|  |  |
| --- | --- |
| http://compsci.appstate.edu/sites/compsci.appstate.edu/files/imagecache/slideshow/slideshow/ASU_compsci_logo.png**The CS4ALL NSF Supported Program** | https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9GcQGzOU-XT8XZWIBUwiPs2jjgixLO3CvrEyNq90lu1dbXJ0BQume[**https://cs.appstate.edu/cs4all/**](https://cs.appstate.edu/cs4all/) |

**Activity Title:** Modeling Projectile Motion in SNAP

**Introduction/Motivation:**

* This module is designed to introduce some of the more robust applications of SNAP. This module will help solidify physics ideas while improving on a student’s programming skills.

**Background:**

* To be successful, students should have a basic understanding of the mathematics behind the kinematic equations and the application of them to solve for position and instantaneous speed. A rudimentary understanding of programming and previous experience with SNAP will provide added benefits.

**Materials List:**

* SNAP available at <https://snap.berkeley.edu/snap/snap.html>
* “PMwithSNAP-Shell.xml” (An incomplete SNAP project.)

**Assessments:**

**Pre-Assessment:**

* Write the kinematic equations for the x and y direction.
* Solve a simple projectile motion problem given a known initial speed and angle.

**Preparation:**

* Access SNAP from <https://snap.berkeley.edu/snap/snap.html>

**Lab Activity:**

1. Read through this entire section so you understand the problem you are being presented with. You should be thinking of a solution as the problem is revealed in subsequent steps. Answer the preliminary questions before trying to “fill in the blanks.” After you have done that; proceed to actually do step 2.
2. Open SNAP.
3. Open the “PMwithSNAP-Shell.xml” file.



1. Your window should look similar to this now.



1. You should notice there are four places where code is missing. You are tasked with filling in these blanks.
2. These blanks are going to be where you initialize your variables. The section under the Green flag is for when the program first runs. The section under the “receive” function is to reset after a missed shot. Hint: The code you will be adding should be the same for both sections.



1. This blank is missing two lines that initialize your speed variables.



1. This last blank is where you will calculate and update your position variables. You should also include a bit of code to check to see if the girl manages to make a basket. Hint: You will need to update all this in a loop.



1. Once your code is complete and it is working you should save the file with a new name and submit it to the instructor for grading.

**Preliminary Questions:**

1. In Step 6, you are asked to initialize variables. What variables will you need to include? Think about the fact that you are moving in two directions. Do you have velocity in both directions? Do you have acceleration in both directions? What will you need to break a velocity into components?
2. What do you need to do in the initialize routines to make sure you can pass the speed and angle checks?
3. In step 7 you need at least two lines to set up your initial speed in each direction. What equations will help you do this?
4. Step 8 requires several things.
* First, you need to update position according to the kinematic equations.
* Next, you need to update a variable that moves things along.
* Lastly, you need a bit of code to see if the girl has reached the goal. If she has, it needs to set the “goal?” variable to true, “1”.
* All of this should be included in a loop that updates at a repeats a reasonable number of times in a reasonable amount of time.

**Post Completion Questions:**

1. What is the shape of motion that the basketball player takes?
2. What direction is the positive x direction? What direction is the positive y direction?
3. Consider the input needed to score. Is this reasonable from any player?
4. Based on your answer from number 3, would this be a good simulator to model actual basketball shots?

**Programming Tips and Challenges:**

1. Save. Often.
2. When working on Step 8, watch out for infinite loops; for an example, see Tip 3.
3. When working on Step 8, watch out for infinite loops; for an example, see Tip 2.
4. Be wary of your inputs for speed and angle. Look carefully at what the code will check for as far as goofy inputs and what it won’t. For an added challenge you can fix the code to only accept numeric inputs.
5. You’ll notice that the basketball player can travel off screen. Add some code that resets her to her initial position and starts the simulation over.