Physics Lab - Teaching Package

Interim Report

Adam Beecham
BSc. Computer Science

School of Computer Science and Informatics,
Cardiff University

Supervisor: Helen R. Phillips

Moderator: Alun D. Preece

Abstract

The physics lab project involves the design and implementation of a visual simulation of a number of scenarios that would typically be encountered in an A-level physics course. The emphasis of this project is on providing a way of visualising the actions of forces on objects and how these actions are reflected by the underlying equations. The package would be used as a teaching aid by an instructor to demonstrate the application of these equations, which would otherwise be difficult or dangerous to perform. This package should also provide a more engaging method of teaching students the mathematical concepts behind various applications of force and help students to better understand these concepts. The aim of this project is to address the issues that students encounter when solving physics problems that require understanding of fundamental physics concepts and mathematical equations.

Acknowledgements

- Dr Helen R. Phillips, who originally proposed the idea of implementing a physics simulation system, and has supervised the development of the project.

- Mr John Ivins of croesyceiliog School, for providing feedback on the selection of appropriate physics concepts to use in simulations.

Table of Contents

Conclusion

Appendices

References

Introduction

Problem Overview

For many students today, physics is considered to be one of the most challenging subjects that can be studied. Studies commonly find that students feel that the mathematics involved is difficult, concepts are too abstract or that the subject is simply not interesting. An investigation into why students perceive physics to be difficult revealed that from a survey of 293 students studying introductory physics, 57% felt physics was difficult, while 41% felt it was too abstract and 33% felt it required a good understanding of mathematics[4](Ornek, F. et al. 2008). It is possible that such perceptions of physics are a result of students having difficulty grasping the fundamental concepts of physics. As argued by Freedman (1996) students often misinterpret the meaning of mathematical equations; although they are able to correctly solve problems, they have difficulty in interpreting their results[3].

Project Aims

The physics lab project aims to address the common difficulties that physics students encounter at an early stage, by producing a software system for use in GCSE or A-level course. This system would provide a number of simulations, covering momentum, velocity and forces, using 3D graphics in order to convey these topics in an interesting manner. These topics require understanding of fundamental concepts of physics, mathematical equations and are often perceived to be abstract.

As it is not assumed that users will be expert computer users, the system produced by this project should be suitable for use by users who are not computer literate by providing a simplistic user interface for interaction with objects in a simulation. The system should also clearly show how the underlying equations change in step with their respective equations, as this is intended to make given concepts feel less abstract. In turn, this should help improve a student's ability to interpret how these equations generate results. The system also aims to use visually stimulating graphics and to provide problems for students to work through. This active learning approach will be more engaging for students, meaning that they are likely to learn more from using the system.

Beneficiaries

This project is primarily aimed at A-level and GCSE students. It aims to benefit these students by providing an effective means of improving their understanding of various equations and concepts required when solving physics problems regarding forces, momentum and velocity. The project aims to achieve this by exploiting the graphics capabilities of modern computer systems in order to produce visually attractive 3D simulations.

Another potential beneficiary would be a GCSE or A-level physics instructor. By incorporating the use of these simulations into a course, the instructor would be more easily able to demonstrate concepts that students might consider to be abstract and difficult to visualise. This would potentially allow the instructor to focus on other aspects of the course that students may have difficulty with.

Scope

In order to achieve the stated aims, the outcome of this project will be a software system compromising a range of interactive 3D physics simulations on forces, momentum and velocity. Simulations will be selectable from a menu that clearly summarises the areas that a particular simulation addresses. Each simulation will have a simple interface, suitable for a novice user that will allow manipulation of settings such as initial velocities, launch angles and mass. Simulations will provide feedback to the user, showing the equations used and values calculated (such as velocity and distance) for a given time in the simulation.

The system will be able to open simulations in demonstration mode, which allows the user to run the simulation as normal, or question mode, which will allow an instructor to set problems based on the given equation, for students to attempt to solve. The system will check answers given, and inform the user whether or not they have correctly answered the question.

Approach

The system will be developed following the incremental development approach. The core components will be designed and implemented first. These core components consist of a basic simulation framework, the menu, and the essential components of the user interface, such as play, pause, reset and exit buttons. Each component will then be tested and evaluated, prior to design and implementation of the next increment.

Each successive iteration will then consist of a new simulation, including any necessary amendments to the interface. The final iteration will add the question mode to the system, once all other components have been tested and are working correctly. Once all iterations have been developed, the system will then undergo acceptance testing by a physics instructor before being considered ready for distribution.

Before development of the main system begins, a prototype system will be developed as a proof of concept. This system will be composed of a projectile motion simulation with a user interface, allowing basic interaction with the system, and options for pausing, playing and resetting the system.

Background

Active Learning

Active learning is the theory that 'we learn more effectively by doing'[5]. There are a number of activities that can be used to encourage active learning, categorised into high and low risk activities. As high risk activities are more likely to fail and require more intricate and longer planning, the physics lab project aims to encourage active learning by providing a question mode, so that students actively engage with the simulations. This was decided to be an effective solutions as quizzing students is considered a low risk activity and the amount of time required to produce a questionnaire is reasonably short. This preparation time can be significantly reduced by recycling questions through the physics lab systems ability to save setups.

As stated by Bonwell, one particular disadvantage brought by Active learning is that 'active learning strategies reduces the amount of available lecture time that can be devoted to content coverage'[1]. While this problem cannot be completely eliminated when using the physics lab system, it could be minimised by using the system to compliment course coverage, as a tool to aid student understanding of concepts of force, velocity and momentum.

JMonkeyEngine

In order to speed up the development process of the physics lab project, the system will be built on using the JMonkeyEngine 3 framework. JMonkeyEngine is an open source game engine built with java that allows rapid development of 3D applications. This framework is in turn based on the LWJGL (Light weight java games library) which provides access to OpenGL's graphics capabilities. JMonkey applications typically extend from the SimpleApplication class, which is responsible for setting up the basic components of a 3D scene. The JMonkey documentation can be found at: http://jmonkeyengine.org/wiki/doku.php . A comparison of JMonkey against other similar systems can be found in Appendix A.

Existing Solutions

At present, few software systems currently exist that attempt to address the issues tackled by this project. Many existing educational physics programs place emphasis on animations displaying a particular concept to the user. While this addresses the issue of physics being abstract, it does not help students to understand the underlying equations. Some systems use complex user interfaces that are simply inappropriate for use by a novice user. These often contain many elements, causing them to be particularly difficult to navigate and unclear use of icons, making it challenging for the user to deduce the functions of each element. In addition to these problems, existing systems typically cannot be purchased as individual copies, but only accessed through monthly or yearly payments. A number of subscription types are normally offered, giving different levels of access, but it is often difficult to know which service is appropriate, as trials are very limited or often not provided.

The main reason that these existing systems are inadequate for addressing the issues tackled by this project is that none make use of the concept of active learning. While all involve displaying animations to the user, none attempt to engage the user further than allowing setup of simulations. Users are then simply expected to observe said simulations, which does not utilize active learning in any form. An analysis of two existing systems can be found in Appendix B

Requirements Specification

The final version of the system should satisfy the following functional and non-functional requirements:

Functional Requirements

- The system shall have simulations that demonstrate concepts covered in GCSE and A-level physics such as:
    - GCSE level
        - Distance, speed and acceleration
        - Velocity
        - Momentum and Force
    - AS/A level
        - Equations of motion, including Projectile motion
        - Components of force and equilibrium
        - Pulley systems
        - Collisions and momentum

  (A specific list of concepts with worked examples can be found in Appendix C)

- Simulations will have 3D graphics.
- Simulation particles will conform to the appropriate physics for the given simulation (e.g. projectiles will move as defined by the corresponding projectile motion equations).
- Simulations will allow interaction through the manipulation of an object's properties via the user interface. Properties that will be changeable will include initial velocity, mass and launch angles.
- Simulations will have a camera that can be moved around the scene by the user so that the scene can be viewed from various angles.
- Simulations will display data related to the objects mass, acceleration and velocity in a graphical manner so that it is easier for a student to understand the underlying mathematical concepts.
- Simulations should have two modes:
    - demo mode -simply runs the simulation with the standard interface for interacting and visualising a given concept
    - question mode - as demo mode, but will allow the user to create mathematical problems on the given simulation (automatically or manually) and provide a means to answer and check these questions.
    - The user should be able to save the setup of any simulation, as well as any questions produced by the user for that simulation.
- The system shall have a menu from which will display a list of available simulations, including an image to help identify the simulation and the level that the simulation is aimed at (GCSE or A-level). The menu will also provide a means to open previously saved simulation setups.

- If a simulation is selected, the system shall display the simulation, as well as an interface providing options to adjust the simulation as required. These options will always include (but will not be limited to):
    - play, pause and reset options for the simulation
    - return to menu and exit application options
    - a save option(to save the setup of a simulation)
- Where appropriate the interface will allow the user to filter the information displayed on a given object, such as its velocity, so that a user can focus on any given area they choose.
- The system will provide help for how to navigate and use components appropriately. As the system is designed for use by physics instructors, it is assumed that no help needs to be provided on the underlying physics of each simulation.
- The system will provide a calculator that can be used to assist in answering.

Non-Functional Requirements

- The system will be suitable for use by novice users. This will be achieved with the following:
    - Risk of user error will be minimised by restricting user input using GUI components such as sliders, combo boxes and spinners. Only values within a defined range will be allowed.
    - The system will be developed with consideration to the Principles of usability[2] (Dix et al.):
        - Learnability - the interface will be simplistic, components logically grouped and behave in a predictable manner. If the user alters values, then the system will display these changes to the user immediately wherever possible. Common GUI components will be used so that the system feels familiar to the user.
        - Flexibility - The user will be able to customize the system by being able to alter values in the simulation and to show/hide non-essential components as needed.
        - Robustness - The system will provide feedback to the user through progress bars when busy processing and will warn the user when irreversible changes are about to be made. The user will be able to reset simulations restore default settings, and also save setups as desired.
- Simulations will be sorted in the menu by topic and level (GCSE or A-level).
- The system will ideally be usable on linux, windows and mac environments.
- Simulations will be graphically detailed, using models, textures and lighting techniques in order to be as appealing to students as possible.
- The user should be able to easily exit the program at any point.
- Simulations should have a frame rate of at least 30fps, so that they appear to animate smoothly.

Algorithms

The majority of algorithms used by the project are simply the equations that are required to calculate an objects velocity and distance travelled. These algorithms will be responsible for animating the simulations, positioning particles correctly in each frame of the animation. The pseudocode for the animation of the projectile motion simulation can be found in the prototype design section.

System Architecture

The system will have a modular design, where each element will be designed as a separate class, with similar elements (such as simulations) inheriting from one generic class containing the functionality required by all these elements. Due to the extensive use of mathematics, helper classes will be utilised, to provide shortcuts to commonly used values. A unit conversion class will also be required for converting between world units(the units used by jMonkey) to meters, as students will be require feedback from simulations as meters. One other essential component will be a screen manager, as the program will frequently need to switch between displaying the menu and selected simulations. The screen manager will be responsible for ensuring this transition occurs in a predictable way, and any transition screens, such as loading screens are displayed.

Prototype Design

Overview

The prototype system was produced for several reasons:

- As a proof of concept, to show that the final system can be realistically achieved.
- To allow potential development problems to be identified and resolved prior to development of the final system.
- To allow familiarisation with the JMonkeyEngine 3 framework and its capabilities.
- To develop the core simulation framework, ready for the implementation of the final system.

The prototype consists only of a projectile motion simulation and its interface, in order to demonstrate how simulations, particles and interfaces can inherit from their abstract parent classes. This allows for effective reuse of core components, so that specific simulations can be rapidly produced.

Prototype Sketch

The first step taken in designing the system was a sketch of the prototype system. This involved sketching the appearance of the projectile motion simulation, as well as its interface and components. The sketch can be seen below:

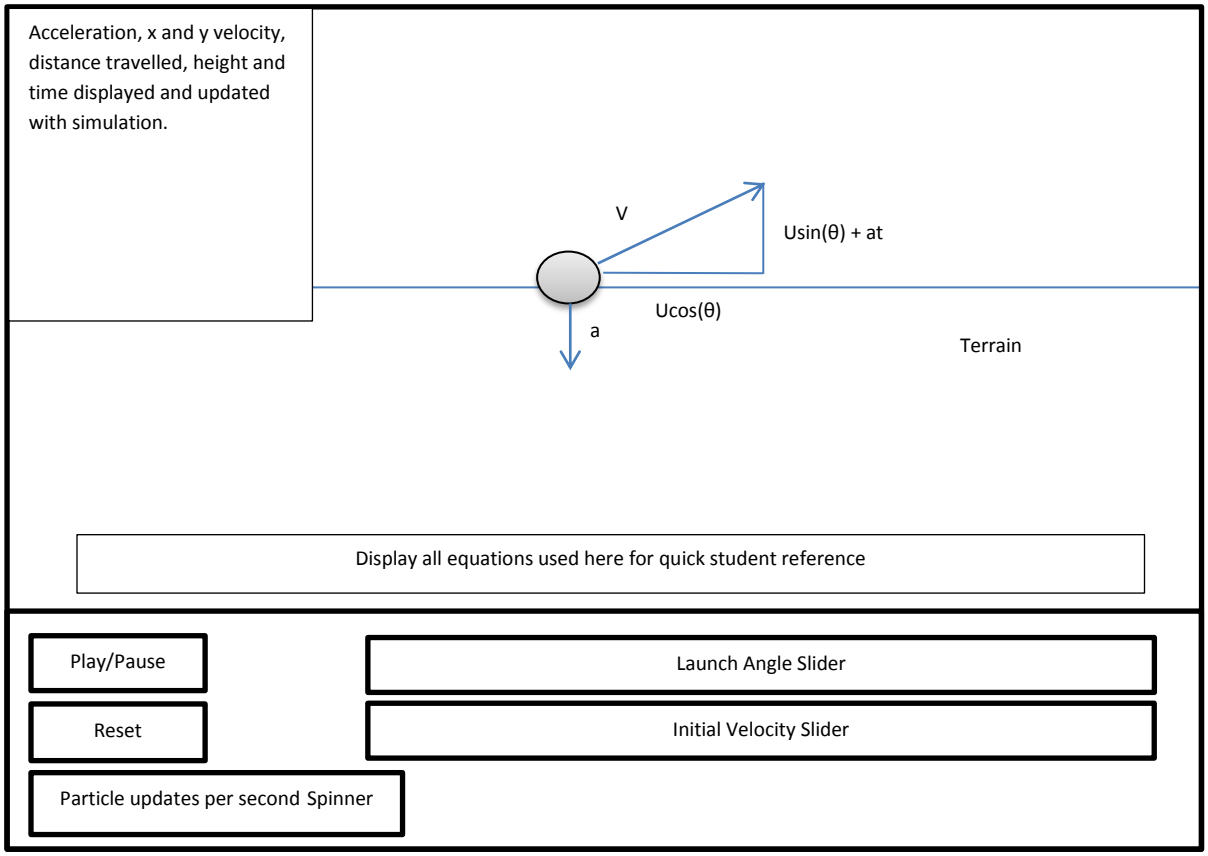


Figure 1 - Prototype Sketch

As can be seen, the sketch shows the simulation as it would appear at some arbitrary time after being run. The particle velocity and acceleration arrows displayed in the scene represent that at this point, the particle would move with a velocity specified by the x and y components, where:

- U = initial (launch) velocity
- a = acceleration (gravity, constantly pulling down at -9.81 m/s)
- t = time elapsed.
- θ = launch angle.

The interface provides all the basic functionality required to interact with the scene. From the interface, the user can start, pause and resume the simulation with a single button, reset the scene at any time, select a launch velocity and angle using sliders, and change how often the particle updates per second. The sliders and spinners will include labels that clearly displays the purpose of each component, and will provide immediate feedback to the user, informing them what value they have selected.

Prototype Class Diagram

The class diagram below shows the structure of the prototype system. The Simulation, and InterfacePanel classes are abstract classes that will be used in the final system to provide each simulation and its accompanying interface with its core features. As each simulation will have uniquely behaving particles, an abstract particle class will also be used, to prevent having to rewrite common variables and methods for each particle.

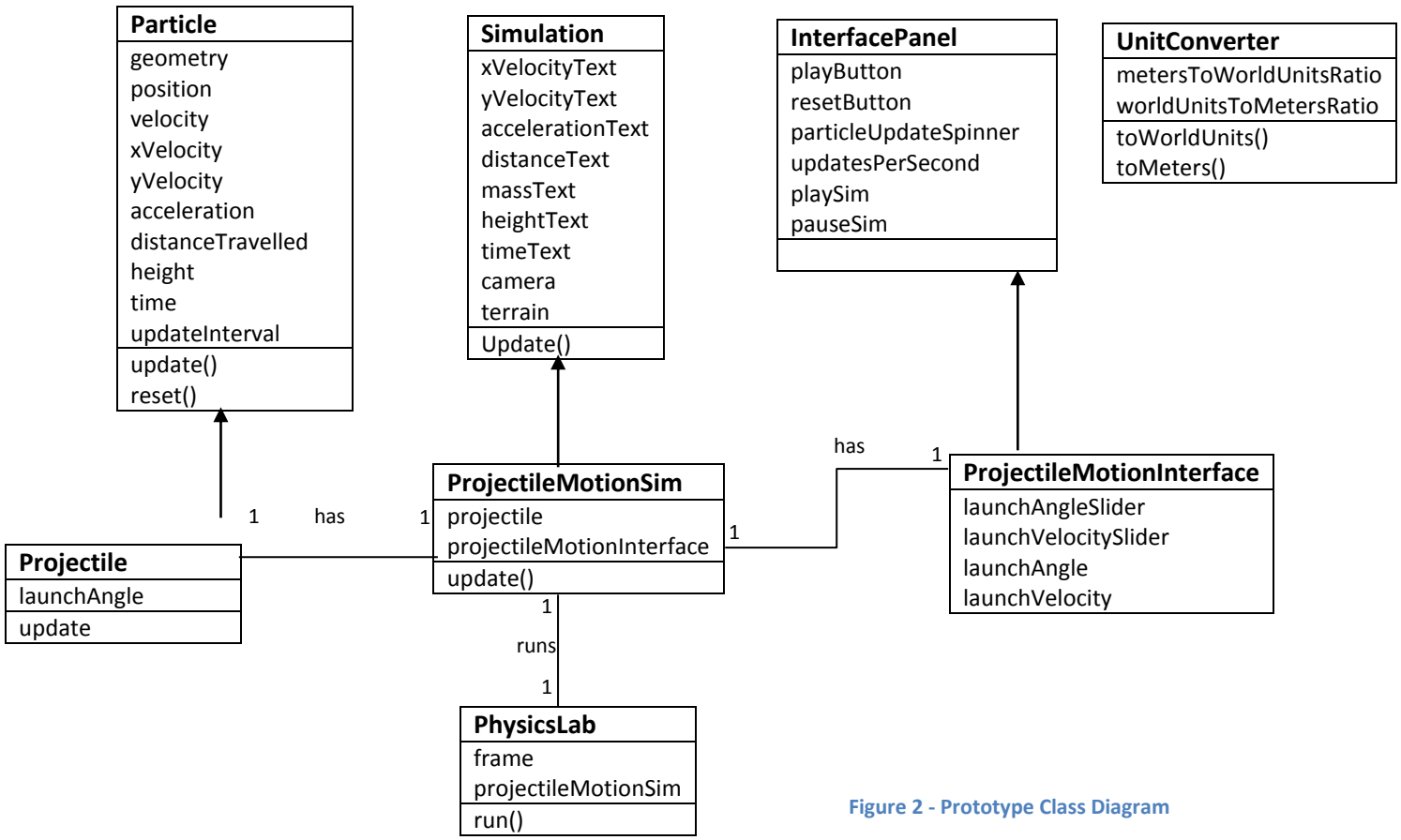


Figure 2 - Prototype Class Diagram

To summarise the above diagram, the Projectile, ProjectileMotionSim and ProjectileMotionInterface classes extend the functionality of their parent classes. The ProjectileMotionSim class has a projectile and a ProjectileMotionInterface, while the PhysicsLab class simply runs the program. The UnitConverter converts between meters and world units, as output will need to be displayed as meters, but any positions and movements in the simulation must be calculated in world units.

Projectile Update Algorithm

The main algorithm in the prototype system involves updating the simulation by calculating the properties of a projectile at a given time and then positioning the projectile accordingly. This should only happen if the user is currently playing the simulation, otherwise the projectile should do nothing. This functionality can be seen in the pseudocode below.

```
Algorithim SimulationUpdate(timePerFrame)
      Input: timePerFrame, the time taken to update from one frame
      to the next.
      Output: new position of projectile, its height, distance, time
      elapsed and x,y velocities.
      if !playSimulation then
            setProjectileLaunchVelocity()
            setProjectileLaunchAngle()
            setProjectileUpdateInterval()
      if playSimulation then
            updateProjectile(timePerFrame)
            updateInterface()
      if resetSimulation then
            resetParticle()
```

Figure 3 - ProjectileMotionSim Update Algorithm

The algorithm above simply allows the user to setup the particles update frequency, launch angle and velocity before starting the simulation. If the simulation is running then the projectile should update and the correct value for each of its properties be displayed. The particle should reset if the simulation has been reset. The pseudocode below shows how the particle will be updated.

```
Algorithm UpdateProjectile(timePerFrame)
      Input: timePerFrame, the time taken to update from one frame
      to the next.
      Output: new position of projectile, its height, distance, time
      elapsed and x,y velocities.
      yVelocity ← u * sin(theta) + a * t
      xVelocity ← u * cos(theta)
      distance  ← u * cos(theta) * t
      height    ← u * sin(theta) * t + 0.5 * a * t²
      if landed then
            stop()
      position = position + velocity * timePerFrame
      if timeToUpdate then
            setPosition()
            updateInterface()
            resetUpdateInterval()
```

Figure 4 - Projectile Update Algorithm

The projectile update algorithm simply computes new values for each property on each time the projectile is updated. The position is calculated by adding the velocity to the position. As the velocity is given in meters per second, it is multiplied by the timePerFrame variable to give the correct distance that the projectile has moved. If the updateInterval has elapsed, then the projectile is placed in its new position, and its values are displayed in the interface. The projectile stops if it hits the ground.

Prototype Implementation

User Interface

The user interface of the prototype system is composed of several swing components. The interface allows the user to setup a projectiles initial velocity and launch angle, and then play, pause or reset the simulation. As specified by the requirements, the interface is simplistic and restricts the users input to suitable ranges. Components are grouped into core components and projectile specific components by JPanels. TheFeedback is provided through JLabels which update immediately after a value is altered. These values are then passed to the ProjectileMotionSim class, which updates the particle. The code snippet below shows how the interface retrieves users initial particle velocity and angles from JSliders and immediately provides feedback to the user by displaying the value selected.

```
/*
 * override the stateChanged method in the simulation
 * class
 */
 @Override
 public void stateChanged(ChangeEvent e)
 {
     //needed to update the upsSpinner
     super.stateChanged(e);

     //get value from the launchAngle slider
     if(e.getSource() == launchAngleSlider)
     {
         //set launchAngleVariable to the slider value
         launchAngle = (int)launchAngleSlider.getValue();
         //show the user the selected value
         angleLabel.setText(launchAngle + " degrees");
     }
     //get value from the velocity slider
     if(e.getSource() == velocitySlider)
     {
        //set the velocity variable to the slider value
         velocity = (float)(velocitySlider.getValue())/10;
         //show the user the selected value
         velocityLabel.setText(velocity + " m/s");
     }
 }
```

Figure 5 - ProjectileMotionInterface stateChanged Method

The prototype interface is created by defining a ProjectileMotionInterface object in the ProjectileMotionSim class. The ProjectileMotionInterface class extends the InterfacePanel class, as specified in the design, which defines the core components of the simulation interface.

Projectile Motion Simulation

The simulation provided by the prototype is a projectile motion simulation, specified by the ProjectileMotionSim class. This class extends the Simulation class, which in turn extends the SimpleApplication class, provides access to the required JMonkeyEngine capabilities, such as the camera and 3D graphics features. The main feature of the ProjectileMotionSim class is its update method. This class is responsible for updating the interface that displays each of the particles properties and the particle itself, positioning it in its appropriate position as the simulation advances. The code snippet below shows the main sections of the ProjectileMotionSim class update method.

```
/*
 * Update the simulation
 */
@Override
public void simpleUpdate(float tpf) {

//if the simulation isn't playing, allow the user
//to setup the simulation.
   if(!panel.playSim())
   {
      particle.setLaunchAngle(panel.getLaunchAngle());
      particle.setLaunchVelocity(panel.getLaunchVelocity());
      particle.setUpdateInterval(1f /
               panel.getParticleFramesPerSecond());
   }

   //update the simulation if its playing
   if(panel.playSim())
   {
       //update the particle
       particle.update(tpf);
       //update the interface
       if(particle.updateInterface())
       {
           xVelocity.setText("x velocity: " +
                 particle.getXVelocity() + " m/s");
           yVelocity.setText("y velocity: " +
           particle.getYVelocity() + " m/s");
           ...
        }
    }
```

```
    //if the user resets the simulation, reset the particle and the
```

```
    interface
    if(panel.resetSim())
    {
       particle.reset();
       panel.setReset(false);
       xVelocity.setText("x velocity: ");
       yVelocity.setText("y velocity: ");
       ...
    }
}
```

Figure 6 - ProjectileMotionSim Update Method

Projectile

The projectile class is responsible for drawing and updating the projectile to the screen. It extends the Particle class, which provides the basic components of all particles. The projectile updates itself by calculating its velocity, then adding the velocity to the current position to compute its new location. The projectile then calculates its height and distance and provides these as feedback to the user. The code snippet below shows the projectiles update method:

```
/*
 * update the projectile
 */
 @Override
 public void update(float tpf)
 {
     //update while the projectile is still airborne
     if(!stop)
     {
         //particle class calculates the time variable
         super.update(tpf);

         //calculate the x and y components of the particle velocity
         yVelocity = (float)(initialVelocity *
                     Math.sin(launchAngle))
                       + (acceleration*time);
         xVelocity = -(float)(initialVelocity *
                     Math.cos(launchAngle));
         velocity = new Vector3f(0,yVelocity,xVelocity);

         //calculate the distance and height
         distanceTravelled = (float)((initialVelocity *
                             Math.cos(launchAngle)) * time);
         height = (float)((initialVelocity * Math.sin(launchAngle))
                  * time)
                 + (0.5f * (acceleration * (time * time)));

        //compute the new position by adding the velocity times the
        //time per frame to the current position
```

```
        position = position.add(UnitConverter.toWorldUnits(
                    velocity.mult(tpf)));

        //position the particle at every update interval. This
        //allows the particle to be updated at a speed specified by
        //the user
        if(interval >= updateInterval)
        {
             updateInterface = true;
             geometry.setLocalTranslation(position);
             interval = 0;
        }

        //stop the particle if it hits the floor
        if(position.y <= -1)
        {
             velocity.zero();
             position.y = -1;
             geometry.setLocalTranslation(position);
             stop = true;
        }
    }
}
```

Figure 7 - Projectile Update Method

Known Issues

While the use of the JMonkeyEngine has simplified the process of developing the systems graphics, it has introduced some complications in developing the interface. As JMonkey applications typically take up an entire frame, a work around had to be used to allow the use of a swing interface. This involved running the application in a Canvas, so that it could be resized and allow the interface to be displayed. This workaround is in the PhysicsLab class, which can be found with the full source code in the appendices.

Another issue is the feedback provided by the system. Although the calculations are accurate, the interface displays values with an accuracy of up to 7 decimal places. This level of accuracy is not required, and makes the interface look untidy. This also casues extremely small values to be displayed, such as the projectiles x velocity when it is launched at 90 degrees. The x component should be zero, however very small values are detected and displayed. The screenshot below displays this behaviour, with the affected properties highlighted.

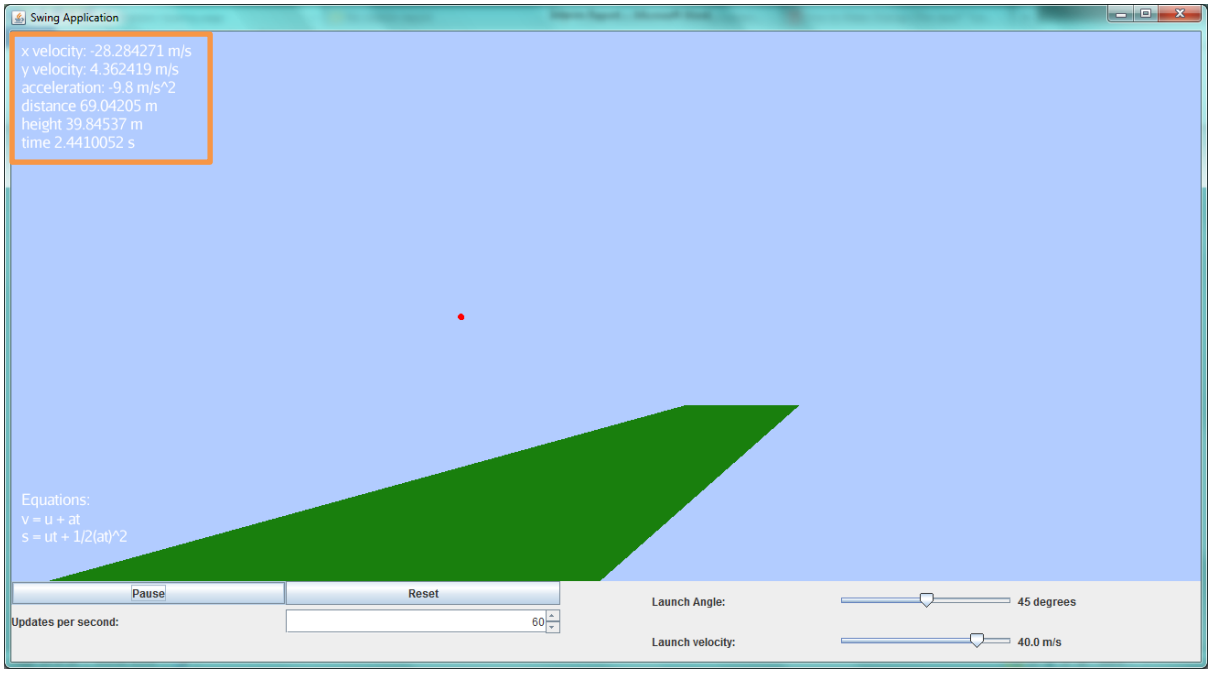


Figure 8 - Issues with the interface

Conclusions

Summary

The main aim of the interim report has been to highlight the development of the prototype version of the physics lab system and the core components of its implementation. The development of a framework for particles, simulations and interfaces, will allow for rapid development of the additional simulations to be included in the final version of the system. The report has also identified the requirements of the final system, justified the use of an iterative development approach and development decisions, such as the use of the JMonkeyEngine framework. The systems use of a question mode to encourage Active Learning has also been explained and finally issues with the prototype have been identified, including issues with number formatting and swing interfaces.

Future Work

Using the framework developed in the prototype system, the next stage of the project will be to develop the remaining simulations for the system. These will be built in the same manner as the projectile motion simulation, with particles, simulations and interfaces extending the abstract classes used in the prototype. The Particle, Simulation and InterfacePanel classes will likely require additional functionality that has not been identified by simply designing the projectile motion simulation alone.

The design and implementation of a menu and screen manager class will be necessary to enable users to select a desired simulation and for the system to be able to manage the transitions between the menu and simulations. Once all the system has been developed and each iteration has been tested, the system will then need to be tested by an A-level instructor, who will then provide feedback on the systems effectiveness.

Appendices

Appendix A - Comparison of 3D programming utilities

Due to the Physics labs systems need for 3D graphics, specialist programming tools for graphics programming will be required. A number of tools that could potentially be used in development of the system are listed below.

Java - JOGL

JOGL is a java wrapper library for the OpenGL graphics language. JOGL provides a wide range of tools for developing 3D applications due to OpenGL being a specialist graphics language. Most coding must be done at a low level, as OpenGL does not provide readymade utilities such as cameras or utilities for managing the scene.

Advantages:

- Low level code allows for a greater level of control over development of simulation graphics
- Open source library so free to use
- Can be cross platform, so long as the system has an openGL compatible graphics card (most modern computers do)

Disadvantages:

- Longer development time will be required (to develop camera classes, update loops and primitive shapes, such as a sphere).
- Can be tricky to setup for development (either place in java directory or in a desired folder and set a classpath to the library)

C# - XNA 4.0 Framework

The XNA framework is a framework developed by Microsoft for indie game developers. XNA provides both 3D and 2D graphics capabilities which can be accessed at a high and low level. XNA's 3D graphics system typically uses Models which can be quickly imported using the XNA content pipeline. Developers must use vectors and Matrices in order to position or transform objects.

Advantages:

- Easy to use library, which is well supported with a large community
- Works with visual studio IDE, which will enable quick development

Disadvantages:

- Not open source, so a risk of legal issues unless proper permission is sought
- Will only run on windows systems
- No tools for creating user interfaces
- Only a very basic set of 3D utilities provided

Java - JMonkeyEngine 3

The JMonkeyEngine 3 is a complete framework, including an IDE, for developing 3D applications in java. The JMonkeyEngine is based on the lightweight java games library, which allows access to openGL functionality at a high level. JMonkey provides a wide range of basic utilities for 3D applications, such as cameras, primitive shapes, materials and an integrated physics engine.

Advantages:

- Wide range of utilities available will allow development to focus on building simulations immediately, rather than developing core components such as cameras and update loops.
- Open source and freely available for use
- Cross platform capability, so long as a machine supports openGL 2.0 or higher

Disadvantages:

- No utilities for developing suitable interfaces, but JMonkey applications can be run in a canvas and allow for the use of java swing components instead.

Conclusion

It is clear that the JMonkeyEngine solution would be ideal for the project, as this provides tools to produce a 3D cross platform system and is open source, so can be used freely.

Appendix B - Analysis of Existing Systems

Physics-online.com by Fable Multimedia

Overview

Physics-online is an online service that provides a range of simulations, videos and articles on a range of GCSE and A-level physics topics. Content is arranged by topic, level and exam board. Each topic has a number of articles, simulations and videos that the user can view, explaining a particular aspect of that topic. Much of the content provided is from external sources, however content produced by Fable multimedia is also provided. The main user interface is simple and intuitive and consists of the navigator, where the user can select the topic they wish to learn more about, and the main screen, where all content is displayed. Help videos and articles are easily accessible and users can also choose to display the content in a separate popup window if desired.

In order to use Physics-online, users must register an account appropriate to their requirements. The types of account available are:

Lite - a free, single user account which provides limited access to available content.

Home - a single user full access account that can be used any day after 4pm and costs £17.50 per year

Single - a single user account that provides full 24 hour access costing £49 per year.

School - a multi user account providing full 24 hour access for teachers and students costing £99 per year.

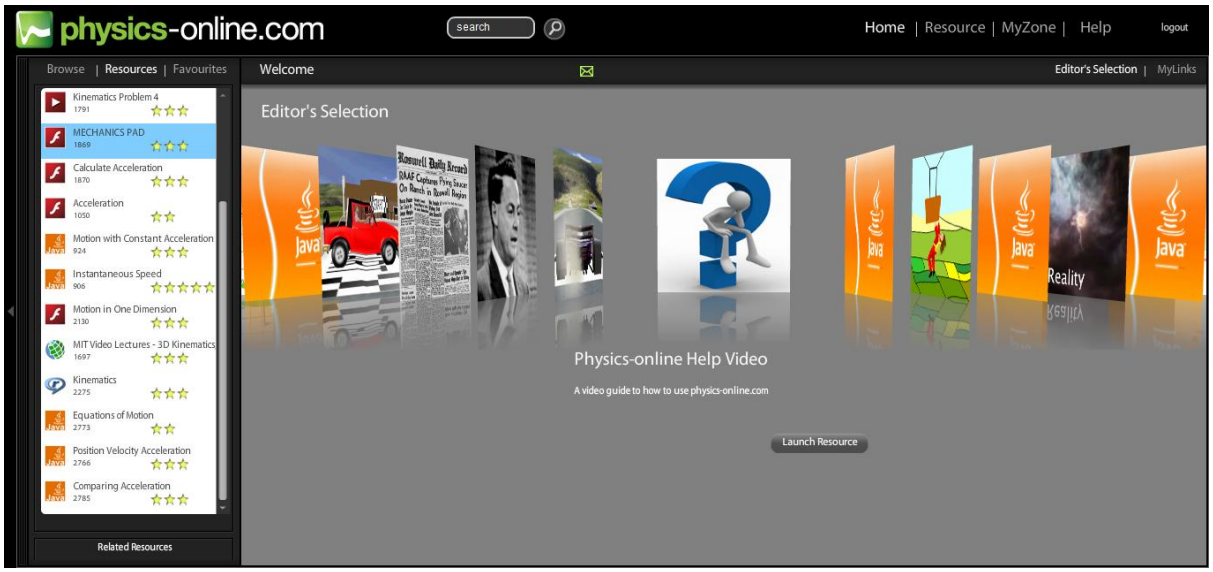


Figure 2 - User Interface

Despite being aimed at British GCSE and A-level physics students, much of the content is pulled from American university websites. While some of this content appears to be useful, a significant proportion seems irrelevant and sometimes too advanced for the target audience. In addition, this also raises questions over the possible legal issues involved with using these resources, as it is unclear if permission has been granted for the inclusion of this content in the system.

Simulations

A number of 3D and 2D simulations are available for users, some of which are linked from external websites and are typically java applets or flash animations. Most simulations produced by Physics-online are produced using mechanics pad, a custom 2D flash program built on top of a flash physics engine derived from the C++ Box2D physics engine. Mechanics pad also provides the ability for users to build their own simulations, with the ability to add bodies, change properties of a given body and also to display forces acting on them. In house 3D simulations are produced using adobe shockwave and are used mainly for illustrating velocity/time graphs.

Most simulations included are built by Fable multimedia and designed specifically for A-level and GCSE students. These simulations are well suited to the intended audience, as they explain the covered concepts well, and are able to hold the interest of the user through the use of interactivity. Simulations from external websites are often from university websites, often too simplistic to be of any use to students. Due to the inclusion of mechanics pad, linking to these resources also seems pointless, as mechanics pad is able to perform the same simulations, while also providing useful information, such as an objects speed.

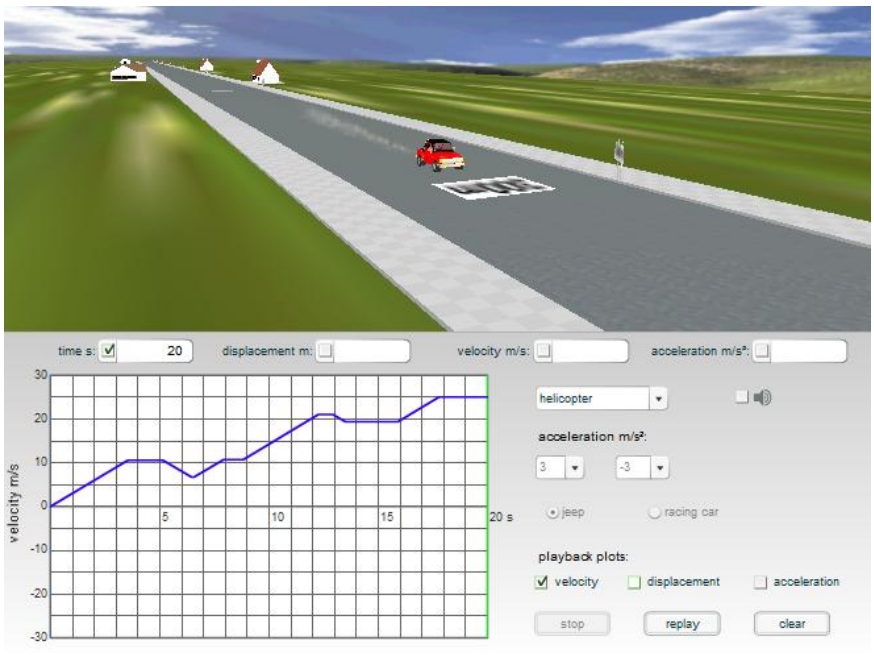


Figure 3 - 3D velocity/time graph demonstration

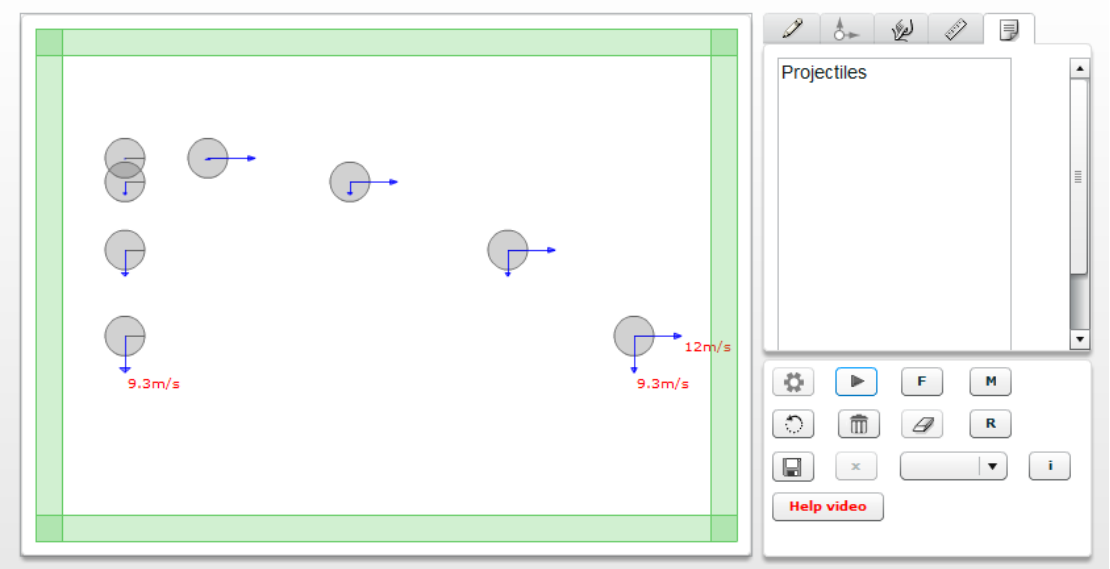


Figure 4 - Mechanics Pad

Videos

Videos are primarily flash based, and are either produced directly by Fable Multimedia or from external sources, normally university websites. The main purposes of videos are for explaining specific concepts or working through example questions. Videos produced by Fable are typically easy to follow and work through examples, clearly explaining how to answer a given question and would be useful for a student who may be struggling to understand the math behind a concept. One downside to this is that some videos appear to assume that the viewer has some prior knowledge and hence doesn't explain why certain values are required, though this knowledge could be acquired from other resources provided.

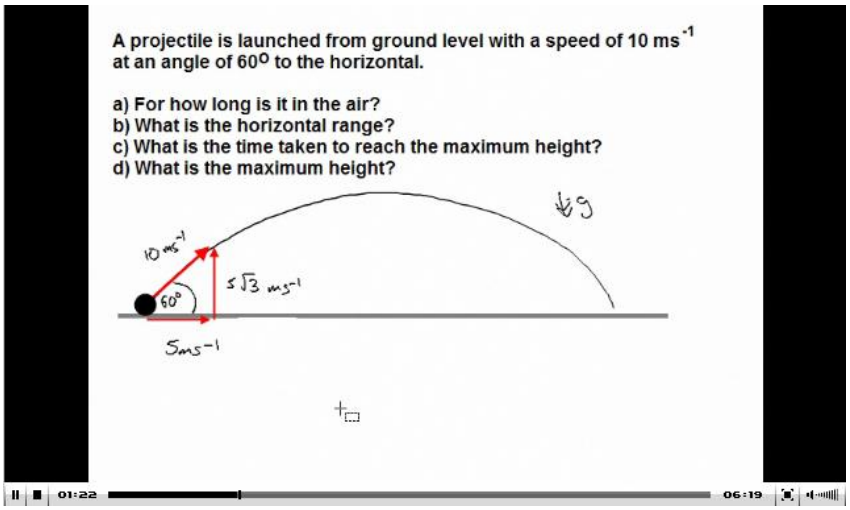


Figure 5 - Example question video

Articles

Some topics contain articles which are typically links to external webpages covering specific topics in great detail, however this is often in much more detail than would be required for a GCSE or A-level student, and can be difficult and dry to read. With the exception of help articles, the inclusion of

these articles seems unnecessary, as the topics covered are almost always covered in the required level of detail by videos or simulations.

Interactive Physics by Design Simulation Technologies

Interactive physics is a standalone program that can be run on Windows or Mac classic mode (available on version 10.4 or lower). Interactive physics claims to be widely used by schools worldwide and has won a number of awards. The most recent release of interactive physics is the 2005 version, meaning that this software is now out of date and would not cover all aspects of current physics syllabuses.

Unlike physics-online, interactive physics is based entirely on simulations, and allows the user to create experiments for themselves, or choose from a library of ready to run simulations, which can also be customised by the user. The library of animations includes (but is not limited to) simulations for momentum, kinematics, projectiles, collisions and pulley systems. All simulations are in 2D and make use of simplistic sprite graphics that are not visually stimulating, however they are very simple to set up and would be appropriate for even non-computer literate users. Many simulations also provide information on each simulation, such as an objects velocity or acceleration, and also provide brief explanations of why objects behave as they do, which would be appropriate for GCSE level, yet would not be in enough detail for an A-level student.

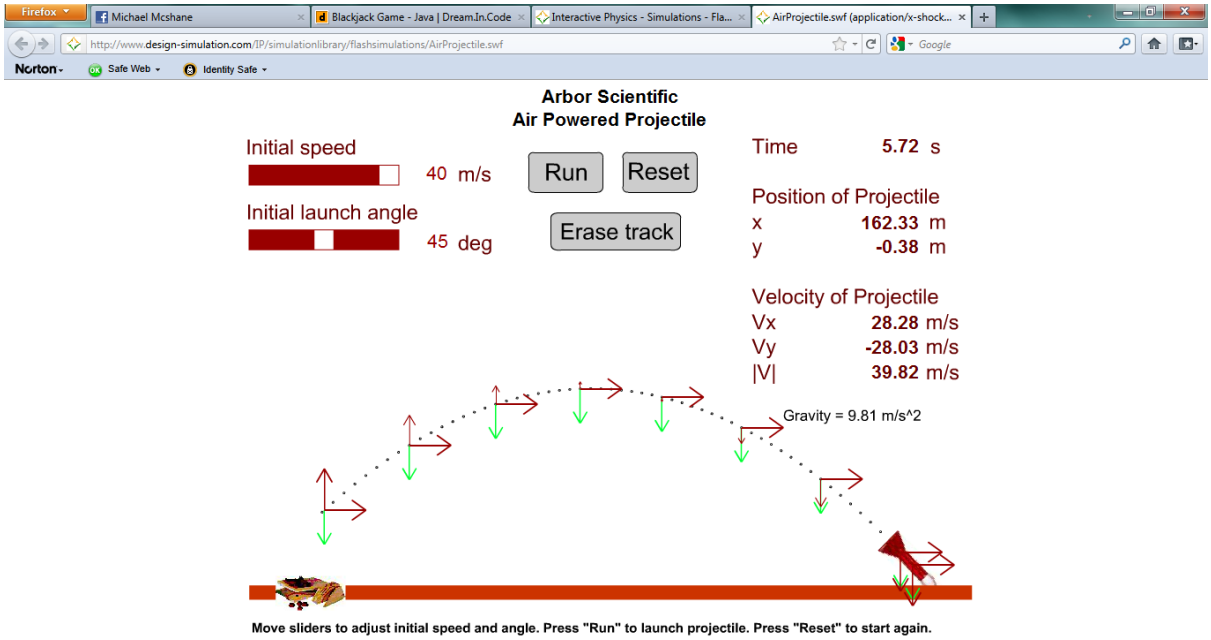


Figure 6 - Projectile Simulation

Interactive physics has a complex user interface, similar to a drawing package such as photoshop. The user can select a variety of components from a menu and alter their properties, then add them to a simulation as desired. The downside to this is that the interface appears cluttered, meaning that it would be difficult for an inexperienced user to create an experiment for themselves. It should be noted however, that to view simulations already included, the user need only know how to access the library of prebuilt simulations, meaning that the user interface can be avoided almost entirely.

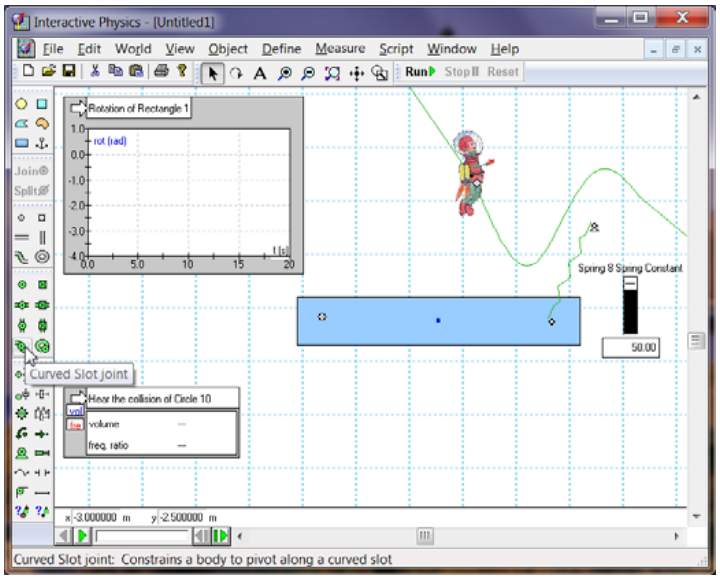


Figure 7 - Interactive physics user interface

There are several editions of physics online available, each with different functionality. The different versions available are as follows:

Demo edition

- Access to demo files only
- Files cannot be saved or copied
- Cannot export movies or data
- Available in 10 international languages

Textbook edition

- Licensed with a Developer edition and is able to play simulations created by the Developer edition
- Files cannot be saved or copied
- Limited printing and exporting capabilities
- Limited access to pre-designed physics experiments
- Created for textbooks, not instructors
- CD must be in CD-ROM drive for program to run
- Available in 10 international languages

Homework edition

- Sold and packaged in increments of 10 CDs and are available to customers with at least a 10-user license of IP
- Full use capability
- Access to all pre-designed physics experiments
- CD must be in CD-ROM drive for program to run
- Does not include user manuals or technical support and must be installed on student-owned computers, not on school equipment
- Available in 10 international languages

Full edition

- Full use capability
- Manual lookup word at first launch
- Allows saving and copying files
- Allows exporting data
- Access to all pre-designed physics experiments
- Available in 10 international languages

Developer edition

- Available to publishers and science curriculum developers whose customer base is on a state-wide or country-wide level
- Sold with a specified number of textbook versions (minimum of 500 textbook versions)
- Full use capability
- Access to all pre-designed physics experiments
- Capability to create files that can be opened by:
    - Textbook edition
    - Homework edition
    - Full edition
    - Developer edition
- Available in 10 international languages

Source: http://www.design-simulation.com/IP/versiondiff.php#Homework

Appendix C - Physics Concepts

The following highlights the concepts that could be covered by the Final system. Some worked examples have been provided to demonstrate how these concepts can be applied

Equations of Motion:

- For objects with constant acceleration
- Air resistance is ignored
- 6 equations:
    1. $v = u + at$
    2. $s = 1/2(u + v)t$
    3. $s = ut + 1/2at^2$
    4. $s = vt - 1/2at^2$
    5. $v^2 = u^2 + 2as$
    6. $a = (v – u)/t$

Where:

s = the distance between initial and final positions (displacement)
u = the initial velocity
v = the final velocity
a = the constant acceleration
t = the time taken to move from the initial state to the final state

Example:

A particle is projected vertically upwards with a speed of 34.3 m/s. Find the velocity of the particle after 4 seconds.

Using $v = u + at$

$v = 34.3 + (-9.8)4$ (-9.8 = gravity slowing particle)

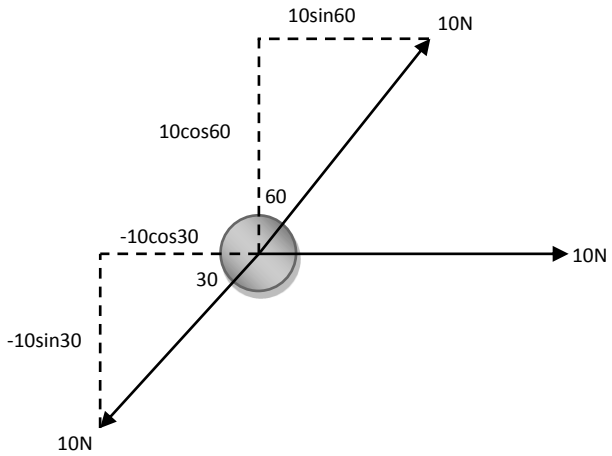
$v = - 4.9$ m/s^2

Components of Force

- For calculating the resultant force (and angle of said force) applied to an object
- Useful when multiple forces are being applied to an object
- Can also be used to deduce the horizontal and vertical components of given forces acting on an object

Example:

Consider an object with the following forces acting upon it:



To deduce the resultant force, first break up forces into horizontal and vertical components and calculate the resultant force of each.

Resultant horizontal = 10 + 10sin60 - 10cos30

= 10N

Resultant vertical = 10cos60 - 10sin30

= 0N

Calculate the total force by taking the square root of the addition of the vertical and horizontal components squared

Resultant force = $\sqrt{10^2}$ (since the vertical force is 0)

= 10N

Angle = $\tan^{-1}(0/10)$

= 0

The resultant force is 10N at an angle of 0 degrees (the two angled forces cancel each other out).

Coefficient of Friction

- A value that represents the friction between two surfaces
- Not constant, but increases until it reaches its maximum value
- Once this value is reached, the object in question will begin to move

To calculate the maximum frictional force:

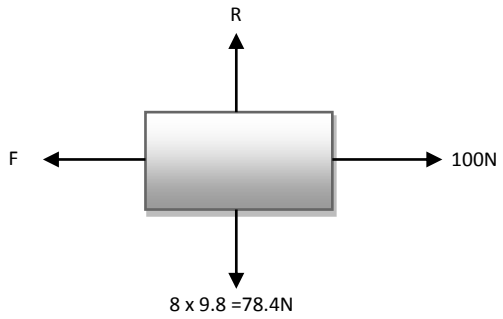
F = mR

Where:

- *F* = maximum frictional force
- *m* = coefficient of friction
- *R* = normal reaction force

Example:

A block of mass 8kg is placed on a rough surface with coefficient of friction 0.7. Find max frictional force and acceleration of the block if a force of 100N was applied to it.



First calculate max frictional force:

F = 0.7 x 78.4

= 54.88N

Now the acceleration of the object can be deduced using the formula:

Resultant force = mass x acceleration

100N - 54.88N = 8a

a = 45.12/8

a = 5.64 m/s^2

Pulleys

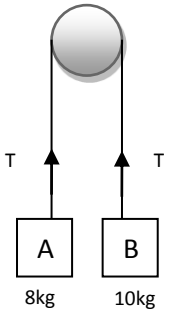
- Two particles connected by an inextensible string (which is assumed to have no mass) over a smooth pulley
- If these particles are of different mass, one will begin to pull the other over the pulley.
- The acceleration of these particles, and the tension in the string are calculated by using the formula:

  R = ma

  for each particle, and solving simultaneously.

Example:

Consider the following system:



To find acceleration:

Consider A:

TN - 8 x 9.8N = 8a

TN - 78.4N = 8a

Consider B:

10 x 9.8N - TN = 10a

98N - TN = 10a

Solve simultaneously:

98 - TN = 10a

-78.4 + TN = 8a

19.6 = 18a

a = 19.6/18

a = 1.08m/s^2

Use the acceleration to find the tension:

98 - T = 10x1.08

98 = 10.8 + T

98 - 10.8 = T

T = 87.2N

Momentum

- The momentum (P) of an object is equal to its mass (m) multiplied by its velocity (v) and measured in Newton seconds (Ns):

  $P = mv$

- Conservation of momentum states that when a collision between two objects occurs, the total momentum is preserved, hence:

  $\text{Momentum}_{\text{before}} = \text{Momentum}_{\text{after}}$

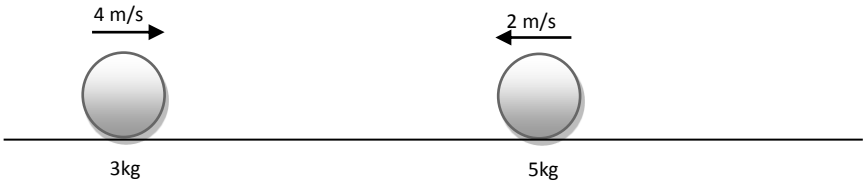
  Or, if two objects m_1 and m_2 collide, travelling at velocities u_1 and u_2 prior to colliding and v_1 and v_2 after, then this becomes:

  $m_1u_1 + m_2u_2 = m_1v_1 + m_2v_2$

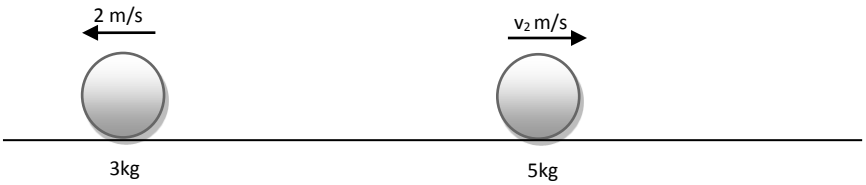
Example:

Two particles, P & Q, of masses 3kg & 5kg respectively are moving along a straight line with velocity 4m/s and -2m/s respectively. After the impact, the direction of P is reversed and its speed is 2m/s. Find the velocity of Q.

Before impact:



After impact:



As the momentum of the system is known to be the same before the collision as it is after, simply apply the values to the equation to find V_2 (it is assumed that movement to the right is positive):

$(3 \times 4) + (5 \times -2) = (3 \times -2) + 5v_2$

$12 - 10 = 5v_2 - 6$

$5v_2 = 8$

$v_2 = 1.6$ m/s

Impulse

- Impulse is the change in momentum of an object caused by a force (F), and is equivalent to the force applied multiplied by the time (t) the force is applied. Also measured in Ns:

  Impulse = change in momentum = Ft

  Or

  Ft = mv - mu (final momentum minus initial momentum)

- Force-time graphs can be used to plot force in a collision over time. Impulse can then be deduced by calculating the area under the graph

Example:

A ball experiences a force of 25N over a time period of 7.25 seconds. What is the impulse of the ball?

Impulse = Ft

= 25 x 7.25

= 181.25 Ns

Summary

After receiving feedback from Mr John Ivins, it was decided that the final system should implement as many of the concepts above as possible, with the exception of coefficient of friction. This is because friction is no longer as broadly covered in GCSE/ A-level physics.

References

References

1. Bonwell, C. 2000. *Active Learning: Creating Excitement in the Classroom* [Online]. Available at: http://www.ydae.purdue.edu/lct/hbcu/documents/Active_Learning_Creating_Excitement_in_the_Classroom.pdf [Accessed: 10 December 2011]

2. Dix, A. et al. 2005. *Human Computer Interaction*[Online]. Available at: http://cognac.ai.ru.nl/studie/MMI_summary.pdf [Accessed: 11 December 2011]

3. Freedman, R. 1996. *Challenges in Teaching and Learning Introductory Physics* [Online]. Available at: http://web.physics.ucsb.edu/~airboy/challenge.html [Accessed: 3 December 2011]

4. Ornek, F. et al. 2008. What makes physics difficult? *International Journal of Environmental & Science Education* 3 (1), pp. 30-34

5. Petty, G. 2004. *Active learning* [Online]. Available at: http://www.geoffpetty.com/activelearning.html [Accessed: 6 December 2011]