IWP Version 4.03 Tutorial

*by Jason Cockrell, North Carolina School of Science and Mathematics*

**Module 1. How to create a simple problem.**

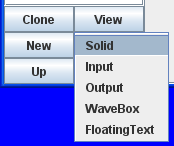
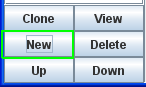
In this tutorial, you will learn how to create a simple projectile problem in which a cannonball is fired horizontally from a cliff and students can observe the path of the trajectory as well as adjust the height of the cliff and the initial velocity of launch and take data on the velocity of the ball at various times. In addition, you will incorporate a hidden adjustable component for gravity that will allow for a simulation of ballistics on another planet.

Open version 4.03 of the IWP author jar file. This will allow you to create and edit problems. The instructions that follow pertain to the designer file.

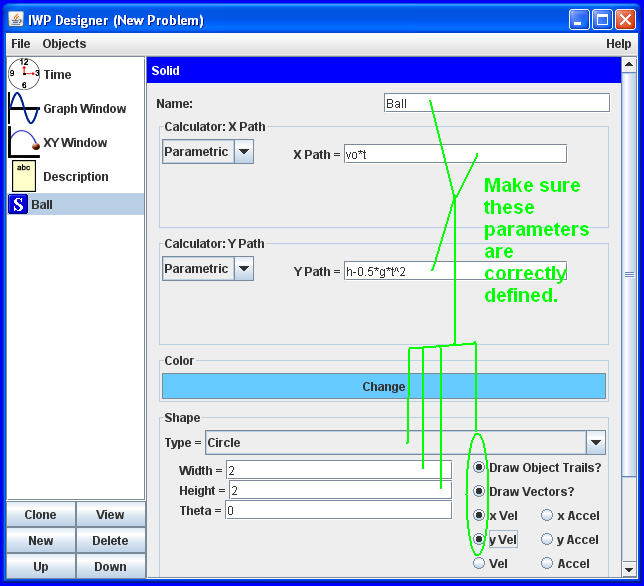
**Ball**

To start, click “New” and select “Solid.”

🡪

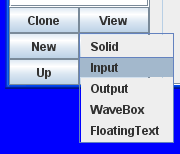


You will be immediately taken to the object editor interface. At the top of this editor is the name; name your projectile something simple like “Ball” since you will need to use the name in equations later. Define the X Path of the Ball as *vo\*t*. The Y Path should be h-*0.5\*g\*t^2*. Under Shape, select “Circle.” Change the Width and Height to 2 each. Check the circles for “Draw Object Trails?” and “Draw Vectors?” Do not check “Graphable?” (We’ll take up graphing later.) Underneath “Draw Vectors?” check “x Vel” and “y Vel.” This will allow the student to see the path of the object and vectors indicating its horizontal and vertical velocity while playing the animation.

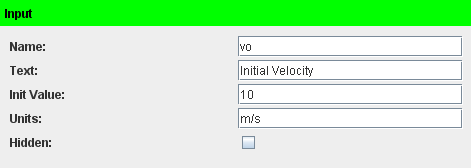


**Inputs**

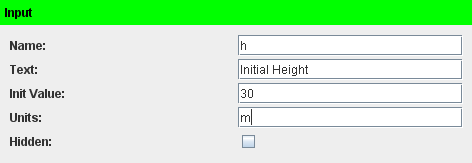
Now you will need to define some variables that students can later modify to adjust the path of the projectile. Click “New” and select “Input.”



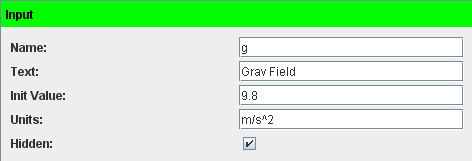
Name this input *vo* and put a one- or two-word description in the “Text” bar such as “Initial Velocity.” Do not name your input “e” or “E.” IWP recognizes this as the mathematical constant “e” and will not understand what you are trying to tell it. (If you must start a name with “e”, type a different letter first. Then slip the “e” in front. The same applies to “pi”.) Set the initial value to 10 and the units to m/s. Make sure to leave the “Hidden” box unchecked.



Click “New” and select “Input” again. Name this input *h* and in the “Text” bar put “Initial Height.” The initial value should be 30 and the units, m. Again, leave the “Hidden” box unchecked.

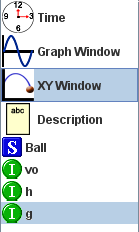


Finally, create a third input, named *g* and labeled “Grav Field.” The initial value is, of course, 9.8, and the units, m/s^2. Do not try to use a superscript to indicate exponents as IWP does not recognize this formatting. Now, click on the “Hidden” checkbox to check it. This will prevent students from seeing or editing this input unless you uncheck it. This means your projectile will be fired on Earth’s surface unless you decide otherwise.

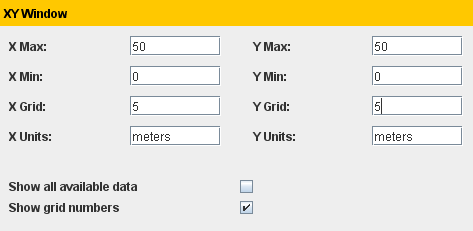


**XY Window**

Click on “XY Window” to view the XY Window editor interface.

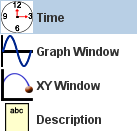


Set your X Max and Y Max to 50 each, the X Min and Y Min to 0, and the X Grid and Y Grid to 5. The units are set to meters by default, but they will not appear to the student. (That’s a bug.) For now, leave “Show all available data” unchecked. As long as “Show grid numbers” is checked, the student will be able to see the scale numbers of the XY Window when viewing the problem. If you do not want the student to know the scale, uncheck it.

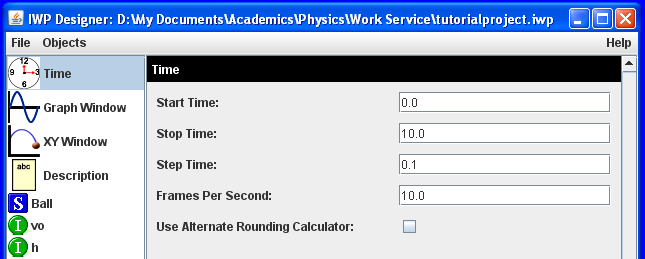


**Time**

Click on “Time.”



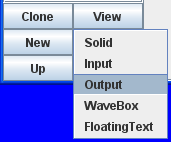
This will allow you to adjust the properties of time in the animation. You will see that the default settings are these:



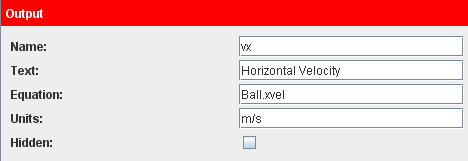
There is no need to adjust the default settings for this problem. Ten seconds should be long enough for a simple projectile problem, and, of course, there is no reason not to start at *t* = 0.0 seconds, although you could start at a negative or positive time if you wished. The Step Time controls how much time passes *in the problem* per frame. Frames Per Second controls how many frames the animator displays per second of *real* time. Note that, since each frame is set by default to represent 0.1 seconds of problem time and there are 10.0 frames per second of real time, problem time and real time are synchronized. If FPS is increased without Step Time being decreased, the animation will run faster than real time, and vice-versa. Do not increase FPS beyond 25, as that will result in the animator trying to do too many calculations too rapidly. If you find that an animation won’t run, that may be because the FPS is too high.

**Outputs**

Click “New” and select “Output”.



Name this output *vx* and label it “Horizontal Velocity.” For the equation, type exactly: Ball.xvel (Note that this is case-sensitive and you must use the correct name of your solid object). The units are m/s.



Now click “Clone.”



This generates a copy of the previous output. Change the name from *vxClone* to *vy,* the label from “Horizontal Velocity” to “Vertical Velocity,” and the equation from “Ball.xvel” to “Ball.yvel.” This format is applicable to a range of possible outputs. You can create as many as you want. Use “name-of-solid.function”. The available functions are xpos, ypos (position), xvel, yvel (velocity), xaccel, yaccel (acceleration).

It is also possible to create more complex equations for Outputs. For example, suppose you want to know the magnitude of the velocity vector of the Ball, not the horizontal and vertical components. You would simply need to define the Output Equation as exactly “(Ball.xvel^2+Ball.yvel^2)^0.5”.

**Using the Animator**

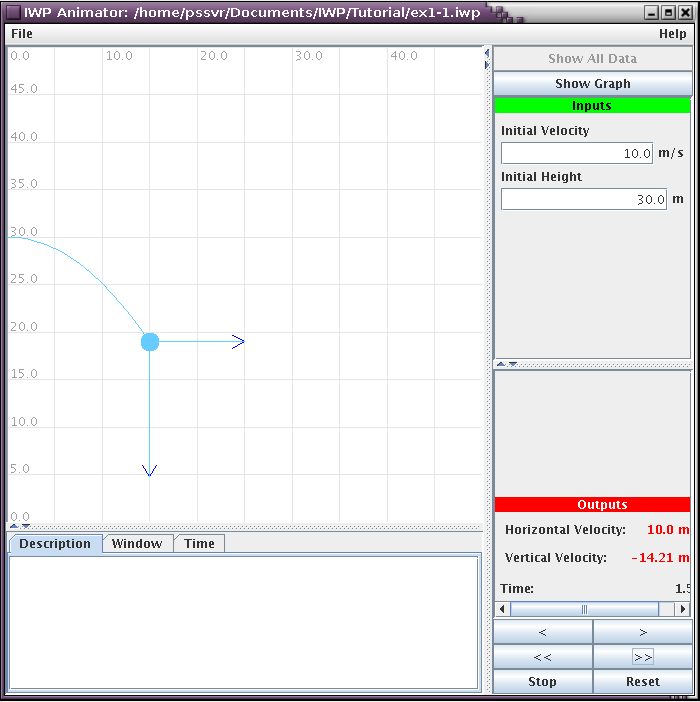
Finally, click View to open up the Animator. This is the view the students will have when they do the problem.



The ball is located on the left side of the XY Window. Play the animation by clicking the single right arrow.

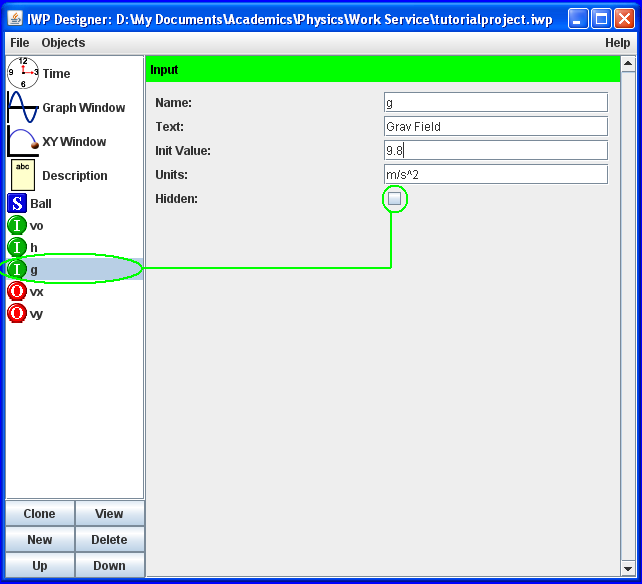


The ball is fired at an initial horizontal velocity and accelerates downward due to gravity. Additionally, the student viewing the Animator can adjust the values of the Initial Velocity and Launch Height, and view the values of the Horizontal Velocity and Vertical Velocity. Here is the animator in action. Note the velocity vectors and the parabolic trail.

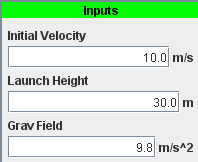


Click Reset to restore the Animator to its initial state. Change the Initial Velocity to 15 m/s and click Reset. Play the animation again. Notice that the projectile travels a greater horizontal distance. Reset the animation again. Now, move through time in 0.1-second intervals by clicking the double right arrow, just underneath the play button. Click this button ten times to see what happens to the ball after one second. Note that the vertical velocity increases at the rate expected considering the value of *g*.

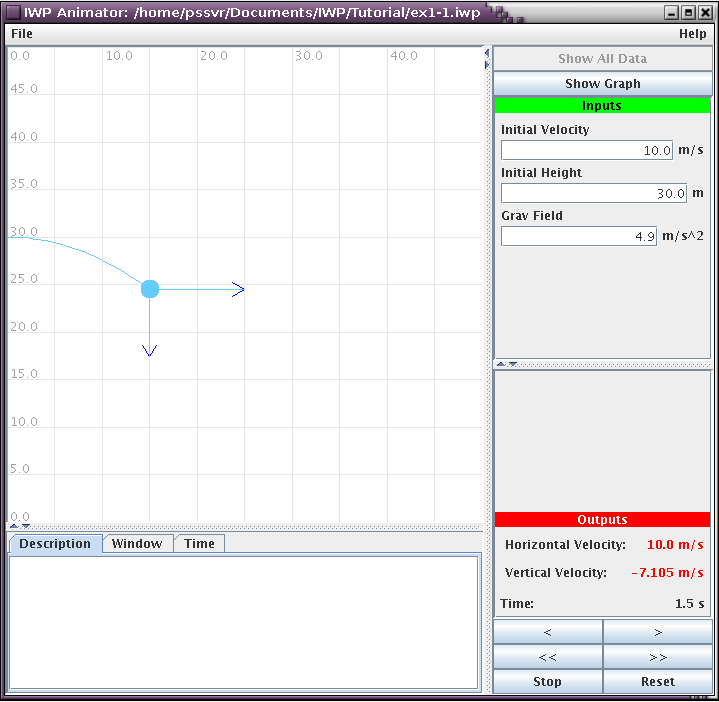
Exit the Animator and return to the Editor. Click on the Input *g* and uncheck the “Hidden” box.



Click View again to return to the Animator. Note that the student can now see and change the value of *g* from the Animator.

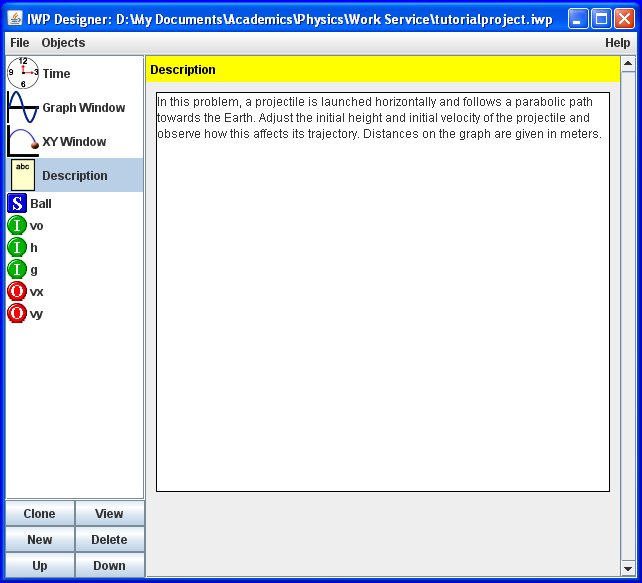


Reduce the value of Grav Field to 4.9 m/s^2, click Reset, and play the animation again. The ball does not accelerate downward as rapidly as before because it has been fired on another planet or in a very large accelerating elevator.



**Adding a Description**

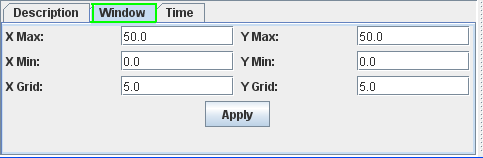
Exit the animator and return to the Editor. Click “Description” and type in a description as shown.



This description will be visible to the student when he or she is viewing the problem.

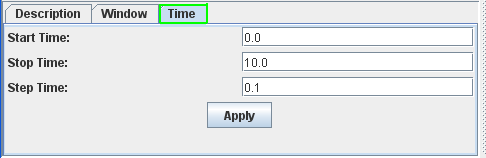
**Window and Time**

Above the description in the Animator, there are three tabs: Description, Window, Time. Click on Window.



The student can adjust these parameters as desired to be able to better see the XY Window in the Animator. Make sure to click Apply after an adjustment. X Grid and Y Grid define the distance between grid lines on the window. It is not advisable to reduce these to less than one-twentieth of the difference between the respective Max and Min values, as that would create far too many grid lines which would slow the Animator and obscure the view.

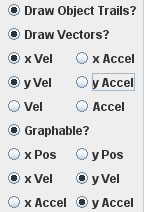
Next, click on Time.



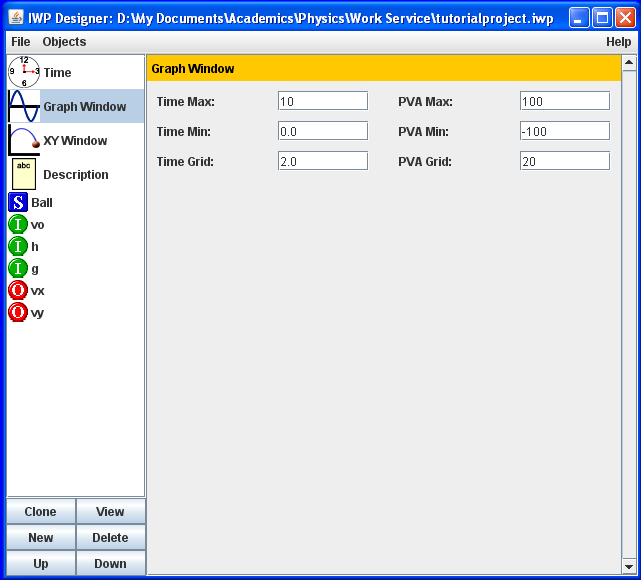
Once again, the student can adjust these parameters and must click Apply. If the Step Time is made too small relative to the difference between the difference between the Stop and Start Times, the Animator will take too long to run.

**Graphing**

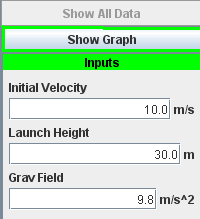
Now close the animator once again. Click “Ball” to edit your projectile. Scroll down and check “Graphable?,” “x Vel,” “y Vel,” and “y Accel.”



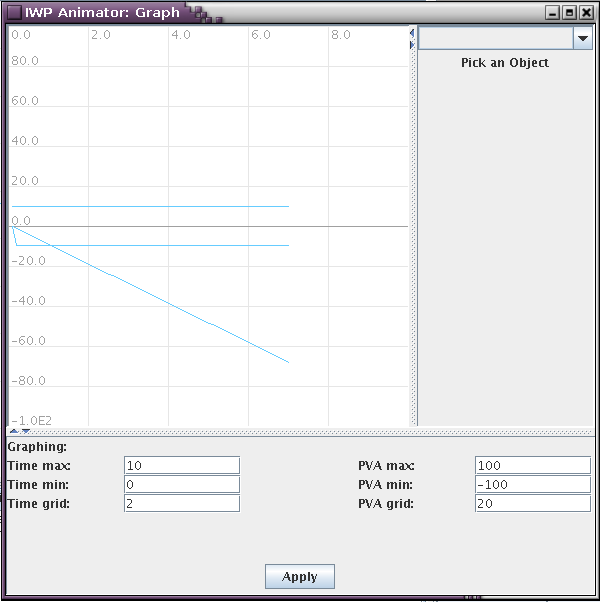
Click “Graph Window” to view the graph editor. The default settings will not work for this problem. Set “PVA Max” and “PVA Min” to 100 and -100, respectively, and “PVA Grid” to 20. This will make the scale of your graph manageable so students can actually see what is being plotted.



Now click “View” again to return to the animator. Click “Show Graph” to display the graph.

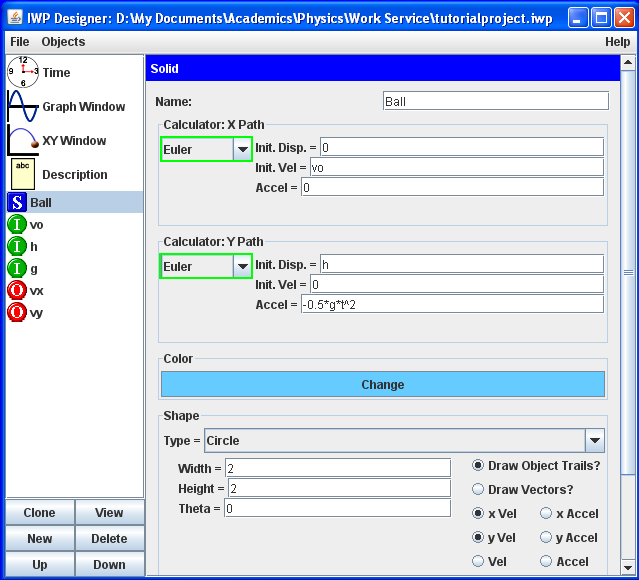


Click the single right arrow to play the animation. Observe the graph as horizontal and vertical velocity as well as vertical acceleration are plotted over time. There is no color coding of the three plots, but it is a good mental exercise for students to figure out which graph goes with which function.



**Euler Equations**

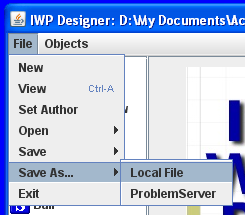
Thus far you have defined the projectile’s path using Parametric equations, but it is possible to use Euler’s Method to approximate the trajectory. In the Editor, click on Ball and set the equation type in the pull-down menus to Euler instead of Parametric. Do not worry about RK2 and RK4, which are higher-order approximations. The X-Path should be described as having an Initial Displacement of 0, an Initial Velocity of *vo*, and an Acceleration of 0. The Y-Path has an Initial Displacement of *h*, an Initial Velocity of 0, and an Acceleration of -0.5\**g*\**t*^2.



Open the Animator and run the animation. The result is essentially the same, but the parameters were a little bit easier to input.

**Saving Files**

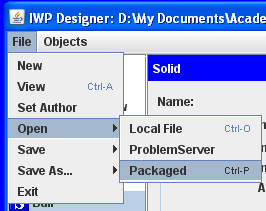
In the Editor, not the Animator, click File, Save As, Local File. Do not bother saving to the Problem Server.



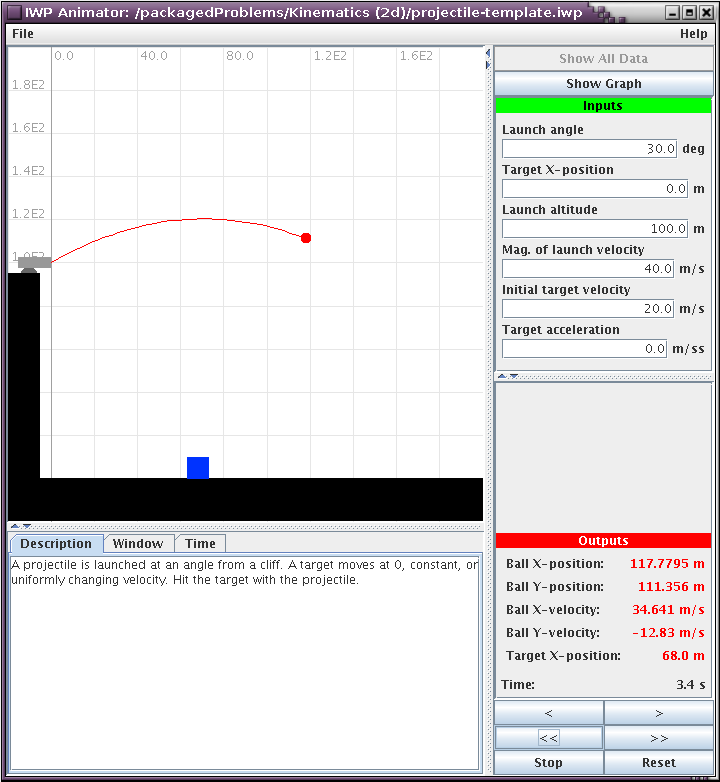
Name the file and save it. Note that IWP will automatically save files to the same directory in which the designer.jar file is located. A file can be manually saved to a different directory. However, if edits are made later to such a file, they must be saved using File, Save, Local File, as opposed to File, Save As, Local File. Save As will always save to the default directory unless manually told to use a different one.

**Opening a Packaged Template**

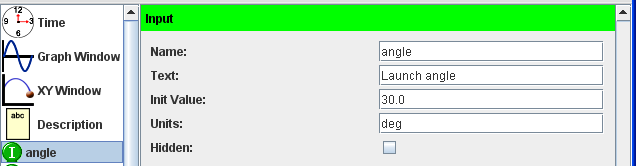
In the Editor, click File, Open, Packaged.



In the file directory, select Kinematics (2d), projectile-template.iwp, and click Open. Go ahead and click View to see the Animator, and run the animation. A cannon fires a ball off of a cliff while a target speeds along the ground below.



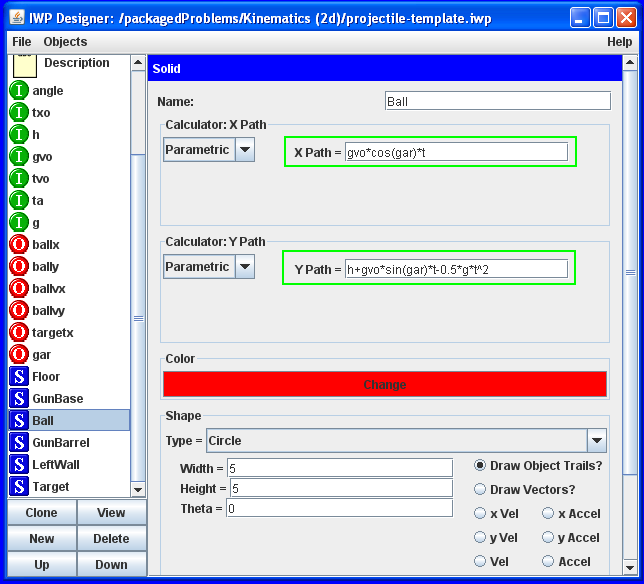
This is a complex problem with many Solids, Inputs, and Outputs. Close the Animator and return to the Editor. Click on the first Input, “angle.” This is the launch angle at which the cannon fires the ball.



Note that the initial value is 30.0 and the units are degrees. IWP handles trigonometric functions in radians, not degrees. To allow the student to adjust an Input in degrees, a simple function is required. Click on the last Output, “gar.” This Output acts as a conversion from degrees to radians.



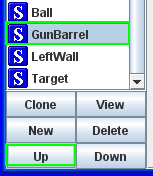
“toRadians(value)” is a function that multiplies the value in parentheses by π/180. The Output is hidden from the student because there is no need for him or her to know it; it is simply rephrasing the value of the “angle” Input. To see this concept in action, click on the Solid “Ball.” Notice that, in the X- and Y-Paths, the Magnitude of the Launch Velocity, represented by *gvo*, is multiplied by cos(gar) and sin(gar), respectively. This resolves the Launch Velocity into its horizontal and vertical components using the *gar* Output. You can also use “toDegrees(value)” to convert into degrees.



In the list of Inputs, Outputs, and Solids on the left side of the Editor, scroll down to see all the Solids. Notice that “Ball” is vertically on top of “GunBarrel.” IWP reads from the top down. When multiple Solids are to be displayed in the same location, the lowest one appears on top of the others in Animator. To see this, return to the Animator and note the specific positions of the Ball and GunBarrel relative to one another.



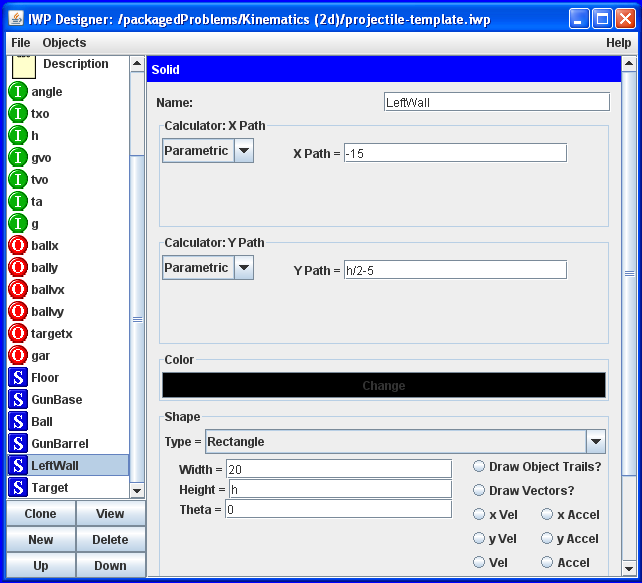
Exit the Animator. In the Editor, select “GunBarrel” and click “Up” to move it up one space in the vertical column.



Now “GunBarrel” is listed on top of “Ball.” Return to the Animator. Notice how the Ball appears to be outside of the barrel.



The layering of Solids can be important when creating scenery and effects. Some of the Solids in this problem exist completely for aesthetic purposes. To see how to create scenery objects, click on the “LeftWall” Solid. The LeftWall is essentially the cliff on the left side of the Animator that supports the cannon. Notice the parameters it has.

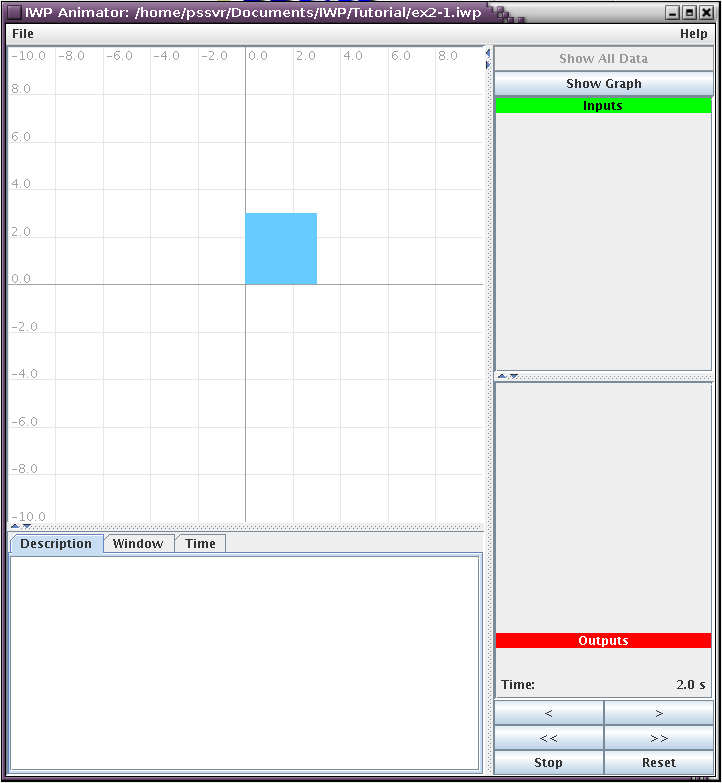


The Shape is, of course, a rectangle. The Width is 20 because that is a convenient width. The Height is *h*, which, in this problem, is defined as an Input for the Launch Height of the projectile. It makes sense that the cliff on which the cannon rests would have to be as high as the height of the launch. This means that the size of LeftWall will change if the student adjusts the Launch Height. More importantly, the Y Path is defined as “h/2-5.” This is because the X and Y Paths define the position of the center of the Solid. The center of the cliff is, of course, located at half of its total height. The “-5” serves to shift the cliff slightly downward of where we might expect it, so that there is room for the cannon. This way, the cannon sits on top of the cliff instead of being embedded inside it.

Note: Files cannot be saved from the Editor to the packaged problem directory. They must be saved locally.

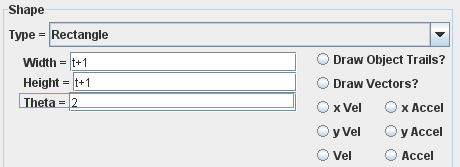
**Miscellaneous Notes**

Just like the X and Y Paths, the dimensions of Solids can be made time dependent. A very simple example would be a square that initially has sides of length 1 meter and grows another meter in both directions every second. To accomplish this, create a Solid of the type Rectangle and define its width and height to be *t* + 1. The square’s center will be at the origin and it will expand uniformly with time. If you want to achieve the same expansion but keep the lower left corner of the square at the origin at all times, set the X and Y Paths to 0.5\*(*t*+1) each. Here are those parameters in action:

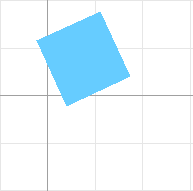


Note that this does not only apply to squares but to any Solids of the type Rectangle. If desired, one dimension of a Rectangle can be made constant and the other time-dependent. In this case, the Rectangle will stretch or compress in one dimension.

The sides of Rectangles do not always have to be parallel to the axes of the XY Window. Rectangles can be inclined by inputting a value for the Theta parameter. Note that Theta operates in radians, not degrees.



Here is the same Solid created in the previous note, but inclined by 2 radians.



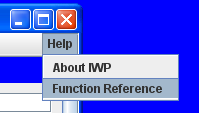
The value of Theta can also be made time-dependent. Try running the animation for a rectangle whose dimensions and angle are all *t*. Note that the rotational feature does not currently function for Solids other than Rectangles.

If desired, you can designate yourself as the Author of a problem. In the Editor, click File, Set Author.



Type your name into the window that appears and hit Enter.

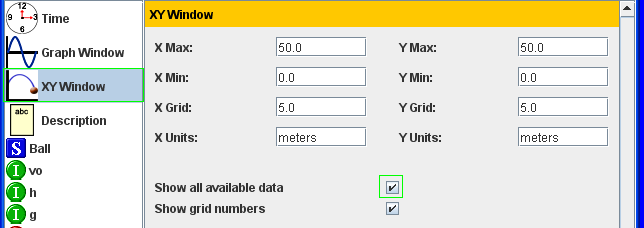
There is a compendium of all functions currently available in IWP. To access it, click Help in the upper right corner and select Function Reference.



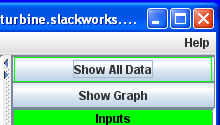
This will display a list of functions as well as recognized constants. The sign and signum functions output a 1 if the input is greater than zero and a -1 if it is less than zero. Signum returns a zero if the input is zero, while sign gives a 1. The modtwo function returns a 1 if the input is odd and a zero if the input is even.

Avoid creating a WaveBox Solid. This feature is currently under maintenance. It works, but is too difficult to learn to use. The interface is being improved, as well.

It is possible to collect large amounts of data from IWP and analyze it with a spreadsheet. Open the projectile problem created in this module and go to the XY Window Editor. Once there, check “Show all available data.”



Now, open the Animator and run the animation. In the upper right corner, click Show All Data.



This will open a window with data. Hit CTRL + A to select, CTRL + C to copy, and CTRL + V to paste this data into Notepad. Save the data in Notepad using the following format, with quotation marks included: “data.csv” Next, open Microsoft Excel. Open the csv file in Excel. Make sure that, when trying to open the file, you are looking through all file types, instead of just files with an xml tag at the end. Excel will open the csv file and separate the data into meaningful columns. The student can do all of this from the Animator, so he or she can analyze data effectively.

Files can also be opened from the Animator by clicking on File, Open, Local File or Packaged in the upper left corner.

Powers of ten notation can be used in Inputs. For example, to indicate the value of the charge of an electron, it is convenient to use 1.6e-19. IWP will read this correctly in an Input, but will not understand how to handle it in an Output, function, parameter of a Solid, et cetera. To circumvent this issue, it is recommended that Inputs be created to represent very large and very small numbers. In a problem that frequently deals with the charge of an electron, define an Input that has the value 1.6e-19 and use that Input in other functions when needed.

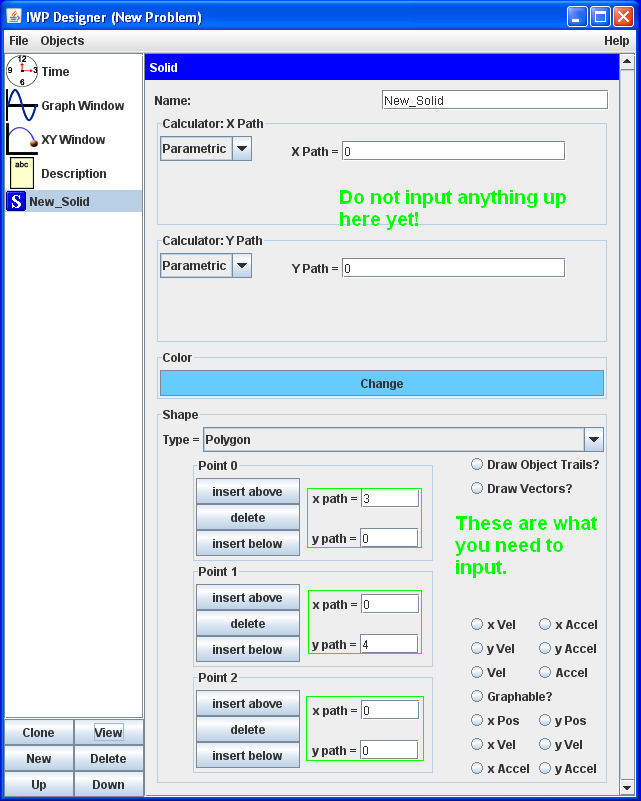
**Module Two. How to Create and Use Various Complex Objects.**

In this module, you will learn how to make more interesting and more versatile objects than circles and rectangles. This includes waves, polygons, lines, vectors, text, and some special images representative of real-world objects. You will learn to adjust their parameters to achieve elastic ropes, accelerating vehicles, and more.

As with Module One, open the designer.jar file to access the problem editor.

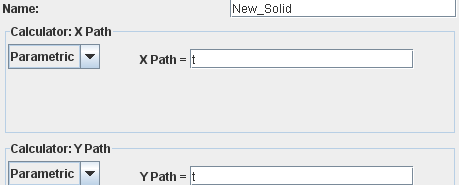
**Polygons**

Create a new Solid. For Type, select “Polygon.” Polygons are defined in terms of the locations of each vertex, however many they may have. There are initially too many points for everything to be displayed in the Editor window. For this reason, you will need to click and drag to enlarge the Editor window until you are able to see the interfaces for inputting the parameters of all three of the default points. There are three because polygons always have at least three vertices. To start, create a simple triangle by defining Point 0 as having an X path of 3, Y path of 0, and Point 1 as having an X path of 0, Y path of 4, and Point 2 as having an X path of 0, Y path of 0. Ignore completely the parametric equations at the top half of the screen. You have created a simple “3-4-5” triangle.



Note that it is completely irrelevant which Point # is defined as which location. You could have just as easily given Point 0 the coordinates (0,4) and Point 1 the coordinates (3,0), and the result would have been the same. Now click View to see the triangle.

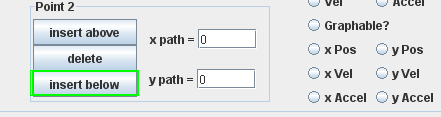
Return to the Editor. Now input “*t*” for the Parametric equations in the upper half of the Editor.



Click on View to see the Animator and run the animation. Note that the entire triangle moves horizontally and vertically at a rate of 1 m/s each. This same effect could have been achieved by adding “+*t*” to the coordinates of each individual points, but with more typing and room for error.

Now set the Parametrics equal to 0 again. Change the coordinates of the (3,0) and (0,4) points to (3+0.75\**t*,0) and (0,4+*t*), respectively. Click View and run the animation again. Observe that two of the points of the triangle move, while one stays at the origin. The shape of the triangle does not change due to the 0.75 multiplier. Name this Solid “Triangle.”

Now create a new Solid. Set its Type to Polygon and name it “Pentagon.” Underneath Point 2, click “Insert Below.”



This creates Point 3. Click “Insert Below” again to create Point 4. Now, input the following coordinates exactly as described. This time, order does matter:

Point 0: (0,0)

Point 1: (1,-1)

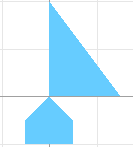
Point 2: (1,-2)

Point 3: (-1,-2)

Point 4: (-1,-1)

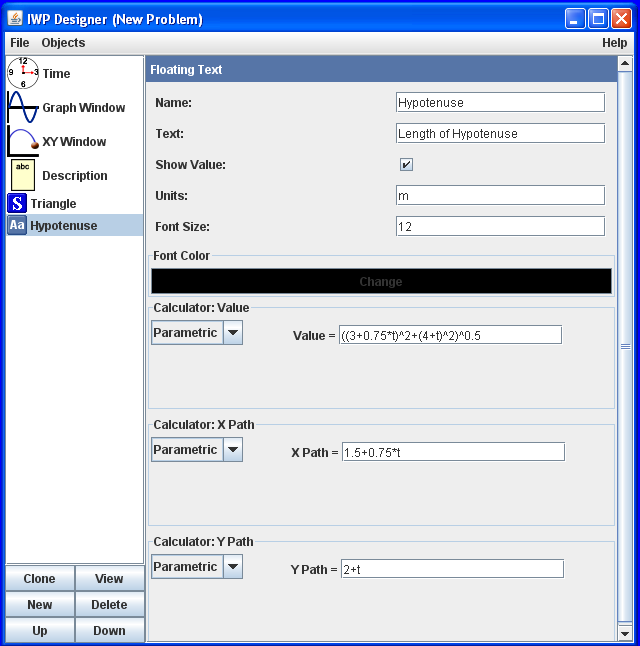
If IWP will not allow you to input all of these coordinates, try deleting the Pentagon, clicking and dragging to increase the size of your Editor window even more (in both the horizontal and vertical directions) and re-creating the Pentagon.

When you have inputted the coordinates, click View to see the Pentagon. Note that these vertices can also be made time-dependent as the Triangle’s are.

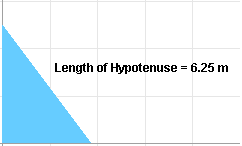


**Floating Text**

Delete the Pentagon, but keep the Triangle. Click “New” and select “FloatingText” (not “Solid”). Name this text “Hypotenuse” and give it the text “Length of Hypotenuse.” Do not uncheck the “Show Value” option. Input “m” for units and keep the font size as the default of 12. Define the value as “((3+0.75\*t)^2+(4+t)^2)^0.5,” the X Path as “1.5+0.75\*t,” and the Y Path as “2+t.” All together, your parameters should be like this:



Now click “View” and run the animation. Observe how the text moves with the Triangle and displays the length of the Hypotenuse every 0.1 seconds.



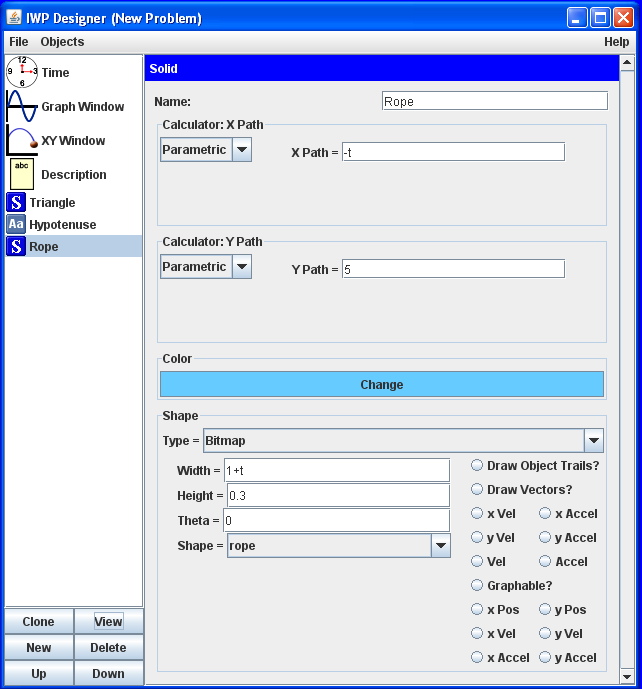
**Lines and Vectors**

Return to the Editor and create a new Solid. Set its Type to “Line.” For Lines, the Parametric Equations in the upper half of the Editor define the position of one end of the Line. The Width and Height parameters define the position of the other end, treating the former end as the origin. For example, if you want to make a line that spans from the point (4,4) to (5,5), set the X and Y Paths to 4 each, but set the Width and Height to 1 each. This means the line begins at (4,4) and then continues to a point that is 1 to the right and 1 above (4,4). As you may have deduced, this means that the Theta parameter does nothing to lines – their angle is always going to be the inverse tangent of the Height divided by the Width. As with other Solids, all four of the parameters that affect lines can be made time-dependent by defining them in terms of *t*.

Change the Type of your Solid to “Vector.” Notice that the available parameters are the same. Vectors are exactly the same as Lines except one end has an arrowhead indicating the direction of the Vector. In this case, it matters which end is defined as which. The base of the Vector is the point defined by the X and Y Paths, and the head is defined by the Width and Height.

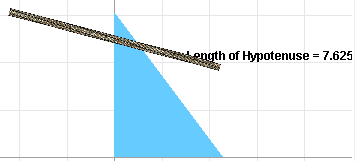
**Bitmaps**

Delete your Vector and create a new Solid. Set its Type to “Bitmap.” As soon as you do this, a new parameter appears, Shape. Set the Shape to “rope.” Name the Solid “Rope,” set its X Path to –*t*, Y Path to 3, Width to *t*+1, and Height to 0.3.

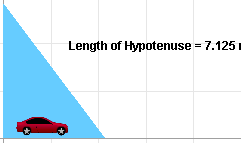


Now click View and play the animation. Notice how the Rope stretches horizontally with respect to time. The right end of the rope does not move (much). This is because the X Path has been defined as “-*t*,” which compensates for the stretching of the Rope.

Next, return to the Editor and set Theta to “pi/4+*t*.” Set the X Path to 0 and the Width to 4. Play the animation again. Notice how the Rope begins tilted at an angle of 45 degrees and spins with time.



Rename your Rope as “Car.” Set its Shape to “car\_final.” Set its X Path to t, Y Path to 0.5, Height and Width to 1 each, and Theta to 0. Run the animation and watch the Car drive along the X-axis.



**Step Function and Its Application to a Problem**

Now, you will combine many of the elements explained thus far into an interesting problem. Go ahead and delete what you’ve created thus far in this Module. Create a new Solid, and set its Type to “Bitmap.” The Shape, of course, is “car\_final.” Name it “Daredevil.” The parameters for this Solid are marvelously complex. The X Path is “step(5-*t*)\*(*t*-5)+step(*t*-5)\*(cos(theta)\**t*-5\*cos(theta))”. The Y Path is “step(*t*-5)\*(sin(theta)\**t*-5\*sin(theta))+0.3”. The Theta Parameter is “step(t-5)\*theta”. Note the distinction between “Theta” and “theta.” The parameters cannot be captured in a screen-take. You will just have to input them exactly as described.

The “step” function deserves an explanation. When the value *x* in the case of “step(*x*)” is **not greater than zero**, the step function simply acts as a **zero**. When *x* is **greater than zero**, the step function acts as a **one**. The key to using step is to multiply it by an expression. This way, that expression affects the Solid in a normal way when *x* is greater than zero, but has no effect at all when *x* is not greater than zero. In the case of the path of Daredevil, the Car will start at (-5,0.3). It will move at a rate of *t* to the right until *t* = 5. At that point, it will climb at an incline of theta, which is an Input you will create shortly. The magnitude of its velocity will be the same, but the direction will not.

Now create a new Input. Name it “theta” and give it the Text “Angle.” Let the initial value be 0.75 and the units, rads.

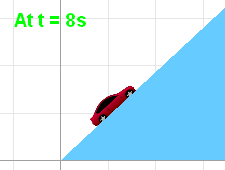
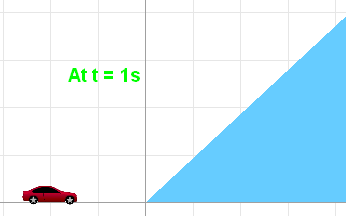
Create a new Solid. Make it a Polygon with the name “Ramp.” Leave the X and Y Paths as zero. Define the coordinates of the three points as follows:

Point 0: (0,0)

Point 1: (10,0)

Point 2: (10,tan(theta)\*10)

Click “View” to open the Animator and play the animation. Notice that the Daredevil drives up a rather steep incline. Students can adjust exactly how steep the incline is, and the Ramp will set itself to compensate automatically.



Adjust the value of theta and click “Reset.” Run the animation again. This can remain amusing for several repetitions.

**IWP Version 4 Appendix**

**Functions**

To see all of IWP’s functions in action, select File, Open, Packaged, Function Reference, and open TEST\_mathFunctions.iwp. Click View and play the animation. The animation simply shows a large number of Outputs that are all functions of time. In order of decreasing vertical position, the functions are:

*sign()* Returns -1 if the input is less than zero and 1 otherwise.

*exp()* Raises the mathematical constant *e* to the power of the input.

*toRadians()* Multiplies the input by π and divides by 180 degrees.

*toDegrees()* Multiplies the input by 180 degrees and divides by π.

*step()* Returns 1 if the input is positive and 0 otherwise. Returns 0 if the input is zero.

*cot()* Takes the cotangent of the input. Assumes an input in radians.

*sec()* Takes the secant of the input. Assumes an input in radians.

*csc()* Takes the cosecant of the input. Assumes an input in radians.

*tan()* Takes the tangent of the input. Assumes an input in radians.

*sqrt()* Takes the positive square root of the input. Returns nothing if the input is negative.

*sin()* Takes the sine of the input. Assumes an input in radians.

*cos()* Takes the cosine of the input. Assumes an input in radians.

*ln()* Takes the natural logarithm of the input. Returns negative infinity if the input is zero and nothing if the input is negative.

*atan()* Takes the arctangent of the input. Returns an output in radians.

*asin()* Takes the arcsine of the input. Returns an output in radians.

*acos()* Takes the arccosine of the input. Returns an output in radians.

*abs()* Takes the absolute value of the input.

*modtwo()* Divides the input by two to obtain an integer and returns the remainder.

*signum()* Returns -1 if the input is negative, 1 if the input is positive, and 0 if the input is zero.

*random()* Returns a random value between the value of the input and zero.

**Constants**

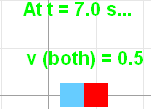
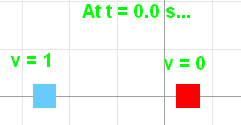
IWP has two constants built in which can be used in functions: *pi* and *e*. Objects created in IWP should not be given the same name as either of these constants, as IWP will not understand such a name. This issue can be circumvented by typing other characters besides the constants’ names first and then adding the troublesome characters afterwards. For example, if you try to name a Solid “Earth,” as soon as you type “E”, IWP will display an error message. However, if you type “arth” and then move the cursor back to the beginning of the word and type “E”, IWP will understand.

**Use of Step**

The following is a very simple inelastic collision problem which makes use of the step function. Create a new Solid and name it “Blue.” Leave its Y Path at 0 but set the X Path to “step(5-*t*)\*(*t*-5)+step(*t*-5)\*0.5\*(*t*-5)”. The first term in the path, “step(5-*t*)\*(*t*-5)”, causes the object to have an initial position of -5 and to progress to the right at a rate of 1 until *t*=5, at which point the object will be at the origin and step(5-*t*) will return 0. The second term, “step(*t*-5)\*0.5\*(*t*-5)”, causes the object to move to the right at a velocity of 0.5 starting at *t*=5 and continuing indefinitely. Prior to *t*=5, step(*t*-5) returns 0.

Next, create a new Solid and name it “Red.” Set its color to Red. Leave its Y Path at 0 but set the X Path to “1+step(*t*-5)\*0.5\*(*t*-5)”. The 1 simply gives the object an initial position. The “step(*t*-5)\*0.5\*(*t*-5)” causes the object to travel to the right at a velocity of 0.5 beginning at *t*=5 and continuing indefinitely afterward.

Now run the animation and observe as Blue collides into Red and the two continue at half of Blue’s original velocity.



Of course, this simple problem has little application. To see a more detailed collision problem which makes use of Step, select File, Open, Packaged, the directory ‘Momentum, Collisions, KE’, and open the file collision-inelastic-01.iwp. This is the same type of problem but much more complex, with variable Inputs.

**Use of Signum**

To demonstrate the use of the Signum function, create two rectangular solids on either side of the screen. Name them simply “Left” and “Right” and color them blue and red. Left should have an X Path of -10 and Right, 10. Both should have a height of 20, width of 1, and Y Path of 0. This is a parallel-plate capacitor.

Now create an Input named “v” for voltage. This will be the voltage on the Left plate; the Right plate shall be defined to have zero voltage. Give the Input an initial value of 200 and the units V. Create a third Solid and set the type to Vector. Make the color black, the height 0, theta 0, and Y Path 8. Set the X Path to -9.5\*signum(*v*) and the width to 19\*signum(*v*). Clone this object four times. Set the first clone to have a Y Path of 4, the second clone a Y Path of 0, the third, -4, and the fourth, -8.

Click on View to see the animation. The five Vectors now represent a uniform electric field. Change the voltage of the Left plate to a negative value and click Reset to see that the field lines switch direction.

**Use of Rand**

Rand is a pseudo-random generator that returns a real number between zero and whatever value is input to the function. It can be used to simulate random natural phenomena such as rainfall. To generate precipitation, first create a new Solid and name it “Drop.” Define its position not in Parametrics but in Euler. Set the initial X Displacement to “-10 + rand(20)”. There is no need for X Velocity or X Acceleration. Set the initial Y Displacement to “10+rand(20)” and the Y Velocity to -3. There is no need for Y Acceleration. Make the Drop a blue Circle of dimension 0.3 X 0.3. Clone the drop forty times. View the animation to see the rain fall.

