**Goals**

The goal of this lab is to learn the following:

* Create objects using Quorum libraries
* Get user input
* Use physics actions
* Use program to run simulation and write a report

**Computer Science Principles Curriculum**

* **Big Idea: Abstraction:**EK 2.1.1A, 2.1.2A, 2.2.1A, 2.2.1B, 2.2.1C, 2.2.2A, 2.2.2B, 2.2.3B, 2.2.3D, 2.2.3F, 2.3.1A, 2.3.1B, 2.3.1D, 2.3.2A, 2.3.2B, 2.3.2C, 2.3.2D, 2.3.2E, 2.3.2F, 2.3.2G, 2.3.2H
* **Big Idea:** **Data and Information:** EK 3.1.1A, 3.1.1D, 3.1.1E, 3.1.2A, 3.1.2B, 3.1.2C, 3.1.2D, 3.1.2E, 3.1.2F, 3.1.3A, 3.1.3B, 3.1.3C, 3.1.3D, 3.1.3E
* **Big Idea:** **Algorithms:**  EK 4.1.1A, 4.1.1B, 4.1.1E, 4.1.1F, 4.1.1G, 4.1.2A, 4.1.2B, 4.1.2C, 4.1.2G, 4.1.2H, 4.1.2I, 4.2.1A, 4.2.1B, 4.2.1C, 4.2.4A, 4.2.4B, 4.2.4C, 4.2.4F, 4.2.4H
* **Big Idea:** **Programming:** EK 5.1.1A, 5.1.1B, 5.1.1C, 5.1.1D, 5.1.2A, 5.1.2B, 5.1.2C, 5.1.2D, 5.1.2E, 5.1.2F, 5.1.2G, 5.1.2I, 5.1.2J, 5.1.3A, 5.1.3B, 5.1.3C, 5.1.3D, 5.1.3E, 5.1.3F, 5.2.1A, 5.2.1B, 5.2.1C, 5.2.1D, 5.2.1E, 5.2.1G, 5.2.1H, 5.2.1I, 5.2.1J, 5.2.1K, 5.3.1A, 5.3.1B, 5.3.1C, 5.3.1D, 5.3.1E, 5.3.1F, 5.3.1G, 5.3.1I, 5.3.1J, 5.3.1L, 5.3.1M, 5.3.1N, 5.3.1O, 5.4.1C, 5.4.1D, 5.4.1I, 5.4.1J, 5.4.1K, 5.5.1A, 5.5.1D, 5.5.1E, 5.5.1F,

**Common Core Standards**

* **English Language Arts Standards » Science & Technical Subjects:**CCSS.ELA-Literacy.RST.9-10.2, CCSS.ELA-Literacy.RST.9-10.3, CCSS.ELA-Literacy.RST.9-10.4, CCSS.ELA-Literacy.RST.9-10.5, CCSS.ELA-Literacy.RST.9-10.7, CCSS.ELA-Literacy.RST.9-10.10, CCSS.ELA-Literacy.RST.11-12.2, CCSS.ELA-Literacy.RST.11-12.3, CCSS.ELA-Literacy.RST.11-12.4, CCSS.ELA-Literacy.RST.11-12.5
* **Mathematics Content: High School Number and Quantity » Quantities:** CCSS Math Content.HSN.CED.A1, CED.A2
* **Mathematics Content: High School Algebra » Creating Equations:** CCSS Math Content.HSA.Q.A1, Q.A2, Q.A3
* **Mathematics Content: High School Functions » Building Functions:**CCSS.Math.Content.HSF.BF.A1
* **Mathematics Content: High School Statistics and Probability » Interpreting Categorical and Quantitative Data:** CCSS Math Content.HSS.ID.A1, ID.C7, ID.C8, ID.C9
* **Standards For Mathematical Practice:**CCSS.Math.Practice.MP1, CCSS.Math.Practice.MP2, CCSS.Math.Practice.MP4, CCSS.Math.Practice.MP5, CCSS.Math.Practice.MP6, CCSS.Math.Practice.MP7, CCSS.Math.Practice.MP8

**Vocabulary**

* Mass
* Gravity
* Objects

**Overview**

For this lesson, you will work with a partner to create a computer simulation to model the effects of gravity and mass on two objects that are the same size. You will use several physics concepts to make objects react correctly in the simulation. After completing the program, your team will use the simulation to collect data and write a report on your findings.

**Goal 1: Create Objects**

You will use the following libraries in this program: Libraries.Game.Game, Libraries.Sound.Audio, Libraries.Game.Graphics.Drawable, and Libraries.Game.Collision.Shapes.Circle. Remember that each of these libraries needs to be at the top of your code, and have the word use before them. You have used the Game, Audio, and Drawable libraries in programs before. There is a new library listed that uses some properties of the Collision class. This will be explained later in the lesson.

Once you have the use statements written, you will need to create several objects to be used in this program. These objects will be created in the main class, but not in an action, so they can be used in any action you wish.

**Example: Create a Drawable object called object1.**

Drawable object1

**Activity: Create other objects you will need**

Create Drawables called object2, cliff, top, and floor. Create two Audio objects called splat and plop. Create a Circle object called shape.

**Goal 2: Get user input**

To make your program easier to use as a simulation, you will ask the user for the mass of the two objects they wish to compare. This code will go in the Main action, before the StartGame() action. This will allow the program to gather the user input and use those variables before starting the simulation.

First, declare and instantiate variables for mass. These variables will be written under the objects, so they can be used in multiple actions. You will instantiate the variables to 0, but they will be overwritten by the user’s input. They can be instantiated to any value, but convention says to use 0.Bottom of Form

**Example: Declare and instantiate a number variable called mass1.**

number mass1 = 0

**Activity: Declare and instantiate the second mass variable.**

Name your second number variable mass2.

Now that you have the variables you need, you can request input from your user. Remember that all input comes in as a text. You will need to cast your input as a number.

**Example: Write an input statement for the mass of the first object**

Bottom of Form

mass1 = cast(number, input(“What is the mass of the first object?”)

**Activity: Write the input statement for the second object.**

Use mass2 for your variable.

If you run your program now, you should see the input statements, but nothing else, yet.

**Goal 3: Use physics statements**

The next step in the program loading images and using physics actions to make them respond to mass and gravity as if they were dropped in the real world. The rest of your code will be written in the CreateGame action.

Start by creating the floor and top of the screen. These two objects will help set up the room that the simulation is in. You will need to EnablePhysics2D for the program, as well as for each object on the screen, so they will react correctly. You will also need to SetGravity for the environment. Without gravity, the objects will not react correctly. You will make the floor and ceiling Unmovable, so that nothing can move them.

**Example: Write the code to add a ceiling to the room.**

You will use the Drawable object top for this.

EnablePhysics2D(true)

//Gravity takes both x and y parameters.

SetGravity2D(0, -100)

//Using GetScreenWidth() for the x will allow the box to fit, without having to figure out how wide the screen is.

//You can use any y height you want.

top:LoadFilledRectangle(GetScreenWidth(), 10)

/The y position is GetScreenHeight() – 6, so the box is set 6 pixels down from the top of the screen.

top:SetPosition(0, GetScreenHeight()-6)

top:EnablePhysics(true)

top:SetUnmovable()

//Don’t forget to add your object!

Add(top)

**Activity: Add a floor to the room.**

**Use the floor object to add a floor to your room. You will need to add one more physics action to this object. SetFriction() lets you set the level of friction an object has. This parameter is a number between 0 and 1. 0 is very little friction, like ice, while 1 is very coarse, like carpet.**

If you run your program now, you should have two long black boxes. One at the top of the screen, and one at the bottom.

Next, you will add the cliff that the other objects will hit. Here is the code to load the image and set its position. To set the cliff in the middle of the screen, you will get the width of the screen, subtract the width of the cliff (150 pixels), and divide by 2.

cliff:Load(“media/BrickWall.png”)

cliff:SetPosition((GetScreenWidth()-150)/2, 0)

**Activity: Use physics and add the cliff.**

**Use EnablePhysics() and SetFriction() to make the cliff react to being hit by an object. Then Add() the cliff.**

If you run your code now, you should see the ceiling and floor, and now a brick wall in the middle of the screen. Next, you will add the falling objects to the simulation. The code below sets up the first object. Comments are added to explain new actions used.

box:Load(“media/Lightning.png”)

box:EnablePhysics(true)

//The next two lines use the Circle object you made. This will make the lightning image react like a circle, even though the image file is a square.

circle:SetRadius(30)

box:SetShape(circle)

box:SetFriction(0.1)

//The next line will set the position, like will the cliff, you need a bit of math to make it line up with the edge of the cliff.

//You use -77 because to take the width of the cliff in account. It had to be moved slightly past half to make the objects fall off the cliff.

box:SetPosition(((GetScreenWidth()-60/2)-77, 450)

//Instead of Unmovable, set this object to Responsive, so it will move and react to physics.

box:SetRepsonsive()

//Restitution is the amount of bounce the object has. 0.5 will make the object return to half the height it dropped.

box:SetRestitution(0.5)

//In this program, the SetMass() action is using the variable from the user input.

box:SetMass(mass1)

box:SetFriction(0.1)

Add(box)

**Activity: Set up the second object.**

**Use the example above to set up the second object. Use the box2 object, and the media/Cork.png file.**

Now that you have all of your code written, you may want to make it accessible to users who cannot see your screen. Accessiblity code has already been added to your program in the Update action. It has been commented out with /\* and \*/. Remove these symbols to allow the code to run.

Run your program and see what happens!

**Goal 3: Use program to run a simulation and write a report.**

Your program will now give you visual, auditory, and numerical information from the simulation. Your team can use this information to make conclusions about the effect of gravity on items with different masses. You can use various methods to analyze your data.

**Next Tutorial…**