Phys Wiz Design Document

Identification

Project Name:
Phys Wiz
Project Leader:
Jeffery Saeteurn (saeteurn@princeton.edu)
Project Members:
Jeffery Saeteurn (saeteurn@princeton.edu)
Yosvani Lopez (ylopez@princeton.edu)
James Lin (jhlin@princeton.edu)

Overview

Phys Wiz is an iPhone application that helps students visualize problems that they will come across in a beginning general physics class. It will dynamically model problems related to kinematics, circular motion, harmonic oscillations, and other topics covered in a typical classical mechanics course.

Our app will allow users to drag and drop certain shapes onto the center of the screen and assign it certain properties such as velocity, acceleration, mass, and etc. The user will also be able to drag and drop pulleys, springs, rope, and etc onto the screen, assign properties to these things, and attach them to their objects. Our app will then dynamically model its behavior in accordance to newtonian mechanics and output certain unknowns given the properties that the user inputted.

The front end of our iPhone application will be written in Swift using Xcode, the integrated developmental environment provided by Apple for iOS applications. The backend will already be handled by the SpriteKit and the built-in physics engine that Apple provides for developers. The backend of this application will also store application states so that users may save specific problems, continue the problems at a later time, or even share the problems with other students!

Target Audiences

1. A high school student taking AP physics is stuck on his homework assignment can use the app to gain a better intuition of the problem he's dealing with.
2. A college student taking Physics 103 is too lazy to draw out problem on paper, so he uses the app to drag and drop everything onto the screen to visualize the problem.
3. A physics student is having trouble with a problem that deals with harmonic oscillations because he just can't visualize the problem by drawing a boring, static picture of a box attached to a spring, not moving or doing anything at all. He remembers seeing an app in the app store the other day called *Phys Wiz*, but he didn't download it because he thought the name sounded too dorky. He sucks up his pride, downloads the app, and models the problem, and to his amazement, the box is moving back and forth on the spring and its

amplitude is slowly going down. He gets harmonic oscillations, solves the problem, and aces the class. The end.

4. A student is on the bus and has nothing to do. He has a physics problem set due in four hours and he hasn't started on it yet, but he doesn't want to take out his pencil or paper to work on the problems. Luckily for him, he downloaded *Phys Wiz* the other day, so he pulls out his phone, opens up the app, models the problem, and BAM! Epiphany! That's one problem down.
5. Even though this app may seem like purely a tool used to assist in solving physics problems, it can also be used for entertainment purposes. Building a bunch of blocks and knocking them over with a bunch of random objects will surely keep people occupied in times of boredom.

Functionality

The app will have a main screen that users can drag and drop shapes and objects onto. This will be the screen that models the problem that the user wants visualized. In order to drag and drop the shapes onto the screen, there will be a bar on the righthand side of the screen that has a list of shapes that the user can place on the main screen. There will also be a bar on the left-hand side of the screen that has items such as pulleys, springs, ropes, and other objects typically seen in courses on classical mechanics. Finally, there will be a bar on the bottom of the screen where the user can input certain properties that the objects have. This bar will also display unknowns that can be calculated using the properties that the user gives us. All of these bars will have a "hide" option so that the user can maximize the screen space used to visualize the problem.

Design

Front End (User Interface Tier)

1. **User Interface**
    1. **Object Listbox:** Contains all the objects that can be added onto the scene.
        1. Allow for adding objects into the scene from the list box.
        2. Ensure that the objects correctly interface with the middleware so that it assumes properties as detailed by the Spritekit.
    2. **Object Properties Status Bar**: Details all the information about each object: Mass, Velocity, Momentum, Acceleration, Net force applied to/by it, Angular velocity, Angular acceleration, Torque, Kinetic Energy
        1. This will be interfaced with the middleware so that it receives information from the physics engine about the properties.
    3. **Contraptions**: Contains all the physical contraptions that can be applied in a system such as Springs, Pulleys, and Ropes
    4. **User Action Listbox**: Contains all the actions that a user can invoke on an object. The actions that the user will be able to apply are *(*Note all directions on vectors will be specified either using gestures or manually inputting the angles *)*:
        1. Apply force, apply torque, set velocity, set acceleration,

5. **The Log:** Contains all the information of user actions (Actions applied by user, highlighted properties, time delta, etc)
6. **The Scene**: The canvas where all the objects will be shown interacting.
    1. Using pinch gestures, the user will be able to set the scale of the scene (change coordinate bounds).

The application canvas will be a screen with a graph paper background that the user can drag and drop objects onto. There will be bars on the sides and on the bottom of the screen that the user can open and close. The bar on the righthand side of the screen will give the user shapes that he or she can drag and drop onto the main screen and give properties to. The bar on the bottom left hand side of the screen will give the user objects that are commonly seen in physics problems such as pulleys, ropes, and springs that the user can drag and drop onto the main screen. The upper left bar will contain vectorial actions that the user can apply/set such as velocities, forces, accelerations, etc.

The bar on the bottom of the screen will display properties of a selected object in the scene. The user can open that bar either by tapping on a "show" icon on the bottom of the screen or by clicking on an object in the main screen. The display will include the objects that are currently on the screen, followed by an icon that either hides or shows all of the properties of that object. These include all of the properties that the user inputted and all of the properties that can be calculated using the information that the user gives us.

On the upper-right hand corner of the application, there will be an icon the pauses/plays the simulation and an icon the resets the simulation. Also, next to these two icons will be an icon the toggles gravity on and off. On the bottom of the screen, right above the properties tab, there will be a slider that the user can use to pause the simulation at a certain time step, so that they can see the properties of a system at that particular point in time (e.g. The amplitude of a pendulum at time $t = 5$ seconds).

Middleware

In the middleware level, we will be interfacing the frontend user interface with the physics engine. We will implement this in Swift by creating Sprite objects using Apple's Sprite Kit Physics Engine. With this, we will be able to impose basic physical functions (set mass, velocity, etc) on the created object and simulate realistic physics problems. We will perform physics calculations for incremental steps guided by a time component. In this level, we would also receive the data from the physics engine and properly display it to the user. The display would include both the simulation provided from the physics engine and the data used to provide the simulation for each object.

Back End (The Physics)

The backend of our app will largely be handled by Apple's SpriteKit and its built-in physics engine. The main part of the back end that we will be implementing is saving and restoring physical states. To save each physical state, the positions, masses, and all other actions imposed on each object will have to be stored in some sort of file system. We will be using Apple's native Core Data as the framework that will save the data states. Restoring states would be almost the same thing but in reverse; we will be recreating scenes from data read from the saved states. All

of this is to ensure that users can leave the app and return to the same exact place or so that the user can share worlds with other users.

Timeline

1. Analysis of problems / Extraction of data
    1. Motion and properties of a single object
        1. Given properties (velocity, mass, friction), locations
    2. Interactions between objects (collisions, tension)
    3. Environment (gravity, levels, grounds, ceilings)
2. Connecting data and translating to use physics engine to solve
3. Displaying output
    1. Moving objects
    2. Displaying calculations
4. Getting incremental output and working out visualization
5. Local data storage
    1. Store history of problems previously entered.
6. Expansions optimizations
    1. myScript type of interface
    2. Equation solver
        1. Would directly use physics engine function to solve equations

April 2

- Creating sprites and performing basic 2-D kinematic motion (no gravity, collision, etc)
- Develop a working link of communication between the user interface and the physics engine. At this point we will be working with a simplification of the physics problems we plan to solve to ensure all parts will work properly and to what extent they will work.

April 9

- Write code for all objects and environmental features for problem modeling. Environmental feature include gravity, levels, grounds, etc.
- Implement the middleware such that the coordinate system would be set to be relative to other objects given the current environmental details. (Block A is 5 feet to the right of block B given that they are both on the same platform)
- At this point given a set of objects and environmental features we would be receiving input data from the user interface and then we would interpret it using our own design paradigm (relative coordinates, etc). All the under-the-hood calculations would be maintained by the middleware communicating with the physics engine (this is to maintain modularity)--with it only giving the frontend certain calculations done by the physics engine to be displayed.

April 16

- Construct the interface such that the user is given the ability to model their problem using both drag and drop objects and a data input box
- Given a model from the user the interface will then call the correct function and obtain the data needed to generate the output.

- Our goal at this point is to have an interface that is as simple and complete as possible without necessarily implementing the simulations yet. We will also be focusing on making the interface intuitive and minimal.

April 23

- Finish interface including the simulation aspect of it. This involves using the data received from the physics engine to create the simulation and also includes optimizing how and when we update the simulation (finding the proper balance of space and time used when updating)
- Add extra but necessary components to Phys Wiz like a history tab that saves the user's recently entered problems.

April 30

- Thoroughly test and debug (having a bunch of random people playing with it would surely be a good method to help test).
- Add non-essential features. These are feature that could make the app more fun and user friendly but that are not necessary. This includes possibly integrating a myScript type of interface that allows the user to draw in objects that would then be replaced by the object that most closely resembles what they drew. Another possible expansion may be a custom object option that would allow the user to draw an object that they could then give properties to and create simulations with.

May 2

- Have demo presentation done and rehearsed.

Risks and Outcomes

1. Because our application gives the answers to the physics problems the user inputs into the app, schools might ban students from using it.
2. No one in our group knows how to program in Swift and only one person in our group has experience in iOS development, so a bit of time will be needed for the members in our group to get used to the new programming language.
3. The use of the iOS physics engine may be a risk as we do not know the exact ins and outs of it. We have made some general assumptions as to its capabilities and if it does not work as we expect it too we would either change the way we implement Phys Wiz so that it works better with what the physics engine allows us to do or we would look for an alternative option like the physics engine in Unity.