# Fourth Year Project: Technical Specification

Author: Conor Reddin
Student number: 13336766

Project Title: “Physics Room”
Supervisor: Dr. Mark Humphrys
Date Finished: 21/05/2017

## Table of Contents

## Abstract

The project described in this document is a ‘physics room’ simulator in which the user has complete control over the laws of physics acting on a sphere. It is a web based application which requires no installation to run.

The application contains built-in, preset questions which can be easily loaded by the user. The application will provide the user will an image of the question directly from a Junior or Leaving Cert exam paper and assist them in answering it. This, along with additional functions and features described below, provides the user with a “hands on” approach to physics and is intended to be both a fun and educational tool in which the student is free to explore.

## Motivation

The motivation for this project was a combination of both my interest in mathematics and my own struggles with studying at secondary level education. Some students, such as myself, can find it difficult to study from text-based documents, lecture notes, text books, etc. Depending on how an individual learns, methods such as this can prove challenging to develop an understanding of a given subject.

The aim of this project was to create an interactive, visual-based environment in which the Newtonian laws of physics may feel more intuitive and students may develop an understanding of the various factors and interactions.

Sample questions from Junior and Leaving Certificate level exams have also been included to better aid students undertaking this material in an academic environment. Visual cues within the environment correspond to explanations provided in the user Interface. These explanations are dynamic and will adjust to the inputs of a given question (which may be manually entered by the user).

Physical forces and phenomena such as acceleration due to gravity, velocity of a moving body, interactions between bodies (elasticity and friction), dimensions and placement of the body can all be controlled. This provides the student with the ability to adjust and manipulate the environment within the application.

Ideas for both the languages and the interface were as a result of collaboration with my supervisor, Dr. Mark Humphrys. Dr. Humphrys had suggested three.js as potential option for multiple reasons. One of the main reasons being that three.js would allow the application to be more accessible for students. By developing the application in Three.js, this allowed it to be web-based and thus no installation is required in order to use it. Aside from this, Three.js remains a new language in development, which may allow additional opportunities for this project to be further developed in the future.

Following a decision on the main language to be used in the development, considerations for the layout of the project took place. Multiple ideas for the layout of the project came to mind. All of these had a common idea of a 3D world with a user interface located at the side of the screen. This became the framework towards-which this project was gradually built.

## Research

Prior to development of this application, research was undertaken in order to scope the project requirements and potential tools to be utilized in development.

Research began on different methods of developing 3D environments in order to run the application. A common method that had been suggested was that of 'Unity'. Additional research was carried out in relation to Unity and projects that had been carried out with its application. Although the projects that were reviewed had been impressive, there was an issue regarding accessibility. Unfortunately, Unity would not have proved to be an accessible method either for developing the application or for users wishing to access it easily.

As mentioned previously, a priority was to make the application as accessible to as many people as possible. It was this reason that resulted in the decision to proceed with the project as a web-based application. Research was carried out on various choices for developing an online simulator. A frequently encountered option was that of webGL. I began researching webGL and how it could be utilized to implement the project in mind. This in turn led me to a webGL API for JavaScript called Three.js. Three.js was found to be a powerful JavaScript API for developing 3D worlds online.

Once well-formed ideas for both 'what I would build' and 'how I would build it' were established, considerations turned to the additional tools that would be needed to carry out this task.

Research was conducted on various different three.js projects for ideas on how the application could be developed. This was challenging at first and required me to refresh my memory with JavaScript and then only later move on to three.js.

Three weeks were dedicated to learning JavaScript in order to gain fluency, prior to attempting three.js. This was conducted with the help of "w3schools.com". This provided me with the knowledge to then move on the Three.js.

Given that it was a relatively new language, the research process did not turn up a great deal of detailed documentation on Three.js. The Three.js home website "threejs.org" was, however, helpful with a large pool of examples from different users. This proved to be a useful the source of guidance on several occasions.

Some of the issues encountered at the early stages of the project were server and browser-related. In these instances, "stackoverflow.com" was found to be a helpful source of information, providing clear explanations for many issues.

Shortly after the outset of this project, it was felt that designing the physics in JavaScript from scratch could prove to be too burdensome. Research therefore commenced on third party APIs to incorporate the physics into the application. One such API, which had been repeatedly encountered, was Phyis.js. Physi.js examples were researched, as was the process by which they could be incorporated into the application.

Initially this course of action showed promise, but unfortunately Physi.js was found to be very poorly documented and so it was decided not to incorporate it. This then led to the decision to develop the physics which would be used within the application. Once this decision was made, further research was carried out on the relevant mathematical formulae that would need to be used.

## Design

The on-screen layout of this application is a 3D environment, within which the user can control and observe a sphere moving under physical forces. The sphere will bounce, reflect off walls, cast shadows and behave differently in changing environments (inclined floor, etc.).

When the user loads the application (by accessing the website) they are presented with a room with a ball located in its center, above the floor. On the left hand side of the screen, the user is asked to select the desired education level from three options ('Junior Cert', 'Leaving Cert' and 'Third Level'). Once the user has selected one of these options, they are then asked to select a subject (either Physics or Applied Maths). Finally, the user can select an exam question from that subject and education level which they would like to simulate in the application.

The design was intended to be as intuitive as possible for the user. The User will only be presented with options which they can make at that time. After each selection (e.g. education level or subject) additional options will be presented. Menus and buttons are sized and colored so as to draw the attention of the user while at the same time offering a visual appeal. At all stages the user can press the 'Home' button to revert to start of the selection process.

Once the user has chosen the question that he/she would like to simulate, the application loads the relevant settings and asks the user to press start. Upon pressing the start button, a simulation of the question will be carried out.

For the sake of illustration, a white trail follows the path of the ball as it moves around the room. The point at which the ball first hits that ground will be marked with a black circle and the highest point reached is marked with a red line. The vertical and horizontal displacement of the ball before hitting the ground is also marked with blue lines. These are adapted for some questions so as to better illustrate what that question in particular has asked, such as the horizontal distance between two points in the balls flight path. At all times a legend can be seen in the bottom right-hand sign of the screen to provide insight into the meaning of these visual aids.

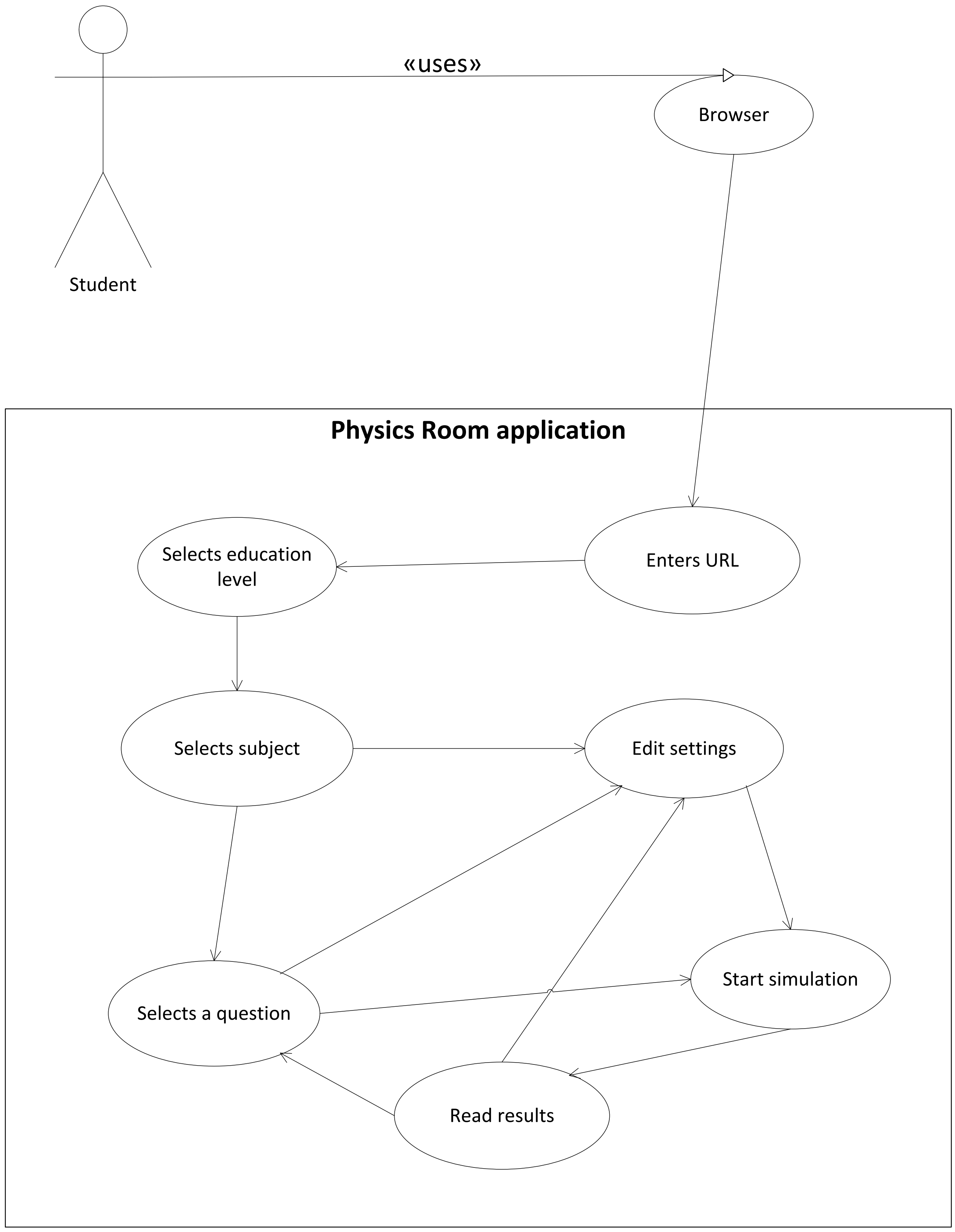
The 'Q&A' button will toggle both the question and an explanation of the answer on and off the screen. The answers which have been provided intend to give a deeper understanding of the underlying physics and will respond to manual changes in the inputs carried out by the user.

These manual changes can be carried out via the 'Build your own' button. Pressing this button will open up a separate menu on the right hand side of the screen showing the specific settings for any given question. The number and type of settings that are shown will be depending on the education level that has been selected by the user. This is intended to make the application more 'user-friendly' and prevent overwhelming students with information that is not applicable to their education level.

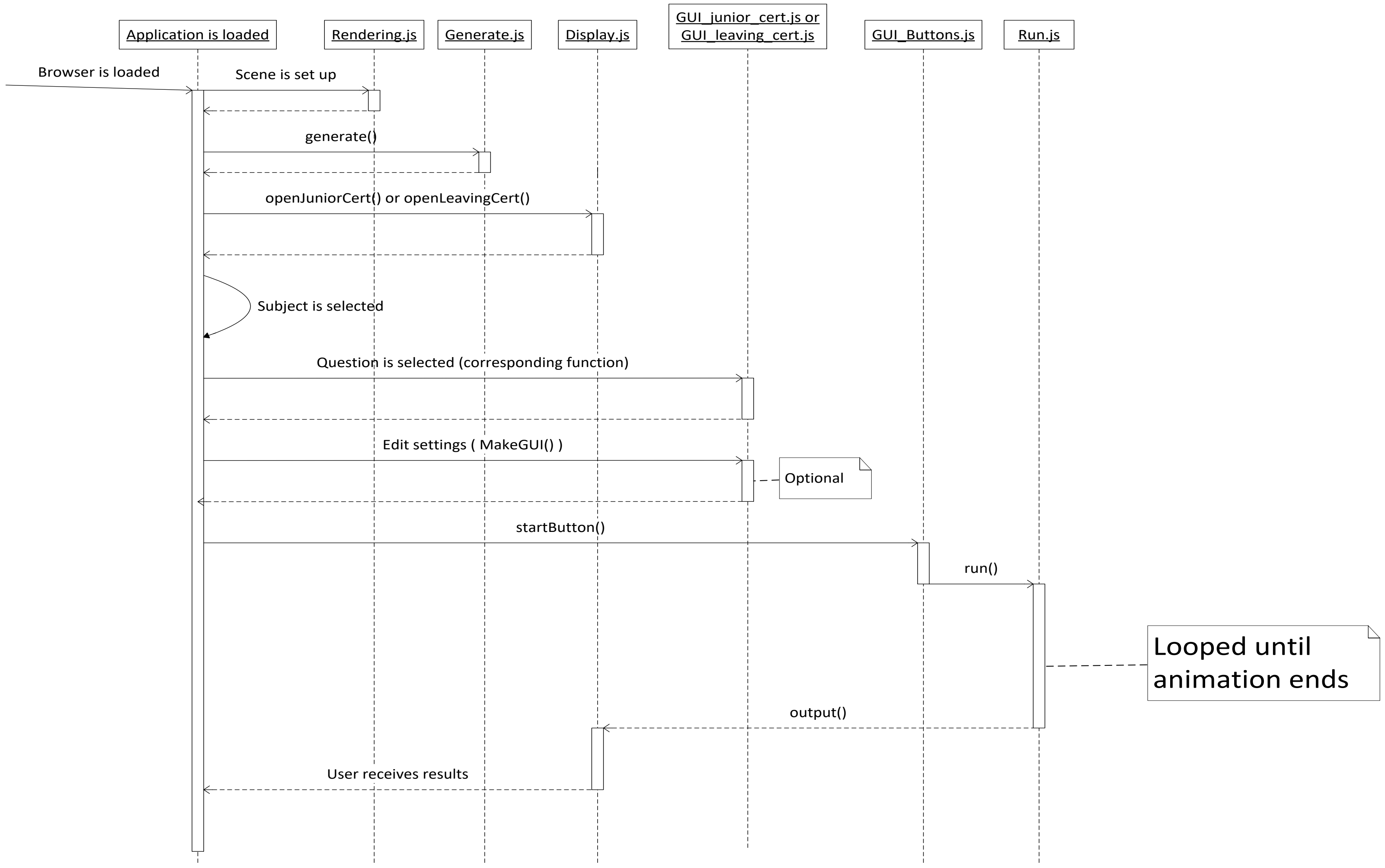
Lighting effects have been incorporated to enhance the aesthetic appeal of the application and provide a better simulation of the motion of a body in 3D space. These effects cause a realistic reflection of light from the surface of the sphere. The resulting shadow also provides useful visual cues to the user about the current location of the sphere within the environment. At any stage the user can click the ‘Light switch’ to toggle these lighting effects on and off. The user may decide to do this if, for example, their internet connection was poor, causing the application to run particularly slowly or if the user’s PC does not have the required processing power to run the application smoothly.

The application also makes use of simple sound effects. It does this by playing a uniform ‘bang’ sound whenever the sphere hits either the ground or a wall. This is to provide audible cues to the user about what is happening within the simulation.

## Use Case diagram



## Sequence Diagram



## Implementation

Constructing all areas of the code involved development within multiple different JavaScript files. It was ensured throughout the course of the project that files which were entirely my own work were kept separate from others that had been adapted or borrowed. The physics used in this application were not adapted or borrowed but developed entirely from scratch applying the basic, first principles that a Leaving Cert physics student could implement.

The html file “PhysicsRoom.html” is the webpage through which the application is accessed. This loads all of the JavaScript files which are used. The html is broken into multiple sections (divs) which display different key pieces of information:

- The “container” div holds the animation,
- The “home” div holds the home, light switch, texture and hide buttons,
- The “menu” div contains the buttons that the user uses to specify the education level,
- The “question” div contains an image of a question from an exam paper,
- The “results” div holds the output of the application.

Using a file called “layout.css” it was possible to alter the display so that every div except for “container” and “home” are hidden and shown on a button click.

In order to run the simulator it was necessary to make use of some third party JavaScript files. The files used were:

- “dat.gui.min.js” which provides the tools for making a user interface,
- “mouse.camera.js” which provides the ability to rotate the camera around the scene,
- “MouseOperation.js” which restricts the mouse effect to the div which it is hovering over,
- “TrailRenderer.js” which provides the tools to simulate a trail behind moving objects
- “three.js” which is the main file and is an API for JavaScript to run WebGL.

The JavaScript files that were entirely original pieces of code were separated along the functional lines of each component. These JavaScript files of original creation were; “GlobalVariables.js”, “Rendering.js”, “Display.js”, “GUI_Buttons.js”, “GUI_junior_cert.js”, “GUI_leaving_cert.js”, “GUI_third_level.js”, “Lighting.js”, “Generate.js”, “Run.js”, the html file which hosts the application “PhysicsRoom.html” and the CSS file for the design “layout.css”.

These html and CSS files both work together to create the general layout of the application. “PhysicsRoom.html” is where the user interface is generated (on the left hand side of the screen) and the space for the simulator behind it is allocated. “layout.css” is used to create the menu over the simulator and center the content of the tables.

Once the webpage is set up, the JavaScript files are then loaded. The first to be loaded are the third party files - original JavaScript files are then called. The first two files called are “GlobalVariables.js” (which contains all the variables used for running the application) and “Rendering.js” (which is used to initialize the page space and set up the simulator). It is important that these two files are loaded before any others in order to prevent errors from occurring. Once these two files are loaded, the remaining files are then loaded which contain all functions used in the application.

## Generate.js

The first file used is the “Generate.js” file which creates all of the objects in the world. This file creates the ball, the walls, the lines (used for illustration), and the spot which are all displayed within the environment. The “Generate.js” also carries out the refreshing of the simulator if the user changes any of the settings.

In order to ensure that the user cannot break the environment, limiters were put in place to keep the user from making the room or ball too big or small.

Once the world is generated, all of the variables are multiplied by a scale variable which increases the physical size of all objects uniformly. The reason for this is related to the light and shading effects which require the objects to be of a large scale for realistic effects. This scaling will have no effect on any of the calculations or physical properties of the environment and are purely a visual concern.

## Lighting.js

Another file used (when the world is initially loaded) is the “Lighting.js” file. This is used to provide lighting to the 3D environment. This illuminates the surface of objects and causes the sphere to cast a shadow.

This was surprisingly difficult to implement. The reason being, that casting shadows creates a large demand on the processing power of the computer. When it was initially implemented into the application, the lighting effect was made to be proportional to the size of the room in the “Generate.js” file. After testing on different machines, the required processing power was found to be too burdensome for some computers to handle. In some cases it even resulted in the computer crashing. In the end, a separate file was created with a hard set value for the lighting. This enabled weaker machines to run the application effectively. As a result, the lighting in the world is no longer adjustable. This is an unfortunate flaw in the design but a necessary one.

As mentioned above a “Light switch” function was also included. This feature was included to enable the application to run on weaker machines. Removing the shadows causes the application to require significantly less processing power.

## Display.js

Navigation of the user interface is handled by “PhysicsRoom.html”, “GUI_buttons.js” and “Display.js” together. When the user selects an education level it calls one of three functions in “Display.js” each of which opens a different menu. The “Display.js” file also handles the output of the final answer to the question. This is an aspect of the code which could be greatly improved from further development as currently the style of the output is hardcoded instead of generated (although the numbers themselves are generated).

## GUI_Buttons.js

The "GUI_Buttons.js" file handles the functionality for all buttons that are not education level-specific. The file provides functionality for the hide and show buttons as well as the start button which runs the application.

The start button first terminates any simulation that may be currently running. The start button then copies all of the settings which the user has specified. It does this to prevent the original values from being altered, allowing the user to quickly restart the simulation without needing to manually re-enter any values. The start button then resets the world back to its initial position and starts the application once again.

## "GUI_junior_cert.js", "GUI_leaving_cert.js" & "GUI_third_level.js"

The "GUI_junior_cert.js", "GUI_leaving_cert.js" and "GUI_third_level.js" files are used to handle the user interfaces for the different education levels.

Once the user selects an education level and subject, they are presented with a series of questions from that subject as well as a "Build your own" button which will open an interface on the right hand side of the screen allowing the user can change the variables manually. The junior cert, leaving cert and third level questions and variable interfaces are tied to the "GUI_junior_cert.js", "GUI_leaving_cert.js" and "GUI_third_level.js" files respectively.

When a question in the interface is clicked, the corresponding function for that question in the corresponding file is run. This is done in order to set up the required settings of the question in order to display and answer it.

## Run.js

When the start button is clicked the "run" function in Run.js is called which begins the animation. This is carried out using the settings that either the question or user have provided. This is the main JavaScript file which demonstrates the laws of physics in action to the user.

The "run" function requests an animation frame on itself in order to put itself in a loop. With each call of the run function, it checks whether the ball in the simulator has stopped moving (at which point the animation is terminated). If the user has specified a stopping time it then checks to see if the stopping time has been reached (at which point the animation is terminated). If neither of these checks are passed then the ball continues to move for the next iteration.

As the application is run the "Run.js" file records the highest point it reached as well as whether or not the ball has touched the ground. If the ball does touch the ground then the "markTheSpot" function in "Generate.js" is called in order to mark where the ball hit the ground.

“Run.js” uses a function called “collisionCheck” to determine whether the ball has made contact with any surface. If the ball does make contact with any surface then the velocity of the ball is reversed and recalculated.

As the ball moves, the “run” function also updates the trail which follows the ball (it in order to show the path which has been taken). The “run” function ends by recording the amount of time that has passed and then begins the next iteration.

## Additional files

In order to provide additional aesthetic appeal and immersion into the application, images and a sound file were also incorporated. The sound file is played whenever the ball makes contact with a surface. This is done in order to provide an audio cue to the user that the ball has bounced and provide additional realism.

The images used have a cosmetic purpose. These were used in order to texture the ground and walls. Images were chosen so as to be subtle (i.e. not overwhelming), neutral, and to give to application visual appeal. Images of the exam paper questions were also included to allow the user to easily see the question that the application was simulating.

## Sample code

The two pieces of code that I feel the most proud of are:

The “run()” function in “Run.js” which runs the animation

```
function stopRunCheck()
{
        var speedBool = Math.round(yVelocityC * 10) / 10 >= -0.2 * scale &&
Math.round(yVelocityC * 10) / 10 <= 0.2 * scale;
        var posBool = theSphere.position.y <= scale + floor.position.y +
(((theSphere.position.z * -Math.sin(radAngle)) + speeds.radius * scale)/
Math.cos(radAngle))

        // Check if ball has stopped bouncing
        if(speedBool && posBool)
        {
                stopCounter += 1;
                if(stopCounter >= 30)
                {
                        endRun();
                }
        }
        else
                stopCounter = 0;
}

function stopTimeReachedCheck()
{
        if(speeds.StopTime != 0 && (t - startTime) >= speeds.StopTime)
                endRun();
}

function run()
{
        animate = requestAnimationFrame( run );

        stopRunCheck();
        t = clock.getElapsedTime();
        stopTimeReachedCheck();

        // Highest point reached
        if(highestPoint < theSphere.position.y / scale)
                highestPoint = theSphere.position.y / scale;

        collisionCheck();

        moveBall();

        // Updates trail to balls new position
        trail.advance();

        // Remembers time elapsed for next run
        t2 = t;

        // Recreates the scene
        render();
}
```

- The “generate()” function in “Generate.js” creates and refreshes the world

```
function generate()
{
        scene.remove(theSphere);
        scene.remove(theSphere2);
        scene.remove(floor);
        scene.remove(wall1);
        scene.remove(wall2);
        scene.remove(wall3);
        scene.remove(wall4);
        scene.remove(net);
        scene.remove(spot);

        for(x = 0; x < 5; x++)
        {
                scene.remove(disLinesY.pop());
                scene.remove(disLinesX.pop());
                scene.remove(highLines.pop());
        }
        linesExist = false;

        // Limiters for the room and ball size
        if(speeds.radius < 0)
                speeds.radius = 0;
        else if(speeds.radius > 30)
                speeds.radius = 30;

        if(speeds.roomSize < 0)
                speeds.roomSize = 0;
        else if(speeds.roomSize > 200)
                speeds.roomSize = 200;

        // Ground and wall positions
        var pos = -(speeds.roomSize * scale)/2;

        // Ground angle
        radAngle = speeds.angle * (Math.PI/180);

        makeBall(pos);
        makeGround(pos, radAngle);
        makeWalls(pos, radAngle);
        makeProps(pos, radAngle);

        updateCameraXYZ();
        camera.updateMatrix();
        render();
}
```

## Problems solved

One of the largest problems encountered during the course of this project was the lighting. Lighting in WebGL requires significant processing power. It was found that some computers would experience issues while attempting to run it. The application was tested on several machines with the lighting effects both ‘activated’ and ‘deactivated’. It was found to be the case that on all computers, the simulation would run much smoother with the lighting effects deactivated.

When lighting was initially incorporated into the application, most machines found it difficult to run and in some instances caused the entire to computer to crash. The application was initially developed in such a way that the lighting effects and the environment were regenerated together. On powerful machines, this was found to be a manageable process without much slowdown. On some standard computers there was a degree of slowdown, but not so much as to be unmanageable. However, for below average strength machines (maybe 5-6 years old) running the application would occasionally cause the computer to crash. As the intention was for this application to be as accessible as possible to all students, this was obviously a serious issue.

The eventual solution to this problem was to create a separate file for generating the lighting effects. This would be separate from that which creates the rest of the environment and would run only once. The file, named “Lighting.js”, contained hard coded settings for the lighting effects. This was found to be a much more manageable solution for all potential types, age and power of hardware which could be used to run the application. The downside to this solution was the facts that this meant that the lighting would not automatically adjust to the dimensions of the environment.

In order to validate the decision to retain the lighting effects, individual testers were asked to observe the application running both with and without the lighting effects. The feedback was unanimous in agreement that the 3D environment could appear ‘strange’ and ‘disorientating’ to the user without shadows as a point of reference as to the sphere’s location within the room. Upon receiving this feedback, it was decided that, although requiring significant processing power, the lighting would be a necessary addition. A decision was made that visual appeal would be important in delivering an application which would engage with users. On this issue, a compromise was struck between efficiency and aesthetics.

As previously mentioned, a light switch button was included in the interface which allows the user to disable shadows.

The user interface was another problem that required a significant amount of attention. During the early phases of the project’s development, the user interface was unclear and incomprehensible to some users. Overcoming this issue required several meetings with Dr. Humphrys which resulted in the ongoing evolution of the user interface. In order to prevent the user from being overwhelmed, it was important to ensure that they would only be provided with information that was considered necessary at that point in time. This lead to the ‘hide’ feature and limited the scope of menu options and information displayed to the user.

Another area which caused some issues during development was the attempt at separating the mouse functionality between the simulator and the user interface. The issue in question was that when the user would ‘click and drag’ the mouse it would affect both the menu and the simulator at the same time. Using the mouse wheel would also scroll the menu on the interface and zoom in/out of the simulator at the same time.

In order to overcome this problem, a solution was required which would recognize when the mouse was over the menu or over the simulator and change the mouse’s functionality accordingly. Upon searching online, a file called “MouseOperation.js” was found which enabled this to be carried out. It achieved this by creating ‘event listeners’ which observe the movement of the mouse and recognize the div over which it was currently located.

An issue which remains part of the application is the structure of the solutions provided to the user for each of the preset questions. As previously mentioned, the layout and structure of the output is currently hardcoded instead of generated (although the numbers themselves are generated). This aspect of the code is the area which would have benefited most from additional time and development, and where future improvement could be most easily made. However, a decision was made in the final weeks of development this it was critical for the application to generate clear answers for each of the preset questions. Had a further work been undertaken at this time, the task may have overrun the deadline for this iteration of the project.

The simulator, in its current form, can also experience issues while handling extremely large values for parameters (velocity of the ball, etc.). Extreme velocities and accelerations can cause the ball to travel beyond the defined borders of the simulation and result in a bug.

On reflection, if this project were to be undertaken again, the variables would be managed differently. As the program developed, the functions and variables grew and become interconnected to the point that majority were made global. This is a very inefficient way of handling variables. The code should have been laid out in such a way that the variables were initialized within the functions before being passed between them. Some variables were made global by necessity; however, had the code been designed differently it is likely that this number could have been reduced.

## Results

### Implementation

The final product which has been developed is an easy-to-access application that is intuitive for the user. This project achieves the target goals that had been identified at its outset. The resulting piece of work both aligns with what had been originally intended and incorporates additional ideas that were conceived throughout development (visual aids, third level parameters, etc.).

This project commenced with a vision of a program that would allow the user to control the laws of physics, and thus, provide them with some insight for studying the subject at a basic level. The result of this work is a web based application that is:

- easy to access for most users,
- provides clear instructions,
- provides insight to exam questions,
- allows the user to alter the physics and environment observe the resulting effects,
- encourages experimentation by the user
- is fun to use

For these reasons, I am proud of the many months' work that have been placed into this project and very pleased with the final product that has been developed.

### Testing

Testing was carried out at several points throughout the development of this project. These resulted in several changes being incorporated into the final version. These tests were conducted with the assistance of third and secondary level students as well and secondary level teachers.

Initial responses from third level students suggested that that the interface was 'difficult to follow' and that some of the information to which the users were presented was difficult to understand. Generally, users were impressed by the realistic behavior of the ball as it moved about the environment. Most enjoyed adapting the variables and observing the resultant behavior. The majority communicated that the application was simple and fun to use. Most users were of the opinion that the application worked well, aside from the interface which 'could have been clearer'.

Many secondary level teachers stated that the application showed promise. They communicated minor issues with loading the application on their machines but they said believed that it could prove to be a very useful tool. A number of those tested suggested that it would likely be more useful for Leaving Cert students, given the larger variety and increased complexity of questions that it could cover.

By and large, the secondary level students offered similar responses to those of the third level students, particularly regarding the interface. As a result several adjustments were made to the final version. Once a brief description of the application was provided to the students, it was found to be universally enjoyable. The testers were observed as they tinkered and experimented with the settings (to make the ball behave in funny ways). It was satisfying to watch the testers enjoy the application as intended while at the same time become familiar with the concepts of velocity, acceleration, displacement, etc.

## Future work

Given the nature of the application, the possibilities for additional work should be limitless. Most obvious would be the inclusion of additional Physics and Applied Maths questions covering projectiles or bodies in motion. A simple evolution could be the ability to reposition objects within the world, thereby allowing the user to witness the behavior of the object in several scenarios.

Outside the scope of the Junior and Leaving Cert, the 'Third level' environment could be further developed. This is currently a 'sandbox' environment, which gives the user free reign to adapt velocities and accelerations in all directions. This could be adapted to account for acceleration in the x-z plane due to the force of wind, etc.

The physical dimensions and shape of the environment can also be adapted to include such changes as inclined floors. Currently, this allows the user to incline the floor along one axis. This could be further developed to allow an incline along two axes. Furthermore, localized distortions or changes in the shape of the room could be included. These distortions could be non-linear and curved (e.g. a curved mound in the center of the floor, surrounded by level ground).

Other objects could be included within the environment, allowing interaction between them. Currently, some questions will include additional objects within the environment (e.g. net in tennis court question, or second ball). However, there is currently no functionality that with allow interaction between these objects. The laws of physics currently written into the application (which cause the ball to be reflected from the walls and floor) could be adapted to account for collisions between multiple bodies within the environment.

The application is currently focused on the motion of bodies in simple systems under Newtonian Physics. These systems could be made more complex - for example, and environment could be created to simulate the motion of astronomical bodies in a 'Solar system'. The Laws of physics currently within the application could be adapted to account for the attraction due to gravity between bodies (related to their mass and inversely proportional to the square of their distance). Laws of circular motion could also be included and the user to adjust the variable concerned to gain a better understanding of these laws in an interesting and entertaining way.

Improvements could also be made to the solutions currently provided by the application. Although the output is dynamic in its response to most inputs which can be overwritten by the user, there are some that are currently fixed (e.g. angle of trajectory to provide maximum displacement up a plane). Further work could be carried out to provide solutions which would be dynamic to all user inputs.