**Fiziks Simulator**

**Feasibility Report**

**Definition**

The program’s intended purpose is to act as a friendly simulator to work as demonstrations of universal gravity on Newtonian objects with user modifiable size, mass, and velocity of objects as well as the underlying gravitational constant. The user will be able to create objects and watch them interact in real time based on the input that was put into the system. The purpose of the program is to simulate the attraction between Newtonian objects in a fake universe with relative forces, magnitudes, distances and timespans.

**Pre-Problem Problem Analysis**

Calculus cannot definitively calculate the interaction of three bodies with gravitational attraction therefore there are no shortcuts for calculating the position of each object to the others. This means that a line for the acceleration of each object will be needed to execute over 16000 times per second for only ten objects. If the display is updating this in real time, it will be difficult to not slow. The handling of this problem will be dealt with experimentally, to find the optimum amount of objects without noticeably slowing performance. This all assumes the program executes a frame change sixty times per second which is the standard for action-based games.

Action-reaction issue, Newton’s third law will be difficult to put into the program for contact forces, even though trigonometry, vector logic, and the inverse square law make the program tick from a distance, when they interact it will vary too much. The distance between objects really close will make their attraction really great and they’ll accelerate more so rigid body physics have to be implemented. The only problem with that is if they realistically rebound and become closer as a fraction of their original forces then their distance will approach zero and the objects will teleport to infinity… Since that’s not ideal I have to build in momentum conservation laws that execute under the condition that the objects contact. Again the program may slow from this, so finding the sweet spot between experience and performance is the most important.

Keyboard support on Mac has been known to have issues so compatible versions may be released if necessary. To do so, the program will be recoded on a Mac accommodating the possibly different key codes for various events.

**Items to Consider:**

-a computer with a reputable 2nd generation processor and integrated math CPU, 4GB RAM, offline adobe flash player software to run on any Windows system, Mac support will be later implemented but unlikely to be on launch

-program should be suitable for all ages, but also ideal for audiences taking early Newtonian physics or calculus so that it can be used in demonstrations. Valid units restrict its use for extensive academic purposes.

-the program is open source and free, it is developed in Adobe Flash CS4 professional, paid for by the school

Note: A highly refined user interface may not be able to come out until after release as to not slow the development of the key parts of the physics engine due to time constraints.

**Research Requirements**

The writers of the program must understand the following:

-classical physics, Newtonian laws of gravitation, inverse square law, conservation laws, basic orbital geometry to make realistic ellipses

-vector logic and trigonometric relationships so that the program can work it out

-Actionscript 3 and how to operate flash, create classes, looping arrays, and manipulate user inputted data

-any other geometry, coding, or physics related to objects that have mass

**End User Requirements**

James Zhu, LCVI Secondary School student recommends (Jan 5, 2015):

-colourful and inviting gameplay

-include a starry background

-add music or sound

Jasmine Hurteau, LCVI Secondary School student recommends (Jan 5, 2016):

-detailed engine and design

-realistic physics including rebound physics

Mr. E. Finn, LCVI Secondary School teacher recommends (Dec 18th 2015):

-realistic forces and motion that have to be recalculated every frame change

**Design and Development:**



On form load:

User is prompted to create objects by clicking stage, when control is pressed it toggles this feature

If not then the user can edit text fields and other interactables

On keyboard event:

Control toggles creating

Space initiates/stops simulation (terminal stop probably on launch)

On stage click when not creating:

Creates and populates array of Newtonian objects,

Generates a text field each for initial x and y velocities editable by user, mass and radius of each object

On starting simulation (from pressing space bar)

Timer ticks

Start individual object timers

Set initial velocities for each objects based on textfield inputs

On Timer tick:

Solve relative attraction between all objects from nested looping of attraction equation

Check to see if objects are touching, when objects touch:

Repel touching objects based on repulsion equation

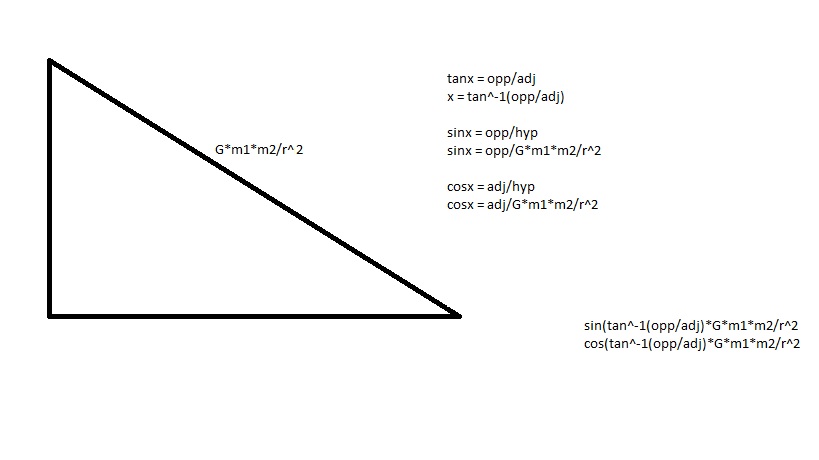
On stopping simulation (from pressing space bar an even number of times):

Stops all objects from moving on stage, refresh math and all that stuff

**Additional (Misc.) Logic**

Some of the inherent public traits of the Newtonian object class will include its velocity, as this will need to be modified quickly and efficiently; the two dimensions of its net velocity (x,y) = netVel[x-value,y-value], because velocity is just the sum of x and y vector components; its mass because this is one of the few variables to change to result in non-uniform accelerations; as well as the actual size of the Newtonian object on screen. The set traits will include acceleration due to gravity of a given Newtonian object which will be determined in the class as a mathematical function of its mass and the distance between another mass using the inverse square law. An arbitrary gravitational constant will have to be built into the program to allow the numbers to work nicely with one another so unit conversion will take place immediately following a public trait’s entrance into the NewtonianObject constructor to allow the program to move smoothly. Distances will be arbitrary as the scale of real gravity would be far too undetectable therefore a given pixel will not directly correspond to a distance unit in real life.

Vector logic:



-a vector can be considered a numeric quantity with direction, and magnitude

DIRECTION: Absolute value of distance between objects determines which way to move, this only works at right angles SO to change the rate along each x, y distance the program has to take the sine of the inverse tangent of the angle to get the direction of acceleration

MAGNITUDE: (Inverse square law) multiply the math above by the gravitational constant in the program, a corresponding mass, then divide by the distance between the two, in this way all objects are attracted to all others at varying magnitudes as a function of their distances and respective masses (just like in real life) ~~Next up special relativity~~

To double an array (the reason for this was explained earlier):

*Loop through array, each step sets the newly generated element the element of the current index of the loop*

To make all things attracted to one another:*(Solving attraction of the array of ‘things’)*

*Loop through half of a doubled array (terms repeat once as stated above)*

*Within this loop is another of the same length and calculates the attraction of one object to the next object denoted as the current index plus the value of the inner loops’ counter variable. With a variable to attract each object to each other object, always ‘i’-places ahead in the array of things*

\*Hopefully this whole thing goes well.