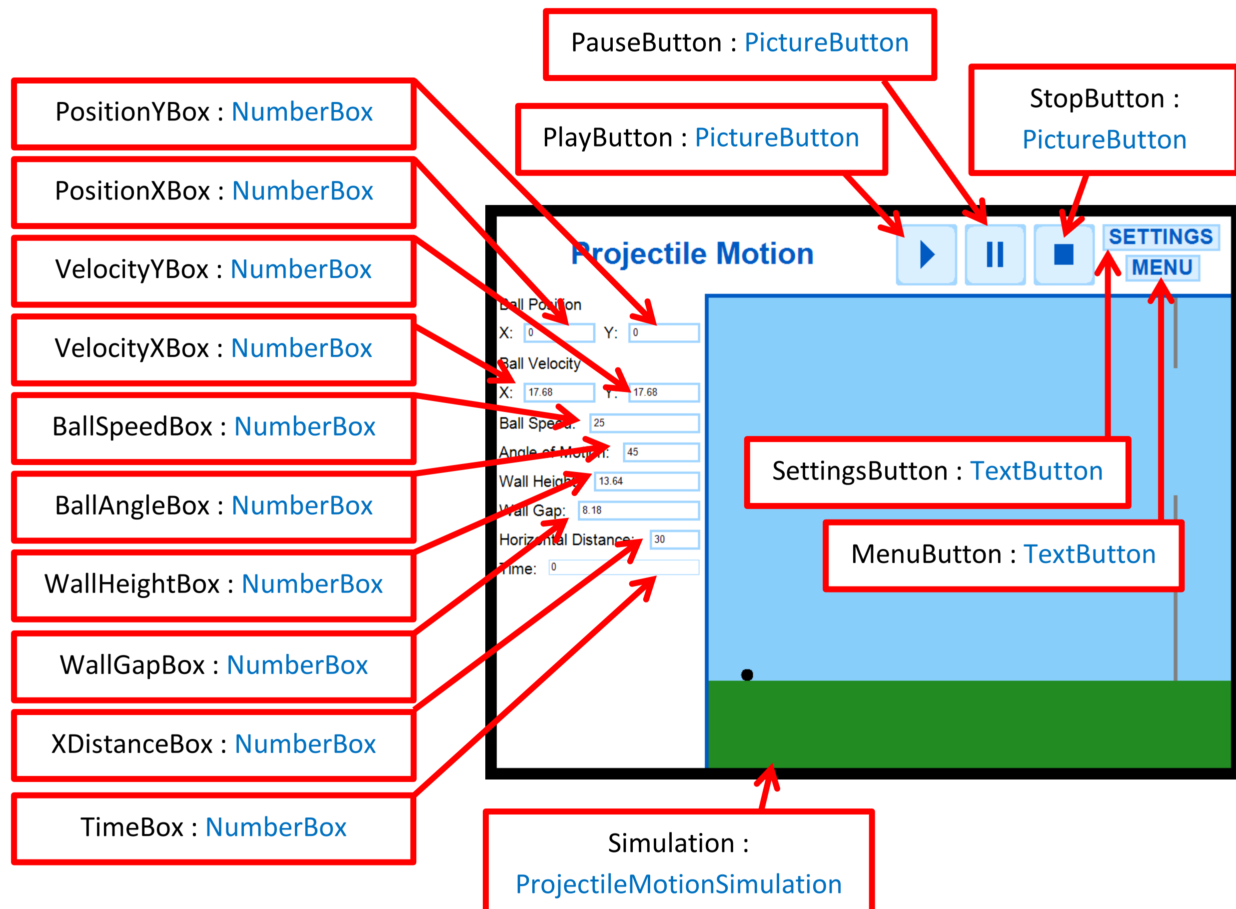
```
        'Update the visible simulation
        Select Case VisibleSimulation
            Case "Forces On Slopes"
                FOSSimulation.Update()
                If FOSSimulation.Finished = True Then
                    FOSSimulation.ResetVariables()
                    FOSSimulation.Finished = False
                    FOSSimulation.Enabled = True
                End If
            Case "Resolving Forces"
                RFSimulation.Update()
                If RFSimulation.Finished = True Then
                    RFSimulation.ResetVariables()
                    RFSimulation.Finished = False
                    RFSimulation.Enabled = True
                End If
            Case "Projectile Motion"
                PMSimulation.Update()
                If PMSimulation.Finished = True Then
                    PMSimulation.ResetVariables()
                    PMSimulation.Finished = False
                    PMSimulation.Enabled = True
                End If
        End Select
    End Sub

    Public Overrides Sub HandleInput()
        'Check all buttons for clicks
        For y = 0 To 2
            If Simulations(y).LaunchButton.Clicked = "Clicked" And
Simulations(y).Enabled = True Then
                ScreenManager.UnloadScreen("SimulationMenu")
                Select Case Simulations(y).Title
                    Case "Projectile Motion"
                        ScreenManager.AddScreen(New ProjectileMotion)
                    Case "Resolving Forces"
                        ScreenManager.AddScreen(New ResolvingForces)
                    Case "Forces On Slopes"
                        ScreenManager.AddScreen(New ForcesOnSlopes)
                End Select
            End If
        Next

        If MainMenuButton.Clicked = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New Title)
            ScreenManager.AddScreen(New SimulationButton)
            ScreenManager.AddScreen(New TestButton)
            ScreenManager.AddScreen(New MyProgressButton)
        End If
        If SettingsButton.Clicked = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New Settings({New SimulationMenu}))
        End If
    End Sub

    Public Overrides Sub Draw()
        'MAIN TITLE BAR
        'Title
```

```
        Main.GFX.DrawString("SIMULATIONS", Main.Arial_50_Bold, New
SolidBrush(Color.FromArgb(0, 90, 194)), New Point(0, 10))
        'Buttons
        MainMenuButton.Draw()
        SettingsButton.Draw()

        'DIVIDING LINES
        For y = 0 To 2
            Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(0, 90, 194)), 5),
New PointF(0, SimulationInfoHeight * (y + 0.5)), New PointF(480, SimulationInfoHeight
* (y + 0.5)))
        Next
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(0, 90, 194)), 5),
SimulationInfoWidth, SimulationInfoHeight \ 2 - 2, SimulationInfoWidth, 960 + 2)

        For y = 0 To 2
            'FOR EACH SIMULATION
            If Simulations(y).Enabled = True Then
                'Button
                Simulations(y).LaunchButton.Draw()
            End If
        Next

        'PREVIEW OF VISIBLE SIMULATION
        Graphics.FromImage(BMP).Clear(Color.White)
        Select Case VisibleSimulation
            Case "Forces On Slopes"
                FOSSimulation.DrawToCustomImage(BMP)
                Main.AutoFitText(550, 360 + BMP.Height \ 2 + 5, BMP.Width,
Main.Arial_15, Simulations(1).Description)
                Graphics.FromImage(BMP).DrawString(VisibleSimulation,
Main.Arial_12_Bold, New SolidBrush(Color.FromArgb(0, 90, 194)), 5, 5)
            Case "Resolving Forces"
                RFSimulation.DrawToCustomImage(BMP)
                Main.AutoFitText(550, 360 + BMP.Height \ 2 + 5, BMP.Width,
Main.Arial_15, Simulations(2).Description)
                Graphics.FromImage(BMP).DrawString(VisibleSimulation,
Main.Arial_12_Bold, New SolidBrush(Color.FromArgb(0, 90, 194)), 5, 5)
            Case "Projectile Motion"
                PMSimulation.DrawToCustomImage(BMP)
                Main.AutoFitText(550, 360 + BMP.Height \ 2 + 5, BMP.Width,
Main.Arial_15, Simulations(0).Description)
                Graphics.FromImage(BMP).DrawString(VisibleSimulation,
Main.Arial_12_Bold, New SolidBrush(Color.FromArgb(0, 90, 194)), 5, 5)
            Case Else
                Graphics.FromImage(BMP).DrawString("Hover over a Simulation for a
preview", Main.Arial_12_Bold, New SolidBrush(Color.FromArgb(0, 90, 194)), 20,
BMP.Height \ 2 - 20)
        End Select
        Graphics.FromImage(BMP).DrawRectangle(New Pen(Brushes.Black, 5), 2, 2,
BMP.Width - 5, BMP.Height - 5)
        Main.GFX.DrawImage(BMP, 550, 360 - BMP.Height \ 2)
    End Sub
End Class
```

ProjectileMotion

This screen is for controlling the Projectile Motion Simulation. There are variables for changing various aspects of the Simulation. There are also buttons for the Play, Pause and Stop/Reset

commands. The main feature of this screen is its instance of the ProjectileMotionSimulation, which is responsible for the actual Simulation, as well as the animation.




```
Imports System.Math

Public Class ProjectileMotion
    Inherits BaseScreen

    Private MenuButton As New TextButton("MENU", Main.Arial_20_Bold,
ProgramSection.Simulation, New Point(822, 50), -1, 35, 3, 1)
    Private SettingsButton As New TextButton("SETTINGS", Main.Arial_20_Bold,
ProgramSection.Simulation, New Point(792, 10), -1, 35, 3, 1)

    Private PlayButton As New PictureButton(New Point(522, 10),
My.Resources.PlayDefault, My.Resources.PlayHover, My.Resources.PlayDown, -1, -1)
    Private PauseButton As New PictureButton(New Point(612, 10),
My.Resources.PauseDefault, My.Resources.PauseHover, My.Resources.PauseDown, -1, -1)
    Private StopButton As New PictureButton(New Point(702, 10),
My.Resources.StopDefault, My.Resources.StopHover, My.Resources.StopDown, -1, -1)

    Private Simulation As New ProjectileMotionSimulation(SimulationMode.Simulation)
    Private PositionXBox, PositionYBox, VelocityXBox, VelocityYBox, BallSpeedBox,
BallAngleBox, WallHeightBox, WallGapBox, XDistanceBox, TimeBox As NumberBox

    Public Sub New()
        Dim TempY, TempX As Integer

        Name = "ProjectileMotion"
```

```
        State = ScreenState.Active
        Location = New Point(0, 0)

        'Create the input boxes in the correct positions for each variable
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15, "Ball
Position", False)
        TempX = Main.GFX.MeasureString("X:", Main.Arial_15).Width + 10
        PositionXBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 / 7 - TempX - 10)
        TempX = 960 / 7 + Main.GFX.MeasureString("Y:", Main.Arial_15).Width + 10
        PositionYBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY += Main.GFX.MeasureString("X:", Main.Arial_15).Height + 15

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Ball
Velocity", False)
        TempX = Main.GFX.MeasureString("X:", Main.Arial_15).Width + 10
        VelocityXBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 / 7 - TempX - 10)
        TempX = 960 / 7 + Main.GFX.MeasureString("Y:", Main.Arial_15).Width + 10
        VelocityYBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY += Main.GFX.MeasureString("X:", Main.Arial_15).Height + 15

        TempX = Main.GFX.MeasureString("Ball Speed:", Main.Arial_15).Width + 10
        BallSpeedBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Ball Speed:",
False)

        TempX = Main.GFX.MeasureString("Angle of Motion:", Main.Arial_15).Width + 10
        BallAngleBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Angle of
Motion:", False)

        TempX = Main.GFX.MeasureString("Wall Height:", Main.Arial_15).Width + 10
        WallHeightBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Wall Height:",
False)

        TempX = Main.GFX.MeasureString("Wall Gap:", Main.Arial_15).Width + 10
        WallGapBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Wall Gap:",
False)

        TempX = Main.GFX.MeasureString("Horizontal Distance:", Main.Arial_15).Width +
10
        XDistanceBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Horizontal
Distance:", False)

        TempX = Main.GFX.MeasureString("Time:", Main.Arial_15).Width + 10
        TimeBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 1, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Time:", False)

        GetValuesFromSim()
```

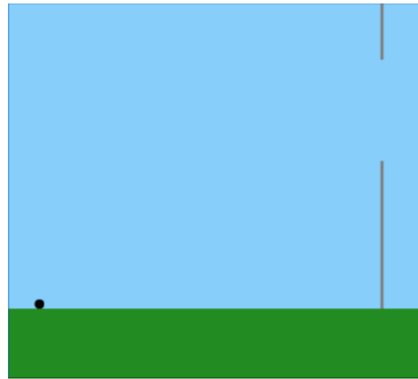
```
    End Sub

    Public Overrides Sub HandleInput()
        Dim ChangeOccured As Boolean = True
        If MenuButton.Clicked = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New SimulationMenu)
        End If
        If SettingsButton.Clicked() = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New Settings({New ProjectileMotion}))
        End If

        'Check input for play, pause, stop
        If PlayButton.Clicked() = "Clicked" Then
            Simulation.TTimer = Now
            Simulation.Enabled = True
        ElseIf PauseButton.Clicked() = "Clicked" Then
            Simulation.Enabled = False
        ElseIf StopButton.Clicked() = "Clicked" Then
            Simulation.Enabled = False
            Simulation.ResetVariables()
            GetValuesFromSim()
        End If
        For Each Key In Main.KeysDown
            If Key = 32 Then
                'Space Bar
                If Simulation.Enabled = True Then
                    Simulation.Enabled = False
                Else
                    Simulation.TTimer = Now
                    Simulation.Enabled = True
                End If
            End If
        Next

        If Simulation.Enabled = False And Simulation.T = 0 Then
            'Check for input from variable input boxes, then update the simulation
with the values.
            'Sometimes when one value changes, other dependent values must change too.
            If PositionXBox.HandleInput() = "Entered" And PositionXBox.Text <> "" Then
                Simulation.InitialBallS.X = Simulation.Pixels(CDec(PositionXBox.Text))
+ 50
            ElseIf PositionYBox.HandleInput() = "Entered" And PositionYBox.Text <> ""
Then
                Simulation.InitialBallS.Y = Simulation.Pixels(CDec(PositionYBox.Text))
            ElseIf VelocityXBox.HandleInput() = "Entered" And VelocityXBox.Text <> ""
Then
                Simulation.FiringV.X = CDec(VelocityXBox.Text)
            ElseIf VelocityYBox.HandleInput() = "Entered" And VelocityYBox.Text <> ""
Then
                Simulation.FiringV.Y = CDec(VelocityYBox.Text)
            ElseIf BallSpeedBox.HandleInput() = "Entered" And BallSpeedBox.Text <> ""
Then
                Simulation.FiringSpeed = CDec(BallSpeedBox.Text)
            ElseIf BallAngleBox.HandleInput() = "Entered" And BallAngleBox.Text <> ""
Then
                Simulation.FiringAngle = CDec(BallAngleBox.Text)
            ElseIf WallHeightBox.HandleInput() = "Entered" And WallHeightBox.Text <>
"" Then
                Simulation.WallY2 = 500 - Simulation.Pixels(CDec(WallHeightBox.Text))
```

```
                Simulation.WallY1 = 500 - Simulation.Pixels(CDec(WallHeightBox.Text) +
CDec(WallGapBox.Text))
            ElseIf WallGapBox.HandleInput() = "Entered" And WallGapBox.Text <> "" Then
                Simulation.WallY2 = 500 - Simulation.Pixels(CDec(WallHeightBox.Text))
                Simulation.WallY1 = 500 - Simulation.Pixels(CDec(WallHeightBox.Text) +
CDec(WallGapBox.Text))
            ElseIf XDistanceBox.HandleInput() = "Entered" And XDistanceBox.Text <> ""
Then
                If XDistanceBox.Text <> "0" Then
                    Simulation.Scale = 550 / CDec(XDistanceBox.Text)
                Else
                    Main.MessageBox("Error: Horizontal Distance cannot be 0")
                End If
            Else
                ChangeOccured = False
            End If

            'Variables entered
            If ChangeOccured = True Then
                Simulation.ResetVariables()
                GetValuesFromSim()
            End If
        End If
    End Sub

    Private Sub GetValuesFromSim()
        'values from simulation into boxes
        PositionXBox.Text = Round(Simulation.Metres(Simulation.BallS.X - 50), 2)
        PositionYBox.Text = Round(Simulation.Metres(Simulation.BallS.Y), 2)
        VelocityXBox.Text = Round(Simulation.BallV.X, 2)
        VelocityYBox.Text = Round(Simulation.BallV.Y, 2)
        BallSpeedBox.Text = Round(Sqrt(Simulation.BallV.X ^ 2 + Simulation.BallV.Y ^
2), 2)
        BallAngleBox.Text = Round(Main.Deg(Atan(Simulation.BallV.Y /
Simulation.BallV.X)), 2)
        WallHeightBox.Text = Round(Simulation.Metres(500 - Simulation.WallY2), 2)
        WallGapBox.Text = Round(Simulation.Metres(500 - Simulation.WallY1) -
WallHeightBox.Text, 2)
        XDistanceBox.Text = Round(550 / Simulation.Scale, 2)
        TimeBox.Text = Round(Simulation.T, 2)
    End Sub

    Public Overrides Sub Update()
        Simulation.Update()

        If Simulation.Enabled = True Then
            GetValuesFromSim()
        End If
    End Sub

    Public Overrides Sub Draw()
        Dim TempY As Integer = 0

        'MAIN TITLE BAR
        'Title
        Main.GFX.DrawString("Projectile Motion", Main.Arial_30_Bold, New
SolidBrush(Color.FromArgb(0, 90, 194)), New Point(261 -
Main.GFX.MeasureString("Projectile Motion", Main.Arial_30_Bold).Width \ 2, 25))
        'Simulation control buttons
        PlayButton.Draw()
        PauseButton.Draw()
```

```
        StopButton.Draw()
        'Other buttons
        MenuButton.Draw()
        SettingsButton.Draw()

        'DIVIDING LINES
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(0, 90, 194)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7), New Point(960, 720 * 1 / 7))
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(0, 90, 194)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7 - 2), New Point(960 * 2 / 7, 720))

        'VARIABLE SETTINGS
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15, "Ball
Position")
        Main.GFX.DrawString("X:", Main.Arial_15, Brushes.Black, 0, TempY)
        PositionXBox.Draw()
        Main.GFX.DrawString("Y:", Main.Arial_15, Brushes.Black, 960 / 7, TempY)
        TempY += Main.GFX.MeasureString("Y:", Main.Arial_15).Height + 15
        PositionYBox.Draw()

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Ball
Velocity")
        Main.GFX.DrawString("X:", Main.Arial_15, Brushes.Black, 0, TempY)
        VelocityXBox.Draw()
        Main.GFX.DrawString("Y:", Main.Arial_15, Brushes.Black, 960 / 7, TempY)
        TempY += Main.GFX.MeasureString("Y:", Main.Arial_15).Height + 15
        VelocityYBox.Draw()

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Ball Speed:")
        BallSpeedBox.Draw()

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Angle of
Motion:")
        BallAngleBox.Draw()

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Wall Height:")
        WallHeightBox.Draw()

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Wall Gap:")
        WallGapBox.Draw()

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Horizontal
Distance:")
        XDistanceBox.Draw()

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Time:")
        TimeBox.Draw()
        'SIMULATION
        Simulation.Draw()

        'MAIN RECT: 277, 106, 683, 614
    End Sub
End Class
```

ProjectileMotionSimulation

This class purely controls the actual Simulation, and only displays the animation for Projectile Motion. The DrawToCustomImage procedure is used for the Simulation previews on the Simulations Menu. It draws the animation with half the size.



```
Imports System.Math

Public Enum SimulationMode
    Simulation
    Test
End Enum

Public Class ProjectileMotionSimulation
    Inherits BaseScreen

    Public Mode As SimulationMode
    Public Finished As Boolean = False
    Public Visible As Boolean = True

    Public Scale As Double = 550 / 30

    Private Size As New Size(683, 614)

    Private GroundY As Integer

    Private BallRadius As Integer
    Public BallMass As Integer
    Public BallV, BallF, InitialBallS, BallS As PointF
    'BallS: X = pixels right from left edge, Y = pixels up from ground
    Public BallAngle, BallSpeed As Double
    Private BallOutOfTop As Boolean = False
    Private AmountOutOfTop As Decimal
    Public g As Double = -9.8

    Public T As Double
    Public Tmicros As Integer
    Public TTimer As Date
    Public Enabled As Boolean

    Private WallX, WallWidth As Integer
    Public WallY1, WallY2 As Double
    Private BallOffset As Point

    Public FiringAngle As Single
    Public FiringV As PointF
    Public FiringSpeed As Single

    Public Sub New(ByVal InputMode As SimulationMode)
        Mode = InputMode
        Name = "ProjectileMotionSimulation"
        State = ScreenState.NoInput
        Location = New Point(277, 106)
```

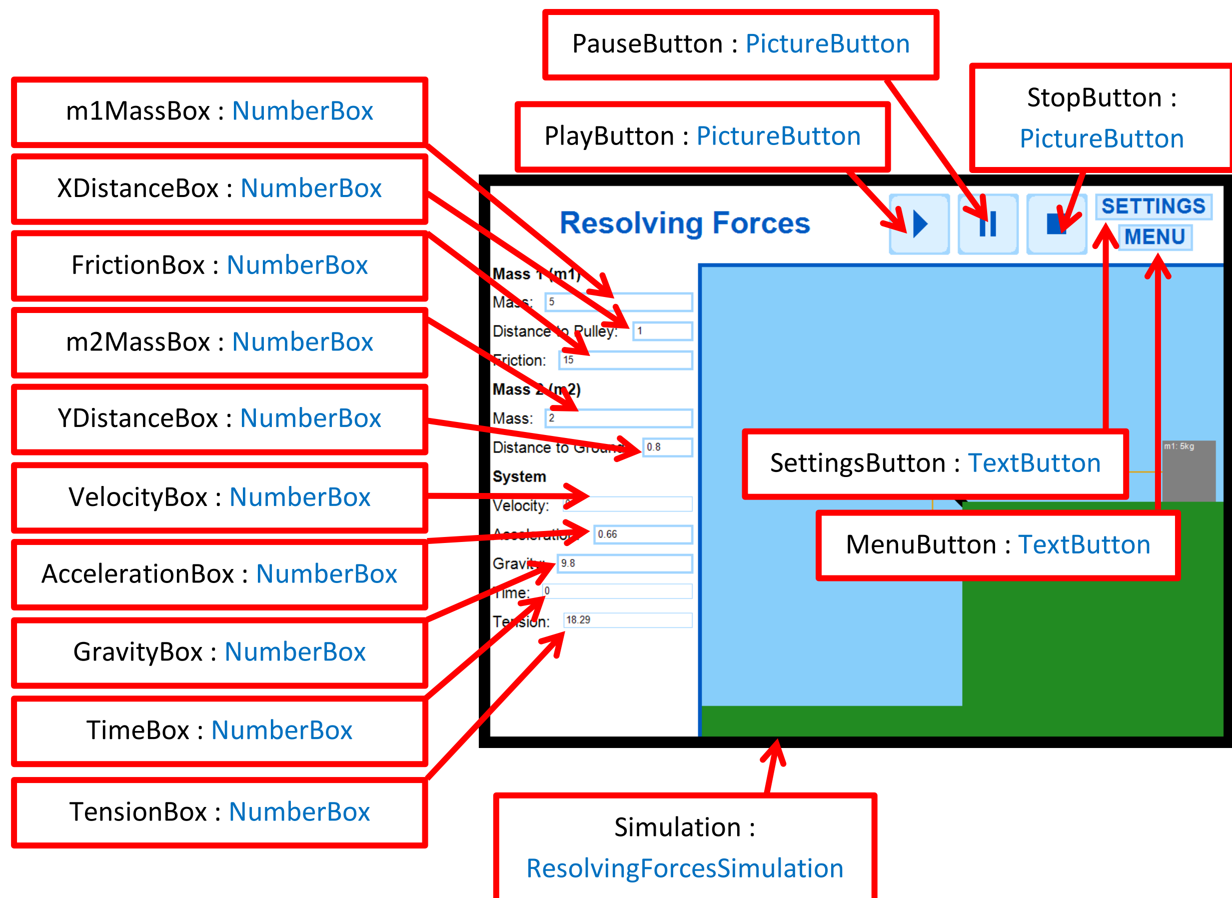
```
        GroundY = 500
        WallX = 600
        WallWidth = 5
        BallRadius = 8
        T = 0
        InitialBallS = New PointF(50, 0)

        If Mode = SimulationMode.Simulation Then
            WallY1 = 100
            WallY2 = 250
            FiringSpeed = 25
            FiringAngle = 45
            ResetVariables()
        End If
    End Sub

    Public Sub ResetVariables()
        'Set variables to their correct initial conditions
        FiringV = New PointF(FiringSpeed * Cos(Main.Rad(FiringAngle)), FiringSpeed *
Sin(Main.Rad(FiringAngle)))

        Tmicros = 0
        T = 0
        BallS = InitialBallS
        BallV = FiringV
        BallAngle = FiringAngle
        BallOutOfTop = False
    End Sub

    Public Sub SetTestVariables(ByVal InputWallHeight As Integer, ByVal InputWallGap
As Integer, ByVal InputFiringSpeed As Single, ByVal InputFiringAngle As Single, ByVal
InputGroundDistance As Single)
        'Allows input of variables other than the defualt.
        'This is needed for the test mode
        Scale = 550 / InputGroundDistance

        WallY2 = 500 - Pixels(InputWallHeight)
        WallY1 = 500 - Pixels(InputWallHeight + InputWallGap)

        FiringSpeed = InputFiringSpeed
        FiringAngle = InputFiringAngle

        ResetVariables()
    End Sub

    Public Function Metres(ByVal Pixels As Double) As Double
        Return Pixels / Scale
    End Function
    Public Function Pixels(ByVal Metres As Double) As Double
        Return Metres * Scale
    End Function

    Public Overrides Sub Update()
        If Enabled = True Then
            If (Now - TTimer).TotalMilliseconds > 25 Then
                TTimer = Now

                Dim NewBallX, NewBallY As Double

                'Every 25 milliseconds (ish)
```

```
                'Gradually increase the time variable
                'Calculate the expected position as if no collision happens, then
                'see if there should be a collision
                For i = 1 To 10000
                    NewBallX = InitialBallS.X + Pixels(FiringV.X * T)
                    NewBallY = InitialBallS.Y + Pixels(FiringV.Y * T + 0.5 * g * T ^
2)

                    'Update ball's velocity
                    BallV.Y = FiringV.Y + g * T

                    If Abs(BallV.X) > 0 Then
                        If NewBallX < 0 Or (BallS.X <= WallX And NewBallX >= WallX And
(GroundY - NewBallY <= WallY1 Or GroundY - NewBallY >= WallY2)) Or NewBallX >
Size.Width - BallRadius Then
                            If NewBallX < 0 Then
                                'Ball reaches left edge
                                BallS.X = 0
                                BallV.X = 0
                            ElseIf NewBallX >= Size.Width - BallRadius Then
                                'Ball has gone through wall and reaches right edge
                                BallS.X = Size.Width - BallRadius
                                BallV.X = 0
                            ElseIf NewBallX > WallX Then
                                'Ball hits wall
                                BallS.X = WallX
                                BallV.X = 0
                            End If
                        Else
                            'No special cases, free space ahead
                            BallS.X = NewBallX
                        End If
                    End If

                    If Abs(BallV.Y) > 0 Then
                        If NewBallY < 0 Or NewBallY > GroundY - 2 * BallRadius Then
                            If NewBallY > GroundY - 2 * BallRadius Then
                                'Ball reaches top edge
                                BallOutOfTop = True
                                AmountOutOfTop = Round(Metres(NewBallY - (GroundY - 2
* BallRadius)), 2)

                                BallS.Y = GroundY - 2 * BallRadius
                                BallV.Y = 0
                            ElseIf NewBallY < 0 Then
                                'Ball Reaches ground
                                BallS.Y = 0
                                BallV.Y = 0
                                BallV.X = 0
                                Finished = True
                            End If
                        Else
                            'No special cases, free space ahead
                            BallS.Y = NewBallY
                            BallOutOfTop = False
                        End If
                    End If

                    'Increase time by 1ms
                    Tmicros += 1
                    T = Tmicros / 1000000
```

```
                Next
            End If
            '0.01s of simulation has passed
        End If
    End Sub

    Public Overrides Sub Draw()
        'SKY
        Main.GFX.FillRectangle(Brushes.LightSkyBlue, Location.X, Location.Y,
Size.Width, GroundY)
        'GROUND
        Main.GFX.FillRectangle(Brushes.ForestGreen, Location.X, Location.Y + GroundY,
Size.Width, Size.Height - GroundY)
        'WALL
        Main.GFX.FillRectangle(Brushes.Gray, Location.X + WallX + BallRadius,
Location.Y, WallWidth, CInt(WallY1) - BallRadius)
        Main.GFX.FillRectangle(Brushes.Gray, Location.X + WallX + BallRadius,
Location.Y + CInt(WallY2) + BallRadius, WallWidth, GroundY - CInt(WallY2) -
BallRadius)
        'BALL (Or OuOfTop arrow)
        If BallOutOfTop = True Then
            Main.GFX.FillEllipse(New SolidBrush(Color.FromArgb(50, Color.Black)),
Location.X + BallS.X - BallRadius, Location.Y + GroundY - BallS.Y - 2 * BallRadius, 2
* BallRadius, 2 * BallRadius)
            Main.GFX.DrawLine(New Pen(Brushes.Red, 3), Location.X + BallS.X,
Location.Y + 3 * BallRadius + 2 * BallRadius, Location.X + BallS.X, Location.Y + 2 *
BallRadius)
            Main.GFX.DrawLine(New Pen(Brushes.Red, 3), Location.X + BallS.X -
BallRadius, Location.Y + BallRadius + 2 * BallRadius, Location.X + BallS.X, Location.Y
+ 2 * BallRadius)
            Main.GFX.DrawLine(New Pen(Brushes.Red, 3), Location.X + BallS.X +
BallRadius, Location.Y + BallRadius + 2 * BallRadius, Location.X + BallS.X, Location.Y
+ 2 * BallRadius)
            Main.GFX.DrawString(AmountOutOfTop & "m", Main.Arial_12_Bold, Brushes.Red,
Location.X + BallS.X + BallRadius * 2, Location.Y + 2 * BallRadius)
        Else
            Main.GFX.FillEllipse(Brushes.Black, Location.X + BallS.X - BallRadius,
Location.Y + GroundY - BallS.Y - 2 * BallRadius, 2 * BallRadius, 2 * BallRadius)
        End If
    End Sub

    Public Sub DrawToCustomImage(ByRef BMP As Image)
        'Used for drawing the simulation when it is used as a preview on the
simulation menu
        'SKY
        Graphics.FromImage(BMP).FillRectangle(Brushes.LightSkyBlue, 0, 0, Size.Width \
2, GroundY \ 2)
        'GROUND
        Graphics.FromImage(BMP).FillRectangle(Brushes.ForestGreen, 0, GroundY \ 2,
Size.Width \ 2, (Size.Height - GroundY) \ 2)
        'WALL
        Graphics.FromImage(BMP).FillRectangle(Brushes.Gray, (WallX + BallRadius) \ 2,
0, WallWidth \ 2, (CInt(WallY1) - BallRadius) \ 2)
        Graphics.FromImage(BMP).FillRectangle(Brushes.Gray, (WallX + BallRadius) \ 2,
(CInt(WallY2) + BallRadius) \ 2, WallWidth \ 2, (GroundY - CInt(WallY2) - BallRadius)
\ 2)
        'BALL (Or OuOfTop arrow)
        Graphics.FromImage(BMP).FillEllipse(Brushes.Black, (BallS.X - BallRadius) \ 2,
(GroundY - BallS.Y - 2 * BallRadius) \ 2, BallRadius, BallRadius)
    End Sub
End Class
```

ResolvingForces

This screen is for controlling the Resolving Forces Simulation. There are variables for changing various aspects of the Simulation. There are also buttons for the Play, Pause and Stop/Reset commands. The main feature of this screen is its instance of the ResolvingForcesSimulation, which is responsible for the actual Simulation, as well as the animation.




```
Imports System.Math

Public Class ResolvingForces
    Inherits BaseScreen

    Private MenuButton As New TextButton("MENU", Main.Arial_20_Bold,
ProgramSection.Simulation, New Point(822, 50), -1, 35, 3, 1)
    Private SettingsButton As New TextButton("SETTINGS", Main.Arial_20_Bold,
ProgramSection.Simulation, New Point(792, 10), -1, 35, 3, 1)

    Private PlayButton As New PictureButton(New Point(522, 10),
My.Resources.PlayDefault, My.Resources.PlayHover, My.Resources.PlayDown, -1, -1)
    Private PauseButton As New PictureButton(New Point(612, 10),
My.Resources.PauseDefault, My.Resources.PauseHover, My.Resources.PauseDown, -1, -1)
    Private StopButton As New PictureButton(New Point(702, 10),
My.Resources.StopDefault, My.Resources.StopHover, My.Resources.StopDown, -1, -1)

    Private Simulation As New ResolvingForcesSimulation(SimulationMode.Simulation)

    Private m1MassBox, m2MassBox, FrictionBox, AccelerationBox, GravityBox,
XDistanceBox, YDistanceBox, TimeBox, VelocityBox, TensionBox As NumberBox
```

```
    Public Sub New()
        Dim TempY, TempX As Integer

        Name = "ResolvingForces"
        State = ScreenState.Active
        Location = New Point(0, 0)

        'Create the input boxes in the correct positions for each variable
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15_Bold,
"Mass 1 (m1)", False)
        TempX = Main.GFX.MeasureString("Mass:", Main.Arial_15).Width + 10
        m1MassBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Mass:", False)

        TempX = Main.GFX.MeasureString("Distance to Pulley:", Main.Arial_15).Width +
10
        XDistanceBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Distance to
Pulley:", False)

        TempX = Main.GFX.MeasureString("Friction:", Main.Arial_15).Width + 10
        FrictionBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Friction:",
False)

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15_Bold, "Mass 2
(m2)", False)
        TempX = Main.GFX.MeasureString("Mass:", Main.Arial_15).Width + 10
        m2MassBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Mass:", False)

        TempX = Main.GFX.MeasureString("Distance to Ground:", Main.Arial_15).Width +
10
        YDistanceBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Distance to
Ground:", False)

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15_Bold, "System",
False)
        TempX = Main.GFX.MeasureString("Velocity:", Main.Arial_15).Width + 10
        VelocityBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 1, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Velocity:",
False)

        TempX = Main.GFX.MeasureString("Acceleration:", Main.Arial_15).Width + 10
        AccelerationBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15,
"Acceleration:", False)

        TempX = Main.GFX.MeasureString("Gravity:", Main.Arial_15).Width + 10
        GravityBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Gravity:",
False)
```

```
        TempX = Main.GFX.MeasureString("Time:", Main.Arial_15).Width + 10
        TimeBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 1, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Time:", False)

        TempX = Main.GFX.MeasureString("Tension:", Main.Arial_15).Width + 10
        TensionBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 1, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Tension:",
False)

        GetValuesFromSim()
    End Sub

    Public Overrides Sub HandleInput()
        Dim ChangeOccured As Boolean = True
        If MenuButton.Clicked = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New SimulationMenu)
        End If
        If SettingsButton.Clicked() = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New Settings({New ResolvingForces}))
        End If

        'Check input for play, pause, stop
        If PlayButton.Clicked() = "Clicked" Then
            Simulation.TTimer = Now
            Simulation.Enabled = True
        ElseIf PauseButton.Clicked() = "Clicked" Then
            Simulation.Enabled = False
        ElseIf StopButton.Clicked() = "Clicked" Then
            Simulation.Enabled = False
            Simulation.ResetVariables()
            GetValuesFromSim()
        End If
        For Each Key In Main.KeysDown
            If Key = 32 Then
                'Space Bar
                If Simulation.Enabled = True Then
                    Simulation.Enabled = False
                Else
                    Simulation.TTimer = Now
                    Simulation.Enabled = True
                End If
            End If
        Next

        If Simulation.Enabled = False And Simulation.T = 0 Then
            'Acceleration = (m2Mass * g - Friction) / (m1Mass + m2Mass)
            'Tension = Acceleration * m1Mass + Friction
            'Friction = m2Mass * g - m1Mass * Acceleration - m2Mass * Acceleration
            'Acceleration = (Tension - Friction) / m1Mass

            'Check for input from variable input boxes, then update the simulation
with the values.
            'Sometimes when one value changes, other dependent values must change too.
            'For example, when the friction is changed, the acceleration and tension
changes
            If m1MassBox.HandleInput() = "Entered" And m1MassBox.Text <> "" Then
```

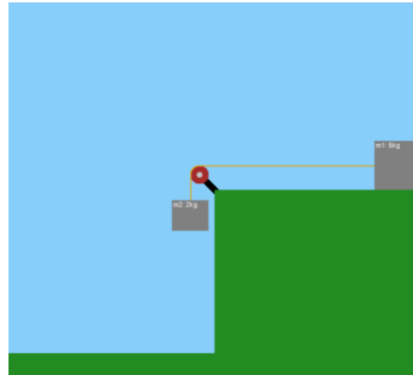
```
                Simulation.m1Mass = CDec(m1MassBox.Text)
            ElseIf XDistanceBox.HandleInput() = "Entered" And XDistanceBox.Text <> ""
Then
                Simulation.xDist = CDec(XDistanceBox.Text)
                Simulation.yDist = Simulation.xDist * 0.8
            ElseIf FrictionBox.HandleInput() = "Entered" And FrictionBox.Text <> ""
Then
                Simulation.Friction = CDec(FrictionBox.Text)
                If Simulation.Friction > Simulation.m2Mass * Simulation.g Then
                    Simulation.Friction = Simulation.m2Mass * Simulation.g
                    Main.MessageBox("Error: Friction cannot be that high")
                End If
                Simulation.Acceleration = (Simulation.m2Mass * Simulation.g -
Simulation.Friction) / (Simulation.m1Mass + Simulation.m2Mass)
                Simulation.Tension = Simulation.Acceleration * Simulation.m1Mass +
Simulation.Friction
            ElseIf m2MassBox.HandleInput() = "Entered" And m2MassBox.Text <> "" Then
                Simulation.m2Mass = CDec(m2MassBox.Text)
            ElseIf YDistanceBox.HandleInput() = "Entered" And YDistanceBox.Text <> ""
Then
                If YDistanceBox.Text <> "0" Then
                    Simulation.yDist = CDec(YDistanceBox.Text)
                    Simulation.xDist = Simulation.yDist / 0.8
                Else
                    Main.MessageBox("Error: Distance to Ground cannot be 0")
                End If
            ElseIf AccelerationBox.HandleInput() = "Entered" And AccelerationBox.Text
<> "" Then
                Simulation.Acceleration = CDec(AccelerationBox.Text)
                Simulation.Friction = Simulation.m2Mass * Simulation.g -
Simulation.m1Mass * Simulation.Acceleration - Simulation.m2Mass *
Simulation.Acceleration
                Simulation.Tension = Simulation.Acceleration * Simulation.m1Mass +
Simulation.Friction
            ElseIf GravityBox.HandleInput() = "Entered" And GravityBox.Text <> "" Then
                Simulation.g = CDec(GravityBox.Text)
            Else
                ChangeOccured = False
            End If

            'Variables entered
            If ChangeOccured = True Then
                Simulation.ResetVariables()
                GetValuesFromSim()
            End If
        End If
    End Sub

    Private Sub GetValuesFromSim()
        'values from simulation into boxes
        m1MassBox.Text = Round(Simulation.m1Mass, 2)
        m2MassBox.Text = Round(Simulation.m2Mass, 2)
        XDistanceBox.Text = Round(Simulation.m1X, 2)
        FrictionBox.Text = Round(Simulation.Friction, 2)
        YDistanceBox.Text = Round(Simulation.yDist - Simulation.m2Y, 2)
        VelocityBox.Text = Round(Simulation.Velocity, 2)
        GravityBox.Text = Round(Simulation.g, 2)
        AccelerationBox.Text = Round(Simulation.Acceleration, 2)
        TimeBox.Text = Round(Simulation.T, 2)
        TensionBox.Text = Round(Simulation.Tension, 2)
    End Sub
```

```
    Public Overrides Sub Update()
        Simulation.Update()

        If Simulation.Enabled = True Then
            GetValuesFromSim()
        End If
    End Sub

    Public Overrides Sub Draw()
        Dim TempY As Integer

        'MAIN TITLE BAR
        'Title
        Main.GFX.DrawString("Resolving Forces", Main.Arial_30_Bold, New
SolidBrush(Color.FromArgb(0, 90, 194)), New Point(261 -
Main.GFX.MeasureString("Resolving Forces", Main.Arial_30_Bold).Width \ 2, 25))
        'Simulation control buttons
        PlayButton.Draw()
        PauseButton.Draw()
        StopButton.Draw()
        'Other buttons
        MenuButton.Draw()
        SettingsButton.Draw()

        'DIVIDING LINES
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(0, 90, 194)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7), New Point(960, 720 * 1 / 7))
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(0, 90, 194)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7 - 2), New Point(960 * 2 / 7, 720))

        'VARIABLE SETTINGS
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15_Bold,
"Mass 1 (m1)")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Mass:")
        m1MassBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Distance to
Pulley:")
        XDistanceBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Friction:")
        FrictionBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15_Bold, "Mass 2
(m2)")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Mass:")
        m2MassBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Distance to
Ground:")
        YDistanceBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15_Bold, "System")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Velocity:")
        VelocityBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15,
"Acceleration:")
        AccelerationBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Gravity:")
        GravityBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Time:")
        TimeBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Tension:")
        TensionBox.Draw()
```

```
        'SIMULATION
        Simulation.Draw()

        'MAIN RECT: 277, 106, 683, 614
    End Sub
End Class
```

ResolvingForcesSimulation

This class purely controls the actual Simulation, and only displays the animation for Resolving Forces. The DrawToCustomImage procedure is used for the Simulation previews on the Simulations Menu. It draws the animation with half the size.



```
Imports System.Math

Public Class ResolvingForcesSimulation
    Inherits BaseScreen

    Public Mode As SimulationMode
    Public Finished As Boolean = False
    Public Visible As Boolean = True

    Public Scale As Double

    Private Size As New Size(683, 614)

    Public g As Double = 9.8

    Public T As Double
    Public Tmicros As Integer
    Public TTimer As Date
    Public Enabled As Boolean

    Public m1Mass, m2Mass, Friction, Acceleration, Velocity, Tension, m1X, m2Y, xDist,
yDist As Single

    Public Sub New(ByVal InputMode As SimulationMode)
        Mode = InputMode
        Name = "ResolvingForcesSimulation"
        State = ScreenState.NoInput
        Location = New Point(277, 106)

        T = 0
        Tmicros = 0

        If Mode = SimulationMode.Simulation Then
            'Default values for variables in simulation mode
```

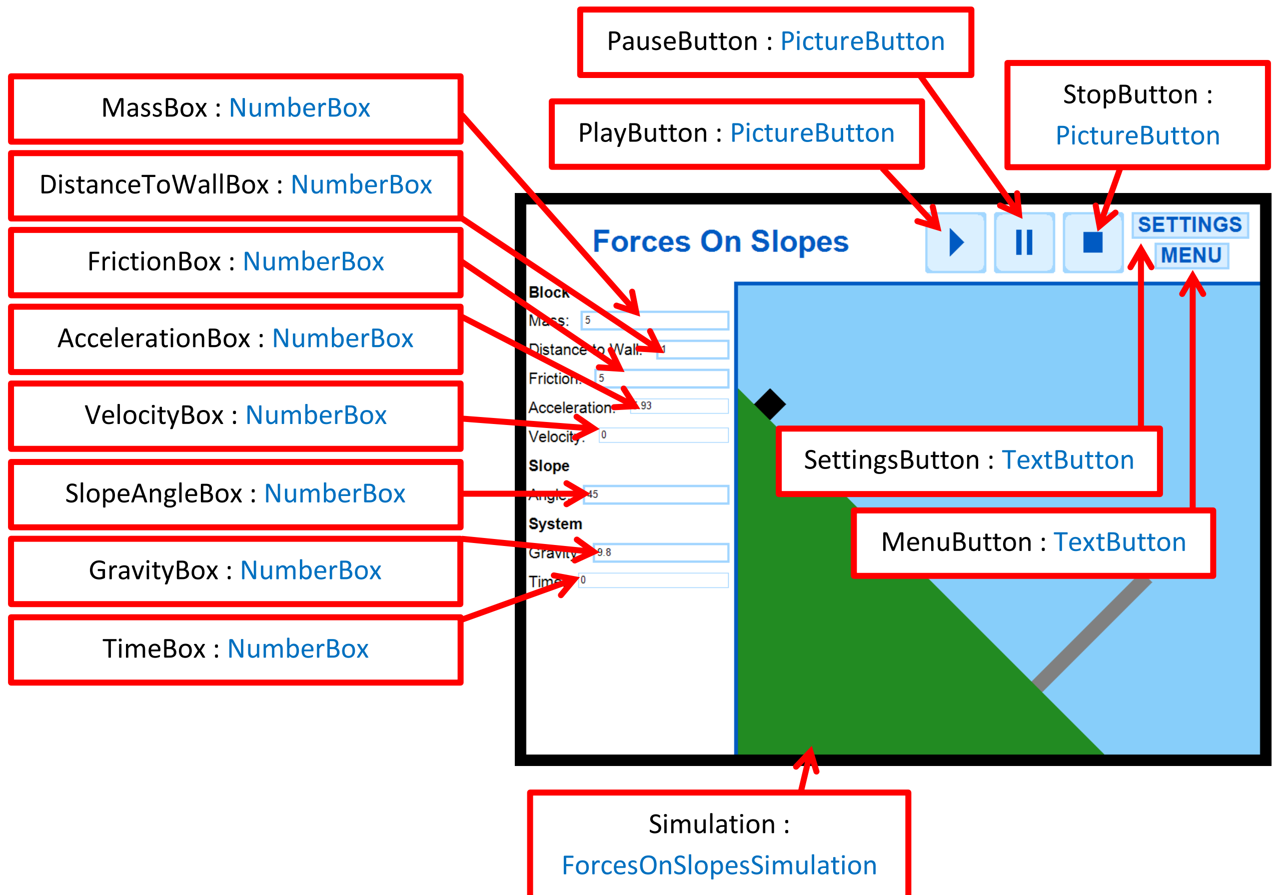
```
            m1Mass = 5
            m2Mass = 2
            'Friction needs to be less than m2Mass * g
            Friction = 15
            xDist = 1
            yDist = xDist * 0.8
            ResetVariables()
        End If
    End Sub

    Public Sub ResetVariables()
        'Set variables to their correct initial conditions
        Scale = 250 / xDist

        m1X = xDist
        m2Y = 0
        Velocity = 0

        Acceleration = (m2Mass * g - Friction) / (m1Mass + m2Mass)
        Tension = Acceleration * m1Mass + Friction

        'Friction = m2Mass * g - m1Mass * Acceleration - m2Mass * Acceleration
        'Acceleration = (Tension - Friction) / m1Mass

        Tmicros = 0
        T = 0
    End Sub

    Public Sub SetTestVariables(ByVal InputM1Mass As Single, ByVal InputM2Mass As
Single, ByVal InputFriction As Single, ByVal InputXDist As Single)
        'Allows input of variables other than the defualt.
        'This is needed for the test mode
        m1Mass = InputM1Mass
        m2Mass = InputM2Mass
        Friction = InputFriction
        xDist = InputXDist
        yDist = xDist * 0.8

        ResetVariables()
    End Sub

    Public Function Metres(ByVal Pixels As Double) As Double
        Return Pixels / Scale
    End Function
    Public Function Pixels(ByVal Metres As Double) As Double
        Return Metres * Scale
    End Function

    Public Overrides Sub Update()
        Dim NewM1X, NewM2Y As Single

        If Enabled = True Then
            If (Now - TTimer).TotalMilliseconds > 25 Then
                TTimer = Now

                'Every 25 milliseconds (ish)

                'Gradually increase the time variable
                'Calculate the expected position as if no collision happens, then
                'see if there should be a collision
                For i = 1 To 10000
```

```
                    NewM1X = xDist - 0.5 * Acceleration * T ^ 2
                    NewM2Y = 0.5 * Acceleration * T ^ 2
                    Velocity = Acceleration * T

                    If Velocity > 0 Then
                        If NewM2Y >= yDist Then
                            'm2 reaches floor, so stop
                            Velocity = 0
                            Acceleration = 0
                            m2Y = yDist
                            m1X = xDist * 0.2
                            Finished = True
                        Else
                            'no collision, so continue as usual
                            m1X = NewM1X
                            m2Y = NewM2Y
                        End If
                    End If

                    Tmicros += 1
                    T = Tmicros / 1000000
                Next
            End If
        End If
    End Sub

    Public Overrides Sub Draw()
        Dim Centre As New Point(Location.X + Size.Width \ 2, Location.Y + Size.Height
\ 2)
        'SKY
        Main.GFX.FillRectangle(Brushes.LightSkyBlue, Location.X, Location.Y,
Size.Width, Size.Height)
        'PULLEY
        Main.GFX.DrawLine(New Pen(Brushes.Black, 10), Centre.X + 10, Centre.Y + 10,
Centre.X - 15, Centre.Y - 15)
        Main.GFX.FillEllipse(Brushes.Brown, Centre.X - 40, Centre.Y - 40, 30, 30)
        Main.GFX.FillEllipse(Brushes.Silver, Centre.X - 30, Centre.Y - 30, 10, 10)
        'TABLE
        Main.GFX.FillRectangle(Brushes.ForestGreen, Centre.X, Centre.Y, Size.Width -
Size.Width \ 2, Size.Height - Size.Height \ 2)

        'MASSES
        'm1
        Main.GFX.FillRectangle(Brushes.Gray, CInt(Round(630 + Pixels(m1X))), Centre.Y
- 80, 70, 80)
        Main.GFX.DrawString("m1: " & m1Mass & "kg", Main.Arial_8, Brushes.White,
CInt(Round(630 + Pixels(m1X))), Centre.Y - 80)
        'm2
        Main.GFX.FillRectangle(Brushes.Gray, Centre.X - 70, CInt(Round(430 +
Pixels(m2Y))), 60, 50)
        Main.GFX.DrawString("m2: " & m2Mass & "kg", Main.Arial_8, Brushes.White,
Centre.X - 70, CInt(Round(430 + Pixels(m2Y))))
        'STRINGS
        Main.GFX.DrawLine(New Pen(Brushes.Goldenrod, 2), Centre.X - 25, Centre.Y - 39,
CInt(Round(630 + Pixels(m1X))), Centre.Y - 39)
        Main.GFX.DrawArc(New Pen(Brushes.Goldenrod, 2), Centre.X - 40, Centre.Y - 40,
30, 30, 180, 90)
        Main.GFX.DrawLine(New Pen(Brushes.Goldenrod, 2), Centre.X - 39, Centre.Y - 25,
Centre.X - 39, CInt(Round(430 + Pixels(m2Y))))
        'FLOOR
```

```
        Main.GFX.FillRectangle(Brushes.ForestGreen, Location.X, CInt(480 +
Pixels(yDist)), Size.Width \ 2, Location.Y + Size.Height - CInt(480 + Pixels(yDist)))
    End Sub

    Public Sub DrawToCustomImage(ByRef BMP As Image)
        'Used for drawing the simulation when it is used as a preview on the
simulation menu
        Dim Centre As New Point(Size.Width \ 4, Size.Height \ 4)
        'SKY
        Graphics.FromImage(BMP).FillRectangle(Brushes.LightSkyBlue, 0, 0, Size.Width \
2, Size.Height \ 2)
        'PULLEY
        Graphics.FromImage(BMP).DrawLine(New Pen(Brushes.Black, 10), Centre.X + 10 \
2, Centre.Y + 10 \ 2, Centre.X - 15 \ 2, Centre.Y - 15 \ 2)
        Graphics.FromImage(BMP).FillEllipse(Brushes.Brown, Centre.X - 40 \ 2, Centre.Y
- 40 \ 2, 30 \ 2, 30 \ 2)
        Graphics.FromImage(BMP).FillEllipse(Brushes.Silver, Centre.X - 30 \ 2,
Centre.Y - 30 \ 2, 10 \ 2, 10 \ 2)
        'TABLE
        Graphics.FromImage(BMP).FillRectangle(Brushes.ForestGreen, Centre.X, Centre.Y,
(Size.Width - Size.Width \ 2) \ 2, (Size.Height - Size.Height \ 2) \ 2)

        'MASSES
        'm1
        Graphics.FromImage(BMP).FillRectangle(Brushes.Gray, Centre.X +
Round(Pixels(m1X)) \ 2, Centre.Y - 80 \ 2, 70 \ 2, 80 \ 2)
        'm2
        Graphics.FromImage(BMP).FillRectangle(Brushes.Gray, Centre.X - 70 \ 2,
Centre.Y + 20 + Round(Pixels(m2Y)) \ 2, 60 \ 2, 50 \ 2)
        'STRINGS
        Graphics.FromImage(BMP).DrawLine(New Pen(Brushes.Goldenrod, 2), Centre.X - 25
\ 2, Centre.Y - 39 \ 2, Centre.X + Round(Pixels(m1X)) \ 2, Centre.Y - 39 \ 2)
        Graphics.FromImage(BMP).DrawArc(New Pen(Brushes.Goldenrod, 2), Centre.X - 40 \
2, Centre.Y - 40 \ 2, 30 \ 2, 30 \ 2, 180 \ 2, 90 \ 2)
        Graphics.FromImage(BMP).DrawLine(New Pen(Brushes.Goldenrod, 2), Centre.X - 39
\ 2, Centre.Y - 25 \ 2, Centre.X - 39 \ 2, Centre.Y + 20 + Round(Pixels(m2Y)) \ 2)
        'FLOOR
        Graphics.FromImage(BMP).FillRectangle(Brushes.ForestGreen, 0, (480 +
Pixels(yDist)) \ 2, Size.Width \ 4, (Size.Height - (480 + Pixels(yDist))) \ 2)
    End Sub
End Class
```

ForcesOnSlopes

This screen is for controlling the Forces On Slopes Simulation. There are variables for changing various aspects of the Simulation. There are also buttons for the Play, Pause and Stop/Reset commands. The main feature of this screen is its instance of the ForcesOnSlopesSimulation, which is responsible for the actual Simulation, as well as the animation.




```
Imports System.Math

Public Class ForcesOnSlopes
    Inherits BaseScreen

    Private MenuButton As New TextButton("MENU", Main.Arial_20_Bold,
ProgramSection.Simulation, New Point(822, 50), -1, 35, 3, 1)
    Private SettingsButton As New TextButton("SETTINGS", Main.Arial_20_Bold,
ProgramSection.Simulation, New Point(792, 10), -1, 35, 3, 1)

    Private PlayButton As New PictureButton(New Point(522, 10),
My.Resources.PlayDefault, My.Resources.PlayHover, My.Resources.PlayDown, -1, -1)
    Private PauseButton As New PictureButton(New Point(612, 10),
My.Resources.PauseDefault, My.Resources.PauseHover, My.Resources.PauseDown, -1, -1)
    Private StopButton As New PictureButton(New Point(702, 10),
My.Resources.StopDefault, My.Resources.StopHover, My.Resources.StopDown, -1, -1)

    Private Simulation As New ForcesOnSlopesSimulation(SimulationMode.Simulation)

    Private MassBox, FrictionBox, DistanceToWallBox, SlopeAngleBox, AccelerationBox,
GravityBox, VelocityBox, TimeBox As NumberBox

    Public Sub New()
        Dim TempY, TempX As Integer

        Name = "ForcesOnSlopes"
        State = ScreenState.Active
        Location = New Point(0, 0)

        'Create the input boxes in the correct positions for each variable
```

```
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15_Bold,
"Block", False)
        TempX = Main.GFX.MeasureString("Mass:", Main.Arial_15).Width + 10
        MassBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Mass:", False)

        TempX = Main.GFX.MeasureString("Distance to Wall:", Main.Arial_15).Width + 10
        DistanceToWallBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Distance to
Wall:", False)

        TempX = Main.GFX.MeasureString("Friction:", Main.Arial_15).Width + 10
        FrictionBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Friction:",
False)

        TempX = Main.GFX.MeasureString("Acceleration:", Main.Arial_15).Width + 10
        AccelerationBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 1, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15,
"Acceleration:", False)

        TempX = Main.GFX.MeasureString("Velocity:", Main.Arial_15).Width + 10
        VelocityBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 1, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Velocity:",
False)

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15_Bold, "Slope",
False)
        TempX = Main.GFX.MeasureString("Angle:", Main.Arial_15).Width + 10
        SlopeAngleBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Angle:",
False)

        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15_Bold, "System",
False)
        TempX = Main.GFX.MeasureString("Gravity:", Main.Arial_15).Width + 10
        GravityBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 3, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Gravity:",
False)

        TempX = Main.GFX.MeasureString("Time:", Main.Arial_15).Width + 10
        TimeBox = New NumberBox(New Point(TempX, TempY), Main.Arial_10,
ProgramSection.Simulation, 1, 960 * 2 / 7 - TempX - 10)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Time:", False)

        GetValuesFromSim()
    End Sub

    Public Overrides Sub HandleInput()
        Dim ChangeOccured As Boolean = True
        If MenuButton.Clicked = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New SimulationMenu)
        End If
```

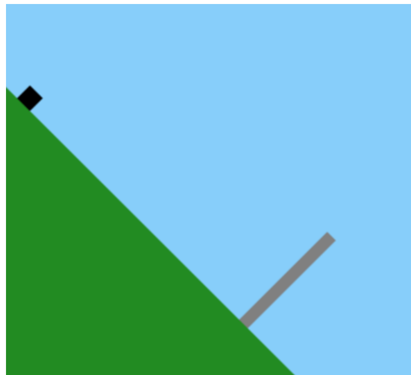
```
        If SettingsButton.Clicked() = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New Settings({New ForcesOnSlopes}))
        End If

        'Check input for play, pause, stop
        If PlayButton.Clicked() = "Clicked" Then
            Simulation.TTimer = Now
            Simulation.Enabled = True
        ElseIf PauseButton.Clicked() = "Clicked" Then
            Simulation.Enabled = False
        ElseIf StopButton.Clicked() = "Clicked" Then
            Simulation.Enabled = False
            Simulation.ResetVariables()
            GetValuesFromSim()
        End If
        For Each Key In Main.KeysDown
            If Key = 32 Then
                'Space Bar
                If Simulation.Enabled = True Then
                    Simulation.Enabled = False
                Else
                    Simulation.TTimer = Now
                    Simulation.Enabled = True
                End If
            End If
        Next

        If Simulation.Enabled = False And Simulation.T = 0 Then
            'Check for input form variable input boxes, then update the simulation
with the values.
            'Sometimes when one value changes, other dependent values must change too.
            If MassBox.HandleInput = "Entered" And MassBox.Text <> "" Then
                Simulation.Mass = CDec(MassBox.Text)
            ElseIf DistanceToWallBox.HandleInput = "Entered" And
DistanceToWallBox.Text <> "" Then
                If DistanceToWallBox.Text <> "0" Then
                    Simulation.SlopeDistance = CDec(DistanceToWallBox.Text)
                Else
                    Main.MessageBox("Error: Distance to Wall cannot be 0")
                End If
            ElseIf FrictionBox.HandleInput = "Entered" And FrictionBox.Text <> "" Then
                'Friction > Mass * g * Sin(Main.Rad(SlopeAngle))
                If CDec(FrictionBox.Text) > Simulation.Mass * Simulation.g *
Sin(Main.Rad(Simulation.SlopeAngle)) Then
                    Main.MessageBox("Error: Friction cannot be that high")
                End If
                Simulation.Friction = CDec(FrictionBox.Text)
            ElseIf SlopeAngleBox.HandleInput = "Entered" And SlopeAngleBox.Text <> ""
Then
                If CDec(SlopeAngleBox.Text) >= 0 And CDec(SlopeAngleBox.Text) <= 90
Then
                    Simulation.SlopeAngle = CDec(SlopeAngleBox.Text)
                Else
                    Main.MessageBox("Error: Slope angle must be between 0 and 90
degrees")
                End If
            ElseIf GravityBox.HandleInput = "Entered" And GravityBox.Text <> "" Then
                Simulation.g = CDec(GravityBox.Text)
            Else
                ChangeOccured = False
```

```
            End If

            'Variables entered
            If ChangeOccured = True Then
                Simulation.ResetVariables()
                GetValuesFromSim()
            End If
        End If
    End Sub

    Private Sub GetValuesFromSim()
        'values from simulation into boxes
        MassBox.Text = Round(Simulation.Mass, 2)
        DistanceToWallBox.Text = Round(Simulation.SlopeDistance -
Simulation.Displacement, 2)
        FrictionBox.Text = Round(Simulation.Friction, 2)
        AccelerationBox.Text = Round(Simulation.Acceleration, 2)
        VelocityBox.Text = Round(Simulation.Velocity, 2)
        SlopeAngleBox.Text = Round(Simulation.SlopeAngle, 2)
        GravityBox.Text = Round(Simulation.g, 2)
        TimeBox.Text = Round(Simulation.T, 2)
    End Sub

    Public Overrides Sub Update()
        Simulation.Update()

        If Simulation.Enabled = True Then
            GetValuesFromSim()
        End If
    End Sub

    Public Overrides Sub Draw()
        Dim TempY As Integer = 0

        'MAIN TITLE BAR
        'Title
        Main.GFX.DrawString("Forces On Slopes", Main.Arial_30_Bold, New
SolidBrush(Color.FromArgb(0, 90, 194)), New Point(261 - Main.GFX.MeasureString("Forces
On Slopes", Main.Arial_30_Bold).Width \ 2, 25))
        'Simulation control buttons
        PlayButton.Draw()
        PauseButton.Draw()
        StopButton.Draw()
        'Other buttons
        MenuButton.Draw()
        SettingsButton.Draw()

        'DIVIDING LINES
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(0, 90, 194)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7), New Point(960, 720 * 1 / 7))
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(0, 90, 194)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7 - 2), New Point(960 * 2 / 7, 720))

        'VARIABLE SETTINGS
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15_Bold,
"Block")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Mass:")
        MassBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Distance to
Wall:")
        DistanceToWallBox.Draw()
```

```
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Friction:")
        FrictionBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15,
"Acceleration:")
        AccelerationBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Velocity:")
        VelocityBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15_Bold, "Slope")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Angle:")
        SlopeAngleBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15_Bold, "System")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Gravity:")
        GravityBox.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "Time:")
        TimeBox.Draw()

        'SIMULATION
        Simulation.Draw()

        'MAIN RECT: 277, 106, 683, 614
    End Sub
End Class
```

ForcesOnSlopesSimulation

This class purely controls the actual Simulation, and only displays the animation for Forces On Slopes. The DrawToCustomImage procedure is used for the Simulation previews on the Simulations Menu. It draws the animation with half the size.



```
Imports System.Math

Public Class ForcesOnSlopesSimulation
    Inherits BaseScreen

    Public Mode As SimulationMode
    Public Finished As Boolean = False
    Public Visible As Boolean = True

    Public Scale As Double

    Private Size As New Size(683, 614)

    Private CritAngle As Single = Main.Deg(Atan(614 / 480))

    Public g As Double = 9.8

    Public T As Double
```

```
    Public Tmicros As Integer
    Public TTimer As Date
    Public Enabled As Boolean

    Public Mass, Acceleration, Velocity, SlopeAngle, Displacement, Friction,
WallPosition, WallHeight As Single
    Public SlopeX, SlopeY, SlopeLength, SlopeDistance As Single

    Public Sub New(ByVal InputMode As SimulationMode)
        Mode = InputMode
        Name = "ForcesOnSlopesSimulation"
        State = ScreenState.NoInput
        Location = New Point(277, 106)

        T = 0
        Tmicros = 0
        WallHeight = 203

        If Mode = SimulationMode.Simulation Then
            'Set default vlaues for simulation mode
            Mass = 5
            SlopeAngle = 45
            'Friction must be less than mgsin(angle)
            Friction = 5
            SlopeDistance = 1

            ResetVariables()
        End If
    End Sub

    Public Sub ResetVariables()
        'Set variables to their correct initial conditions

        If SlopeAngle <= CritAngle Then
            SlopeX = 480
            SlopeY = SlopeX * Tan(Main.Rad(SlopeAngle))
        Else
            SlopeY = 614
            SlopeX = 614 / Tan(Main.Rad(SlopeAngle))
        End If
        SlopeLength = Sqrt(SlopeX ^ 2 + SlopeY ^ 2)

        Scale = (0.8 * SlopeLength - 60) / SlopeDistance

        If Friction > Mass * g * Sin(Main.Rad(SlopeAngle)) Then
            Friction = Mass * g * Sin(Main.Rad(SlopeAngle))
        End If

        Acceleration = (Mass * g * Sin(Main.Rad(SlopeAngle)) - Friction) / Mass
        Friction = Mass * g * Sin(Main.Rad(SlopeAngle)) - Mass * Acceleration

        Displacement = 0
        Velocity = 0

        Tmicros = 0
        T = 0
    End Sub

    Public Sub SetTestVariables(ByVal InputMass As Single, ByVal InputDistanceToWall
As Single, ByVal InputFriction As Single, ByVal InputSlopeAngle As Single)
        'Allows input of variables other than the defualt.
```

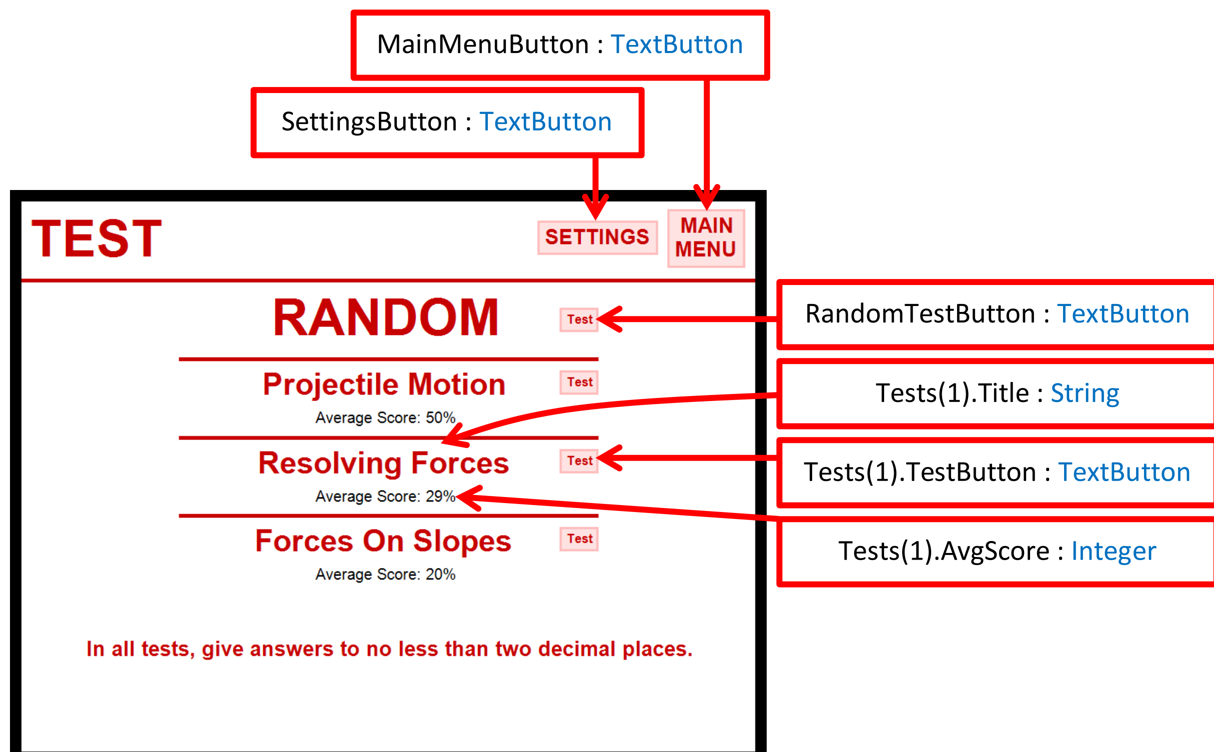
```
        'This is needed for the test mode

        Mass = InputMass
        SlopeDistance = InputDistanceToWall
        Friction = InputFriction
        SlopeAngle = InputSlopeAngle

        ResetVariables()
    End Sub

    Public Function Metres(ByVal Pixels As Double) As Double
        Return Pixels / Scale
    End Function
    Public Function Pixels(ByVal Metres As Double) As Double
        Return Metres * Scale
    End Function

    Public Overrides Sub Update()
        Dim NewDisplacement As Single

        If Enabled = True Then
            If (Now - TTimer).TotalMilliseconds > 25 Then
                TTimer = Now
                'Every 25 milliseconds (ish)

                'Gradually increase the time variable
                'Calculate the expected position as if no collision happens, then
                'see if there should be a collision
                For i = 1 To 10000
                    NewDisplacement = 0.5 * Acceleration * T ^ 2
                    Velocity = Acceleration * T

                    If Velocity > 0 Then
                        If NewDisplacement >= SlopeDistance Then
                            'Hits Wall
                            Velocity = 0
                            Displacement = SlopeDistance
                            Acceleration = 0
                            Finished = True
                        Else
                            Displacement = NewDisplacement
                        End If
                    End If

                    Tmicros += 1
                    T = Tmicros / 1000000
                Next
            End If
        End If
    End Sub

    Public Overrides Sub Draw()
        Dim SlopePoints(3), WallPoints(3), MassPoints(3) As Point
        'SKY
        Main.GFX.FillRectangle(Brushes.LightSkyBlue, Location.X, Location.Y,
Size.Width, Size.Height)
        'SLOPE
        If SlopeX = 480 Then
            SlopePoints(0) = New Point(Location.X + 480, Location.Y + Size.Height)
            SlopePoints(1) = New Point(Location.X, Location.Y + Size.Height)
            SlopePoints(2) = New Point(Location.X, Location.Y + Size.Height - SlopeY)
```

```
            SlopePoints(3) = New Point(Location.X, Location.Y + Size.Height - SlopeY)
        ElseIf SlopeX < 480 Then
            SlopePoints(0) = New Point(Location.X + 480, Location.Y + Size.Height)
            SlopePoints(1) = New Point(Location.X, Location.Y + Size.Height)
            SlopePoints(2) = New Point(Location.X, Location.Y)
            SlopePoints(3) = New Point(Location.X + 480 - SlopeX, Location.Y)
        End If
        Main.GFX.FillPolygon(Brushes.ForestGreen, SlopePoints)
        'WALL
        WallPoints(0) = New Point(Location.X + 480 - (0.2 * SlopeLength - 20) *
Cos(Main.Rad(SlopeAngle)), Location.Y + Size.Height - (0.2 * SlopeLength - 20) *
Sin(Main.Rad(SlopeAngle)))
        WallPoints(1) = New Point(Location.X + 480 - 0.2 * SlopeLength *
Cos(Main.Rad(SlopeAngle)), Location.Y + Size.Height - 0.2 * SlopeLength *
Sin(Main.Rad(SlopeAngle)))
        WallPoints(2) = New Point(WallPoints(1).X + WallHeight * Cos(Main.Rad(90 -
SlopeAngle)), WallPoints(1).Y - WallHeight * Sin(Main.Rad(90 - SlopeAngle)))
        WallPoints(3) = New Point(WallPoints(0).X + WallHeight * Cos(Main.Rad(90 -
SlopeAngle)), WallPoints(0).Y - WallHeight * Sin(Main.Rad(90 - SlopeAngle)))
        Main.GFX.FillPolygon(Brushes.Gray, WallPoints)
        'MASS - 50 pixels along slope
        MassPoints(0) = New Point(WallPoints(1).X - (0.8 * SlopeLength - 60 -
Pixels(Displacement)) * Cos(Main.Rad(SlopeAngle)), WallPoints(1).Y - (0.8 *
SlopeLength - 60 - Pixels(Displacement)) * Sin(Main.Rad(SlopeAngle)))
        MassPoints(1) = New Point(WallPoints(1).X - (0.8 * SlopeLength - 30 -
Pixels(Displacement)) * Cos(Main.Rad(SlopeAngle)), WallPoints(1).Y - (0.8 *
SlopeLength - 30 - Pixels(Displacement)) * Sin(Main.Rad(SlopeAngle)))
        MassPoints(2) = New Point(MassPoints(1).X + 30 * Cos(Main.Rad(90 -
SlopeAngle)), MassPoints(1).Y - 30 * Sin(Main.Rad(90 - SlopeAngle)))
        MassPoints(3) = New Point(MassPoints(0).X + 30 * Cos(Main.Rad(90 -
SlopeAngle)), MassPoints(0).Y - 30 * Sin(Main.Rad(90 - SlopeAngle)))
        Main.GFX.FillPolygon(Brushes.Black, MassPoints)
    End Sub

    Public Sub DrawToCustomImage(ByRef BMP As Image)
        'Used for drawing the simulation when it is used as a preview on the
simulation menu
        Dim SlopePoints(3), WallPoints(3), MassPoints(3) As Point
        'SKY
        Graphics.FromImage(BMP).FillRectangle(Brushes.LightSkyBlue, 0, 0, Size.Width \
2, Size.Height \ 2)
        'SLOPE
        If SlopeX = 480 Then
            SlopePoints(0) = New Point(240, Size.Height / 2)
            SlopePoints(1) = New Point(0, Size.Height / 2)
            SlopePoints(2) = New Point(0, (Size.Height - SlopeY) / 2)
            SlopePoints(3) = New Point(0, (Size.Height - SlopeY) / 2)
        ElseIf SlopeX < 480 Then
            SlopePoints(0) = New Point(240, Size.Height / 2)
            SlopePoints(1) = New Point(0, Size.Height / 2)
            SlopePoints(2) = New Point(0, 0)
            SlopePoints(3) = New Point((480 - SlopeX) / 2, 0)
        End If
        Graphics.FromImage(BMP).FillPolygon(Brushes.ForestGreen, SlopePoints)
        'WALL
        WallPoints(0) = New Point((480 - (0.2 * SlopeLength - 20) *
Cos(Main.Rad(SlopeAngle))) / 2, (Size.Height - (0.2 * SlopeLength - 20) *
Sin(Main.Rad(SlopeAngle))) / 2)
        WallPoints(1) = New Point((480 - 0.2 * SlopeLength *
Cos(Main.Rad(SlopeAngle))) / 2, (Size.Height - 0.2 * SlopeLength *
Sin(Main.Rad(SlopeAngle))) / 2)
```

```
        WallPoints(2) = New Point(WallPoints(1).X + (WallHeight * Cos(Main.Rad(90 - SlopeAngle))) / 2, WallPoints(1).Y - (WallHeight * Sin(Main.Rad(90 - SlopeAngle))) / 2)
        WallPoints(3) = New Point(WallPoints(0).X + (WallHeight * Cos(Main.Rad(90 - SlopeAngle))) / 2, WallPoints(0).Y - (WallHeight * Sin(Main.Rad(90 - SlopeAngle))) / 2)
        Graphics.FromImage(BMP).FillPolygon(Brushes.Gray, WallPoints)
        'MASS - 50 pixels along slope
        MassPoints(0) = New Point(WallPoints(1).X - ((0.8 * SlopeLength - 60 - Pixels(Displacement)) * Cos(Main.Rad(SlopeAngle))) / 2, WallPoints(1).Y - ((0.8 * SlopeLength - 60 - Pixels(Displacement)) * Sin(Main.Rad(SlopeAngle))) / 2)
        MassPoints(1) = New Point(WallPoints(1).X - ((0.8 * SlopeLength - 30 - Pixels(Displacement)) * Cos(Main.Rad(SlopeAngle))) / 2, WallPoints(1).Y - ((0.8 * SlopeLength - 30 - Pixels(Displacement)) * Sin(Main.Rad(SlopeAngle))) / 2)
        MassPoints(2) = New Point(MassPoints(1).X + (30 * Cos(Main.Rad(90 - SlopeAngle))) / 2, MassPoints(1).Y - (30 * Sin(Main.Rad(90 - SlopeAngle))) / 2)
        MassPoints(3) = New Point(MassPoints(0).X + (30 * Cos(Main.Rad(90 - SlopeAngle))) / 2, MassPoints(0).Y - (30 * Sin(Main.Rad(90 - SlopeAngle))) / 2)
        Graphics.FromImage(BMP).FillPolygon(Brushes.Black, MassPoints)
    End Sub
End Class
```

TestMenu

This screen presents each category of Test. Each Test is a list item in Tests() and each one has a title, average score, and selection button. These attributes are in the structure TestInfo. When the screen is loaded, the User's text file is processed to find the average score for each of their Test categories, and to see if there are any categories that they haven't yet completed a Test for.




```
Imports System.IO

Public Class TestMenu
    Inherits BaseScreen
```

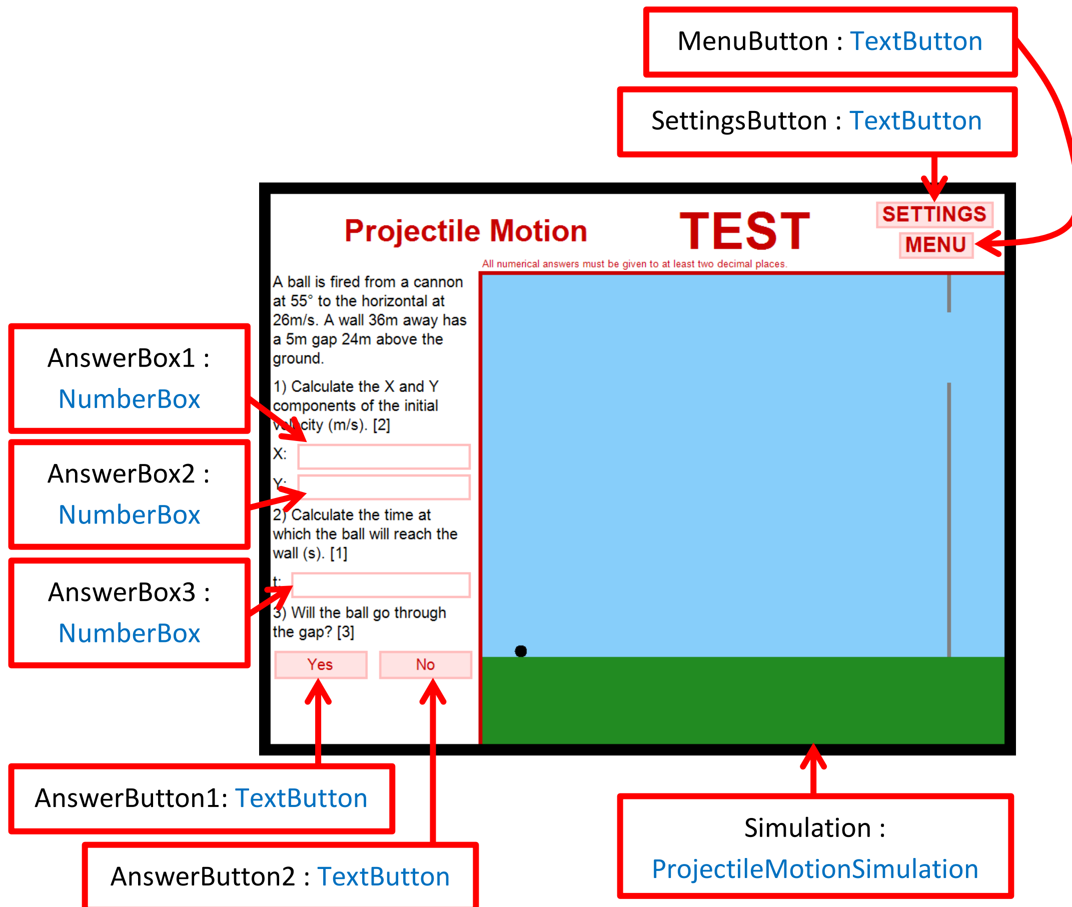
```
    Private Const TestInfoHeight As Single = 720 * 1 / 7

    Private MainMenuButton As New TextButton(" MAIN" & vbNewLine & "MENU",
Main.Arial_20_Bold, ProgramSection.Test, New Point(845, 10), -1, -1, 3)
    Private SettingsButton As New TextButton("SETTINGS", Main.Arial_20_Bold,
ProgramSection.Test, New Point(675, 25), -1, -1, 3)
    Private RandomTestButton As New TextButton("Test", Main.Arial_12_Bold,
ProgramSection.Test, New Point(704, TestInfoHeight + 36), -1, -1, 3)

    Private Structure TestInfo
        Dim Title As String
        Dim AvgScore As Integer
        Dim TestButton As TextButton
        Dim Location As Point
        Dim Enabled As Boolean
    End Structure

    Private Tests(4) As TestInfo
    Private NumOfTests As Integer = 0

    Public Sub New()
        'Read and decrypt the user's text file
        Dim UserContent As String =
Main.DecryptString(File.ReadAllText(Main.CurrentUser & ".sv"))
        If UserContent.Length > 0 Then
            UserContent = UserContent.Substring(0, UserContent.Length - 1)
        End If

        Name = "TestMenu"
        State = ScreenState.Active
        Location = New Point(0, 0)

        'Set the titles of the tests I want to display
        Tests(0).Title = "Projectile Motion"
        Tests(1).Title = "Resolving Forces"
        Tests(2).Title = "Forces On Slopes"

        'Set up all tests
        For y = 0 To 4
            If Tests(y).Title <> Nothing Then
                Tests(y).Enabled = True
            End If
            Tests(y).Location = New Point(TestInfoHeight, (2 + y) * TestInfoHeight)
            Tests(y).TestButton = New TextButton("Test", Main.Arial_12_Bold,
ProgramSection.Test, New Point(704, Tests(y).Location.Y + 15), -1, -1, 3)
            If Tests(y).Enabled = True Then
                'Get average score from user file. Set to -1 if not yet completed.

                'Process user file to find average score for each category of test.
                If InStr(UserContent, Tests(y).Title) = 0 Then
                    'Test title is not found in user file. This means they haven't yet
completed a test of that type yet.
                    Tests(y).AvgScore = -1
                Else
                    'Split user file content into tests, then look at the right ones
                    Dim strTestResults() As String = Split(UserContent, "|")
                    Dim TotalScore As Integer = 0
                    Dim NumScores As Integer = 0

                    'Add up all scores of the category
                    For Each strTestResult In strTestResults
```

```
                    If Split(strTestResult, ",")(0) = Tests(y).Title Then
                        TotalScore += Split(strTestResult, ",")(1)
                        NumScores += 1
                    End If
                Next

                'find the average
                Tests(y).AvgScore = TotalScore / NumScores
            End If

            NumOfTests += 1
        End If
    Next
End Sub

Public Overrides Sub HandleInput()
    Dim Result As String = ""

    If MainMenuButton.Clicked = "Clicked" Then
        ScreenManager.UnloadScreen(Name)
        ScreenManager.AddScreen(New Title)
        ScreenManager.AddScreen(New SimulationButton)
        ScreenManager.AddScreen(New TestButton)
        ScreenManager.AddScreen(New MyProgressButton)
    End If
    If SettingsButton.Clicked() = "Clicked" Then
        ScreenManager.UnloadScreen(Name)
        ScreenManager.AddScreen(New Settings({New TestMenu}))
    End If

    If RandomTestButton.Clicked = "Clicked" Then
        'Random Test
        Result = Tests(Main.Rand.Next(0, NumOfTests)).Title
    End If
    For y = 0 To 4
        If Tests(y).TestButton.Clicked = "Clicked" And Tests(y).Enabled = True
Then
            'Specific Test
            Result = Tests(y).Title
        End If
    Next
    If Result <> "" Then
        ScreenManager.UnloadScreen(Name)
        Select Case Result
            Case "Projectile Motion"
                ScreenManager.AddScreen(New ProjectileMotionTest)
            Case "Resolving Forces"
                ScreenManager.AddScreen(New ResolvingForcesTest)
            Case "Forces On Slopes"
                ScreenManager.AddScreen(New ForcesOnSlopesTest)
        End Select
    End If

End Sub

Public Overrides Sub Draw()
    Dim Output As String = ""

    'MAIN TITLE BAR
    'Title
```

```
        Main.GFX.DrawString("TEST", Main.Arial_50_Bold, New
SolidBrush(Color.FromArgb(199, 0, 0)), New Point(0, 10))
        'Buttons
        MainMenuButton.Draw()
        SettingsButton.Draw()

        'DIVIDING LINES
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(199, 0, 0)), 5), 0,
TestInfoHeight, 960, TestInfoHeight)

        'RANDOM TEST
        'Title
        Main.GFX.DrawString("RANDOM", Main.Arial_50_Bold, New
SolidBrush(Color.FromArgb(199, 0, 0)), New Point(2 * TestInfoHeight + 110,
TestInfoHeight + 10))
        'Button
        RandomTestButton.Draw()
        'OTHER TESTS
        For y = 0 To 4
            If Tests(y).Enabled = True Then
                'Line Above
                Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(199, 0, 0)),
5), 2 * TestInfoHeight, Tests(y).Location.Y, 960 - 2 * TestInfoHeight,
Tests(y).Location.Y)
                'Title
                Main.GFX.DrawString(Tests(y).Title, Main.Arial_30_Bold, New
SolidBrush(Color.FromArgb(199, 0, 0)), New Point(480 -
Main.GFX.MeasureString(Tests(y).Title, Main.Arial_30_Bold).Width \ 2,
Tests(y).Location.Y + 10))
                'Avg Score
                If Tests(y).AvgScore = -1 Then
                    Output = "You have not completed a test for " & Tests(y).Title & "
yet."
                Else
                    Output = "Average Score: " & Tests(y).AvgScore & "%"
                End If
                Main.GFX.DrawString(Output, Main.Arial_15, Brushes.Black, New
Point(480 - Main.GFX.MeasureString(Output, Main.Arial_15).Width \ 2,
Tests(y).Location.Y + 65))
                'Button
                Tests(y).TestButton.Draw()
            End If
        Next
        '2dp warning
        Main.GFX.DrawString("In all tests, give answers to no less than two decimal
places.", Main.Arial_20_Bold, New SolidBrush(Color.FromArgb(199, 0, 0)), 480 -
Main.GFX.MeasureString("In all tests, give answers to no less than two decimal
places.", Main.Arial_20_Bold).Width \ 2, 570)
    End Sub
End Class
```

ProjectileMotionTest

This screen is for testing the User on the Projectile Motion category. The question is presented on the left. Once the user gives valid answers, the class's instance of the ProjectileMotionSimulation starts, using the starting variables randomly generated by this class. Then the class's instance of TestReport shows the User's Test results. A diagram showing my plan for this process for any test can be found in the design section on page 15.




```
Public Class ProjectileMotionTest
    Inherits BaseScreen

    Private MenuButton As New TextButton("MENU", Main.Arial_20_Bold,
ProgramSection.Test, New Point(822, 50), -1, 35, 3, 1)
    Private SettingsButton As New TextButton("SETTINGS", Main.Arial_20_Bold,
ProgramSection.Test, New Point(792, 10), -1, 35, 3, 1)

    Private AnswerBox1, AnswerBox2, AnswerBox3 As NumberBox
    Private AnswerButton1, AnswerButton2 As TextButton

    Private CorrectFinalAnswer As String = ""
    Private CorrectTimeAnswer As Decimal
    Private BallReachesWall As Boolean
    Private CorrectButton As Boolean

    Private Simulation As New ProjectileMotionSimulation(SimulationMode.Test)
    Private Report As TestReport

    Private twoDPWarning As Boolean = False

    Private WallHeight, WallGap As Integer
    Private XDistance, Speed, Angle As Single

    Public Sub New()
        Dim TempY, TempX, CurrentQNumber As Integer
        Dim HeightAtWall As Single

        Name = "ProjectileMotionTest"
```

```
        State = ScreenState.Active
        Location = New Point(0, 0)

        'Generate starting variables
        XDistance = Main.Rand.Next(15, 50 + 1)
        WallGap = Main.Rand.Next(3, 5 + 1)
        WallHeight = Main.Rand.Next(WallGap, (XDistance - WallGap) * 0.8 + 1)
        Angle = Main.Rand.Next(20, 60 + 1)
        Speed = Main.Rand.Next(XDistance / 2, XDistance + 1)

        'Plug starting variables into Simulation
        Simulation.SetTestVariables(WallHeight, WallGap, Speed, Angle, XDistance)

        'Calculate correct answers
        CorrectTimeAnswer = Math.Round(XDistance / (Speed *
Math.Cos(Main.Rad(Angle))), 2)
        HeightAtWall = Math.Round(Speed * Math.Sin(Main.Rad(Angle)) *
CorrectTimeAnswer - 4.9 * CorrectTimeAnswer ^ 2, 2)
        If HeightAtWall < 0 Then
            BallReachesWall = False
            CorrectFinalAnswer = "No"
        Else
            BallReachesWall = True
        End If
        If BallReachesWall = True Then
            If HeightAtWall > WallHeight And HeightAtWall < WallHeight + WallGap Then
                CorrectFinalAnswer = "Yes"
            Else
                CorrectFinalAnswer = "No"
            End If
        End If

        CurrentQNumber = 1
        'Create answer boxes in the right places
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15, "A ball
is fired from a cannon at " & Angle & "° to the horizontal at " & Speed & "m/s. A wall
" & XDistance & "m away has a " & WallGap & "m gap " & WallHeight & "m above the
ground.", False)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, CurrentQNumber
& ") Calculate the X and Y components of the initial velocity (m/s). [2]", False)
        CurrentQNumber += 1
        TempX = Main.GFX.MeasureString("X:", Main.Arial_15).Width + 10
        AnswerBox1 = New NumberBox(New Point(TempX, TempY), Main.Arial_15,
ProgramSection.Test, 3, 960 * 2 / 7 - TempX - 14)
        TempY += Main.GFX.MeasureString("X:", Main.Arial_15).Height + 15
        TempX = Main.GFX.MeasureString("Y:", Main.Arial_15).Width + 10
        AnswerBox2 = New NumberBox(New Point(TempX, TempY), Main.Arial_15,
ProgramSection.Test, 3, 960 * 2 / 7 - TempX - 14)
        TempY += Main.GFX.MeasureString("Y:", Main.Arial_15).Height + 15
        If BallReachesWall = True Then
            TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15,
CurrentQNumber & ") Calculate the time at which the ball will reach the wall (s).
[1]", False)
            CurrentQNumber += 1
            TempX = Main.GFX.MeasureString("t:", Main.Arial_15).Width + 10
            AnswerBox3 = New NumberBox(New Point(TempX, TempY), Main.Arial_15,
ProgramSection.Test, 3, 960 * 2 / 7 - TempX - 14)
            TempY += Main.GFX.MeasureString("t:", Main.Arial_15).Height + 15
        Else
            AnswerBox3 = New NumberBox(New Point(TempX, TempY), Main.Arial_15,
ProgramSection.Test, 3, 960 * 2 / 7 - TempX - 14)
```

```
            AnswerBox3.Text = "bleh"
        End If
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, CurrentQNumber
& ") Will the ball go through the gap? [3]", False)
        AnswerButton1 = New TextButton("Yes", Main.Arial_15, ProgramSection.Test, New
Point(5, TempY), 960 * 1 / 7 - 15, -1, 3)
        AnswerButton2 = New TextButton("No", Main.Arial_15, ProgramSection.Test, New
Point(960 * 1 / 7 + 5, TempY), 960 * 1 / 7 - 15, -1, 3)
    End Sub

    Public Overrides Sub HandleInput()
        Dim YesClicked As Boolean = False
        Dim NoClicked As Boolean = False

        If MenuButton.Clicked = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New TestMenu)
        End If
        If SettingsButton.Clicked() = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New Settings({New TestMenu}))
        End If

        'Handle answers input
        If Simulation.Finished = False And Simulation.Enabled = False Then
            AnswerBox1.HandleInput()
            AnswerBox2.HandleInput()
            If BallReachesWall = True Then
                AnswerBox3.HandleInput()
            End If
            If AnswerButton1.Clicked() = "Clicked" Then
                YesClicked = True
            End If
            If AnswerButton2.Clicked() = "Clicked" Then
                NoClicked = True
            End If

            If YesClicked = True Or NoClicked = True Then
                If AnswerBox1.CheckFilled And AnswerBox2.CheckFilled And
((BallReachesWall = False) Or (BallReachesWall = True And AnswerBox3.CheckFilled))
Then
                    'Other answers have been filled
                    If (YesClicked = True And CorrectFinalAnswer = "Yes") Or
(NoClicked = True And CorrectFinalAnswer = "No") Then
                        'Correct Button Clicked
                        CorrectButton = True
                    Else
                        'Incorrect Button Clicked
                        CorrectButton = False
                    End If
                    twoDPWarning = False
                    'Start Running Simulation
                    Simulation.Enabled = True
                Else
                    twoDPWarning = True
                End If
            End If
        End If
    End Sub

    Public Overrides Sub Update()
```

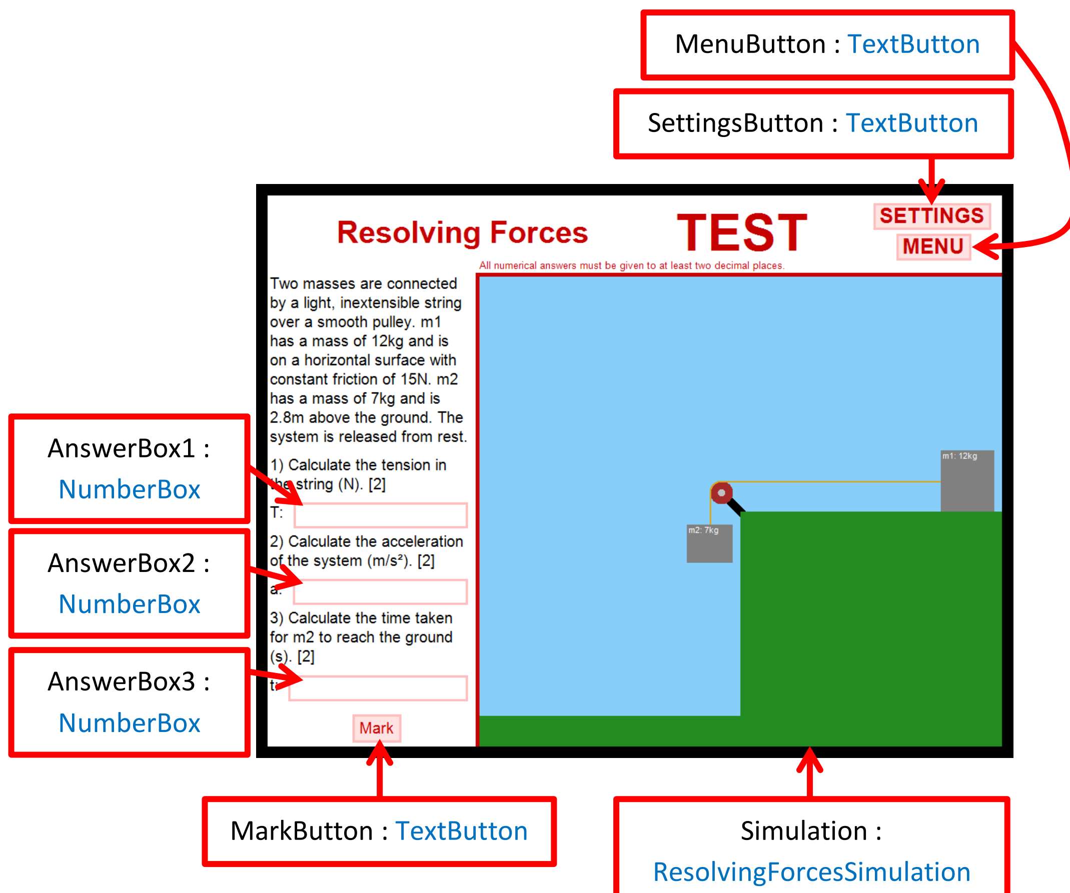
```
        If Simulation.Finished = True And Simulation.Enabled = True Then
            Dim Parts As New List(Of TestQuestionPart)
            Dim TempPart As TestQuestionPart

            Simulation.Enabled = False
            Simulation.Visible = False

            'SCORE ANSWERS AND CREATE REPORT

            'Part 1 - Components of velocity
            TempPart.ScoreOutOf = 2
            TempPart.ScoreAchieved = 0
            TempPart.CorrectAnswer = "X: " & Math.Round(Simulation.FiringV.X, 2) &
"m/s Y: " & Math.Round(Simulation.FiringV.Y, 2) & "m/s"
            If Math.Round(CDbl(AnswerBox1.Text), 2) = Math.Round(Simulation.FiringV.X,
2) Then
                TempPart.ScoreAchieved += 1
            End If
            If Math.Round(CDbl(AnswerBox2.Text), 2) = Math.Round(Simulation.FiringV.Y,
2) Then
                TempPart.ScoreAchieved += 1
            End If
            Parts.Add(TempPart)

            If BallReachesWall = True Then
                'Part 2 - Time at wall
                TempPart.ScoreOutOf = 1
                TempPart.ScoreAchieved = 0
                TempPart.CorrectAnswer = "t: " & CorrectTimeAnswer & "s"
                If Math.Round(CDbl(AnswerBox3.Text), 2) = CorrectTimeAnswer Then
                    TempPart.ScoreAchieved = 1
                End If
                Parts.Add(TempPart)
            End If

            'Part 3 - Buttons
            TempPart.ScoreOutOf = 3
            TempPart.ScoreAchieved = 0
            TempPart.CorrectAnswer = CorrectFinalAnswer
            If CorrectButton = True Then
                TempPart.ScoreAchieved = 3
            End If
            Parts.Add(TempPart)

            Report = New TestReport("Projectile Motion", Parts)
        End If

        Simulation.Update()
        If Simulation.Finished = True And Simulation.Enabled = False Then
            Report.Update()
        End If
    End Sub

    Public Overrides Sub Draw()
        Dim TempY, CurrentQNumber As Integer

        'MAIN TITLE BAR
        'Title
        Main.GFX.DrawString("Projectile Motion", Main.Arial_30_Bold, New
SolidBrush(Color.FromArgb(199, 0, 0)), New Point(261 -
Main.GFX.MeasureString("Projectile Motion", Main.Arial_30_Bold).Width \ 2, 25))
```

```
        Main.GFX.DrawString("TEST", Main.Arial_50_Bold, New
SolidBrush(Color.FromArgb(199, 0, 0)), New Point(522, 10))
        If twoDPWarning Then
            If Now.Millisecond < 800 Then
                Main.GFX.DrawString("All numerical answers must be given to at least
two decimal places.", Main.Arial_10, Brushes.Blue, 960 * 2 / 7, 720 / 7 - 20)
            End If

        Else
            Main.GFX.DrawString("All numerical answers must be given to at least two
decimal places.", Main.Arial_10, New SolidBrush(Color.FromArgb(199, 0, 0)), 960 * 2 /
7, 720 / 7 - 20)
        End If
        'Buttons
        MenuButton.Draw()
        SettingsButton.Draw()

        'DIVIDING LINES
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(199, 0, 0)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7), New Point(960, 720 * 1 / 7))
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(199, 0, 0)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7 - 2), New Point(960 * 2 / 7, 720))

        'SIMULATION OR REPORT
        If Simulation.Visible = True Then
            Simulation.Draw()
        Else
            Report.Draw()
        End If

        'QUESTION
        CurrentQNumber = 1
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15, "A ball
is fired from a cannon at " & Angle & "° to the horizontal at " & Speed & "m/s. A wall
" & XDistance & "m away has a " & WallGap & "m gap " & WallHeight & "m above the
ground.")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, CurrentQNumber
& ") Calculate the X and Y components of the initial velocity (m/s). [2]")
        CurrentQNumber += 1
        Main.GFX.DrawString("X:", Main.Arial_15, Brushes.Black, 0, TempY)
        AnswerBox1.Draw()
        TempY += Main.GFX.MeasureString("X:", Main.Arial_15).Height + 15
        Main.GFX.DrawString("Y:", Main.Arial_15, Brushes.Black, 0, TempY)
        AnswerBox2.Draw()
        If BallReachesWall = True Then
            TempY += Main.GFX.MeasureString("Y:", Main.Arial_15).Height + 15
            TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15,
CurrentQNumber & ") Calculate the time at which the ball will reach the wall (s).
[1]")
            CurrentQNumber += 1
            Main.GFX.DrawString("t:", Main.Arial_15, Brushes.Black, 0, TempY)
            AnswerBox3.Draw()
        End If
        TempY += Main.GFX.MeasureString("t:", Main.Arial_15).Height + 15
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, CurrentQNumber
& ") Will the ball go through the gap? [3]")
        AnswerButton1.Draw()
        AnswerButton2.Draw()

        'MAIN RECT: 277, 106, 683, 614
    End Sub
```

```
End Class
```

ResolvingForcesTest

This screen is for testing the User on the Resolving Forces category. The question is presented on the left. Once the user gives valid answers, the class's instance of the ResolvingForcesSimulation starts, using the starting variables randomly generated by this class. Then the class's instance of TestReport shows the User's Test results. A diagram showing my plan for this process for any test can be found in the design section on page 15.




```
Imports System.Math

Public Class ResolvingForcesTest
    Inherits BaseScreen

    Private MenuButton As New TextButton("MENU", Main.Arial_20_Bold,
ProgramSection.Test, New Point(822, 50), -1, 35, 3, 1)
    Private SettingsButton As New TextButton("SETTINGS", Main.Arial_20_Bold,
ProgramSection.Test, New Point(792, 10), -1, 35, 3, 1)

    Private AnswerBox1, AnswerBox2, AnswerBox3 As NumberBox
    Private MarkButton As TextButton

    Private CorrectTimeAnswer As Decimal

    Private Simulation As New ResolvingForcesSimulation(SimulationMode.Test)
    Private Report As TestReport
```

```
    Private twoDPWarning As Boolean = False

    Private m1Mass, m2Mass, Friction, xDist As Single

    Public Sub New()
        Dim TempY, TempX As Integer

        Name = "ResolvingForcesTest"
        State = ScreenState.Active
        Location = New Point(0, 0)

        'Generate starting variables
        xDist = Main.Rand.Next(1, 20 + 1) / 2
        m1Mass = Main.Rand.Next(1, 15 + 1)
        m2Mass = Main.Rand.Next(1, 15 + 1)
        Friction = Main.Rand.Next(5, m2Mass * Simulation.g)

        'Plug starting variables into Simulation
        Simulation.SetTestVariables(m1Mass, m2Mass, Friction, xDist)

        'Calculate correct answers
        CorrectTimeAnswer = Round(Sqrt(2 * Simulation.yDist /
Simulation.Acceleration), 2)

        'Create answer boxes in the right places
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15, "Two
masses are connected by a light, inextensible string over a smooth pulley. m1 has a
mass of " & m1Mass & "kg and is on a horizontal surface with constant friction of " &
Friction & "N. m2 has a mass of " & m2Mass & "kg and is " & xDist * 0.8 & "m above the
ground. The system is released from rest.", False)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "1) Calculate
the tension in the string (N). [2]", False)
        TempX = Main.GFX.MeasureString("T:", Main.Arial_15).Width + 10
        AnswerBox1 = New NumberBox(New Point(TempX, TempY), Main.Arial_15,
ProgramSection.Test, 3, 960 * 2 / 7 - TempX - 14)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "T:", False)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "2) Calculate
the acceleration of the system (m/s²). [2]", False)
        TempX = Main.GFX.MeasureString("a:", Main.Arial_15).Width + 10
        AnswerBox2 = New NumberBox(New Point(TempX, TempY), Main.Arial_15,
ProgramSection.Test, 3, 960 * 2 / 7 - TempX - 14)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "a:", False)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "3) Calculate
the time taken for m2 to reach the ground (s). [2]", False)
        TempX = Main.GFX.MeasureString("t:", Main.Arial_15).Width + 10
        AnswerBox3 = New NumberBox(New Point(TempX, TempY), Main.Arial_15,
ProgramSection.Test, 3, 960 * 2 / 7 - TempX - 14)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "t:", False)
        TempY += Main.GFX.MeasureString("t:", Main.Arial_15).Height \ 2
        MarkButton = New TextButton("Mark", Main.Arial_15, ProgramSection.Test, New
Point(960 / 7 - Main.GFX.MeasureString("Mark", Main.Arial_15).Width \ 2, TempY), -1, -
1, 3)
    End Sub

    Public Overrides Sub HandleInput()
        If MenuButton.Clicked = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New TestMenu)
        End If
        If SettingsButton.Clicked() = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
```

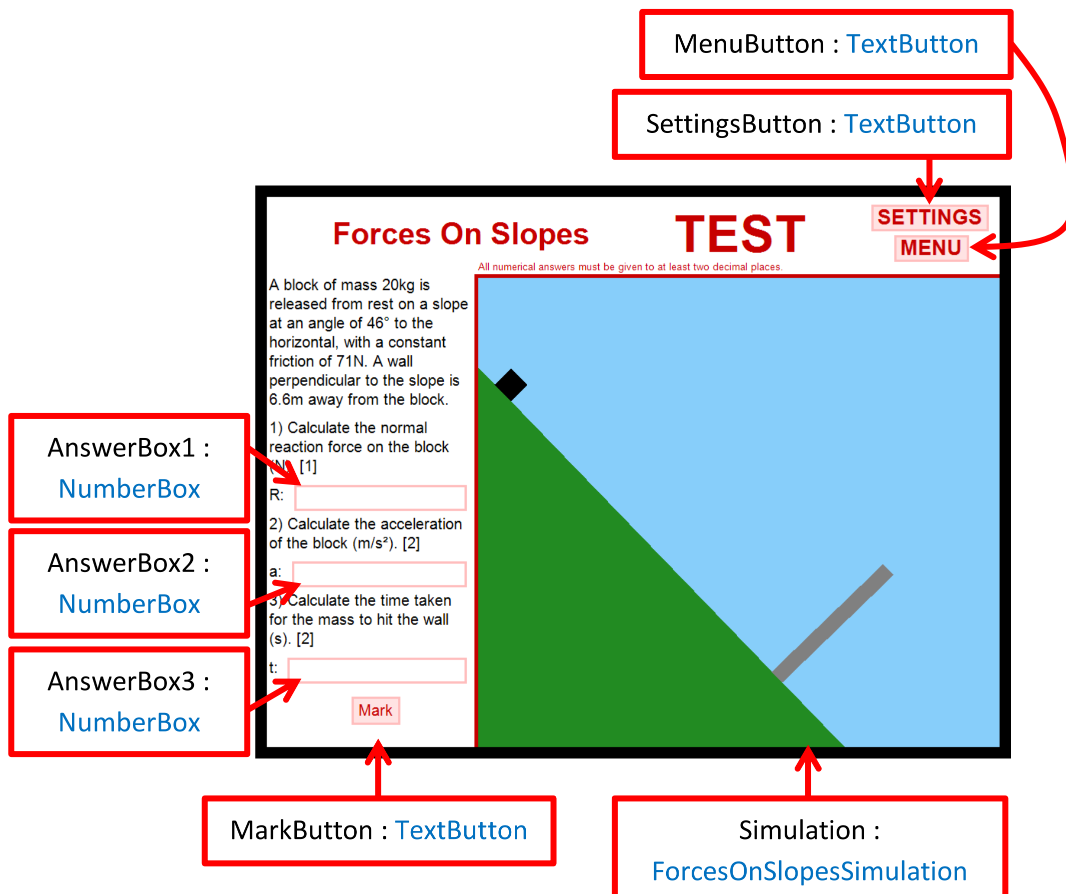
```
            ScreenManager.AddScreen(New Settings({New TestMenu}))
        End If

        'Handle answers input
        If Simulation.Finished = False And Simulation.Enabled = False Then
            AnswerBox1.HandleInput()
            AnswerBox2.HandleInput()
            AnswerBox3.HandleInput()

            If MarkButton.Clicked = "Clicked" Then
                If AnswerBox1.CheckFilled And AnswerBox2.CheckFilled And
AnswerBox3.CheckFilled Then
                    'Other answers have been filled
                    twoDPWarning = False

                    'Start Running Simulation
                    Simulation.Enabled = True
                Else
                    twoDPWarning = True
                End If
            End If
        End If
    End Sub

    Public Overrides Sub Update()
        If Simulation.Finished = True And Simulation.Enabled = True Then
            Dim Parts As New List(Of TestQuestionPart)
            Dim TempPart As TestQuestionPart

            Simulation.Enabled = False
            Simulation.Visible = False

            'SCORE ANSWERS AND CREATE REPORT
            'Part 1 - Calculating Tension
            TempPart.ScoreOutOf = 2
            TempPart.CorrectAnswer = "T: " & Round(Simulation.Tension, 2) & "N"
            If Round(CDbl(AnswerBox1.Text), 2) = Round(Simulation.Tension, 2) Then
                TempPart.ScoreAchieved = 2
            Else
                TempPart.ScoreAchieved = 0
            End If
            Parts.Add(TempPart)

            'Part 2 - Calculating Acceleration
            TempPart.ScoreOutOf = 2
            TempPart.CorrectAnswer = "a: " & Round((Simulation.Tension -
Simulation.Friction) / m1Mass, 2) & "m/s²"
            If Round(CDbl(AnswerBox2.Text), 2) = Round((Simulation.Tension -
Simulation.Friction) / m1Mass, 2) Then
                TempPart.ScoreAchieved = 2
            Else
                TempPart.ScoreAchieved = 0
            End If
            Parts.Add(TempPart)

            'Part 3 - Calculating Time
            TempPart.ScoreOutOf = 2
            TempPart.CorrectAnswer = "t: " & CorrectTimeAnswer & "s"
            If Round(CDbl(AnswerBox3.Text), 2) = CorrectTimeAnswer Then
                TempPart.ScoreAchieved = 2
            Else
```

```
                TempPart.ScoreAchieved = 0
            End If
            Parts.Add(TempPart)

            Report = New TestReport("Resolving Forces", Parts)
        End If

        Simulation.Update()

        If Simulation.Finished = True And Simulation.Enabled = False Then
            Report.Update()
        End If
    End Sub

    Public Overrides Sub Draw()
        Dim TempY As Integer

        'MAIN TITLE BAR
        'Title
        Main.GFX.DrawString("Resolving Forces", Main.Arial_30_Bold, New
SolidBrush(Color.FromArgb(199, 0, 0)), New Point(261 -
Main.GFX.MeasureString("Resolving Forces", Main.Arial_30_Bold).Width \ 2, 25))
        Main.GFX.DrawString("TEST", Main.Arial_50_Bold, New
SolidBrush(Color.FromArgb(199, 0, 0)), New Point(522, 10))
        If twoDPWarning Then
            If Now.Millisecond < 800 Then
                Main.GFX.DrawString("All numerical answers must be given to at least
two decimal places.", Main.Arial_10, Brushes.Blue, 960 * 2 / 7, 720 / 7 - 20)
            End If

        Else
            Main.GFX.DrawString("All numerical answers must be given to at least two
decimal places.", Main.Arial_10, New SolidBrush(Color.FromArgb(199, 0, 0)), 960 * 2 /
7, 720 / 7 - 20)
        End If
        'Buttons
        MenuButton.Draw()
        SettingsButton.Draw()

        'DIVIDING LINES
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(199, 0, 0)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7), New Point(960, 720 * 1 / 7))
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(199, 0, 0)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7 - 2), New Point(960 * 2 / 7, 720))

        'SIMULATION OR REPORT
        If Simulation.Visible = True Then
            Simulation.Draw()
        Else
            Report.Draw()
        End If

        'QUESTION
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15, "Two
masses are connected by a light, inextensible string over a smooth pulley. m1 has a
mass of " & m1Mass & "kg and is on a horizontal surface with constant friction of " &
Friction & "N. m2 has a mass of " & m2Mass & "kg and is " & xDist * 0.8 & "m above the
ground. The system is released from rest.")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "1) Calculate
the tension in the string (N). [2]")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "T:")
```

```
        AnswerBox1.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "2) Calculate
the acceleration of the system (m/s²). [2]")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "a:")
        AnswerBox2.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "3) Calculate
the time taken for m2 to reach the ground (s). [2]")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "t:")
        AnswerBox3.Draw()
        MarkButton.Draw()
        'MAIN RECT: 277, 106, 683, 614
    End Sub
End Class
```

ForcesOnSlopesTest

This screen is for testing the User on the Forces On Slopes category. The question is presented on the left. Once the user gives valid answers, the class's instance of the ForcesOnSlopesSimulation starts, using the starting variables randomly generated by this class. Then the class's instance of TestReport shows the User's Test results. A diagram showing my plan for this process for any test can be found in the design section on page 15.




```
Imports System.Math

Public Class ForcesOnSlopesTest
    Inherits BaseScreen
```

```
    Private MenuButton As New TextButton("MENU", Main.Arial_20_Bold,
ProgramSection.Test, New Point(822, 50), -1, 35, 3, 1)
    Private SettingsButton As New TextButton("SETTINGS", Main.Arial_20_Bold,
ProgramSection.Test, New Point(792, 10), -1, 35, 3, 1)

    Private AnswerBox1, AnswerBox2, AnswerBox3 As NumberBox
    Private MarkButton As TextButton

    Private CorrectTimeAnswer As Decimal

    Private Simulation As New ForcesOnSlopesSimulation(SimulationMode.Test)
    Private Report As TestReport

    Private twoDPWarning As Boolean = False

    Private Mass, DistanceToWall, Friction, SlopeAngle As Single

    Public Sub New()
        Dim TempY, TempX As Integer

        Name = "ForcesOnSlopesTest"
        State = ScreenState.Active
        Location = New Point(0, 0)

        'Generate starting variables
        Mass = 20 'Main.Rand.Next(5, 30 + 1)
        DistanceToWall = 6.6 'Main.Rand.Next(1, 100 + 1) / 10
        SlopeAngle = 46 'Main.Rand.Next(15, 70 + 1)
        Friction = 71 'Main.Rand.Next(0, 2 * Mass * Simulation.g *
Sin(Main.Rad(SlopeAngle))) / 2

        'Plug starting variables into Simulation
        Simulation.SetTestVariables(Mass, DistanceToWall, Friction, SlopeAngle)

        'Calculate correct answers
        CorrectTimeAnswer = Round(Sqrt(2 * DistanceToWall / Simulation.Acceleration),
2)

        'Create answer boxes in the right places
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15, "A block
of mass " & Mass & "kg is released from rest on a slope at an angle of " & SlopeAngle
& "° to the horizontal, with a constant friction of " & Friction & "N. A wall
perpendicular to the slope is " & DistanceToWall & "m away from the block.", False)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "1) Calculate
the normal reaction force on the block (N). [1]", False)
        TempX = Main.GFX.MeasureString("R:", Main.Arial_15).Width + 10
        AnswerBox1 = New NumberBox(New Point(TempX, TempY), Main.Arial_15,
ProgramSection.Test, 3, 960 * 2 / 7 - TempX - 14)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "T:", False)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "2) Calculate
the acceleration of the block (m/s²). [2]", False)
        TempX = Main.GFX.MeasureString("a:", Main.Arial_15).Width + 10
        AnswerBox2 = New NumberBox(New Point(TempX, TempY), Main.Arial_15,
ProgramSection.Test, 3, 960 * 2 / 7 - TempX - 14)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "a:", False)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "3) Calculate
the time taken for the mass to hit the wall (s). [2]", False)
        TempX = Main.GFX.MeasureString("t:", Main.Arial_15).Width + 10
        AnswerBox3 = New NumberBox(New Point(TempX, TempY), Main.Arial_15,
ProgramSection.Test, 3, 960 * 2 / 7 - TempX - 14)
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "t:", False)
```

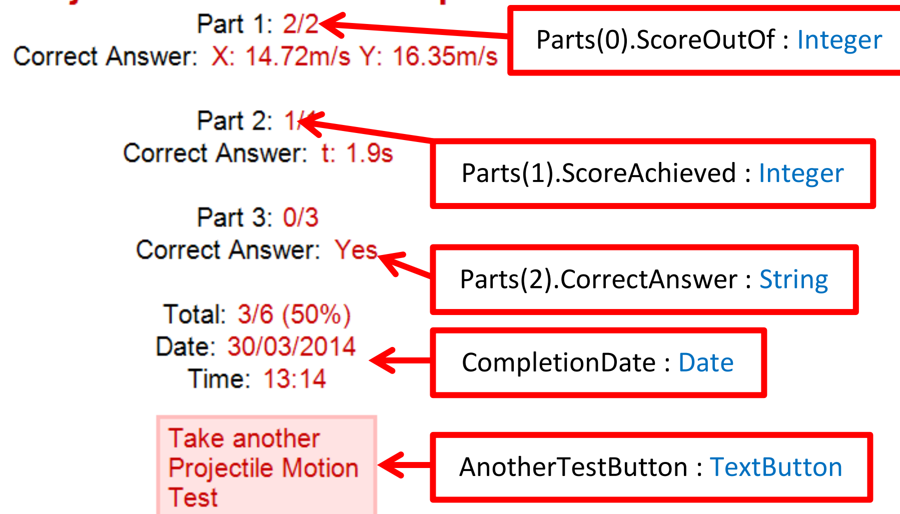
```
        TempY += Main.GFX.MeasureString("t:", Main.Arial_15).Height \ 2
        MarkButton = New TextButton("Mark", Main.Arial_15, ProgramSection.Test, New
Point(960 / 7 - Main.GFX.MeasureString("Mark", Main.Arial_15).Width \ 2, TempY), -1, -
1, 3)
    End Sub

    Public Overrides Sub HandleInput()
        If MenuButton.Clicked = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New TestMenu)
        End If
        If SettingsButton.Clicked() = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New Settings({New TestMenu}))
        End If

        'Handle answers input
        If Simulation.Finished = False And Simulation.Enabled = False Then
            AnswerBox1.HandleInput()
            AnswerBox2.HandleInput()
            AnswerBox3.HandleInput()

            If MarkButton.Clicked = "Clicked" Then
                If AnswerBox1.CheckFilled And AnswerBox2.CheckFilled And
AnswerBox3.CheckFilled Then
                    'Other answers have been filled
                    twoDPWarning = False

                    'Start Running Simulation
                    Simulation.Enabled = True
                Else
                    twoDPWarning = True
                End If
            End If
        End If
    End Sub

    Public Overrides Sub Update()
        If Simulation.Finished = True And Simulation.Enabled = True Then
            Dim Parts As New List(Of TestQuestionPart)
            Dim TempPart As TestQuestionPart

            Simulation.Enabled = False
            Simulation.Visible = False

            'SCORE ANSWERS AND CREATE REPORT
            'Part 1 - Calculating the normal reaction
            TempPart.ScoreOutOf = 1
            TempPart.CorrectAnswer = "R: " & Round(Simulation.Mass * Simulation.g *
Cos(Main.Rad(Simulation.SlopeAngle)), 2) & "N"
            If Round(CDbl(AnswerBox1.Text), 2) = Round(Simulation.Mass * Simulation.g
* Cos(Main.Rad(Simulation.SlopeAngle)), 2) Then
                TempPart.ScoreAchieved = 1
            Else
                TempPart.ScoreAchieved = 0
            End If
            Parts.Add(TempPart)

            'Part 2 - Calculating the acceleration
            TempPart.ScoreOutOf = 2
```

```
            TempPart.CorrectAnswer = "a: " & Round((Simulation.Mass * Simulation.g *
Sin(Main.Rad(Simulation.SlopeAngle)) - Simulation.Friction) / Simulation.Mass, 2) &
"m/s²"
            If Round(CDbl(AnswerBox2.Text), 2) = Round((Simulation.Mass * Simulation.g
* Sin(Main.Rad(Simulation.SlopeAngle)) - Simulation.Friction) / Simulation.Mass, 2)
Then
                TempPart.ScoreAchieved = 2
            Else
                TempPart.ScoreAchieved = 0
            End If
            Parts.Add(TempPart)

            'Part 3 - Calculating the time for the mass to reach the wall
            TempPart.ScoreOutOf = 2
            TempPart.CorrectAnswer = "t: " & CorrectTimeAnswer & "s"
            If Round(CDbl(AnswerBox3.Text), 2) = CorrectTimeAnswer Then
                TempPart.ScoreAchieved = 2
            Else
                TempPart.ScoreAchieved = 0
            End If
            Parts.Add(TempPart)

            Report = New TestReport("Forces On Slopes", Parts)
        End If

        Simulation.Update()

        If Simulation.Finished = True And Simulation.Enabled = False Then
            Report.Update()
        End If
    End Sub

    Public Overrides Sub Draw()
        Dim TempY As Integer

        'MAIN TITLE BAR
        'Title
        Main.GFX.DrawString("Forces On Slopes", Main.Arial_30_Bold, New
SolidBrush(Color.FromArgb(199, 0, 0)), New Point(261 - Main.GFX.MeasureString("Forces
On Slopes", Main.Arial_30_Bold).Width \ 2, 25))
        Main.GFX.DrawString("TEST", Main.Arial_50_Bold, New
SolidBrush(Color.FromArgb(199, 0, 0)), New Point(522, 10))
        If twoDPWarning Then
            If Now.Millisecond < 800 Then
                Main.GFX.DrawString("All numerical answers must be given to at least
two decimal places.", Main.Arial_10, Brushes.Blue, 960 * 2 / 7, 720 / 7 - 20)
            End If

        Else
            Main.GFX.DrawString("All numerical answers must be given to at least two
decimal places.", Main.Arial_10, New SolidBrush(Color.FromArgb(199, 0, 0)), 960 * 2 /
7, 720 / 7 - 20)
        End If
        'Buttons
        MenuButton.Draw()
        SettingsButton.Draw()

        'DIVIDING LINES
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(199, 0, 0)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7), New Point(960, 720 * 1 / 7))
```

```
        Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(199, 0, 0)), 5), New
Point(960 * 2 / 7, 720 * 1 / 7 - 2), New Point(960 * 2 / 7, 720))

        'SIMULATION OR REPORT
        If Simulation.Visible = True Then
            Simulation.Draw()
        Else
            Report.Draw()
        End If

        'QUESTION
        TempY = Main.AutoFitText(0, 720 * 1 / 7, 960 * 2 / 7, Main.Arial_15, "A block
of mass " & Mass & "kg is released from rest on a slope at an angle of " & SlopeAngle
& "° to the horizontal, with a constant friction of " & Friction & "N. A wall
perpendicular to the slope is " & DistanceToWall & "m away from the block.")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "1) Calculate
the normal reaction force on the block (N). [1]")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "R:")
        AnswerBox1.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "2) Calculate
the acceleration of the block (m/s²). [2]")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "a:")
        AnswerBox2.Draw()
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "3) Calculate
the time taken for the mass to hit the wall (s). [2]")
        TempY = Main.AutoFitText(0, TempY, 960 * 2 / 7, Main.Arial_15, "t:")
        AnswerBox3.Draw()
        MarkButton.Draw()
        'MAIN RECT: 277, 106, 683, 614
    End Sub
End Class
```

TestReport

This object replaces the Simulation in each Test after the Simulation has finished running. It shows how well the User performed in the Test, including the correct answers for each question. It also decrypts the User's text file, appends their latest Test data, and encrypts it again. The DrawRow function is for easily creating the lines of text in the image below where one half is one colour and the other half is another.

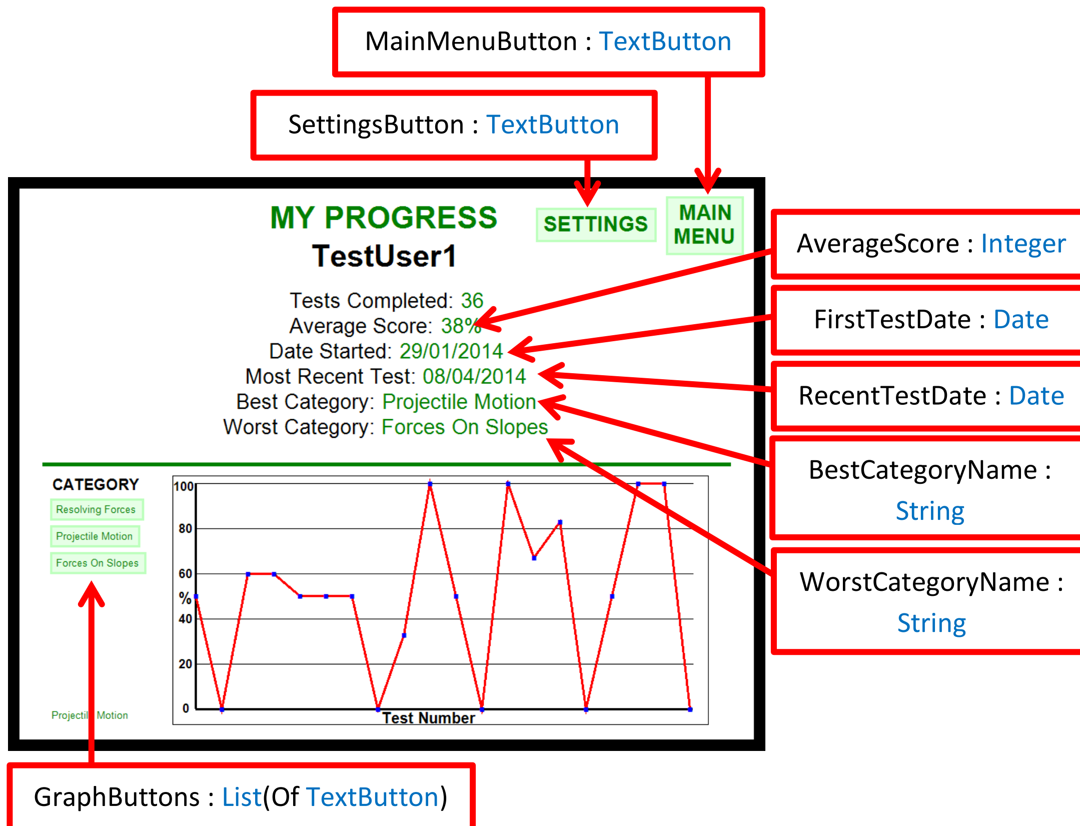


```
Imports System.IO

Public Structure TestQuestionPart
    Dim ScoreAchieved, ScoreOutOf As Integer
    Dim CorrectAnswer As String
End Structure

Public Class TestReport
    Inherits BaseScreen

    Private Size As New Size(683, 614)

    Private AnotherTestButton As TextButton

    Private CompletionDate As Date
    Private Parts As New List(Of TestQuestionPart)
    Private TotalAchieved, TotalOutOf As Integer

    Public Sub New(ByVal InputTestName As String, ByVal InputParts As List(Of
TestQuestionPart))
        Name = InputTestName
        Location = New Point(277, 106)

        CompletionDate = Now
        Parts = InputParts

        'Calculate total score
        For i = 0 To Parts.Count - 1
            TotalAchieved += Parts(i).ScoreAchieved
            TotalOutOf += Parts(i).ScoreOutOf
        Next

        AnotherTestButton = New TextButton("Take another" & vbNewLine & Name &
vbNewLine & "Test", Main.Arial_15, ProgramSection.Test, New Point(Location.X +
Size.Width \ 2 - Main.GFX.MeasureString(Name, Main.Arial_15).Width \ 2, Location.Y +
Size.Height * 1 / 7 + Main.GFX.MeasureString("Hello", Main.Arial_15).Height * (5 + 3 *
Parts.Count)), -1, -1, 3)

        'ADD TEST RESULT TO USER FILE
        File.WriteAllText(Main.CurrentUser & ".sv",
Main.EncryptString(Main.DecryptString(File.ReadAllText(Main.CurrentUser & ".sv")) &
Name & "," & Math.Round(TotalAchieved / TotalOutOf * 100) & "," & CompletionDate &
"|"))
    End Sub

    Public Overrides Sub Update()
        If AnotherTestButton.Clicked() = "Clicked" Then
            Select Case Name
                Case "Projectile Motion"
                    ScreenManager.UnloadScreen("ProjectileMotionTest")
                    ScreenManager.AddScreen(New ProjectileMotionTest)
                Case "Resolving Forces"
                    ScreenManager.UnloadScreen("ResolvingForcesTest")
                    ScreenManager.AddScreen(New ResolvingForcesTest)
                Case "Forces On Slopes"
                    ScreenManager.UnloadScreen("ForcesOnSlopesTest")
                    ScreenManager.AddScreen(New ForcesOnSlopesTest)
            End Select
        End If
```

```
    End Sub

    Private Function DrawRow(ByVal String1 As String, String2 As String, TempY As Integer)
        'Draw one line of text where two parts are of different colours
        Dim StringLength1 As Integer = Main.GFX.MeasureString(String1, Main.Arial_15).Width
        Dim StringLength2 As Integer = Main.GFX.MeasureString(String2, Main.Arial_15).Width

        Main.GFX.DrawString(String1, Main.Arial_15, Brushes.Black, Location.X + Size.Width \ 2 - (StringLength1 + StringLength2) \ 2, TempY)
        Main.GFX.DrawString(String2, Main.Arial_15, New SolidBrush(Color.FromArgb(199, 0, 0)), Location.X + Size.Width \ 2 - (StringLength1 + StringLength2) \ 2 + StringLength1, TempY)

        Return Main.GFX.MeasureString(String1 & String2, Main.Arial_15_Bold).Height
    End Function
    Public Overrides Sub Draw()
        Dim TempY As Integer

        'USER NAME
        Main.GFX.DrawString("User: ", Main.Arial_10, Brushes.Black, Location)
        Main.GFX.DrawString(Main.CurrentUser, Main.Arial_10, New SolidBrush(Color.FromArgb(199, 0, 0)), New Point(Location.X + Main.GFX.MeasureString("User: ", Main.Arial_10).Width, Location.Y))
        'TITLE
        TempY = Location.Y + Size.Height * 1 / 7
        Main.GFX.DrawString(Name & " Test Report", Main.Arial_20_Bold, New SolidBrush(Color.FromArgb(199, 0, 0)), Location.X + Size.Width \ 2 - Main.GFX.MeasureString(Name & " Test Report", Main.Arial_20_Bold).Width \ 2, TempY)
        TempY += Main.GFX.MeasureString(Name & " Test Report", Main.Arial_20_Bold).Height
        'PARTS
        For i = 0 To Parts.Count - 1
            TempY += DrawRow("Part " & i + 1 & ": ", Parts(i).ScoreAchieved & "/" & Parts(i).ScoreOutOf, TempY)
            TempY += DrawRow("Correct Answer: ", Parts(i).CorrectAnswer, TempY)
            TempY += DrawRow(" ", " ", TempY)
        Next
        'TOTAL
        TempY += DrawRow("Total: ", TotalAchieved & "/" & TotalOutOf & " (" & Math.Round(TotalAchieved / TotalOutOf * 100) & "%)", TempY)
        'DATE AND TIME
        TempY += DrawRow("Date: ", CompletionDate.Date, TempY)

        If CompletionDate.Minute < 10 Then
            TempY += DrawRow("Time: ", CompletionDate.Hour & ":0" & CompletionDate.Minute, TempY)
        Else
            TempY += DrawRow("Time: ", CompletionDate.Hour & ":" & CompletionDate.Minute, TempY)
        End If
        'BUTTONS
        AnotherTestButton.Draw()
    End Sub
End Class
```

MyProgressReport

This screen is for the User to see how they are performing on Tests. It reads their text file and processes it to find useful information, such as their best and worst categories. The top half of the screen shows overall statistics and the bottom half shows a graph with Test score against Test number for specific categories. A diagram of my plan for this process can be found in the design section on page 16.




```
Imports System.IO

Public Class MyProgressReport
    Inherits BaseScreen

    Public Structure TestReportInfo
        Dim Category As String
        Dim Score As Integer
        Dim CompletionDate As Date
    End Structure

    Private TestReports As New List(Of TestReportInfo)
    Private AverageScore As Integer
    Private FirstTestDate, RecentTestDate As Date
    Private BestCategoryName, WorstCategoryName As String

    Private MainMenuButton As New TextButton(" MAIN" & vbNewLine & "MENU",
Main.Arial_20_Bold, ProgramSection.MyProgress, New Point(845, 10), -1, -1, 3)
    Private SettingsButton As New TextButton("SETTINGS", Main.Arial_20_Bold,
ProgramSection.MyProgress, New Point(675, 25), -1, -1, 3)

    Private GraphButtons As New List(Of TextButton)
    Private GraphPoints() As Point
```

```
    Private CurrentCategory As String

    Public Sub New()
        Dim TestMsgBox As String = ""

        'Get the current users raw file, and decrypt it
        Dim UserContent As String =
Main.DecryptString(File.ReadAllText(Main.CurrentUser & ".sv"))

        If UserContent.Length > 0 Then
            'Remove the '|' symbol from the end
            UserContent = UserContent.Substring(0, UserContent.Length - 1)
        End If
        'Split the raw data string into separate reports
        Dim strTestReports() As String = Split(UserContent, "|")
        Dim TempTestReport As TestReportInfo
        Dim TotalScore As Integer
        Dim BestCategory, WorstCategory As Integer
        Dim CategoriesWithData As New List(Of String)
        Dim TempGraphButton As TextButton

        Name = "MyProgressReport"
        Location = New Point(0, 0)
        State = ScreenState.Active

        If UserContent.Length > 0 Then
            FirstTestDate = Split(strTestReports(0), ",")(2)
            RecentTestDate = FirstTestDate

            For Each strTestReport In strTestReports
                'Split each report into the three fields.
                '0: Category
                '1: Score
                '2: Completion Date
                TempTestReport.Category = Split(strTestReport, ",")(0)
                TempTestReport.Score = Split(strTestReport, ",")(1)
                'Add up all of the scores.
                TotalScore += TempTestReport.Score
                TempTestReport.CompletionDate = Split(strTestReport, ",")(2)
                If TempTestReport.CompletionDate < FirstTestDate Then
                    FirstTestDate = TempTestReport.CompletionDate
                End If
                If TempTestReport.CompletionDate > RecentTestDate Then
                    RecentTestDate = TempTestReport.CompletionDate
                End If
                TestReports.Add(TempTestReport)
            Next

            'Calculate the average score
            AverageScore = TotalScore / TestReports.Count

            'Find the Best and Worst categories using the GetAverageScore() function
            BestCategory = GetAverageScore(TestReports(0).Category)
            WorstCategory = BestCategory
            BestCategoryName = TestReports(0).Category
            WorstCategoryName = BestCategoryName

            For Each TestReport In TestReports
                If Not CategoriesWithData.Contains(TestReport.Category) Then
                    If GetAverageScore(TestReport.Category) > BestCategory Then
                        BestCategory = GetAverageScore(TestReport.Category)
```

```
                        BestCategoryName = TestReport.Category
                    End If
                    If GetAverageScore(TestReport.Category) < WorstCategory Then
                        WorstCategory = GetAverageScore(TestReport.Category)
                        WorstCategoryName = TestReport.Category
                    End If

                    'This is so the program only makes graph buttons for
                    'categories with tests completed
                    CategoriesWithData.Add(TestReport.Category)
                End If
            Next

            CurrentCategory = TestReports(0).Category
            SetGraphPoints(CurrentCategory)
        End If

        'Add graph buttons only for categories with data
        For i = 0 To CategoriesWithData.Count - 1
            TempGraphButton = New TextButton(CategoriesWithData(i), Main.Arial_10,
ProgramSection.MyProgress, New Point(40, 405 + i * 35), -1, -1, 3)
            GraphButtons.Add(TempGraphButton)
        Next
    End Sub

    Private Function GetAverageScore(ByVal Category As String) As Integer
        'Find the average score for all tests with a specified category
        Dim TotalScore As Integer = 0
        Dim NumTests As Integer = 0

        For Each TestReport In TestReports
            If TestReport.Category = Category Then
                TotalScore += TestReport.Score
                NumTests += 1
            End If
        Next

        'Return the average score as a percentage to the nearest integer
        Return Math.Round(TotalScore / NumTests)
    End Function

    Private Sub SetGraphPoints(ByVal Category As String)
        'Set the x and y coordiantes for the data points on the graph for
        'tests with a specified category
        Dim NumPoints As Integer = 0
        Dim XDistance As Integer = 0
        Dim RelevantReports As New List(Of TestReportInfo)

        'Generate a list of all of the reports with the correct category
        For Each Report In TestReports
            If Report.Category = Category Then
                RelevantReports.Add(Report)
                NumPoints += 1
            End If
        Next

        'XDistance is the number of horizontal pixels between each point
        'As there are more points, XDistance decreases
        If NumPoints > 1 Then
            XDistance = 650 / (NumPoints - 1)
        End If
```

```
        ReDim GraphPoints(NumPoints - 1)
        For i = 0 To NumPoints - 1
            GraphPoints(i) = New Point(230 + i * XDistance, 680 -
RelevantReports(i).Score / 100 * 295)
        Next

        'PixWidth = 650
        'PixHeight = 295
    End Sub

    Public Overrides Sub HandleInput()
        If MainMenuButton.Clicked = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New Title)
            ScreenManager.AddScreen(New SimulationButton)
            ScreenManager.AddScreen(New TestButton)
            ScreenManager.AddScreen(New MyProgressButton)
        End If
        If SettingsButton.Clicked = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New Settings({New MyProgressReport}))
        End If

        For Each Button In GraphButtons
            If Button.Clicked() = "Clicked" Then
                CurrentCategory = Button.Text
                SetGraphPoints(CurrentCategory)
            End If
        Next
    End Sub

    Private Function DrawRow(ByVal String1 As String, String2 As String, TempY As
Integer)
        'Draw one line of text where two parts are of different colours
        Dim StringLength1 As Integer = Main.GFX.MeasureString(String1,
Main.Arial_20).Width
        Dim StringLength2 As Integer = Main.GFX.MeasureString(String2,
Main.Arial_20).Width

        Main.GFX.DrawString(String1, Main.Arial_20, Brushes.Black, 480 -
(StringLength1 + StringLength2) \ 2, TempY)
        Main.GFX.DrawString(String2, Main.Arial_20, New SolidBrush(Color.FromArgb(0,
128, 0)), 480 - (StringLength1 + StringLength2) \ 2 + StringLength1, TempY)

        Return Main.GFX.MeasureString(String1 & String2, Main.Arial_20).Height
    End Function
    Public Overrides Sub Draw()
        Dim TempY As Integer = 15

        'TITLE
        Main.GFX.DrawString("MY PROGRESS", Main.Arial_30_Bold, New
SolidBrush(Color.FromArgb(0, 128, 0)), 480 - Main.GFX.MeasureString("MY PROGRESS",
Main.Arial_30_Bold).Width \ 2, TempY)
        TempY += Main.GFX.MeasureString("MY PROGRESS", Main.Arial_30_Bold).Height
        Main.GFX.DrawString(Main.CurrentUser, Main.Arial_30_Bold, Brushes.Black, 480 -
Main.GFX.MeasureString(Main.CurrentUser, Main.Arial_30_Bold).Width \ 2, TempY)
        TempY += Main.GFX.MeasureString(Main.CurrentUser, Main.Arial_30_Bold).Height +
15
        'Menu
        MainMenuButton.Draw()
```

```
        SettingsButton.Draw()

        If TestReports.Count > 0 Then
            'TOP HALF INFO
            'No. Tests completed
            TempY += DrawRow("Tests Completed: ", TestReports.Count, TempY)
            'Average Score
            TempY += DrawRow("Average Score: ", AverageScore & "%", TempY)
            'Date Started
            TempY += DrawRow("Date Started: ", FirstTestDate.Date, TempY)
            'Most Recent Test
            TempY += DrawRow("Most Recent Test: ", RecentTestDate.Date, TempY)
            'Best Category
            TempY += DrawRow("Best Category: ", BestCategoryName, TempY)
            'Worst Category
            TempY += DrawRow("Worst Category: ", WorstCategoryName, TempY)
            'DIVIDING LINE
            Main.GFX.DrawLine(New Pen(New SolidBrush(Color.FromArgb(0, 128, 0)), 5),
30, 360, 930, 360)
            'BOTTOM HALF BUTTONS
            Main.GFX.DrawString("CATEGORY", Main.Arial_15_Bold, Brushes.Black, 40,
375)
            For Each Button In GraphButtons
                Button.Draw()
            Next
            Main.GFX.DrawString(CurrentCategory, Main.Arial_10, New
SolidBrush(Color.FromArgb(0, 128, 0)), 40, 680)
            'GRAPH
            Main.GFX.DrawRectangle(Pens.Black, 200, 375, 700, 325)
            'Y-Axis
            Main.GFX.DrawLine(New Pen(Brushes.Black, 3), 230, 680, 230, 385)
            Main.GFX.DrawString("100", Main.Arial_12_Bold, Brushes.Black, 198, 380)
            Main.GFX.DrawString("80", Main.Arial_12_Bold, Brushes.Black, 205, 435)
            Main.GFX.DrawString("60", Main.Arial_12_Bold, Brushes.Black, 205, 494)
            Main.GFX.DrawString("%", Main.Arial_15_Bold, Brushes.Black, 205, 523)
            Main.GFX.DrawString("40", Main.Arial_12_Bold, Brushes.Black, 205, 553)
            Main.GFX.DrawString("20", Main.Arial_12_Bold, Brushes.Black, 205, 612)
            Main.GFX.DrawString("0", Main.Arial_12_Bold, Brushes.Black, 210, 670)
            'X-Axis
            Main.GFX.DrawLine(New Pen(Brushes.Black, 3), 230, 680, 880, 680)
            Main.GFX.DrawString("Test Number", Main.Arial_15_Bold, Brushes.Black, 470,
680)
            'Horizontal Dividing lines
            For i = 1 To 5
                Main.GFX.DrawLine(Pens.Black, 230, 680 - i * 59, 880, 680 - i * 59)
            Next
            'Plot Line
            If GraphPoints.Length > 1 Then
                Main.GFX.DrawLines(New Pen(Brushes.Red, 3), GraphPoints)
            End If
            'Plot Points
            For Each Point In GraphPoints
                Main.GFX.DrawString(".", Main.Arial_30_Bold, Brushes.Blue, Point.X -
12, Point.Y - 33)
            Next
        Else
            'If no test data
            Main.GFX.DrawString("You have not yet completed any tests.",
Main.Arial_30_Bold, New SolidBrush(Color.FromArgb(0, 128, 0)), 480 -
Main.GFX.MeasureString("You have not yet completed any tests.",
Main.Arial_30_Bold).Width \ 2, 360)
```

```
        End If
    End Sub
End Class
```

UserSelection

This screen is the parent/base class of the two User Selection screens (Test and My Progress). The reason for this is that these two subclasses are identical, except from their colour and which screens they point to. User selection screens will look at the text file directory, and firstly delete any files with unwanted file extensions (all of my save files end in ".sv"). It will generate a list of all possible users. It handles input for clicking on these User Names, as well as creating new users.

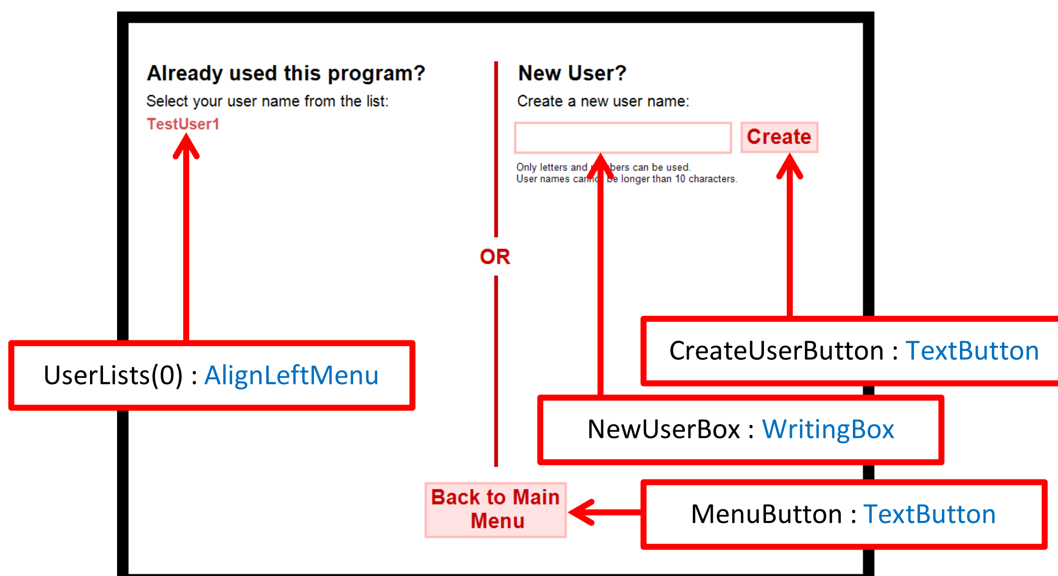
```
Imports System.IO

Public Class UserSelection
    Inherits BaseScreen

    Protected MenuButton As TextButton
    Protected NewUserBox As WritingBox
    Protected CreateUserButton As TextButton

    Protected UserLists As New List(Of AlignLeftMenu)
    Protected Users As New List(Of String)

    Protected UserAlreadyExistsError As Date

    Protected SectionColour As Color

    Protected Sub RefreshExistingUserLists()
        Users.Clear()
        UserLists.Clear()

        For Each FoundFile In Directory.GetFiles(Environment.CurrentDirectory)
            If InStr(FoundFile, ".sv") = 0 Then
                'Invalid file in directory
                File.Delete(FoundFile)
            Else
                'Get the user name from the file path
                FoundFile = FoundFile.Replace(Environment.CurrentDirectory & "\", "")
                FoundFile = FoundFile.Substring(0, FoundFile.Length - 3)
                Users.Add(FoundFile)
            End If
        Next

        'There will be potentially two lists of user names. Each list can be 21 names
long
        'Create the user lists
        UserLists.Add(New AlignLeftMenu(New Point(20, 120), Main.Arial_15_Bold,
Color.FromArgb(80 / 100 * 255, SectionColour), SectionColour))
        If Users.Count > 21 Then
            UserLists.Add(New AlignLeftMenu(New Point(240, 120), Main.Arial_15_Bold,
Color.FromArgb(70 / 100 * 255, SectionColour), SectionColour))
        End If

        'Add usernames to the lists
        Dim Count As Integer = 1
        For Each User In Users
            If Count <= 21 Then
                'First List
                UserLists(0).AddOption(User)
```

```
            Else
                'Second List
                UserLists(1).AddOption(User)
            End If
            Count += 1
        Next
    End Sub

    Public Overrides Sub HandleInput()
        If MenuButton.Clicked() = "Clicked" Then
            ScreenManager.UnloadScreen(Name)
            ScreenManager.AddScreen(New Title)
            ScreenManager.AddScreen(New SimulationButton)
            ScreenManager.AddScreen(New TestButton)
            ScreenManager.AddScreen(New MyProgressButton)
        End If

        Dim Result As String
        For Each FoundList In UserLists
            Result = FoundList.Update()
            If Result <> "" Then
                'Option has been chosen
                Advance(Result)
            End If
        Next

        If NewUserBox.HandleInput() = "Entered" Or (CreateUserButton.Clicked =
"Clicked" And NewUserBox.Text <> "") Then
            'Check against existing
            For Each User In Users
                If NewUserBox.Text = User Then
                    'Initialises a 2 second timer to display the error message for.
                    UserAlreadyExistsError = Now
                    Exit Sub
                End If
            Next

            File.WriteAllText(NewUserBox.Text & ".sv", "")
            RefreshExistingUserLists()

            NewUserBox.Text = ""
        End If
    End Sub

    Protected Overridable Sub Advance(ByVal ChosenOption As String)
        Main.CurrentUser = ChosenOption
        ScreenManager.UnloadScreen(Name)
    End Sub

    Public Overrides Sub Draw()
        'LEFT SIDE
        Main.GFX.DrawString("Already used this program?", Main.Arial_20_Bold,
Brushes.Black, 20, 50)
        Main.GFX.DrawString("Select your user name from the list:", Main.Arial_15,
Brushes.Black, 20, 90)
        For Each FoundList In UserLists
            FoundList.Draw()
        Next
        'CENTRE DIVIDER
        Main.GFX.DrawLine(New Pen(New SolidBrush(SectionColour), 5), 480, 50, 480,
280)
```

```
        Main.GFX.DrawString("OR", Main.Arial_20_Bold, New SolidBrush(SectionColour),
455, 290)
        Main.GFX.DrawLine(New Pen(New SolidBrush(SectionColour), 5), 480, 330, 480,
580)
        'Menu Button
        MenuButton.Draw()
        'RIGHT SIDE
        Main.GFX.DrawString("New User?", Main.Arial_20_Bold, Brushes.Black, 505, 50)
        Main.GFX.DrawString("Create a new user name:", Main.Arial_15, Brushes.Black,
505, 90)
        NewUserBox.Draw()
        CreateUserButton.Draw()
        Main.GFX.DrawString("Only letters and numbers can be used.", Main.Arial_10,
Brushes.Black, 505, 180)
        If NewUserBox.ReachedMaxChars = True Then
            Main.GFX.DrawString("User names cannot be longer than 10 characters.",
Main.Arial_10, Brushes.Red, 505, 195)
        Else
            Main.GFX.DrawString("User names cannot be longer than 10 characters.",
Main.Arial_10, Brushes.Black, 505, 195)
        End If
        If UserAlreadyExistsError <> Nothing Then
            If (Now - UserAlreadyExistsError).TotalMilliseconds > 2000 Then
                UserAlreadyExistsError = Nothing
            End If

            Main.GFX.DrawString("That user name already exists.", Main.Arial_10,
Brushes.Red, 505, 210)
        End If

    End Sub
End Class
```

TestUserSelection

The Test User Selection screen is red and point to the Test Menu.

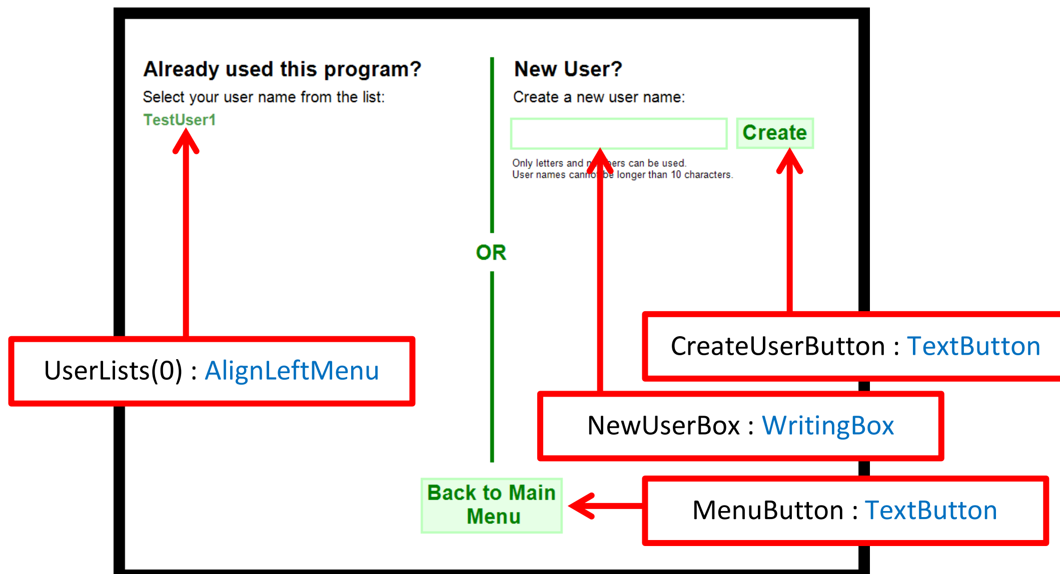



```
Imports System.IO

Public Class TestUserSelection
    Inherits UserSelection
```

```
    Public Sub New()
        'MAX USERNAME LENGTH IS 10

        MenuButton = New TextButton("Back to Main" & vbNewLine & "        Menu",
Main.Arial_20_Bold, ProgramSection.Test, New Point(387, 600), -1, -1, 3, 1)
        NewUserBox = New WritingBox(New Point(505, 130), Main.Arial_20_Bold,
ProgramSection.Test, 3, "WWWWWWWWWW")
        CreateUserButton = New TextButton("Create", Main.Arial_20_Bold,
ProgramSection.Test, New Point(800, 129), -1, -1, 3, 1)

        Name = "TestUserSelection"
        Location = New Point(0, 0)
        State = ScreenState.Active

        SectionColour = Color.FromArgb(199, 0, 0)

        RefreshExistingUserLists()
    End Sub

    Protected Overrides Sub Advance(ChosenOption As String)
        MyBase.Advance(ChosenOption)
        ScreenManager.AddScreen(New TestMenu)
    End Sub
End Class
```

MyProgressUserSelection

The My Progress User Selection screen is green and point to the My Progress Report.




```
Imports System.IO

Public Class MyProgressUserSelection
    Inherits UserSelection

    Public Sub New()
        'MAX USERNAME LENGTH IS 10

        MenuButton = New TextButton("Back to Main" & vbNewLine & "        Menu",
Main.Arial_20_Bold, ProgramSection.MyProgress, New Point(387, 600), -1, -1, 3, 1)
```

```
        NewUserBox = New WritingBox(New Point(505, 130), Main.Arial_20_Bold,
ProgramSection.MyProgress, 3, "WWWWWWWWWWW")
        CreateUserButton = New TextButton("Create", Main.Arial_20_Bold,
ProgramSection.MyProgress, New Point(800, 129), -1, -1, 3, 1)

        Name = "MyProgressUserSelection"
        Location = New Point(0, 0)
        State = ScreenState.Active

        SectionColour = Color.FromArgb(0, 128, 0)

        RefreshExistingUserLists()
    End Sub

    Protected Overrides Sub Advance(ChosenOption As String)
        MyBase.Advance(ChosenOption)
        ScreenManager.AddScreen(New MyProgressReport)
    End Sub
End Class
```

BaseButton

The Base Button is the parent/base class of the button tools. There are two inheriting buttons: PictureButton and TextButton. The most important function, Clicked, is used by the buttons' parent classes to see the state of the button: Clicked, MouseDown, Hover or nothing. The button will also draw differently depending on what state it's in.

```
Public Class BaseButton
    Public Location As New Point
    Public Size As New Size
    Public MouseHover, MouseDown As Boolean

    Public Function Clicked() As String
        If Windows.Forms.Form.MousePosition.X - Main.Left - 15 >= Location.X And
Windows.Forms.Form.MousePosition.X - Main.Left - 15 <= Location.X + Size.Width And
Windows.Forms.Form.MousePosition.Y - Main.Top - 15 >= Location.Y And
Windows.Forms.Form.MousePosition.Y - Main.Top - 15 <= Location.Y + Size.Height Then
            ' If the mouse cursor is inside the button
            MouseHover = True
            For Each Click In Main.MouseButtonsUp
                If Click.Button = MouseButtons.Left Then
                    ' If the left mouse button was held down and is now up (has been
clicked)
                    Return "Clicked"
                End If
            Next
            If Windows.Forms.Form.MouseButtons = MouseButtons.Left Then
                'If the left mouse button is held down
                MouseDown = True
                Return "MouseDown"
            Else
                MouseDown = False
            End If
            Return "Hover"
        Else
            MouseHover = False
        End If

        Return ""
    End Function
```

```
    Public Overridable Sub DrawDefault()
        'Draw if the mouse cursor is outside the button
    End Sub

    Public Overridable Sub DrawMouseHover()
        'Draw if the mouse cursor is in the button, but is not held down
    End Sub

    Public Overridable Sub DrawMouseDown()
        'Draw if the left mouse button is held down inside the cursor
    End Sub

    Public Sub Draw()
        'Choose the correct Draw procedure based on the state of the button:
        'Default, MouseHover or MouseDown
        If MouseHover = True Then
            If MouseDown = True Then
                DrawMouseDown()
            Else
                DrawMouseHover()
            End If
        Else
            DrawDefault()
        End If
    End Sub
End Class
```

PictureButton

This is a type of button for which each possible state of the button is a different picture. The only times in the program when I use Picture Buttons are for the Play, Pause and Stop buttons on each Simulation.

Example of a Picture Button. From left: Default, MouseHover, MouseDown

```
Public Class PictureButton
    Inherits BaseButton
    Private DefaultImage, MouseHoverImage, MouseDownImage As Image

    ''' <summary>
    '''
    ''' </summary>
    ''' <param name="InputLocation"></param>
    ''' <param name="InputDefaultImage"></param>
    ''' <param name="InputMouseHoverImage"></param>
    ''' <param name="InputMouseDownImage"></param>
    ''' <param name="InputWidth">Set to -1 for horizontal auto-sizing based on the
default image size.</param>
    ''' <param name="InputHeight">Set to -1 for vertical auto-sizing based on the
default image size.</param>
    ''' <remarks></remarks>
    Public Sub New(ByVal InputLocation As Point, ByVal InputDefaultImage As Image,
ByVal InputMouseHoverImage As Image, ByVal InputMouseDownImage As Image, ByVal
InputWidth As Integer, ByVal InputHeight As Integer)
```

```
        Location = InputLocation
        DefaultImage = InputDefaultImage
        MouseHoverImage = InputMouseHoverImage
        MouseDownImage = InputMouseDownImage

        If InputWidth = -1 Then
            Size.Width = DefaultImage.Width
        Else
            Size.Width = InputWidth
        End If
        If InputHeight = -1 Then
            Size.Height = DefaultImage.Height
        Else
            Size.Height = InputHeight
        End If
    End Sub

    Public Overrides Sub DrawDefault()
        Main.GFX.DrawImage(DefaultImage, Location)
    End Sub

    Public Overrides Sub DrawMouseHover()
        Main.GFX.DrawImage(MouseHoverImage, Location)
    End Sub

    Public Overrides Sub DrawMouseDown()
        Main.GFX.DrawImage(MouseDownImage, Location)
    End Sub
End Class
```

TextButton

This is a type of button for which each possible state of the button has a different Text, Background and Border colour. The button has a fixed Text which is set at instantiation. There are two ways to instantiate a Text Button: by manually setting all possible colours or by inputting a Program Section (Simulation, Test, MyProgress, Other), for which there are pre-set colours. I make a huge use of Text Buttons in my program.

Example of a Text Button. From left: Default, MouseHover, MouseDown

```
Public Enum ProgramSection
    Simulation
    Test
    MyProgress
    Other
End Enum

Public Class TextButton
    Inherits BaseButton

    Public Text As String
```

```
    Public DefaultBackColour, HoverBackColour, MouseDownBackColour, DefaultTextColour,
HoverTextColour, MouseDownTextColour, DefaultBorderColour, HoverBorderColour,
MouseDownBorderColour As Color
    Private TextFont As Font
    Private Margin As Point
    Private BorderThickness As Integer

    ''' <summary>
    ''' For completely custom buttons.
    ''' </summary>
    ''' <param name="InputText"></param>
    ''' <param name="InputTextFont"></param>
    ''' <param name="InputDefaultTextColour"></param>
    ''' <param name="InputDefaultBackColour"></param>
    ''' <param name="InputDefaultBorderColour"></param>
    ''' <param name="InputHoverTextColour"></param>
    ''' <param name="InputHoverBackColour"></param>
    ''' <param name="InputHoverBorderColour"></param>
    ''' <param name="InputPosition"></param>
    ''' <param name="InputWidth">Set to -1 for horizontal auto-sizing.</param>
    ''' <param name="InputHeight">Set to -1 for vertical auto-sizing.</param>
    ''' <param name="InputBorderThickness"></param>
    ''' <param name="InputMinimumMargin"></param>
    ''' <remarks></remarks>
    Public Sub New(ByVal InputText As String, ByVal InputTextFont As Font, ByVal
InputDefaultTextColour As Color, ByVal InputDefaultBackColour As Color, ByVal
InputDefaultBorderColour As Color, ByVal InputHoverTextColour As Color, ByVal
InputHoverBackColour As Color, ByVal InputHoverBorderColour As Color, ByVal
InputMouseDownTextColour As Color, ByVal InputMouseDownBackColour As Color, ByVal
InputMouseDownBorderColour As Color, ByVal InputPosition As Point, ByVal InputWidth As
Integer, ByVal InputHeight As Integer, ByVal InputBorderThickness As Integer, Optional
ByVal InputMinimumMargin As Integer = 3)
        Location = InputPosition
        Text = InputText
        TextFont = InputTextFont
        DefaultBackColour = InputDefaultBackColour
        HoverBackColour = InputHoverBackColour
        MouseDownBackColour = InputMouseDownBackColour
        DefaultTextColour = InputDefaultTextColour
        HoverTextColour = InputHoverTextColour
        MouseDownTextColour = InputMouseDownTextColour
        BorderThickness = InputBorderThickness
        DefaultBorderColour = InputDefaultBorderColour
        HoverBorderColour = InputHoverBorderColour
        MouseDownBorderColour = InputMouseDownBorderColour

        If InputWidth > -1 Then
            Size.Width = InputWidth
        Else
            'HORIZONTAL AUTOSIZING BASED ON THE WIDTH OF THE TEXT
            Size.Width = (BorderThickness + InputMinimumMargin) * 2 +
Main.GFX.MeasureString(Text, TextFont).Width
        End If
        If InputHeight > -1 Then
            Size.Height = InputHeight
        Else
            'VERTICAL AUTOSIZING BASED ON THE HEIGHT OF THE TEXT
            Size.Height = (BorderThickness + InputMinimumMargin) * 2 +
Main.GFX.MeasureString(Text, TextFont).Height
        End If
```

```
        Margin = New Point(Size.Width \ 2 - Main.GFX.MeasureString(Text,
TextFont).Width \ 2, Size.Height \ 2 - Main.GFX.MeasureString(Text, TextFont).Height \
2)
    End Sub

    ''' <summary>
    ''' For section-specific buttons to match the program colour scheme.
    ''' </summary>
    ''' <param name="InputText"></param>
    ''' <param name="InputTextFont"></param>
    ''' <param name="InputProgramSection"></param>
    ''' <param name="InputPosition"></param>
    ''' <param name="InputWidth">Set to -1 for horizontal auto-sizing.</param>
    ''' <param name="InputHeight">Set to -1 for vertical auto-sizing.</param>
    ''' <param name="InputBorderThickness"></param>
    ''' <param name="InputMinimumMargin"></param>
    ''' <remarks></remarks>
    Public Sub New(ByVal InputText As String, ByVal InputTextFont As Font, ByVal
InputProgramSection As ProgramSection, ByVal InputPosition As Point, ByVal InputWidth
As Integer, ByVal InputHeight As Integer, ByVal InputBorderThickness As Integer,
Optional ByVal InputMinimumMargin As Integer = 3)
        Location = InputPosition
        Text = InputText
        TextFont = InputTextFont
        Select Case InputProgramSection
            Case ProgramSection.Simulation
                DefaultBackColour = Color.FromArgb(217, 238, 255) 'Back with s: 26%
                HoverBackColour = Color.FromArgb(161, 213, 255) 'Back with s: 64%
                MouseDownBackColour = Color.FromArgb(107, 188, 255) 'Back on main
button
                DefaultTextColour = Color.FromArgb(0, 90, 194) 'Text on main button
                HoverTextColour = DefaultTextColour
                MouseDownTextColour = DefaultTextColour
                DefaultBorderColour = HoverBackColour
                HoverBorderColour = DefaultTextColour
                MouseDownBorderColour = DefaultTextColour
            Case ProgramSection.Test
                DefaultBackColour = Color.FromArgb(255, 227, 227) 'Back with s: 26%
                HoverBackColour = Color.FromArgb(255, 189, 189) 'Back with s: 64%
                MouseDownBackColour = Color.FromArgb(255, 150, 150) 'Back on main
button
                DefaultTextColour = Color.FromArgb(199, 0, 0) 'Text on main button
                HoverTextColour = DefaultTextColour
                MouseDownTextColour = DefaultTextColour
                DefaultBorderColour = HoverBackColour
                HoverBorderColour = DefaultTextColour
                MouseDownBorderColour = DefaultTextColour
            Case ProgramSection.MyProgress
                DefaultBackColour = Color.FromArgb(230, 255, 230) 'Back with s: 26%
                HoverBackColour = Color.FromArgb(189, 255, 189) 'Back with s: 64%
                MouseDownBackColour = Color.FromArgb(153, 255, 153) 'Back on main
button
                DefaultTextColour = Color.FromArgb(0, 128, 0) 'Text on main button
                HoverTextColour = DefaultTextColour
                MouseDownTextColour = DefaultTextColour
                DefaultBorderColour = HoverBackColour
                HoverBorderColour = DefaultTextColour
                MouseDownBorderColour = DefaultTextColour
            Case ProgramSection.Other
                DefaultBackColour = Color.FromArgb(248, 230, 255) 'Back with s: 26%
                HoverBackColour = Color.FromArgb(236, 189, 255) 'Back with s: 64%
```

```
                MouseDownBackColour = Color.FromArgb(226, 153, 255) 'Back on main
button
                DefaultTextColour = Color.FromArgb(166, 0, 232) 'Text on main button
                HoverTextColour = DefaultTextColour
                MouseDownTextColour = DefaultTextColour
                DefaultBorderColour = HoverBackColour
                HoverBorderColour = DefaultTextColour
                MouseDownBorderColour = DefaultTextColour
        End Select
        BorderThickness = InputBorderThickness

        If InputWidth > -1 Then
            Size.Width = InputWidth
        Else
            'HORIZONTAL AUTOSIZING BASED ON THE WIDTH OF THE TEXT
            Size.Width = (BorderThickness + InputMinimumMargin) * 2 +
Main.GFX.MeasureString(Text, TextFont).Width
        End If
        If InputHeight > -1 Then
            Size.Height = InputHeight
        Else
            'VERTICAL AUTOSIZING BASED ON THE HEIGHT OF THE TEXT
            Size.Height = (BorderThickness + InputMinimumMargin) * 2 +
Main.GFX.MeasureString(Text, TextFont).Height
        End If

        Margin = New Point(Size.Width \ 2 - Main.GFX.MeasureString(Text,
TextFont).Width \ 2, Size.Height \ 2 - Main.GFX.MeasureString(Text, TextFont).Height \
2)
    End Sub

    Public Overrides Sub DrawDefault()
        Main.GFX.FillRectangle(New SolidBrush(DefaultBackColour), Location.X,
Location.Y, Size.Width, Size.Height)
        Main.GFX.DrawRectangle(New Pen(New SolidBrush(DefaultBorderColour),
BorderThickness), Location.X + BorderThickness \ 2, Location.Y + BorderThickness \ 2,
Size.Width - BorderThickness, Size.Height - BorderThickness)
        Main.GFX.DrawString(Text, TextFont, New SolidBrush(DefaultTextColour),
Location.X + Margin.X, Location.Y + Margin.Y)
    End Sub

    Public Overrides Sub DrawMouseHover()
        Main.GFX.FillRectangle(New SolidBrush(HoverBackColour), Location.X,
Location.Y, Size.Width, Size.Height)
        Main.GFX.DrawRectangle(New Pen(New SolidBrush(HoverBorderColour),
BorderThickness), Location.X + BorderThickness \ 2, Location.Y + BorderThickness \ 2,
Size.Width - BorderThickness, Size.Height - BorderThickness)
        Main.GFX.DrawString(Text, TextFont, New SolidBrush(HoverTextColour),
Location.X + Margin.X, Location.Y + Margin.Y)
    End Sub

    Public Overrides Sub DrawMouseDown()
        Main.GFX.FillRectangle(New SolidBrush(MouseDownBackColour), Location.X,
Location.Y, Size.Width, Size.Height)
        Main.GFX.DrawRectangle(New Pen(New SolidBrush(MouseDownBorderColour),
BorderThickness), Location.X + BorderThickness \ 2, Location.Y + BorderThickness \ 2,
Size.Width - BorderThickness, Size.Height - BorderThickness)
        Main.GFX.DrawString(Text, TextFont, New SolidBrush(MouseDownTextColour),
Location.X + Margin.X, Location.Y + Margin.Y)
    End Sub
End Class
```

BaseMenu

This is the parent/base class for the Menu tools. There is a list of items and when an item is clicked, it's value is returned.

```
Public Class BaseMenu
    Public MenuFont As Font
    Public MenuLocation As Point
    Public MenuOptionY As Integer

    Public MenuOptions As New List(Of String)
    Public OptionDefaultColor, OptionMouseHoverColor As Color
    Public OptionDropShadow As Boolean
    Public DropShadowDepth As Integer

    Public Sub New(ByVal InputMenuLocation As Point, ByVal InputMenuFont As Font,
ByVal InputOptionDefaultColor As Color, ByVal InputOptionMouseHoverColor As Color,
Optional ByVal InputOptionDropShadow As Boolean = False)
        OptionDefaultColor = InputOptionDefaultColor
        OptionMouseHoverColor = InputOptionMouseHoverColor
        MenuFont = InputMenuFont
        OptionDropShadow = InputOptionDropShadow
        MenuLocation = InputMenuLocation

        If OptionDropShadow Then
            If MenuFont.Size > 35 Then
                DropShadowDepth = 2
            Else
                DropShadowDepth = 1
            End If
        End If

        MenuOptionY = MenuFont.Size * 1.5
    End Sub

    Public Sub AddOption(ByVal OptionName As String)
        MenuOptions.Add(OptionName)
    End Sub
End Class
```

AlignLeftMenu

This is a type of Menu for which all items start at the X coordinate given. These menus are used on the title screen and user selection screens.

Example of an Align Left Menu (from the Title screen)

```
Public Class AlignLeftMenu
    Inherits BaseMenu

    Public Sub New(ByVal InputMenuLocation As Point, ByVal InputMenuFont As Font,
ByVal InputOptionDefaultColor As Color, ByVal InputOptionMouseHoverColor As Color,
Optional ByVal InputOptionDropShadow As Boolean = False)
        MyBase.New(InputMenuLocation, InputMenuFont, InputOptionDefaultColor,
InputOptionMouseHoverColor, InputOptionDropShadow)
    End Sub
```

```
    Public Function Update()
        Dim Width, Height As Integer
        Dim count As Integer = 0

        If Main.MouseButtonsUp.Count > 0 Then
            For Each MouseThingy In Main.MouseButtonsUp
                If MouseThingy.Button = MouseButtons.Left Then
                    ' If the left mouse button has been clicked
                    'CHECK POSITION FOR EACH OPTION
                    For Each MenuOption In MenuOptions
                        Width = Main.GFX.MeasureString(MenuOption, MenuFont).Width
                        Height = Main.GFX.MeasureString(MenuOption, MenuFont).Height
                        If Windows.Forms.Form.MousePosition.X - Main.Left - 15 >
MenuLocation.X And Windows.Forms.Form.MousePosition.X - Main.Left - 15 <
MenuLocation.X + Width And Windows.Forms.Form.MousePosition.Y - Main.Top - 15 >
MenuLocation.Y + MenuOptionY * count And Windows.Forms.Form.MousePosition.Y - Main.Top
- 15 < MenuLocation.Y + MenuOptionY * (count + 1) Then
                            'Return the name of the Menu Option which has been clicked
                            Return MenuOption
                        End If
                        count += 1
                    Next

                    Exit For
                End If
            Next
        End If
        Return ""
    End Function

    Public Sub Draw()
        Dim Count As Integer = 0
        Dim Width, Height As Integer

        For Each MenuOption In MenuOptions
            Width = Main.GFX.MeasureString(MenuOption, MenuFont).Width
            Height = Main.GFX.MeasureString(MenuOption, MenuFont).Height

            If Windows.Forms.Form.MousePosition.X - Main.Left - 15 > MenuLocation.X
And Windows.Forms.Form.MousePosition.X - Main.Left - 15 < MenuLocation.X + Width And
Windows.Forms.Form.MousePosition.Y - Main.Top - 15 > MenuLocation.Y + MenuOptionY *
Count And Windows.Forms.Form.MousePosition.Y - Main.Top - 15 < MenuLocation.Y +
MenuOptionY * (Count + 1) Then
                'If Mouse is in option
                Main.GFX.DrawString(MenuOption, MenuFont, New
SolidBrush(OptionMouseHoverColor), New Point(MenuLocation.X, MenuLocation.Y +
MenuOptionY * Count))
            Else
                If OptionDropShadow Then
                    Main.GFX.DrawString(MenuOption, MenuFont, New
SolidBrush(Color.FromArgb(60 / 100 * 255, 0, 0, 0)), New Point(MenuLocation.X -
DropShadowDepth * 2, MenuLocation.Y + MenuOptionY * Count - DropShadowDepth * 2))
                    Main.GFX.DrawString(MenuOption, MenuFont, New
SolidBrush(Color.FromArgb(60 / 100 * 255, OptionDefaultColor)), New
Point(MenuLocation.X - DropShadowDepth, MenuLocation.Y + MenuOptionY * Count -
DropShadowDepth))
                End If
                Main.GFX.DrawString(MenuOption, MenuFont, New
SolidBrush(OptionDefaultColor), New Point(MenuLocation.X, MenuLocation.Y + MenuOptionY
* Count))
            End If
```

```
            Count += 1
        Next
    End Sub
End Class
```

AlignCentreMenu

This is a type of Menu for which all items are centred with the X coordinate given. Although I never make use of this tool in my program, I created it near the beginning as I thought that I may have a need for it.

```
Public Class AlignLeftMenu
    Inherits BaseMenu

    Public Sub New(ByVal InputMenuLocation As Point, ByVal InputMenuFont As Font,
ByVal InputOptionDefaultColor As Color, ByVal InputOptionMouseHoverColor As Color,
Optional ByVal InputOptionDropShadow As Boolean = False)
        MyBase.New(InputMenuLocation, InputMenuFont, InputOptionDefaultColor,
InputOptionMouseHoverColor, InputOptionDropShadow)
    End Sub

    Public Function Update()
        Dim Width, Height As Integer
        Dim count As Integer = 0

        If Main.MouseButtonsUp.Count > 0 Then
            For Each MouseThingy In Main.MouseButtonsUp
                If MouseThingy.Button = MouseButtons.Left Then
                    ' If the left mouse button has been clicked
                    'CHECK POSITION FOR EACH OPTION
                    For Each MenuOption In MenuOptions
                        Width = Main.GFX.MeasureString(MenuOption, MenuFont).Width
                        Height = Main.GFX.MeasureString(MenuOption, MenuFont).Height
                        If Windows.Forms.Form.MousePosition.X - Main.Left - 15 >
MenuLocation.X And Windows.Forms.Form.MousePosition.X - Main.Left - 15 <
MenuLocation.X + Width And Windows.Forms.Form.MousePosition.Y - Main.Top - 15 >
MenuLocation.Y + MenuOptionY * count And Windows.Forms.Form.MousePosition.Y - Main.Top
- 15 < MenuLocation.Y + MenuOptionY * (count + 1) Then
                            'Return the name of the Menu Option which has been clicked
                            Return MenuOption
                        End If
                        count += 1
                    Next

                    Exit For
                End If
            Next
        End If
        Return ""
    End Function

    Public Sub Draw()
        Dim Count As Integer = 0
        Dim Width, Height As Integer

        For Each MenuOption In MenuOptions
            Width = Main.GFX.MeasureString(MenuOption, MenuFont).Width
            Height = Main.GFX.MeasureString(MenuOption, MenuFont).Height
```

```
            If Windows.Forms.Form.MousePosition.X - Main.Left - 15 > MenuLocation.X
And Windows.Forms.Form.MousePosition.X - Main.Left - 15 < MenuLocation.X + Width And
Windows.Forms.Form.MousePosition.Y - Main.Top - 15 > MenuLocation.Y + MenuOptionY *
Count And Windows.Forms.Form.MousePosition.Y - Main.Top - 15 < MenuLocation.Y +
MenuOptionY * (Count + 1) Then
                'If Mouse is in option
                Main.GFX.DrawString(MenuOption, MenuFont, New
SolidBrush(OptionMouseHoverColor), New Point(MenuLocation.X, MenuLocation.Y +
MenuOptionY * Count))
            Else
                If OptionDropShadow Then
                    Main.GFX.DrawString(MenuOption, MenuFont, New
SolidBrush(Color.FromArgb(60 / 100 * 255, 0, 0, 0)), New Point(MenuLocation.X -
DropShadowDepth * 2, MenuLocation.Y + MenuOptionY * Count - DropShadowDepth * 2))
                    Main.GFX.DrawString(MenuOption, MenuFont, New
SolidBrush(Color.FromArgb(60 / 100 * 255, OptionDefaultColor)), New
Point(MenuLocation.X - DropShadowDepth, MenuLocation.Y + MenuOptionY * Count -
DropShadowDepth))
                End If
                Main.GFX.DrawString(MenuOption, MenuFont, New
SolidBrush(OptionDefaultColor), New Point(MenuLocation.X, MenuLocation.Y + MenuOptionY
* Count))
            End If

            Count += 1
        Next
    End Sub
End Class
```

NumberBox

This is a text box intended for the input of numbers only. It only accepts numerical input, with the exception of one dot (decimal place) anywhere except the beginning. It first calculates the Maximum number of characters allowed in the box based on the width of the box and the font size given. Its HandleInput function returns “Entered” when the enter key is pressed.

Number Boxes can be focused by clicking in them, and unfocused by pressing escape, enter, or clicking outside the box. A focused box will be listening for input, and will have a darker border.

There is a CheckFilled function which returns true only if the number in the box has at least 2 decimal places. This is used for the Test questions. The border of the box will flash if this function returns false.

```
Public Class NumberBox
    Public Text As String = ""
    Public Location As Point
    Private Size As Size
    Private BorderThickness, MaxChars As Integer
    Private Font As Font
    Public Focused As Boolean = False
    Public ReachedMaxChars As Boolean = False
    Private DefaultBorderColour, FocusedBorderColour As Color
    Private twoDPWarning As Boolean = False
```

```
    Public Sub New(ByVal InputLocation As Point, ByVal InputFont As Font, ByVal
InputProgramSection As ProgramSection, ByVal InputBorderThickness As Integer, ByVal
InputMaximumWidth As Integer)
        Location = InputLocation
        Font = InputFont
        BorderThickness = InputBorderThickness
        Size = New Size(InputMaximumWidth, Main.GFX.MeasureString("|", Font).Height +
2 * BorderThickness)
        Dim TempWidth As Integer
        Dim TempString As String

        'FIND THE MAXIMUM NUMBER OF CHARACTERS BASED ON THE MAXIMUM PIXEL WIDTH
        MaxChars = -1
        Do
            'Need to leave space for the decimal point and the input cursor
            TempString = ".|"
            MaxChars += 1
            For i = 1 To MaxChars
                TempString += "0"
            Next
            TempWidth = Main.GFX.MeasureString(TempString, Font).Width
            'See if the width of a string with a decimal point, and MaxChars '0's and
the | cursor is too large
        Loop Until TempWidth >= InputMaximumWidth
        'If it's slightly too large, then the max chars should be one less
        MaxChars -= 1

        Select Case InputProgramSection
            Case ProgramSection.Simulation
                DefaultBorderColour = Color.FromArgb(161, 213, 255)
                FocusedBorderColour = Color.FromArgb(0, 90, 194)
            Case ProgramSection.Test
                DefaultBorderColour = Color.FromArgb(255, 189, 189)
                FocusedBorderColour = Color.FromArgb(199, 0, 0)
            Case ProgramSection.MyProgress

            Case ProgramSection.Other

        End Select
    End Sub

    Public Function HandleInput()
        If Focused = True Then
            For Each Key In Main.KeysDown

                If Key >= 96 And Key <= 105 Then
                    'If it's a number on the numPad, change the code so that it's the
same
                    Key -= 48
                End If

                If Key = 190 Then
                    'If it's the dot on the numpad, change the code as if it's on the
main keyboard
                    Key = 110
                End If

                Select Case Key
                    Case 13
                        'Enter
                        Focused = False
```

```
                    If Text.Length > 0 Then
                        Return "Entered"
                    End If
                Case 27
                    'Escape
                    Text = ""
                    Focused = False
                Case 48 To 57
                    'Number
                    If Text.Length < MaxChars Then
                        Text &= Chr(Key)
                    Else
                        ReachedMaxChars = True
                    End If
                Case 110
                    'Dot
                    If Text.Length < MaxChars And InStr(Text, ".") = 0 And
Text.Length > 0 Then
                        Text &= "."
                    Else
                        ReachedMaxChars = True
                    End If
                Case 8
                    'Backspace
                    If Text.Length > 0 Then
                        Text = Text.Substring(0, Text.Length - 1)
                    End If
            End Select

        Next
    End If

    If Text.Length < MaxChars Then
        ReachedMaxChars = False
    End If

    For Each Click In Main.MouseButtonsUp
        If Click.Button = MouseButtons.Left Then
            If Click.Location.X >= Location.X And Click.Location.X < Location.X +
Size.Width And Click.Location.Y >= Location.Y And Click.Location.Y < Location.Y +
Size.Height Then
                'Mouse up in text box
                Focused = True
                twoDPWarning = False
            Else
                'Mouse up out of text box
                Focused = False
            End If
        End If
    Next

    Return ""
End Function

Public Function CheckFilled() As Boolean
    'Checks whether the number box has been typed into, and that is contains a
number
    'with at least two decimal places

    If Text <> "" Then
        If InStr(Text, ".") <> 0 Then
```

```
                'if is not empty and contains the decimal point
                If Split(Text, ".")(1).Length >= 2 Then
                    'if the number of chars after the decimal point is at least 2
                    Return True
                Else
                    twoDPWarning = True
                End If
            Else
                twoDPWarning = True
            End If
        End If
        Return False
    End Function

    Public Sub Draw()
        'Draw Border
        If twoDPWarning = True And Now.Millisecond < 500 Then
            Main.GFX.DrawRectangle(New Pen(New SolidBrush(FocusedBorderColour),
BorderThickness), Location.X, Location.Y, Size.Width, Size.Height)
        Else
            If Focused = True Then
                Main.GFX.DrawRectangle(New Pen(New SolidBrush(FocusedBorderColour),
BorderThickness), Location.X, Location.Y, Size.Width, Size.Height)
            Else
                Main.GFX.DrawRectangle(New Pen(New SolidBrush(DefaultBorderColour),
BorderThickness), Location.X, Location.Y, Size.Width, Size.Height)
            End If
        End If

        'Draw Text
        If Focused = True And Now.Millisecond < 500 Then
            'This means that the "|" symbol only shows every other half second
            Main.GFX.DrawString(Text & "|", Font, Brushes.Black, New Point(Location.X
+ BorderThickness, Location.Y + BorderThickness))
        Else
            Main.GFX.DrawString(Text, Font, Brushes.Black, New Point(Location.X +
BorderThickness, Location.Y + BorderThickness))
        End If
    End Sub
End Class
```

WritingBox

This is a type of Text Box which only accepts uppercase letters, lowercase letters and numbers. Other than this, it works similarly to the Number box. It is used only for the user selection screens for the new user text boxes.

```
Public Class WritingBox
    Public Text As String = ""
    Private Location As Point
    Private Size As Size
    Private BorderThickness, MaxChars As Integer
    Private Font As Font
    Private Focused As Boolean = False
    Public ReachedMaxChars As Boolean = False
    Private DefaultBorderColour, FocusedBorderColour As Color
```

```
    Public Sub New(ByVal InputLocation As Point, ByVal InputFont As Font, ByVal
InputProgramSection As ProgramSection, ByVal InputBorderThickness As Integer, ByVal
InputMaximumLengthString As String)
        Location = InputLocation
        Font = InputFont
        BorderThickness = InputBorderThickness
        MaxChars = InputMaximumLengthString.Length
        Size = New Size(Main.GFX.MeasureString(InputMaximumLengthString & "|",
Font).Width + 2 * BorderThickness, Main.GFX.MeasureString(InputMaximumLengthString &
"|", Font).Height + 2 * BorderThickness)

        Select Case InputProgramSection
            Case ProgramSection.Simulation
                DefaultBorderColour = Color.FromArgb(161, 213, 255)
                FocusedBorderColour = Color.FromArgb(0, 90, 194)
            Case ProgramSection.Test
                DefaultBorderColour = Color.FromArgb(255, 189, 189)
                FocusedBorderColour = Color.FromArgb(199, 0, 0)
            Case ProgramSection.MyProgress
                DefaultBorderColour = Color.FromArgb(189, 255, 189)
                FocusedBorderColour = Color.FromArgb(0, 128, 0)
            Case ProgramSection.Other

        End Select
    End Sub

    Public Function HandleInput()
        If Focused = True Then
            For Each Key In Main.KeysDown

                If Key >= 96 And Key <= 105 Then
                    'If it's a number on the numPad, change the code so that it's the
same
                    Key -= 48
                End If

                Select Case Key
                    Case 13
                        'Enter
                        Focused = False
                        If Text.Length > 0 Then
                            Return "Entered"
                        End If
                    Case 27
                        'Escape
                        Text = ""
                        Focused = False
                    Case 48 To 57
                        'Number
                        If Text.Length < MaxChars Then
                            Text &= Chr(Key)
                        Else
                            ReachedMaxChars = True
                        End If
                    Case 65 To 90
                        'Letter
                        If Text.Length < MaxChars Then
                            If Windows.Forms.Form.ModifierKeys = Keys.Shift Then
                                Text &= UCase(Chr(Key))
                            Else
```

```
                            Text &= LCase(Chr(Key))
                        End If
                    Else
                        ReachedMaxChars = True
                    End If
                Case 8
                    'Backspace
                    If Text.Length > 0 Then
                        Text = Text.Substring(0, Text.Length - 1)
                    End If
            End Select

        Next
    End If

    If Text.Length < MaxChars Then
        ReachedMaxChars = False
    End If

    For Each Click In Main.MouseButtonsUp
        If Click.Button = MouseButtons.Left Then
            If Click.Location.X >= Location.X And Click.Location.X < Location.X +
Size.Width And Click.Location.Y >= Location.Y And Click.Location.Y < Location.Y +
Size.Height Then
                'Mouse up in text box
                Focused = True
            Else
                'Mouse up out of text box
                Focused = False
            End If
        End If
    Next

    Return ""
End Function

Public Sub Draw()
    'Draw Border
    If Focused = True Then
        Main.GFX.DrawRectangle(New Pen(New SolidBrush(FocusedBorderColour),
BorderThickness), Location.X, Location.Y, Size.Width, Size.Height)
    Else
        Main.GFX.DrawRectangle(New Pen(New SolidBrush(DefaultBorderColour),
BorderThickness), Location.X, Location.Y, Size.Width, Size.Height)
    End If
    'Draw Text
    If Focused = True And Now.Millisecond < 500 Then
        Main.GFX.DrawString(Text & "|", Font, Brushes.Black, New Point(Location.X
+ BorderThickness, Location.Y + BorderThickness))
    Else
        Main.GFX.DrawString(Text, Font, Brushes.Black, New Point(Location.X +
BorderThickness, Location.Y + BorderThickness))
    End If

End Sub
End Class
```

User Manual

I have created a User Guide for my program. It can be found at Appendix 1 at the end of this document, on page 200.

Appraisal

Completion of Project Objectives

General Objectives

Below are the seven general objectives that I established before creating the program, along with a brief description about whether or not I have met each objective.

1. Create a VB.NET Windows Forms Application which could be used to help to teach students Mechanics principles for the first time.

Objective Met

One of the three main sections of my program is the Simulation section. This contains all three Simulations completely unlocked. For each one, there are lots of variables which can be changed before the Simulation is started. Changing these variables change the outcome of the Simulation. This mode could be useful for teacher demonstrations, since the teacher could set up a situation that they want the whole class to work with, and project it onto the whiteboard.

2. The program should also act as an effective revision tool for students.

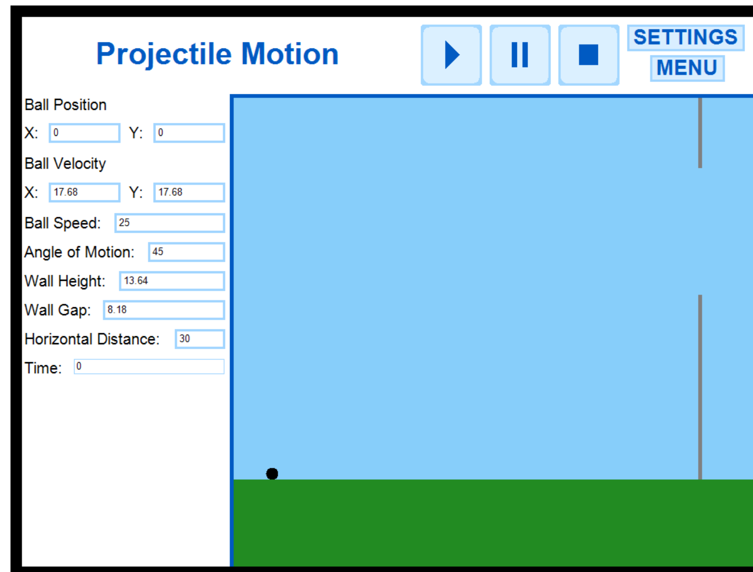
Objective Met

The second section of the program is the Test section which is used (as the name suggests) to test students' performances in the different categories. Once the theory has been learned, students could revise by practising on the questions in this section.

3. There should be at least one simulation about projectile motion.

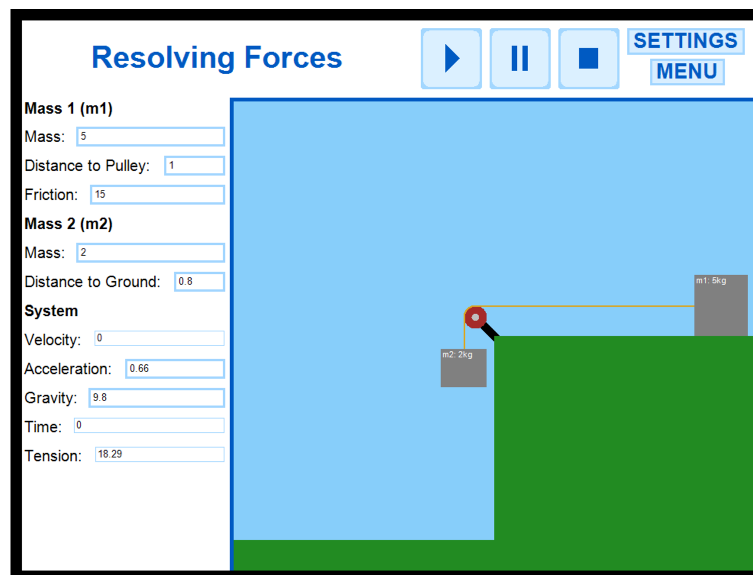
Objective Met

The image below shows the Simulation for Projectile Motion which I have created.



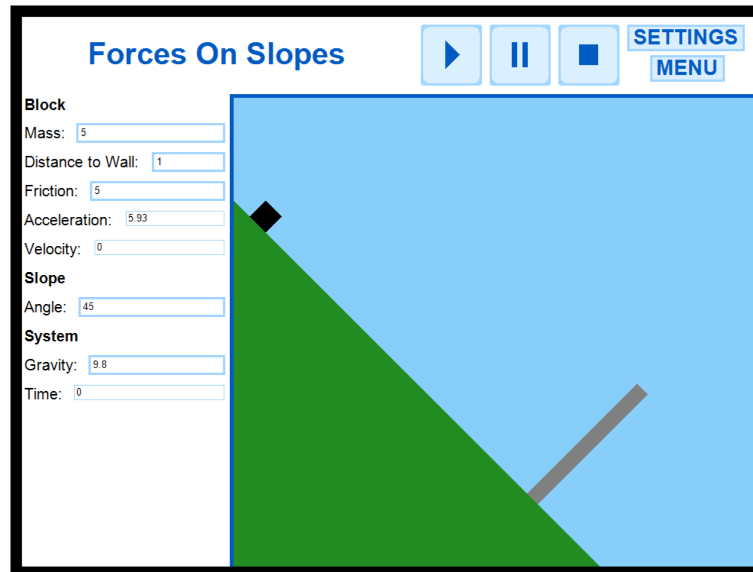
4. **There should be at least one simulation about resolving forces.**

Objective Met The image below shows the Simulation for Resolving Forces which I have created.



5. **There should be at least one simulation about resolving forces at angles ("Stuff on slopes").**

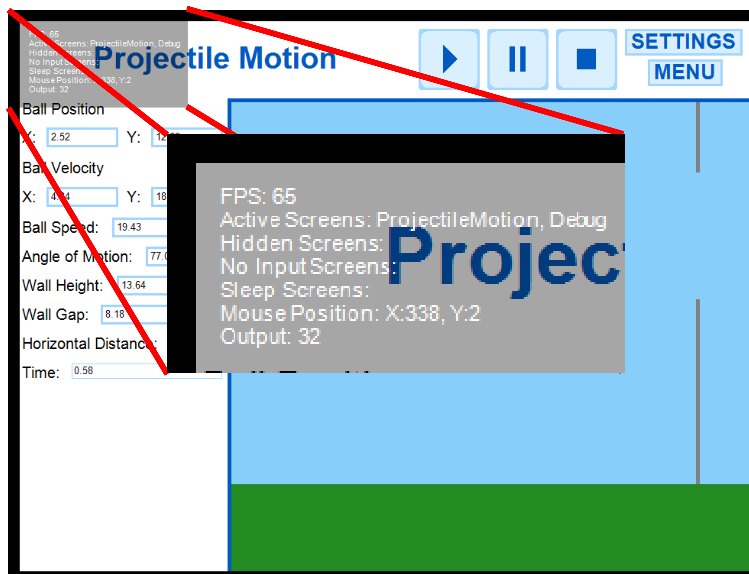
Objective Met The image below shows the Simulation for Forces On Slopes which I have created.



6. **There should be a graphics system in place which ensures that the simulations run smoothly without any flickering or 'lag' on an average machine.**

Objective Met

The conventional Windows Forms graphics method, of moving around pre-designed objects from design view, updates the user's view each time an object is moved. If an object is moved very frequently, or if multiple objects are being moved at once, this is likely to cause flickering. Conversely to this, with my graphics drawing method the user's view isn't updated until all of the drawing for a cycle has finished. This single update is called a frame. The image below shows that my program runs at 65fps (frames per second) while the Projectile Motion Simulation runs, which is much more than the human eye can distinguish between. That image was taken using one of the computers at college, which would be expected to be used by students at college. Also, my graphics drawing system is also not so complicated and intensive that it causes lag (unnaturally long pauses between screen updates or input) due to too many calculations.

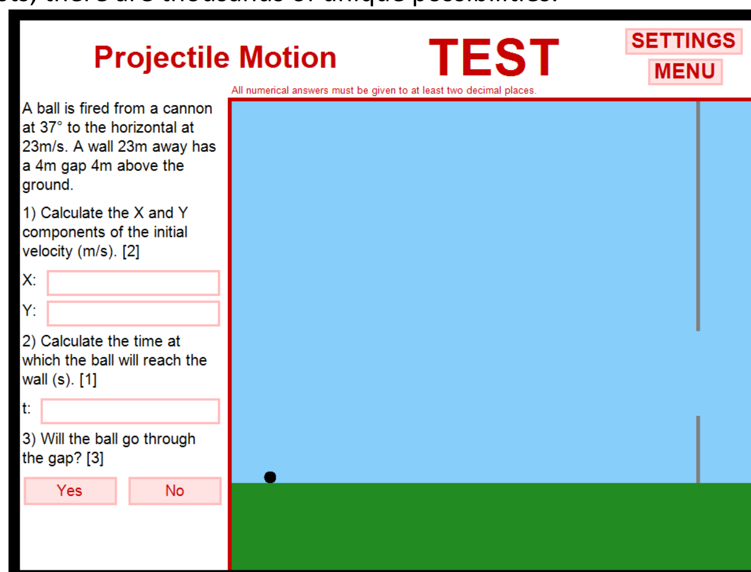


7. **As well as the simulations the program should include a test mode, in which the user is asked an exam-style question based on the starting condition of a prepared situation before seeing a simulation that reveals the answer.**

Objective Met and Exceeded

For each of the categories in the Simulation section of the program there is a corresponding Test. The test consists of an exam-style question based on that category. There is a Simulation next to the question which starts running as soon as the User finishes entering answers to all parts of the question. The Simulation shows the outcome of the situation described by the question.

This objective has been exceeded because the starting conditions (numbers in the question) each Test question is randomly generated. This means that, although there are only three Tests, there are thousands of unique possibilities.



Specific Objectives

Below are the eleven specific objectives that I established before creating the program, along with a brief description about whether or not I have met each objective.

1. **In the test mode the questions asked should have a total mark and the user's answers should be marked as a percentage.**

Objective Met

After the Test has been completed and the Simulation has run, the Test Report screen is shown. This shows the correct answers to the question. This screen adds up all of the marks for each part of the question, and calculates a percentage score. See tests 10.1, 10.2 and 10.3 (starting on page 68) for the system tests of this feature.

2. **Each time a user answers a question in test mode, the score, date/time of answering and question category should be saved in a text file.**

Objective Met

Test results are save in the format "Category,Score,TimeScored|". This is important, as a consistent format is needed for understanding the user data later. See system test series 5 (Page 52) and test 8.1 (Page 63) for evidence of the program correctly saving tests in this format, and writing data to the text file.

3. **Each user of the program on a machine should have their own progress text file assigned to them. If a new user uses the program, a new text file should be created.**

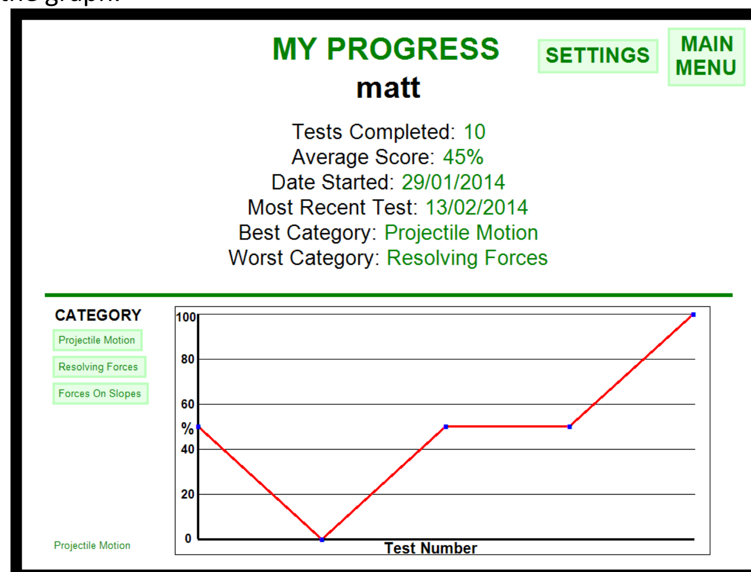
Objective Met

A new text file is created each time someone creates a new User Profile. See system test 9.5 on page 64 for evidence of this working. Each text file is named using the format “UserName.sv”. The “.sv” file extension hides the fact that it is a text file, and lets my program distinguish it from an ordinary text file. This system is one reason why all User Names on a machine need to be unique.

4. **A user should be able to view their progress over time with the test mode for any particular question category. This could be displayed as a graph.**

Objective Met

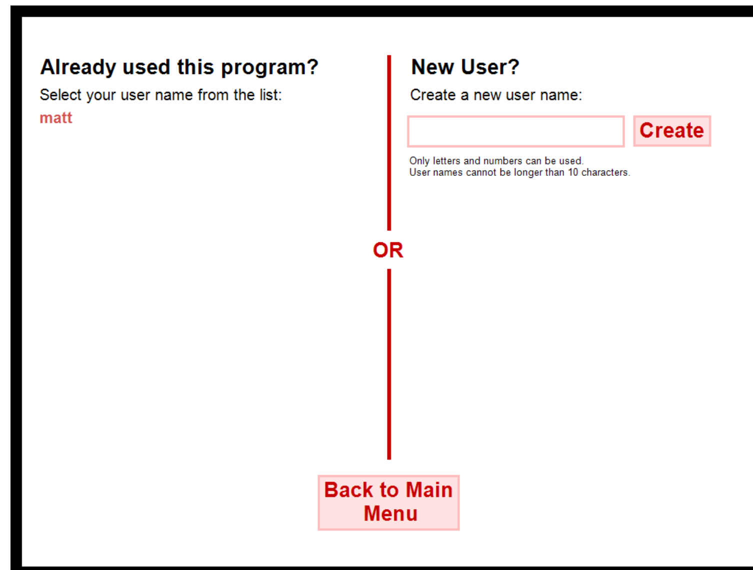
The third and final main section of the program is the My Progress section. This is designed for Users to review their performance in the Test section. The top half of the screen displays statistics about their performance as a whole, and the bottom half displays a graph showing information about individual categories. The graph shows the percentage score for each test completed in a category in chronological order, and draws lines between these data points. The buttons left of the graph can be clicked on to change the category displayed on the graph.



5. **When the test mode or ‘My Progress’ is selected, a list of existing users should be displayed. If the user has used the program on that machine before, they can select their name from the list. If they are not on the list, there will be a text box for them to create a new user name, thus creating a new progress text file.**

Objective Met

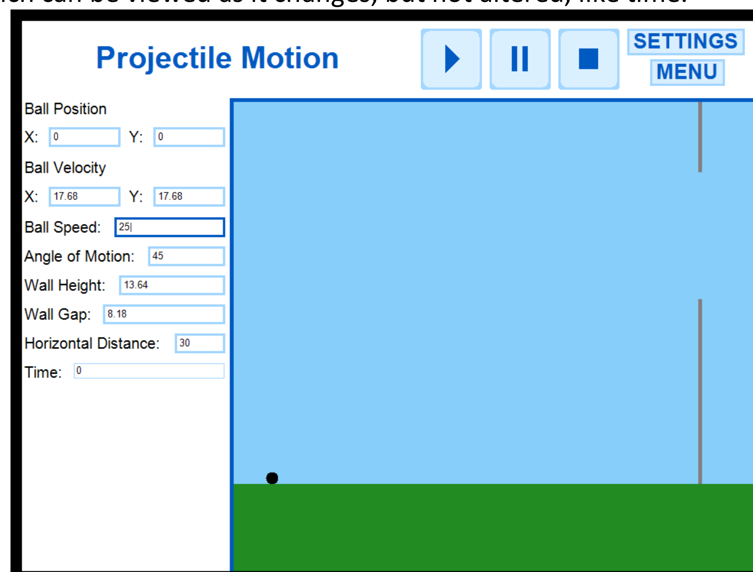
This has been exactly implemented. The image below shows the User Selection screen for the Test section. The one for the My Progress section functions identically, except from it is green and it points toward the My Progress Report screen and not the Tests Menu. The left side of the screen contains a list of up to 42 existing User Names which is found by reading the User text file directory. Clicking on one of the names will select it. The left side of the screen is for creating a new User Name. This has various validation procedures which are tested in system test series 9 (starting on page 64).



6. **The simulations should be visually pre-set, but users should be able to input/alter starting variables and constants before running the simulation.**

Objective Met

On each Simulation screen, the first frame of the animation is presented immediately, and the situation for each one is always the same. For example, the Projectile Motion Simulation (pictured below) will always involve a ball being fired towards a wall with a gap in it and the Resolving Forces Simulation will always involve two masses connected by a string over a pulley. However, the important variables which affect the exact outcome of the Simulation (e.g. Mass, initial velocity or angles) can be changed before the Simulation starts running. On each Simulation, a variable text box which has a thinner border is one for a variable which can be viewed as it changes, but not altered, like time.



7. **Simulations should be able to be paused at any time.**

Objective Met

Each Simulation has a big, easy to see Pause button which will instantly pause the running of the Simulation once clicked. This could be used for viewing the values of variables at a particular time. The Simulation can easily be resumed by clicking the Play button.

8. **There should be keyboard bindings to the simulation play, pause and reset functions. For example, the user could press the space bar to pause the simulation. This would make those functions easier to use, and gives an alternative to clicking with the mouse.**

Objective Partially Met

There is a keyboard binding: when the Space Bar is pressed, the running of the Simulation is toggle on or off. So, if the Space Bar is pressed while the Simulation is running, it will be paused. If the Space Bar is pressed when the Simulation is paused, it will continue to run. However, there is not keyboard shortcut for resetting the Simulation, although this would be fairly easy to implement.

9. **I will need to be able to use traditional SI units for quantities, such as "metres per second" for velocity, rather than "pixels per tick". For this reason, I will create a method for converting between the pixel and metre forms.**

Objective Met

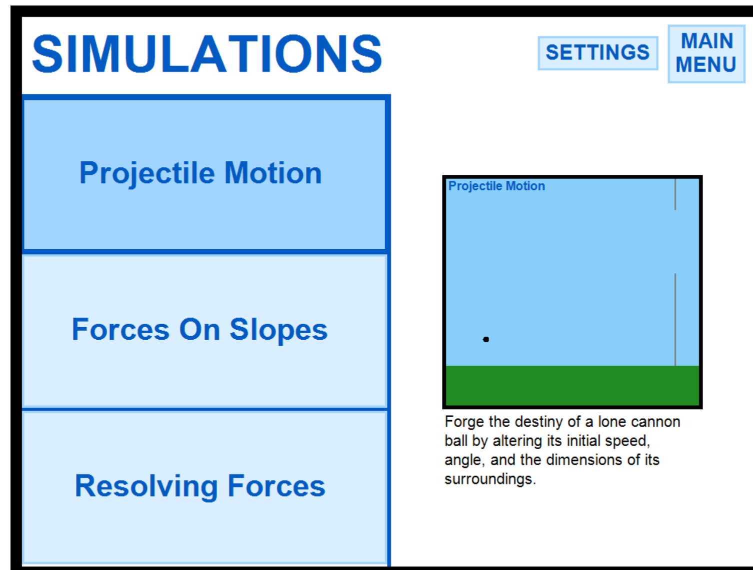
For each Simulation, I have functions for converting between pixels and metres. The code for these is shown below. These functions make use of a variable called Scale. This is a value which represents the number of pixels per metre. The scale value can be set by knowing what a set distance needs to be, in metres, given that you know its distance on the screen, in pixels. For example, in the Projectile Motion Simulation, it is known that the pixel distance between the initial ball position and the wall is 550px. To get a value for scale, the program finds out what the horizontal distance needs to be in metres (either by user input, or generated by a Test) and does the calculation 550/MetreDistance. A similar method is used for the other two Simulations.

```
Public Function Metres(ByVal Pixels As Double) As Double
    Return Pixels / Scale
End Function

Public Function Pixels(ByVal Metres As Double) As Double
    Return Metres * Scale
End Function
```

10. **On the menu for selecting simulations, there should be an image previewing each simulation. This would make the program look more interesting, as well as giving the user a taster of each simulation before needing to run them.**

Objective Met and Exceeded

The image below shows the Simulations Menu. There are three big buttons down the left side, one for selecting each Simulation. When the mouse cursor is hovered over one of these buttons, a preview for the corresponding Simulation is shown on the right side of the screen. This preview involves a brief description, the Simulation title, and not just an image, but and animation sample. This animation is made by running the actual Simulation with some set initial conditions. The Simulation is drawn to the screen half-sized.



11. **Each user's progress data string should be encrypted before being written to file, to prevent users from cheating by altering their scores.**

Objective Met I have created functions for encrypting and decrypting strings and I use this to secure user data before saving it to their text files. A detailed explanation of these functions can be found in the System Maintenance section, on page 85, and evidence of them working can be found in system test series 7, starting on page 56.

Evidence of Authenticated User Feedback

Below are the E-mails sent by my end user as feedback after I sent him the finished program.

Subject: Re: Computing Project
Created By: TDW@barton.ac.uk
Creation Date: 3/31/2014 11:05 AM
From: Tristan White

Recipient	Action	Date & Time	Comment
To: Matthew ARNOLD (4MAR1006@barton.ac.uk)			

Hi Matt

The program is excellent. I liked the user friendliness; it was a visually pleasing program to use! Would it be possible to remove the settings button from most places? I found myself clicking on it when trying to get back to the main menu by mistake, and it isn't relevant whilst doing the test or the simulations.

The simulations were really useful, and showed what was going on well. To improve them, could you have a pop up of a quick guide on how to actually work the maths through for those students who have forgotten that bit? Also, could you have some way for the timer to stop when the ball hits the floor etc? That way it would be possible to check your answers were correct!

The tests were great; they clearly well thought out, and it was useful to be able to track progress. The only other thought is that they don't quite go to exam difficulty. Consider adding a 4th problem, where the thing on the slope has another force added in at varying angles.

Thanks very much Matt for this program, I will be sending it out to my AS mechanics classes to aid in their revision.

Tristan.

Subject: Re: Computing Project
Created By: TDW@barton.ac.uk
Creation Date: 3/31/2014 11:05 AM
From: Tristan White

Recipient	Action	Date & Time	Comment
To: Matthew ARNOLD (4MAR1006@barton.ac.uk)			

Final thought,

Could you have an exit button everywhere so I don't have to click all the way back to get out?

T

Analysis of User Feedback

I have read through my User's feedback and condensed it into a table of positive feedback, negative feedback and possible improvements.

Positive	Negative	Improvements
User Friendly	Easy to accidentally click the settings button instead of the menu button	Remove settings button from unnecessary places
Visually Pleasing	Tests are not difficult enough compared to some exam questions	Add a pop up for the Simulations showing some of the theory associated with that topic
The Simulations showed what was going on well	It seems unnecessary to click lots of buttons to have to exit the program	Allow the Simulation to pause when important events happen
Tests were well thought-out		Add a new simulation where there are forces at angles on an object on a slope
Useful to be able to track progress		Add an exit button everywhere

My User liked how nice the program looked and how the Simulations showed the situation clearly. I think that this is due to my graphics drawing method, as I can make many different shapes. He also said that the program was easy to use, which I think is partly down to the buttons being large, obvious and consistent.

However, he thought that there were too many buttons pointing to the settings screen and this caused him to accidentally click that button when wanting to click the "back to menu" button. There are even settings buttons on each simulation and test. This is unnecessary because none of the program settings affect those screens.

Possible Extensions

After analysing my User feedback, I have made a list of possible future improvements which I could implement into the program if I had more time:

1. Remove the Settings Buttons from all screens except the title screen. The settings screen is not needed enough to warrant having a button to it on almost every screen. Not only are the buttons not needed, but they take up space and make screens more confusing
2. Add an option to each Simulation to pause it when it is "finished". A Simulation could be classed as finished when the main event has stopped happening. Each existing Simulation could be finished when:
    a. Projectile Motion: The ball reaches the wall
    b. Resolving Forces: The second mass hits the floor, and the system stops moving
    c. Forces On Slopes: The block hits the wall
3. Add a button which is always visible for exiting the program. This could be put on the border of the main program window.
4. On all Simulations, add a help button which brings up a pop-up. This would show the Mechanics theory associated with the topic of the Simulation. For example, for Projectile Motion it could show some of the equations of motion which are used for the Simulation. This feature would make the system easier to learn from
5. An obvious improvement would be to add more situations for Simulations and Tests. This would make the program useful for more than just three sub-topics of Mechanics. My User suggested a situation similar to the Forces On Slopes one, but with a force acting on the block at an angle. This would be a more difficult situation to deal with and would make the program reach the highest difficulty possible for actual exams.

Appendices

Appendix 1 – User Guide

Starting on the next page is the User guide for my program.