PHY 1200: Physics for Video Games

with GlowScript VPython

Programs and Games

Aaron Titus

High Point University

Spring 2016

http://bit.ly/phy1200

Contents

1   PROGRAM – Introduction to GlowScript and VPython

Apparatus

Computer
GlowScript – www.glowscript.org

Goal

The purpose of this activity is to write your first program in a language called Python. We will use the web app GlowScript that converts Python to JavaScript so the program can run in a web browser. GlowScript provides the same functions available in the Python module called Visual. Together Python and Visual are named VPython. As a result, GlowScript can be considered the web-based version of VPython. You will probably read or hear the terms GlowScript and VPython used interchangeably; however, there are some important differences. I tend to think of the language as VPython (Python + Visual) and the web app as GlowScript.

We use VPython because it allows you to do vector algebra and to create 3D objects in a 3D scene. The capability of 3D graphics with vector mathematics makes it a great tool for simulating physics phenomena. In this activity, you will learn:

- how to use GlowScript, the web-based integrated development editor (IDE) for writing and running VPython.

- how to structure a simple computer program in VPython.

- how to create 3D objects such as spheres and arrows.

Setup

Go to http://www.glowscript.org/ and create an account. You will need a Google account because GlowScript uses your Google account for authentication. After logging in, you will see a link to "your programs are here." Click this link to enter the IDE.

Procedure

Creating folders and files

1. Once you log in and follow the link to your programs, you are in the GlowScript IDE. Click the **Add Folder** tab to create a new folder. A pop-up window appears as shown in Figure 1.1. Because I must run your programs, make the folder public. Name it "phy1200" if you wish.

2. With the folder name highlighted orange (showing you are in the folder), click the link **Create New Program** and name the program intro .

Starting a program: Setup statements

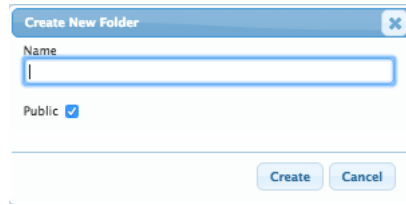3. Notice GlowScript types the first line of the program for you.

Figure 1.1: Create a new folder in GlowScript.

```
GlowScript 2.1 VPython
```

Every GlowScript program begins with this setup statement. It tells GlowScript you are writing VPython code. The version number (2.1 in this case) is included to ensure that older programs will continue to run without errors, even if GlowScript is updated to newer versions.

4. Also, notice there is no "save" menu. Like Google Docs, GlowScript automatically saves your program as you are typing it.

Creating an object

5. As your first VPython command, let's make a sphere. Skip a line in order to make your code more readable, and on line 3, type:

```
sphere()
```

This statement tells the computer to create a sphere object.

6. Run the program by clicking **Run this program** . GlowScript exits the edit mode and enters the run mode. You should see a white sphere on a black background like Figure 1.2. This is called the `scene` .
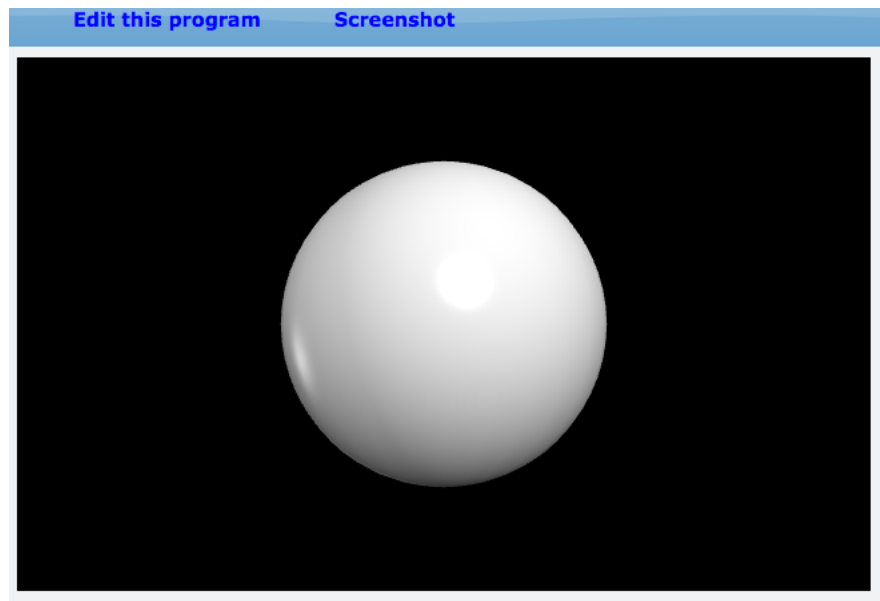

Figure 1.2: Your first VPython program–a sphere.

The 3-D graphics scene

By default the sphere is at the center of the scene, and the "camera" (your point of view) is looking directly at the center.

7. If you are on a PC or have a two-button mouse, hold down both mouse buttons and move the mouse forward and backward to make the camera move closer or farther away from the center of the scene. On a Mac, hold down the option key and mouse button while moving the mouse forward and backward. This is how you *zoom* in VPython. A scroll wheel also zooms in and out.

8. Hold down the right mouse button alone and move the mouse to make the camera "revolve" around the scene, while always looking at the center. On a Mac, in order to rotate the view, hold down the Control key while you click and drag the mouse. This is how you *rotate* the scene in VPython. Because this is a sphere, you won't notice a significant change except for lighting.

   By default, when you first run the program, the coordinate system is defined with the positive x direction to the right, the positive y direction pointing up toward the top edge of the monitor, and the positive z direction coming out of the screen toward you. You can then rotate the camera view to make these axes point in other directions relative to the camera.

Error messages: Making and fixing an error

GlowScript tells you when there is a syntax error in your program. (Logic errors are much more difficult to fix!) To see an example of an error message, let's try making a spelling mistake.

9. Click **Edit** to return to editing mode, and change line 3 of the program to the following:

```
phere()
```

10. Run the program.

    There is no function or object in VPython called `phere()`. As a result, an error message pops up. The message gives the *approximate* line number where the error occurred and a description of the error, as shown in Figure 1.3.
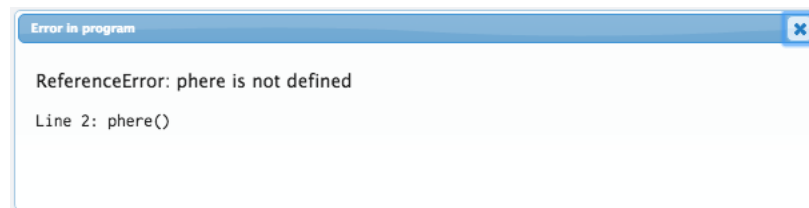
Figure 1.3: An error message in GlowScript.

The line number may be off but is usually close.

11. Correct the error in the program by clicking **Edit this program** and returning to the editor. Once in editing mode, you can click the ⊠ to close the error message.

    There are two types of errors: (1) syntax errors which might be a typing or coding mistake and (2) programmatic errors so the program runs correctly but does something other than what you intended. The error message helps you find the first of these. Finding errors that cause a program to act differently than you intended is much more difficult and is a skill you will develop in this course.

Changing attributes (position, size, color, shape, etc.) of an object

Now let's give the sphere a different position in space and a radius.

12. Change line 3 of the program to the following:

```
sphere(pos=vector(-5,2,3), radius=0.40, color=color.red)
```

13. Run the program. Experiment with other changes to `pos` , `radius` , and `color` . Run the program each time you change an attribute.

14. Answer the following questions:

> What does changing the `pos` attribute of a sphere do?
>
>
> What does changing the `radius` attribute of a sphere do?
>
>
> What does changing the `color` attribute of a sphere do? What colors can you use? You can try `color=vector(1,0.5,0)` for example. The numbers stand for RGB (Red, Green, Blue) and can have values between 0 and 1. Can you make a purple sphere? Note that colors such as cyan, yellow, and magenta are defined, but not all possible colors are defined. Choose random numbers between 0 and 1 for the (Red, Green, Blue) and see what you get.

Autoscaling and units

VPython automatically zooms the camera in or out so all objects appear in the window. Because of this autoscaling, the numbers for the `pos` and `radius` can be in any consistent set of units, like meters, centimeters, inches, etc. For example, this could represent a sphere with a radius 0.20 m at the position $(2, 4, 0)$ m. In this course we will often use SI units in our programs ("Systeme International", the system of units based on meters, kilograms, and seconds).

Creating a box object

Another object we will often create is a box. A box is defined by its position, axis, length, width, and height as shown in Figure 1.4.
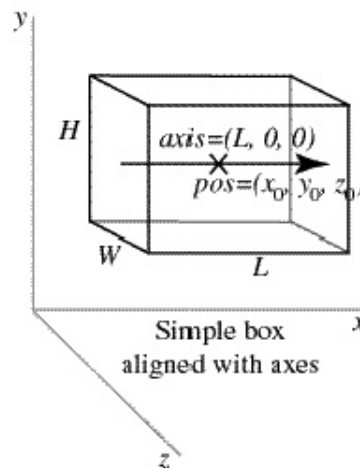
Figure 1.4: Attributes of a box. (Image from `http://www.glowscript.org/docs/VPythonDocs/box.html`)

15. Type the following on a new line, then run the program:

```
box(pos=vector(0,0,0), size=vector(2,1,0.5), color=color.orange)
```

The length, width, and height of the box are expressed as a vector with the attribute:
`size=vector(L,H,W)` .

16. Change the length to 4 and rerun the program.

17. Now change its height and rerun the program.

18. Similarly change its width and position.

> Which dimension (length, height, or width) should be changed to make a box longer along the y-axis? Change your code now to check your answer.
>
>
> What point does the position of the box refer to?
> (a) the center of the box
> (b) one of its corners
> (c) the center of one of its faces
> (d) some other point

Comment lines (lines ignored by the computer)

Comment lines start with a # (pound sign). A comment line can be a note to yourself, such as:

```
# units are meters
```

Or a comment can be used to remove a line of code temporarily, without erasing it.

19. Put a # at the beginning of the line creating the box, as shown below.

```
#box(pos=vector(0,0,0), size=vector(2,1,0.5), color=color.orange)
```

20. Run the program. What did you observe?

21. Uncomment this line by deleting the # and run the program again. The box now appears.

Naming objects; Using object names and attributes

We will draw a tennis court and will change the position of a tennis ball.

22. Clean up your program so it contains only the following objects:

A green box that represents a tennis court. Make it 78 ft long, 36 ft wide, and 4 ft tall. Place its center at the origin.

An orange sphere (representing a tennis ball) at location $\langle -28, 5, 8 \rangle$ ft, with radius 1 ft. Of course a tennis ball is much smaller than this in real life, but we have to make it big enough to see it clearly in the scene. Sometimes we use unphysical sizes just to make the scene pretty.

(Remember, you don't type the units into your program. But rather, you should use a consistent set of units and know what they are.)

23. Run your program and verify that it looks as expected. Use your mouse to rotate the scene so you can see the ball relative to the court. Your program should look like the one below.

```
1  GlowScript 1.1 VPython
2
3  box(pos=vector(0,0,0), size=vector(78,4,36), color=color.green)
4
5  sphere(pos=vector(-28, 5, 8), radius=1, color=color.orange)
```

24. Change the position of the tennis ball to $\langle 0, 6, 0 \rangle$ ft.

25. Run the program.

26. Sometimes we want to change the position of the ball after we defined it. Thus, give a name to the sphere by changing the `sphere` statement in the program to the following:

```
tennisball=sphere(pos=vector(0, 6, 0), radius=1, color=color.orange)
```

We've now given a name to the sphere. We can use this name later in the program to refer to the sphere. Furthermore, we can specifically refer to the attributes of the sphere by writing, for example, `tennisball.pos` to refer to the tennis ball's position attribute, or `tennisball.color` to refer to the tennis ball's color attribute. To see how this works, do the following exercise.

27. Start a new line at the end of your program (perhaps line 7) and type:

```
print(tennisball.pos)
```

28. Run the program.

29. Look at the text below the 3D scene. The printed vector should be the same as the tennis ball's position.

30. Add a new line to the end of your program (perhaps line 9) and type:

```
tennisball.pos=vector(32,7,−12)
```

When running the program, the ball is first drawn at the original position but is then drawn at the last position. (Note: whenever you set the position of the tennis ball to a new value in your program, the tennis ball will be drawn at that position.) This may happen so quickly that you do not notice the tennis ball drawn at the two locations.

31. Add a new line to the end of your program (perhaps line 11) and type:

```
print(tennisball.pos)
```

(Or just copy and paste your previous print statement.)

32. Run your program. It now draws the ball, prints its position, redraws the ball at a new position, and prints its position again. As a result, you should see the following two lines printed:

```
<0, 6, 0>
<32, 7, -12>
```

Of course, this happens faster than your eye can see it which is why printing the values is so useful.

Analysis

All games with graphics include objects on the screen. The game programmer must specify the positions and dimensions (sizes) of the objects using 2D or 3D vectors.

C Do all of the following. You are going to create objects for the game *Frogger*. We will only use spheres and boxes for this part.

1. Click the link to your username to return to your folders.

2. If necessary, click the phy1200 folder. Create a new blank file and name it *frogger-C*.

3. Create a green box for the frog that is at the location $< 0, -100, 0 >$, has a length=10, height=10, and width=10 units. Name the box `frog`.

4. Create a yellow sphere for a lily pad at $< -60, 100, 0 >$ with a radius of 10. Name the sphere `lilypad`.

5. Create a blue box for the water that is at the location $< 0, 0, -10 >$, has a length=150, height=220, and width=10 units. Name the box `water` .

6. Rotate the scene. Is the lily pad inside the water or on top of the water? Is the frog inside the water or on top of the water?

7. Is physics used in this program? Why does the frog not move in this program?

B Do everything for **C** along with the following modifications and additions.

1. Return to your phy1200 folder and create a new blank file and name it *frogger-B*.

2. Copy from your previous program (labeled C) and paste it into this program. Often, this is the fastest way to start a new program.

3. Create another yellow sphere for a lily pad at $< 60, 100, 0 >$ with a radius of 10. Name the sphere `lilypad4` .

4. In between these two lily pads, create two more named `lilypad2` and `lilypad3` so the lily pads are equally spaced.

5. Create a gray road that is exactly half the height of the water. It should extend from the middle of the blue box to the bottom end of the blue box.

6. Create a long cyan box on the left side of the road and a short magenta box on the right side of the road, between the frog and the water. Name them `car1` and `car2` .

7. Print the positions of the cars and the frog.

Figure 1.5 is an example program that fits the criteria for a B.
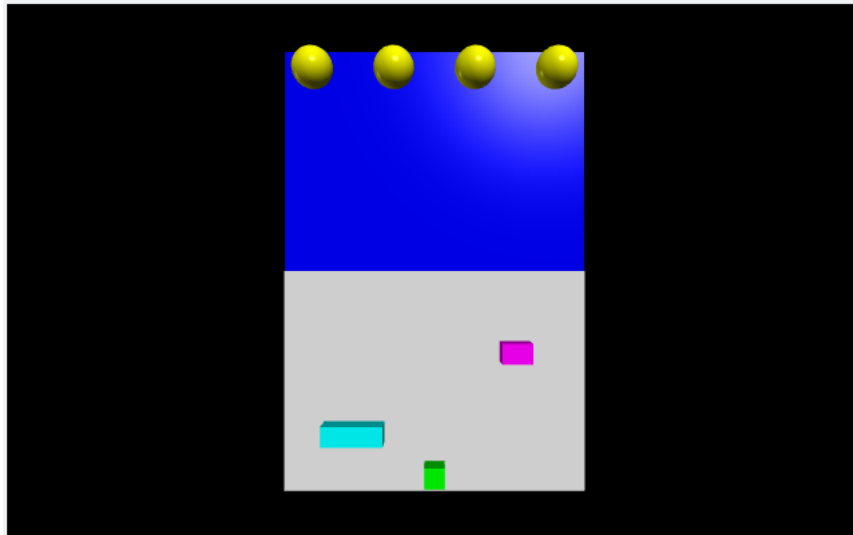
Figure 1.5: The scene required for a B.

A Do everything for **B** along with the following modifications and additions.

1. Return to your phy1200 folder and csreate a new blank file and name it *frogger-A*.

2. Copy from your previous program (labeled B) and paste it into this program.

3. In the top right corner of the GlowScript window, click the link to **Help**. This opens the documentation window. Click the menu to **Choose a 3D object** and view the list of objects shown in Figure 1.5.

4. Select the cylinder and read how to create a cylinder.
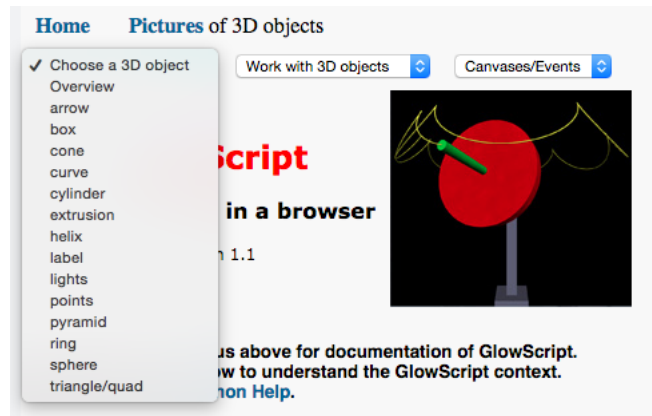
Figure 1.6: GlowScript documentation

5. Change the lily pads so they are thin cylinders that appear to float on top of the water.

6. Use the cylinder object to create 3 logs of different lengths in the water.

7. Now, click the menu to **Work with 3D objects** in the documentation and select **Materials/-Textures**. Read how to specify a texture. You will probably want to click the link to the example program that demonstrates the pre-defined textures.

8. Change the three wooden logs so they use the wood texture.

Figure 1.7 is an example program that fits the criteria for program A.
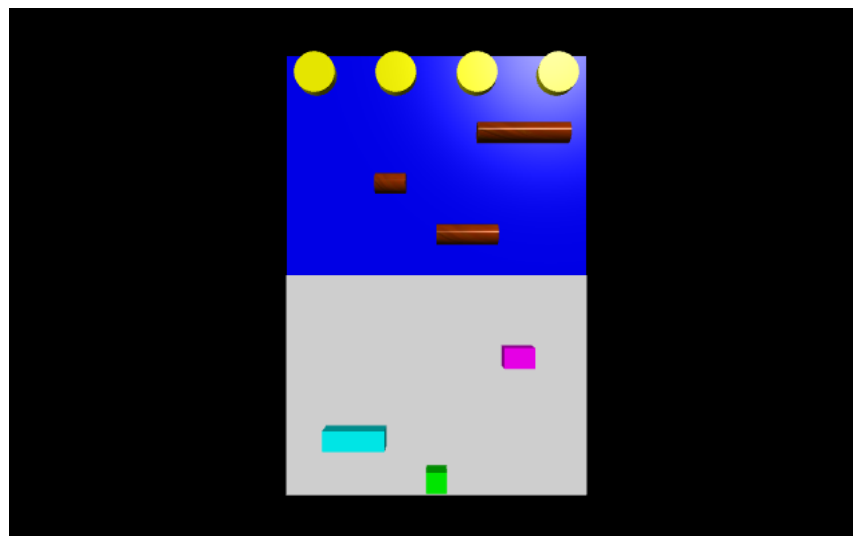
Figure 1.7: The scene required for an A.

2   PROGRAM – Uniform Motion

Apparatus

Computer
GlowScript – www.glowscript.org

Goal

The purpose of this activity is to learn how to use GlowScript (VPython) to model uniform motion (i.e. motion with a constant velocity).

Introduction

General structure of a program

In general, every program that models the motion of physical objects has two main parts:

1. **Before the loop**: The first part of the program tells the computer to:

   (a) Create 3D objects.
   (b) Give them initial positions and velocities.
   (c) Define numerical values for constants we might need.

2. **The `while` loop**: The second part of the program, the loop, contains the lines that the computer reads to tell it how to update the positions of the objects over and over again, making them move on the screen.

To learn how to model the motion of an object, we will write a program to model the motion of a ball moving with a constant velocity.

Procedure

Before you begin, it will be useful to look back at your notes or a previous program to see how you created a sphere and box.

1. Create a new program in GlowScript and save this file with a new name like `ball-uniform-motion`.

2. Add the line below to create a track that is at the origin and has a length of 3 m, a height of 0.05 m, and a width of 0.1 m. Note that the y-position is -0.075 m (below zero) so that we can place a ball at $y = 0$ such that it appears to be on top of the track..

   ```
   track=box(pos=vector(0,−0.075,0), size=vector(3,0.05,0.1), color=color.
       white)
   ```

3. Create a ball (i.e. sphere) at the position $(-1.4, 0, 0)$ m. Choose its radius to be an appropriate size so that the ball appears to be on top of the track and name it `ball`.

Figure 2.1: A ball on a track.

4. Run your program. The ball should appear to be on the top of the track and should be on the left side of the track as shown in Figure 2.1.

Now, we will define the velocity of the ball to be to the right with a speed of 0.3 m/s. A unit vector that points to the right is (1,0,0). So, the velocity of the ball can be written on paper as:

$$\begin{aligned} \vec{v} &= |\vec{v}|\,\hat{v} \\ &= 0.3 * (1,0,0) \end{aligned}$$

Next we will see how to write this in VPython.

5. Just as the position of the ball is referenced as `ball.pos` , let's define the ball's velocity as `ball.v` which indicates that v is a property of the object named `ball` . To do all of this, type this line at the end of your program.

```
ball.v=0.3*vector(1,0,0)
```

This statement creates a property of the ball `ball.v` that is a vector quantity with a magnitude 0.3 that points to the right.

6. Whenever you want to refer to the velocity of the ball, you must refer to `ball.v` . For example, type the following at the end of your program.

```
print(ball.v)
```

7. When you run the program, it will print the velocity of the ball as a 3-D vector as shown below:

```
<0.3, 0, 0>
```

Define values for constants we might need

To make an object move, we will update its position every Δt seconds. In general, Δt should be small enough such that the displacement of the object is small. The size of Δt also affects the speed at which your program runs. If it is exceedingly small, then the computer has to do lots of calculations just to make your object move across your screen. This will slow down the computer.

8. For now, let's use 1 hundredth of a second as the time step, Δt. At the end of your program, define a variable dt for the time interval.

```
dt=0.01
```

9. Also, let's define the total time t for the clock. The clock starts out at $t = 0$, so type the following line.

```
t=0
```

That completes the first part of the program which tells the computer to:

(a) Create the 3D objects and name them.
(b) Give the ball an initial position and velocity.
(c) Define variable names for the clock reading t and the time interval dt.

Create a "while" loop to continuously calculate the position of the object.

We will now create a `while` loop. Each time the program runs through this loop, it will do two things:

    (a) Calculate the displacement of the ball and add it to the ball's previous position in order to find its new position. This is known as the "position update".

    (b) Calculate the total time by incrementing t by an amount dt through each iteration of the loop.

    (c) Repeat.

10. For now, let's run the animation for 10.0 s. On a new line, begin the `while` statement as shown below. This tells the computer to repeat these instructions as long as $t < 10.0$ s.

```
while t < 10.0:
```

Make sure that the `while` statement ends with `:` because Python uses this to identify the beginning of a loop.

To understand what a while loop does, let's update and then print the clock reading.

11. Below the `while` statement, add the following line. Note that it must be indented.

```
t=t+dt
```

After adding this line, your `while` loop will look like:

```
while t < 10.0:
        t=t+dt
```

Note that this line takes the clock reading t, adds the time step dt, and then assigns the result to the clock reading. Thus, through each pass of the loop, the program updates the clock reading.

12. Print the clock reading by typing the following line at the end of the while loop (again, make sure it's indented) and run your program.

```
print(t)
```

After adding the `print` statement, check that your `while` loop looks like:

```
while t < 10.0:
        t=t+dt
        print(t)
```

13. Run the program. View the clock readings printed below the 3D scene.

14. You can make it run indefinitely (i.e. without stopping) by saying "while true" so you can change the `while` statement to read:

```
while 1:
        rate(100)
```

The `rate(100)` statement tells the computer to try to run the loop 100 times in one second.

Check that your program (loop) now looks like:

```
while 1:
        rate(100)
        t=t+dt
        print(t)
```

15. Run the program. Now, it will print clock readings continually until you click the **Edit this program** link and return to the editor.

> Stop and reflect on what is going on in this `while` loop. Your understanding of this code is essential for writing games.

Just as we updated the clock using `t=t+dt` , we also want to update the object's position. Physics tells us that the object's new position is given by:

$$\text{new position coordinates} = \text{current position coordinates} + \text{velocity} \times \text{time step}$$
$$\vec{r}_f = \vec{r}_i + \vec{v}\Delta t$$

This is called the *position update equation.* It says, "take the current position of the object, add its displacement, and the result is the new position of the object." In VPython the "=" sign is an *assignment* operator. It takes the result on the right side of the = sign and assigns its value to the variable on the left.

Now we will update the ball's position after each time step `dt` .

16. Inside the `while` loop *before you update the clock*, update the position of the ball by typing:

```
ball.pos=ball.pos+ball.v*dt
```

After typing this line, check that your `while` loop looks like:

```
while 1:
        rate(100)
        ball.pos=ball.pos+ball.v*dt
        t=t+dt
        print(t)
```

17. Change the print statement to print both the clock reading and the position of the ball. Separate the variables by commas as shown:

```
print(t, ball.pos)
```

18. Run your program. You will see the ball move across the screen to the right. Because we have an infinite loop, it will continue to move to the right until you return to the editor. After the ball travels past the edge of the track, the camera will zoom backward to keep all of the objects in the scene.

19. Printing the values of the time and the ball's position may slow down the computer. Comment out your print statement by typing the # sign in front of the `print` statement (as in `#print`) . Run your program again.

    Sometimes you need to print data in order to check the computer's calculations. However, it can also be distracting and unnecessary. In general, print when you need to check the computer's calculation and debug your program. Otherwise, don't print.

20. Adjust the `rate` statement and try values of 10 or 200, for example. How does increasing or decreasing the argument of the rate function affect the animation?

    The `rate(100)` statement specifies that the while loop will not be executed more than 100 times per second, even if your computer is capable of many more than 100 loops per second. (The way it works is that each time around the loop VPython checks to see whether $1/100$ second of real time has elapsed since the previous loop. If not, VPython waits until that much time has gone by. This ensures that there are no more than 100 loops performed in one second.)

    If you want time to advance in the simulation at the same rate as a real clock (meaning, as nearly as possible, the simulation time is equal to real time), then set the values of `dt` and `rate()` so the product is equal to one. For example, if `dt=0.01` , then choose `rate(100)` because $0.01 * 100 = 1$. Or if `dt=0.02` , then choose `rate(50)` .

Analysis

C Do all of the following.

    1. Start a new program in GlowScript and save this file with a new name like `ball-uniform-motion-C`.

    2. Copy the code from your previous program and paste it into this new program.

    3. Simulate the motion of a ball that starts on the right end of the track and travels to the left with a speed of 0.5 m/s for 5 s. The ball's initial position should be $(1.5, 0, 0)$ m. The `while` loop should run while $t < 5$ s. Print the time and position of the ball.

B Do everything for **C** and the following.

    1. Start a new program in GlowScript and save this file with a new name like `ball-uniform-motion-B`.

    2. Create two balls on a track: Ball A starts on the left side at $(-1.5, 0, 0)$ m and Ball B starts on the right side at $(1.5, 0, 0)$ m. Name them `ballA` and `ballB` in your program.

    3. Ball A travels to the right with a speed of 0.3 m/s and Ball B travels to the left with a speed of 0.5 m/s. Define each of their velocities as `ballA.v` and `ballB.v`, respectively.

    4. Set the `while` loop to run while $t < 5$ s.

    5. Print the clock reading `t` and the position of each ball up to $t = 5$ s.

    6. At what clock reading t do they pass through each other?

A Do everything for **B** and the following.

    1. Start a new program in GlowScript and save this file with a new name like `ball-uniform-motion-A`.

    2. Create a similar track as before, but with the width as 3 m. When the scene is rotated, the track appears as a table top.

    3. Create three balls that all start at $x = -1.5, y = 0$; however, stagger their z-positions so that one travels down the middle of the table, one travels down one edge of the table, and the other travels down the other edge of the table. Name them `ballA`, `ballB`, and, `ballC`, respectively, and give them different colors.

    4. Set the x-velocities of the balls to: (A) 0.25 m/s, (B) 0.5 m/s, and (C) 0.75 m/s.

    5. At what time does Ball C reach the end of the table? (Use a print statement to determine this.)

    6. What are the positions of all three balls when Ball C reaches the end of the table? (Use a print statement to determine this.)

3   PROGRAM – Lists, Loops, and Ifs

Apparatus

Computer
GlowScript – www.glowscript.org

Goal

The purpose of this activity is to learn how to use lists, `for` loops, and `if` statements in VPython.

Introduction

When writing a game, you will typically have multiple objects moving on a screen at one time. As a result, it is convenient to store the objects in a list. Then, you can loop through the list and for each object in the list, update the position of the object.

Procedure

Before you begin, it will be useful to look back at your notes or a previous program to see how you create objects such as spheres and boxes and how you make objects move. The instructions in this chapter do not repeat the VPython code that you learned in previous activities. Have those chapters and programs available for reference as you do this activity.

1. Create a new program and save it with a name like `move-objects.py`.

   The `for` loop and the `range()` list

2. Type the `for` loop shown below.

```
for i in range(0,10,1):
    print(i)
```

3. Save and run your program. The program should print:

```
0
1
2
3
4
5
6
7
8
9
```

   In order to see all of the digits, you may have to scroll in the text box or click and drag on the bottom right corner of the box to expand it.

The statement `range(0,10,1)` creates a list of numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The `for` loop goes through this list, one item at a time, starting with the first item. For each *iteration* through the loop, it executes the code within the loop, but the value of `i` is replaced with the item from the list. Thus for the first iteration of the loop, the value of `i` is 0. Then for the next iteration of the loop, it has the value 1. The loop continues until it has accomplished 10 iterations and `i` has taken on the values of 0 through 9, respectively. Note that the number 10 is not in the list.

4. Change the arguments in the `range(0,10,1)` function. Change 0 to 5, for example. Or change 1 to 2. You can even change the 1 to –1 to see what this does. Run the program each time you change one of the arguments and figure out how each argument affects the resulting list. Write your answers below.

In the function `range(0,10,1)`, how does changing each argument affect the resulting list of numbers?

0:

10:

1:

---

5. Delete the entire `for` loop for now, and we'll come back to it later.

Lists

When writing games, you may have a lot of moving objects. As a result, it is convenient to store your objects in a list. Then you can loop through your list and move each object or check for collisions, etc.

6. To show how this works, first create 4 balls that are all at $x = -5, z = 0$. However, give them y values that are $y = -3, y = -1, y = 1, y = 3$, respectively. Name them `ball1`, `ball2`, etc. Give them different colors and make their radius something that that looks good on the screen.

7. Run your program to verify that you have four balls at the given locations. The screen should look like Figure 3.1 but perhaps with a black background and different color balls.

8. Define the balls' velocity vectors such that they will all move to the right but with speeds of 0.5 m/s, 1 m/s, 1.5 m/s, and 2 m/s. Remember that to define a ball's velocity, type:

```
ball1.v=0.5*vector(1,0,0)
```

You'll have to do this for all four balls. Be sure to change the name of the object and speed. You should have four different lines which specify the velocities of the four balls.

Now we will create a list of the four balls. VPython uses the syntax: `[item1, item2, item3,...]` to create a list where item1, item2, etc. are the list items and the square brackets [ ] denote a list. These items can be integers, strings, or even objects like the balls in this example.

9. To create a list of the four balls, type the following line at the end of your program.

```
ballsList = [ball1, ball2, ball3, ball4]
```

Notice that the names of the items in our list are the names we gave to the four spheres. The name of our list is `ballsList`. We could have called the list any name we wanted.

Motion

We are going to make the balls move. Remember, there are three basic steps to making the objects move.

- Define variables for the clock and time step.
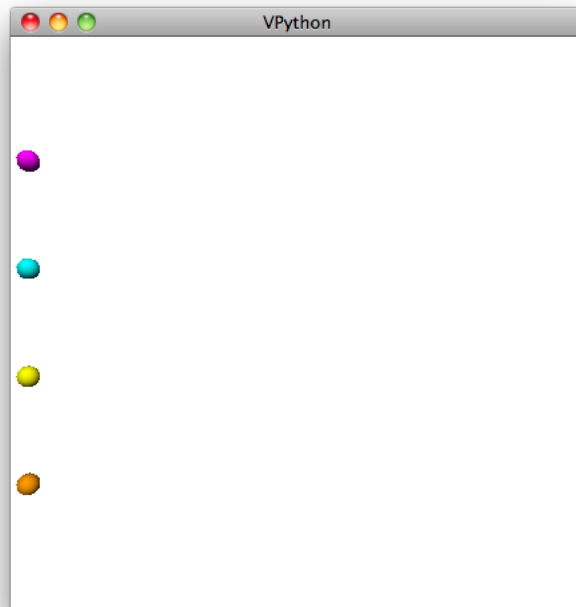
Figure 3.1: Four balls

- Create a `while` loop.
- Update the object's position and update the clock reading.

10. Define variables for the clock and for the time step.

```
t=0
dt=0.01
```

11. Create an infinite `while` loop and use a `rate()` statement to slow down the animation.

```
while 1:
    rate(100)
```

12. We are now ready to update the position of each ball. However instead of updating each ball individually, we will use a `for` loop and our list of balls. Type the following loop to update the position of each ball. Note that it should be indented.

```
    for thisball in ballsList:
        thisball.pos=thisball.pos+thisball.v*dt
```

This loop will iterate through the list of balls. It begins with `ball1` and assigns the value of `thisball` to `ball1`. Then, it updates the position of `ball1` using its velocity. On the next iteration, it uses `ball2`. After iterating through all objects in the list, it completes the loop. And at this point it has updated the position of each ball.

13. Now update the clock. Your while loop should ultimately look like the following:

```
while 1:
    rate(100)
    for thisball in ballsList:
```

```
        thisball.pos=thisball.pos+thisball.v*dt
    t=t+dt
```

Note that the line `t=t+dt` is indented beneath the `while` statement but is not indented beneath the `for` loop. As a result, the clock is updated upon each iteration in the `while` loop, not the `for` loop. The for loop merely iterates through the balls in the ballsList.

Using a `for` loop in this manner saves you from having to write a separate line for each ball. Imagine that if you had something like 20 or 50 balls, this would save you a lot of time writing code to update the position of each ball.

14. Run your program. You should see the four balls move to the right with different speeds.

15. When a ball reaches the right side of the window, the camera will automatically zoom out so that the scene remains in view. In game, we wouldn't want this. Therefore, let's set the size of our window and tell the camera not to zoom. Near the beginning of your program, perhaps on line 2 or 3, add the following lines:

```
scene.width=500
scene.height=500
scene.range=5
scene.autoscale=False
```

The height and width attributes set the size (in pixels) of the scene. The range attribute of `scene` sets the right edge of the window at $x = +5$ and the left edge at $x = -5$. The `autoscale` attribute determines whether the camera automatically zooms to keep the objects in the scene. We set autoscale to `False` in order to turn it off. Set it to `True` if you want to turn on autoscaling.

16. Run your program.

IF statements

We are going to keep the balls in the window. As a result, our code must check to see if a ball has left the window. If it has, then reverse the velocity. When you need to check *if* something has happened, then you need an `if` statement.

Let's check the x-position of the ball. If it exceeds the edge of our window, then we will reverse the velocity. If the x-position of a ball is greater than $x = 5$ or is less than $x = -5$, then multiply its velocity by -1. Though we can write this with a single `if` statement, it might make more sense to you if we use the *if-else* statement. The general syntax is:

```
if condition1 :
    indentedStatementBlockForTrueCondition1
elif condition2 :
    indentedStatementBlockForFirstTrueCondition2
elif condition3 :
    indentedStatementBlockForFirstTrueCondition3
elif condition4 :
    indentedStatementBlockForFirstTrueCondition4
else:
    indentedStatementBlockForEachConditionFalse
```

The keyword "elif" is short for "else if". There can be zero or more `elif` parts, and the `else` part is optional.

17. After updating the velocity of each ball inside the `for` loop, add the following `if-elif` statement:

```
        if thisball.pos.x>5:
            thisball.v=-1*thisball.v
        elif thisball.pos.x<-5:
            thisball.v=-1*thisball.v
```

Note that it should be indented inside the `for` loop because you need to check each ball in the list.

After inserting your code, your `while` loop should look like:

```python
while 1:
    rate(100)
    for thisball in ballsList:
        thisball.pos=thisball.pos+thisball.v*dt
        if thisball.pos.x>5:
            thisball.v=-1*thisball.v
        elif thisball.pos.x<-5:
            thisball.v=-1*thisball.v
    t=t+dt
```

18. Run your program. You should see each ball reverse direction after reaching the left or right edge of the scene.

Analysis

C Do all of the following.

1. Start with your program from this activity and save it as a different name.

2. When a ball bounces off the right side of the scene, change its color to yellow.

3. When a ball bounces off the left side of the scene, change its color to magenta.

B Do everything for **C** and the following.

1. Create a new program and give it a different name.

2. Create 10 balls that move horizontally and bounce back and forth within the scene. Make the scene 10 units wide and give the balls initial positions of $x = -10$, and $z = 0$, but with y positions that are equally spaced from $y = 0$ to $y = 9$. Give them different initial velocities. Make their radii and colors such that they can be easily seen but do not overlap.

A Do everything for **B** with the following modifications and additions.

1. Create a new program and give it a different name.

2. Copy your program in part (B) and paste it into your new program.

3. Set the initial velocity of each ball to be identical. Give them the same speed, but set their velocities to be in the $-y$ direction.

4. When a ball reaches the bottom of the scene ($y = -10$), change its velocity to be in the $+x$ direction. When a ball reaches the right side of the scene change its velocity to be in the $+y$ direction. When a ball reaches the top of the scene, change its velocity to be in the $-x$ direction. Finally, when it reaches the left side of the scene, change its velocity to be in the $-y$ direction. In this way, make the balls move around the edge of the scene.

5. Run your program. You might find that the balls do not move as you expect. The reason is that if you update a ball's position and it just barely goes out of the scene, then you need to move the ball back within the scene. For example, in the python code below, if the ball's position is updated and it goes past the right edge of the scene at $x = 10$, then the line within the IF statement moves the ball one step backward, back into the scene again. In other words, it reverses the position update statement. (Note the negative sign.)

```
thisball.pos=thisball.pos+thisball.v*dt
if thisball.pos.x>10:
    thisball.pos=thisball.pos-thisball.v*dt
```

You need to make sure that in each `if` or `elif` statement where you check that the ball is at the edge of the screen, you move the ball back to its previous position.

4 PROGRAM – Keyboard Interactions

Apparatus

Computer
GlowScript – www.glowscript.org

Goal

The purpose of this activity is to incorporate keyboard and mouse interactions into a VPython program running in GlowScript.

Procedure

Using the keyboard to set the velocity of an object

1. Open the program from *PROGRAM–Lists, Loops, and Ifs* of the four balls bouncing back and forth within the scene. We will use this program as our starting point. If you did not do this exercise, then the code for the program is shown below.

```
GlowScript 2.0 VPython

scene.width=500
scene.height=500
scene.range=5
scene.autoscale=False

ball1=sphere(pos=vector(-5,3,0), radius=0.2, color=color.magenta)
ball2=sphere(pos=vector(-5,1,0), radius=0.2, color=color.cyan)
ball3=sphere(pos=vector(-5,-1,0), radius=0.2, color=color.yellow)
ball4=sphere(pos=vector(-5,-3,0), radius=0.2, color=color.orange)

ball1.v=0.5*vector(1,0,0)
ball2.v=1*vector(1,0,0)
ball3.v=1.5*vector(1,0,0)
ball4.v=2*vector(1,0,0)

ballsList = [ball1, ball2, ball3, ball4]

t=0
dt=0.01

while 1:
    rate(100)
    for thisball in ballsList:
        thisball.pos=thisball.pos+thisball.v*dt
        if thisball.pos.x>5:
            thisball.v=-1*thisball.v
        elif thisball.pos.x<-5:
```

```
              thisball.v=−1*thisball.v
     t=t+dt
```

2. Above your `while` loop, create a box that is at the position $(-4.5, -4.5, 0)$. Name it `shooter` and make its width, length, and height appropriate units so that it looks like it is sitting at the bottom left corner of the window.

3. Run your program and verify that the box is of correct dimensions and is in the left corner of the screen without appearing off screen.

4. Define the velocity of the box to be to the right with a speed of 2 m/s. Name it `shooter.v`.

5. Inside the `while` loop, after the `for` loop updates the positions of the balls, add a line to update the position of the shooter as shown below.

```
     shooter.pos = shooter.pos + shooter.v*dt
```

6. Run your program. You should see the shooter move to the right, and it will continue moving off the screen. If this does not occur, then check for errors of logic in your program.

   Now we want to use the keyboard to control the velocity of the box. We will use the following strategy:

   - Look to see if a key is pressed.
   - Check to see which key is pressed.
   - If the right-arrow is pressed, set the velocity of the shooter to be to the right.
   - If the left-arrow is pressed, set the velocity of the shooter to be to the left.
   - If any other key is pressed, set the velocity of the shooter to be zero.
   - Move the box.

   Moving the box occurs inside the while loop. However, we need the GlowScript environment to continually monitor whether a key has been pressed on the keyboard. Then, when the key is pressed, our code will take over by checking which key it is and setting the velocity of the shooter.

7. On line 3 of your program, immediately after the "GlowScript 2.0 VPython" statement, write the following function.

```
def keyboard(event):
    if event.type=='keydown':
        k = event.which
        print(k)

scene.bind('keydown', keyboard)
```

   Let me explain what this code is doing. GlowScript continually monitors for keyboard and mouse events. The `scene.bind('keydown', keyboard)` function tells VPython that it should call the *keyboard* function whenever a *keydown* event is detected. The *keyboard* function is a custom-defined function. We could have named it anything. (I picked the name *keyboard* just because it made sense to me.) In this function, I first check to see what `type` of event occurred. If it is *keydown* then I get the key and print it.

8. Run your program. Press various keys and note the number that is printed.

What are the numbers that correspond to these keys?
j:
k:
l:
a:
s:
d:
spacebar:
left arrow:
right arrow:
up arrow:
down arrow:

In the previous code, we printed the key because we wanted to see the unique number for each key. But now we want to use the keyboard to control the velocity of the box. Let's set the velocity of the box to be to the right, if the right arrow is pressed, and to the left, if the left arrow is pressed.

9. Comment out the `print` statement since you already figured out the numbers that correspond to the arrow keys.

10. Change the `keyboard` function to be the following:

```
def keyboard(event):
    if event.type=='keydown':
        k = event.which
#        print(k)
        if k == 39:
            shooter.v=2*vector(1,0,0)
        elif k == 37:
            shooter.v=2*vector(-1,0,0)
        else:
            shooter.v=vector(0,0,0)
```

The function *keyboard* checks to see which key is pressed and sets the velocity accordingly.

11. Run your program. Press various keys to see if the program works as expected.

12. Change your program so that pressing `a` causes a fast leftward velocity and pressing `s` causes a fast rightward velocity. In summary, the left and right arrows create slow velocities to the left and right; the `a` and `s` keys create fast velocities to the left and right.

13. Run your program and verify that it works as expected.

14. You probably noticed that it's annoying when the box moves past the edge of the edge of the screen. Use an `if` statement in the `while` loop to check if the box passes the edge of the screen. If it does, then reverse its velocity. The best way to reverse the velocity is to multiply it by -1 as shown below.

```
shooter.v=-shooter.v
```

15. Run your program. The box should reverse, with the same speed, whenever it reaches the edge of the screen. Furthermore, you should be able to set the velocity (fast or slow) of the box using right arrow, left arrow, a, or s, and stop the box with any other key.

Using the keyboard to create a moving object

We are now going to use the keyboard to launch bullets from our shooter. We need another list where we can store the bullets. Before the `while` loop, create an empty list called `bulletsList`.

```
bulletsList=[ ]
```

16. In your `if` statement where you check for keyboard events, add the following `elif` statement to check for the spacebar.

```
        elif k==32:
            bullet=sphere(pos=shooter.pos, radius=0.1, color=color.white)
            bullet.v=3*vector(0,1,0)
            bulletsList.append(bullet)
```

Study this section of code and know what each line does. If you press the spacebar, a white sphere is created at the center of the shooter. Its name is assigned to be `bullet`. Then, its velocity is set to be in the $+y$ direction with a speed of 3 m/s. Finally, and this is really important, the bullet is added (i.e. appended) to the end of the `bulletsList`. Later, in the `while` loop, we can update the positions of all the bullets in this list.

17. Now we have to update the positions of the bullets (i.e. make them move). In your `while` statement before you update the clock, add a `for` loop that updates the positions of the bullets in the bulletsList.

```
    for thisbullet in bulletsList:
        thisbullet.pos=thisbullet.pos+thisbullet.v*dt
```

Your final `while` loop should look like this:

```
while 1:
    rate(100)
    for thisball in ballsList:
        thisball.pos=thisball.pos+thisball.v*dt
        if thisball.pos.x>5:
            thisball.v=-thisball.v
        elif thisball.pos.x<-5:
            thisball.v=-thisball.v

    shooter.pos = shooter.pos + shooter.v*dt
    if(shooter.pos.x>5):
        shooter.v=-shooter.v
    elif(shooter.pos.x<-5):
        shooter.v=-shooter.v

    for thisbullet in bulletsList:
        thisbullet.pos=thisbullet.pos+thisbullet.v*dt

    t=t+dt
```

You should study this code and know what each line means. There are three important sections. One section updates the positions of the balls and reverses their velocities if they reach the edge of the screen. The next section updates the position of the shooter and reverses its velocity if it reaches the edge of the screen. The last section updates the positions of the bullets.

18. Run your program and verify that all aspects work as expected.

Analysis

C Do all of the following.

1. Create a new file and give it an appropriate name. Copy and paste previous code to do the following tasks.

2. Assign variables to the speed of the shooter (both slow and fast) and the speed of the bullet. Call them: `sfast`, `sslow`, and `sbullet`. Write these at the top of your program since you will use them later in the program.

3. When setting the velocity of the shooter in your keyboard function, use the variable for the speed of the shooter. Here is an example:

```
if k == 39:
    shooter.v=sslow*vector(1,0,0)
```

4. Set `sslow` to be very low, like 0.5. And set `sfast` to be very fast, perhaps 5. Also change `sbullet`. See how changing these values affects the game. By using variables, it makes it much easier to change their values for the purpose of gameplay. If you do not use variables, then you have many lines to change if you want to test higher or lower speeds.

5. Give instructions about your game by adding this code at line 2 or 3 of your program, just after the line that says, `GlowScript 2.0 VPython`. These statements add text to the `title` of the scene and will appear above the scene. HTML tags like `<b>` are needed to format the text.

```
scene.title.append('<h2>Instuctions</h2>')
scene.title.append('<br><br>')
scene.title.append('Use the following keys to control the shooter.')
scene.title.append('<br> -- Right arrow - slow, to the right')
scene.title.append('<br> -- Left arrow - slow, to the left')
scene.title.append('<br> -- s - fast, to the right')
scene.title.append('<br> -- a - fast, to the right')
scene.title.append('<br> -- spacebar - shoot a bullet')
scene.title.append('<br> -- any other key - stop')
```

B Do everything for **C** and the following.

1. Create a new file and give it an appropriate name. Copy and paste previous code to do the following tasks.

2. It looks strange for the bullets to come from the center of the box. Fire the bullets from the center of the top plane of the box instead of its center. To do this, you'll have to change the initial position of the bullet when it is created.

3. Check to see if the up arrow key is pressed or the down arrow key is pressed. If one of these keys is pressed, set the velocity of the shooter to be up or down, respectively.

4. Add additional keystrokes that will fire a bullet to the left, to the right, or downward. You may wish to use the arrow keys to fire bullets and other keys for changing the velocity of the shooter. Feel free to reassign keys to whatever makes sense. Change the instructions at the top to match the keys you choose.

A Do everything for **B** with the following modifications and additions.

1. Create a new file and give it an appropriate name. Copy and paste previous code to do the following tasks.

2. Add a counter variable called `shots` and set shots=0 before your `while` loop. Update the value of `shots` and print the value of `shots` every time a bullet is fired. To do this, you must add the following line immediately after the defining the keyboard function with **def** keyboard(). It should look like the lines shown below.

```
def keyboard(event):
        global shots
```

This line makes the shots variable, which was defined outside the function, available within the function.

3. Suppose that the shooter only has 10 bullets. Write code so that if the shooter reaches a maximum of 10 bullets, hitting the spacebar will no longer file a bullet.

4. Create a keystroke that will replenish the shooter, meaning that after hitting this keystroke, you can fire 10 more bullets.

5   PROGRAM – Collision Detection

Apparatus

Computer
GlowScript – www.glowscript.org

Goal

The purpose of this activity is to detect collisions between moving objects. You will learn to create a function, and you will learn about boolean variables that are either `True` or `False` .

Introduction

The idea of collision detection is a fairly simple one: *check to see if two objects overlap.* If their boundaries overlap, then the objects have collided.

Distance between spheres

Suppose that two spheres have radii R_1 and R_2, respectively. Define the center-to-center distance between the two spheres as d.
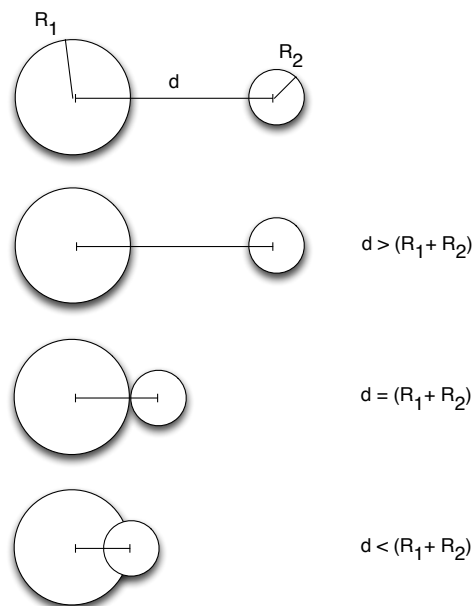
Figure 5.1: Condition for whether two spheres collide.

As shown in Figure 5.1:

if $d > (R_1 + R_2)$ the spheres do not overlap.

if $d < (R_1 + R_2)$ the spheres overlap.

if $d = (R_1 + R_2)$ the spheres exactly touch. Note that this will never happen in a computer game because calculations of the positions of the spheres result in 16-digit numbers (or more) that will never be exactly the same.

If the spheres are at coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2), then the distance between the spheres is:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

This is the magnitude of a vector that points from one sphere to the other sphere, as shown in Figure 5.2.
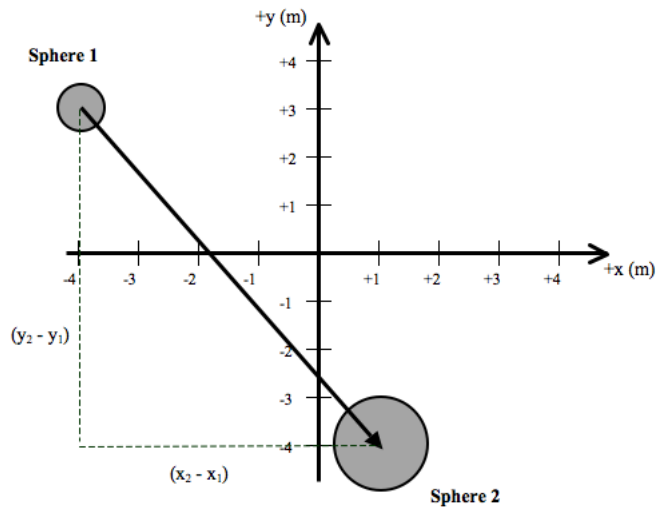
Figure 5.2: Distance between two spheres.

Because we only want the magnitude of the vector from one sphere to the other, it does not matter which sphere you call Sphere 1. Thus, you can just as easily calculate the distance using:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Because you square the vector's components, the sum of the squares of the components will always be positive.

Exercises

Ball1 is at $(-3, 2, 0)$ m and has a radius of 0.05 m. Ball2 is at $(1, -5, 0)$ m and has a radius of 0.1 m. What is the distance between them?

Ball1 is at $(1, 2, 0)$ m and has a radius of 0.05 m. Ball2 is at $(1.08, 1.88, 0)$ m and has a radius of 0.1 m. What is the distance between them? At this instant, have the balls collided?

Procedure

Starting program

1. Begin with the program that you wrote in *Chapter 9 PROGRAM – Keyboard Interactions*. It should have a shooter (that moves horizontally and shoots missiles) and four balls that move horizontally and bounce back and forth within the window.

   If you do not have that program, type the one shown below.

```
GlowScript 2.0 VPython

def keyboard(event):
    if event.type=='keydown':
        k = event.which
#           print(k)
        if k == 39:
            shooter.v=2*vector(1,0,0)
        elif k == 37:
            shooter.v=2*vector(-1,0,0)
        elif k == 65:
            shooter.v=4*vector(-1,0,0)
        elif k == 83:
            shooter.v=4*vector(1,0,0)
        elif k==32:
            bullet=sphere(pos=shooter.pos, radius=0.1, color=color.white)
            bullet.v=3*vector(0,1,0)
            bulletsList.append(bullet)
        else:
            shooter.v=vector(0,0,0)

scene.bind('keydown', keyboard)

scene.width=500
scene.height=500
scene.range=5
scene.autoscale=False

ball1=sphere(pos=vector(-5,3,0), radius=0.2, color=color.magenta)
ball2=sphere(pos=vector(-5,1,0), radius=0.2, color=color.cyan)
ball3=sphere(pos=vector(-5,-1,0), radius=0.2, color=color.yellow)
ball4=sphere(pos=vector(-5,-3,0), radius=0.2, color=color.orange)

ball1.v=0.5*vector(1,0,0)
ball2.v=1*vector(1,0,0)
ball3.v=1.5*vector(1,0,0)
```

```
ball4.v=2*vector(1,0,0)

ballsList = [ball1, ball2, ball3, ball4]

shooter=box(pos=vector(-4.5,-4.5,0), width=1, height=1, length=1, color=
    color.red)
shooter.v=2*vector(1,0,0)

t=0
dt=0.01

bulletsList=[ ]

while 1:
    rate(100)
    for thisball in ballsList:
        thisball.pos=thisball.pos+thisball.v*dt
        if thisball.pos.x>5:
            thisball.v=-thisball.v
        elif thisball.pos.x<-5:
            thisball.v=-thisball.v

    shooter.pos = shooter.pos + shooter.v*dt
    if(shooter.pos.x>5):
        shooter.v=-shooter.v
    elif(shooter.pos.x<-5):
        shooter.v=-shooter.v

    for thisbullet in bulletsList:
        thisbullet.pos=thisbullet.pos+thisbullet.v*dt

    t=t+dt
```

Defining a function

When you have to do a repetitive task, like check whether each missile collides with a ball, it is convenient to define a function. This section will teach you how to write a function, and then we will write a custom function to check for a collision between two spheres.

A function has a *signature* and a *block*. In the signature, you begin with `def` and an *optional parameter list*. In the block, you type the code that will be executed when the function is called.

2. To see how a function works, type the following code near the top of your program after the `GlowScript` statement, perhaps on line 2 or 3.

```
def printDistance(object1, object2):
    distance=mag(object1.pos-object2.pos)
    print(distance)
```

This function accepts two parameters named `object1` and `object2`. It then calculates the distance between the objects by finding the magnitude of the difference in the positions of the objects. (Note that `mag()` is also a function. It calculates the magnitude of a vector.) Then, it prints the distance to the web page.

3. Inside of and at the end of the `while` loop, call your function to print the distance between a ball and the shooter by typing this line. Now each iteration through the loop, it will print the distance between the shooter and `ball1`.

```
        printDistance(shooter, ball1)
```

4. Run the program. You will notice that it prints the distance between the shooter and `ball1` after each iteration (each time step) of the loop.

5. Change your program to print the distance between `ball1` and `ball4` and run your program.

   Note that you didn't have to reprogram the function. You just changed the parameters sent to the function. This is what makes functions such a valuable programming tool.

   Many functions return a value or object. For example, the `mag()` function returns the value obtained by calculating the square root of the sum of the squares of the components of a vector. This way, you can write `distance=mag(object1.pos-object2.pos)`, and the variable `distance` will be assigned the value obtained by finding the magnitude of the given vector. To return a value, the function must have a `return` statement.

6. You can delete the `printDistance` function and the `printDistance` statement because will not use them in the rest of our program.

7. Near the top of your program, after the `GlowScript 2.0 VPython` statement, write the following function. It determines whether two spheres collide or not.

```
def collisionSpheres(sphere1, sphere2):
    dist=mag(sphere1.pos-sphere2.pos)
    if(dist<sphere1.radius+sphere2.radius):
        return True
    else:
        return False
```

   Study the logic of this function. Its parameters are two spheres, so when you call the function, you have to give it the names of two spheres. The function then calculates the distance between the spheres. If this distance is less than the sum of the radii of the spheres, the function returns `True`, meaning that the spheres indeed collided. Otherwise, it returns `False`, meaning that the spheres did not collide.

   This function will only work for two spheres because we are comparing the distance between them to the sum of their radii. Detecting collisions between boxes and spheres will come later.

8. Inside the `for` loop that updates the position of the *bullet*, add the following lines:

```
        for thisball in ballsList:
            if collisionSpheres(thisbullet, thisball):
                thisball.pos=vector(0,-10,0)
                thisball.v=vector(0,0,0)
```

   After adding these lines, the bullet `for` loop will look like this: (You should not need to type this, just compare it to your program.)

```
    for thisbullet in bulletsList:
        thisbullet.pos=thisbullet.pos+thisbullet.v*dt
        for thisball in ballsList:
            if collisionSpheres(thisbullet, thisball):
                thisball.pos=vector(0,-10,0)
                thisball.v=vector(0,0,0)
```

   For each bullet in the bulletsList, the program updates the position of the given bullet and then loops through each ball in the ballsList. For each ball, the program checks to see if the given bullet collides with the given ball. If they collide, then it sets the position of the ball to be below the scene at $y = -10$, and it sets the velocity of the ball to be zero. If they do not collide, nothing happens because there is no `else` statement.

9. Run your program. You will notice that when a bullet hits a ball, the ball disappears from the scene. Note that it is technically still there, and the computer is still calculating its position with each time step. It is simply not in the scene, and its velocity is zero. If you zoom outward, you will see the balls. (They are drawn on top of each other, so you might only see one of them.)

Analysis

We now have the tools to make a game. In a future chapter you will have the freedom to create a game of your choice based on what we've learned. However, in these exercises, you will merely add functionality to this program to make it a more interesting game.

C Create a new file. Copy and paste your program from this lesson. Add all of the following features.

1. If a bullet exits the scene (i.e. `bullet.pos.y > 5`), set its velocity to zero.

2. Create a variable called `hits` and add one to this variable every time a missile hits a sphere. Remember, you increment a variable like the example shown below.

   $$hits = hits + 1$$

3. Print `hits` every time a missile hits a ball.

B Create a new file. Copy and paste your program from **C**. Add the following features.

1. Make 10 balls that move back and forth on the screen and set their y-positions to be greater than $y = 0$ so that they are all on the top half of the screen.

2. Add a variable called `shots` and increment this variable every time a bullet is fired.

A Create a new file. Copy and paste your program from **B**. Add the following features.

1. The score should not be simply based on whether a bullet hits a ball, but it should also be based on how many missiles are needed. For example, if you hit all four balls with only four bullets shot, then you should get a higher score. Also, if you hit all four balls with only four bullets shot in only 1 s, then you should get a higher score than if it required 10 s. Design a scoring system based on bullets fired, hits, and time. This will require a mathematical function of your choosing that gives you the desired outcome. Write your scoring system below. Describe the goals of your scoring system, how points are awarded or subtracted, and write a mathematical function that either computes the score as a function of shots, hits, and time or write a function that updates the score whenever a shot, hit, or time changes.

2. Program your scoring system into the code. Use a variable `score` for the total score. Use a `print()` statement to update the score every time it changes.

3. After you are confident that it is working, write down your top 5 scores.

4. Ask at least three friends to play the game one or more times and write down the top score(s) by each friend.

5. What would you like to change about your scoring system or game based on the experience of your friends?

6   GAME – Pong

Apparatus

Computer
GlowScript – www.glowscript.org

Goal

The purpose of this activity is to study the classic arcade game Pong and use GlowScript VPython to develop both a physical and an unphysical version of the game.

Procedure

Playing Pong

1. Go to `http://www.ponggame.org/` and play the classic Pong Game. Try both the keyboard and mouse to control the paddle.

Pay attention to the motion of the puck when colliding with the wall and a paddle.

1. Is the collision of the puck and a side wall elastic or inelastic? Explain your answer by referring to observations of the motion of the puck.

2. Are the side walls frictionless or not? Explain your answer by referring to observations of the motion of the puck.

3. Is the collision of the puck and a paddle elastic or inelastic? Explain your answer by referring to observations of the motion of the puck.

4. Is the paddle frictionless or not? Explain your answer by referring to observations of the motion of the puck.

Creating a "bouncing" puck in a box

We are going to simulate a puck on an air hockey table that is bouncing around the table. In this simulation, we will assume that the walls and paddle are rigid, frictionless barriers. We will also assume

that the puck and barriers make elastic collisions. Thus, the COR is 1 for all collisions. For simplicity, we will draw the puck as a puck and refer to it as a puck.

2. Begin a new program. Import the visual package.

3. Set the size of the scene. You may want to set the height and width of the window in pixels. The example below will set the range to be 10 (meters or whatever unit you wish you use), the width to be 600 pixels, and the height to be 600 pixels.

```
scene.range=20
scene.width=600
scene.height=600
```

4. Create walls at the top, bottom, and sides of the screen.

5. Run your program and verify that you have four walls around the perimeter.

6. Create a puck at the center using the cylinder object. For a cylinder, the axis determines the length (or height) and orientation of the cylinder. In this case, we want a top view of the cylinder so the axis points in the $+z$ direction.

```
puck=cylinder(pos=vector(0,0,0), axis=vector(0,0,0.1), radius=0.5, color=
    color.white)
```

7. Define the initial velocity of the puck (`puck.v=vector(5,8,0)`), the initial clock reading (`t=0`), and the time step (`dt=0.01`).

8. Create an infinite while loop.

9. Use `rate(100)` to slow down the simulation so that the motion is smooth.

10. Update the position of the puck.

```
    puck.pos=puck.pos+puck.v*dt
```

11. Use an `if-elif` statement to check for a collision between the puck and each wall. If there is a collision, change the velocity of the puck in an appropriate way.

12. Run your program and verify that it works properly.

Creating inelastic collisions

In the real world, a puck would lose energy upon colliding with a rigid barrier. Said another way, the coefficient of restitution is always less than 1. Now we will change the last program by adding friction and a coefficient of restitution.

13. Create a new program with a different name and copy all of your code to this new program. This way, you are saving your previous work as a reference.

14. Make the left wall a "real wall" that causes the puck to lose speed upon colliding with the wall. In other words, after colliding with the left wall, the puck's perpendicular velocity component would be reduced by a factor less than 1. Define a variable COR which you can change to be whatever value you want (between 0 and 1). A COR of 0 means that the puck will not bounce off the wall. A COR of 1 is an elastic collision. A COR greater than one is superelastic, like a bumper in pinball.

Since the left wall is vertical, then the perpendicular component of the puck's velocity is its x component.

```
    puck.v.x = –COR*puck.v.x
```

It helps to use a smaller COR, like 0.5 or less, to notice the effect after one collision.

<div style="border:1px solid black; padding:10px;">
Describe the motion of the puck after a long time. Could we have predicted this given the fact that only one wall results in inelastic collisions?

</div>

Suppose that a wall has a spring in it that "punches" the puck during the collision, similar to bumpers in a pinball machine. Then, you could model this wall by giving it a COR greater than 1.

15. Make one of the walls "super elastic" by giving it a COR greater than 1. (This is like the bumper in a pinball machine.)

16. Run your program and observe the effect of the collisions on the motion of the puck.

Adding friction to a collision

Friction acts parallel to the wall in order to reduce the parallel component of the velocity of the puck.

17. Create a new program with a different name and copy all of your code to this new program. This way, you are saving your previous work as a reference.

18. Start by make all collisions elastic collisions (i.e. $COR = 1$).

Now, we will add friction to the left wall by changing the y-component of the velocity of the puck when it collides with the wall. Exactly how friction affects the velocity of the puck is a bit complicated. Let's use a simple (albeit unphysical) model that reduces the parallel component of the velocity of the puck by a certain percentage. This is similar to the COR for the perpendicular component of the velocity.

19. When the puck collides with the left wall, change the y-component of the velocity by a factor of 20% or something like that. (A factor of 1 is no friction and a factor of 0 is maximum friction. 20% is a factor of 0.2.)

```
puck.v.y=0.2*puck.v.y
```

20. Run your program.

<div style="border:1px solid black; padding:10px;">
Describe the motion of the puck after a long time. Could we have predicted this given the fact that only one wall has friction?

</div>

Making a 1-player Pong game

21. Create a new program. We will start with a blank page.

22. It might be nice to set the width and height of the window in pixels. Use the code below to set the range to 20 (m or whatever units you want to imagine), the width to 600 pixels, and the height to 450 pixels. You are welcome to use a larger width and height if you wish.

```
scene.range=20
scene.width=600
scene.height=450
```

23. Create walls for the ceiling and floor. Also create a wall on the left side that has a hole in it that represents a goal. This is similar to what you see in air hockey, for example. You'll need two boxes on the left side, with a space between them for the goal.

24. Create small box as a paddle on the right side. You will eventually use your mouse to move this box up and down.

25. Create a puck and make its initial velocity something like (15,12,0) m/s.

26. Create variables for the clock and time step.

    A this point, there is no motion, and your scene should look something like Fig. 6.1.
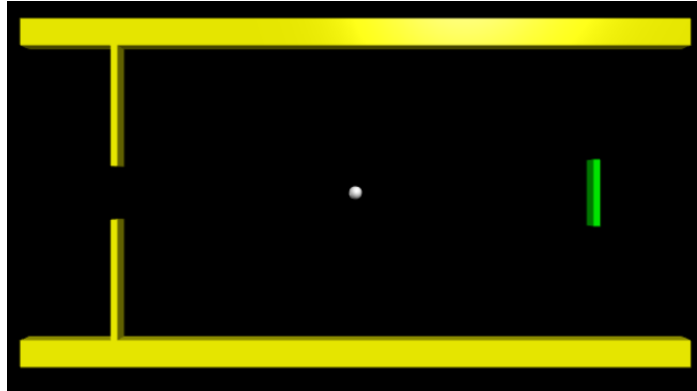

Figure 6.1: The initial scene of our version of Pong.

27. Create an infinite while loop. Update the position of the puck. To check for collisions with the walls and paddle, set the initial velocity of the puck so it collides off the various objects. For now, assume elastic, frictionless collisions. Run your program, check every wall and the paddle, and verify that everything works as expected.

    Controlling the paddle

    Think about how you want to control the paddle. You can use the up and down arrow on the keyboard, click and drag with the mouse, or simply hover with the mouse and move the paddle up and down in sync with the mouse as the mouse moves up and down. All of these ideas are possible, but with different levels of difficulty and experience required. We will explore these options.

    Controlling the paddle with the keyboard

28. Create a new program with a different name and copy all of your previous code to this new program. This way, you are saving your previous work as a reference.

29. We are going to make the paddle move up and down by pressing the up and down arrow keys. Therefore, create a velocity for the paddle and set its initial value to zero. At the same place in the code where you define the puck's velocity, define the paddle's velocity. I called my object `paddle2` so you will have to be consistent with using the name of your paddle object.

    ```
    paddle2.v=vector(0,0,0)
    ```

30. We will have to update the position of the paddle (i.e. make it move). In the same place in the code where you update the position of the puck (inside the `while` loop) , add a line to update the position of the paddle.

    ```
    paddle2.pos=paddle2.pos+paddle2.v*dt
    ```

31. Now you need to create a function that checks for a keypress of the up and down arrow keys. Near the top of the program, at approximately line 3, add the following `movePaddlewithKeyboard` function. Read what it does! If an up or down arrow key is pressed, it sets the velocity of the paddle to be upward or downward. When the key is released (`keyup`) The `scene.bind` function tells GlowScript to look for a `keydown` or `keyup` event and called the `movePaddlewithKeyboard` function. Note that the name of the function `movePaddlewithKeyboard` is not important. I could have named this anything I want. In fact, I happened to pick a really long name, but at least it's descriptive.

```
def movePaddle(event):
#     print(event.type, event.which)
    if event.type=='keydown':
            k = event.which
            if k == 38:
                paddle2.v=50*vector(0,1,0)
            elif k == 40:
                paddle2.v=50*vector(0,-1,0)
    elif event.type=='keyup':
            paddle2.v=vector(0,0,0)

scene.bind('keydown keyup', movePaddle)
```

32. Run the program and see if you can control the paddle. You might find that it is quite dissatisfying. For example, the paddle can go through the top and bottom walls. Also, it is hard to stop the paddle at just the right moment to collide with the puck. To improve the paddle control, adjust the paddle's velocity that is set in this function.

Analysis

C Complete this exercise.

B Do everything for **C** and the following.

    1. Make one of the left walls a super-elastic wall with $COR > 1$ and one of the left walls an inelastic wall with $COR < 1$.

    2. Make half your paddle a super elastic paddle with $COR > 1$ and half your paddle an inelastic paddle with $COR < 1$. Give each half different colors.

A Do everything for **B** with the following modifications and additions.

    1. Place your infinite loop inside another infinite loop. When the puck goes past the paddle or through the goal, increment a score, reset the puck to the middle of the scene, and pause the game and wait for a mouse click or key press. Use the function below to pause the game. This function pauses the program and waits for a mouse click to continue.

```
scene.waitfor('click')
```

    2. Change your program so that the puck bounces off the paddle in a similar way as the *Pong* game that you played at the beginning of this chapter. Note that the physics is incorrect unless you invent a mechanical device that would cause the puck to bounce in this way.

7   PROGRAM – Modeling motion of a fancart

Apparatus

GlowScript
computer

Goal

In this activity, you will learn how to use a computer to model motion with a constant net force. Specifically, you will model the motion of a fan cart on a track.

Introduction

We are going to model the motion of a cart using the following data.

mass of cart	0.8 kg
$\vec{F}_{\text{net on cart}}$	$< 0.15, 0, 0 >$ N

Procedure

1. Begin with a program that simulates a cart moving with constant velocity on a track.

```
1   GlowScript 2.1 VPython
2
3   track = box(pos=vector(0,−0.05,0), size=vector(3.0,0.05,0.1), color=color.
        white)
4   cart = box(pos=vector(−1.4,0,0), size=vector(0.1,0.04,0.05), color=color.
        green)
5
6   cart.m = 0.8
7   cart.v = vector(1,0,0)
8
9   dt = 0.01
10  t = 0
11
12  scene.waitfor("click")
13
14  while cart.pos.x < 1.5 and cart.pos.x >−1.5:
15      rate(100)
16      cart.pos = cart.pos + cart.v*dt
17      t = t+dt
```

2. Run the program

What does line 12 do? It may help to comment it out and re-run your program to see how it changes things.

What line updates the position of the cart for each time step?

What line updates the clock for each time step?

Is the clock used in any calculations? Is it required for our program?

What line causes the program to stop if the cart goes off the end of the track?

We will now apply Newton's second law in order to apply a force to the cart and update its velocity for each time step. There are generally three things that must be done in each iteration of the loop:

(a) calculate the net force (thought it will be constant in this case)

(b) update the velocity of the cart

(c) update the position of the cart

(d) update the clock (*this is not necessary but is often convenient*)

Your program is already doing the third and fourth items in this list. However, the first two items must be added to your program.

3. Between the `rate()` statement and the position update calculation (i.e. between lines 15 and 16), insert the following two lines of code:

```
Fnet=vector(-0.15,0,0)
cart.v = cart.v + (Fnet/cart.m)*dt
```

The first line calculates the net force on the cart (though it is just constant in this case). The second line updates the velocity of the cart in accordance with Newton's second law. After making this change, your `while` loop will look like:

```
while cart.pos.x < 1.5 and cart.pos.x >-1.5:
        rate(100)
        Fnet=vector(-0.15,0,0)
        cart.v = cart.v + (Fnet/cart.m)*dt
        cart.pos = cart.pos + cart.v*dt
        t = t+dt
```

This block of code performs the necessary calculations of net force, velocity, position, and clock reading.

4. Run your program and view the motion.

> What is the direction of the net force on the cart? Sketch a side view of the fancart that shows the orientation of the fan.

Now we will add an arrow object in order to visualize the net force on the cart. An arrow in VPython is specified by its position (the location of the tail) and its axis (the vector that the arrow represents), as shown in Figure 7.1. The axis contains both magnitude and direction information. The magnitude of the axis is the arrow's length and the unit vector of the axis is the arrow's direction. The components of the axis are simply the components of the vector that the arrow represents.
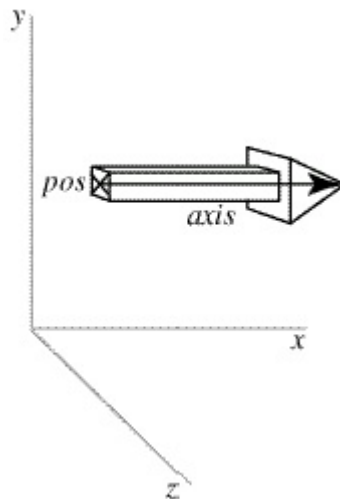
Figure 7.1: A arrow in VPython.

5. Near the top of your program after creating the track and cart, add the following two lines to define a scale and to create an arrow that has the same components $(-0.15, 0, 0)$ as the net force on the cart.

```
scale=1.0
forcearrow = arrow(pos=cart.pos, axis=scale*vector(-0.15,0,0), color=color
    .yellow)
```

6. Run your program.

7. Increase the scale and re-run your program.

> What does changing the scale do? Why do we want to use this variable and adjust it?

> Why does the arrow not move with the cart?

8. We want to make the arrow move with the cart. Thus, in our loop we need to update the position of the arrow after we update the position of the cart. Also, in some situations, the force changes, so in general it's a good idea to update the arrow's axis as well. At the bottom of your `while` loop, after you've updated the clock, add the following lines in order to update the position of the arrow and the axis of the arrow.

```
forcearrow.pos=cart.pos
forcearrow.axis=scale*Fnet
```

9. Run your program.

Lab Report

C Complete the experiment and report your answers for the following questions.

1. Does the simulation behave like a real fancart?

2. Though the velocity of the cart changes as it moves, does the force change or is the force constant?

3. Does the acceleration of the cart change or is the car's acceleration constant?

4. When the cart passes $x = 0$, turn off the fan (i.e. set the net force on the cart to zero). Describe the resulting motion of the cart. What is the velocity of the cart after the fan turns off?

B Do all parts for **C** do the following.

1. Add a second arrow that represents the velocity of the cart. Update its position and its axis. Give it an appropriate scale.

A Do all parts for **C** and **B** and do the following.

1. Add keyboard interactions that allow the user to make the force zero (i.e. turn off the fan), turn on a constant force to the right, or turn on a constant force to the left. In all of these cases, the arrow should indicate the state of the fan.

2. Check that your code results in correct motion. Compare to how a real fan cart would behave. Describe what you did to test your code, and describe what observations you made that convince you that it works correctly. Your description of the motion of the cart should be accompanied by pictures with force and velocity arrows.

8 GAME – Lunar Lander

Apparatus

Computer
GlowScript – www.glowscript.org

Goal

The purpose of this activity is to create a Lunar Lander game where you have to land the lunar module on the moon with as small a speed as possible and as quickly as possible. If the speed is too high, it crashes. If it takes you forever, then you run out of fuel.

Procedure

In the previous simulation that you wrote, you learned how to model the motion of an object on which the net force is constant. In that case, the object was a fancart. You learned how to apply Newton's second law to update the velocity of an object given the net force on the object. Once you can do this, you can model the motion of *any* object. As a reminder, the important steps in each iteration of the loop are to:

1. calculate the net force (although in some cases it is constant)

2. update the velocity of the cart

3. update the position of the cart

4. update the clock (*this is not necessary but is often convenient*).

The net force may not be constant. For example, you can check for keyboard interactions and turn a force on or off, or the net force might depend on direction of motion (such as friction) or speed (such as drag) or position (such as gravitational force of a star on a planet). This is why you have to calculate the net force during each iteration of the loop.

To develop a lunar lander game, we are going to begin with a bouncing ball that makes an elastic collision with the floor.

A bouncing ball

1. Here is a template for a program that simulates a bouncing ball. **However, a few essential lines are missing.** Type the template below but do not run the code (since lines are missing). It is fine if the first line references a more recent version of GlowScript.

```
1  GlowScript 2.1 VPython
2
3  scene.range=20
4
5  ground = box(pos=vector(0,-10.05,0), size=vector(40.0,1,1), color=color.
       white)
6  ball = sphere(pos=vector(0,9,0), radius=2, color=color.yellow)
7
```

```
8    ball.m = 1
9    ball.v = vector(0,0,0)
10   g=vector(0,-10,0)
11
12   dt = 0.01
13   t = 0
14
15   scale=1
16   FgravArrow = arrow(pos=ball.pos, axis=scale*ball.m*g, color=color.red)
17
18   scene.waitfor("click")
19
20   while 1:
21        rate(100)
22   #      Fgrav=
23   #      Fnet=
24   #      ball.v =
25   #      ball.pos =
26        if(ball.pos.y-ball.radius < ground.pos.y+ground.height/2):
27                ball.v=-ball.v
28        t = t+dt
29        FgravArrow.pos=ball.pos
30        FgravArrow.axis=scale*Fgrav
```

Line 10 defines a vector $\vec{g}$. What is this vector called? What is its direction, and what is its magnitude?

2. Line 22 should compute the gravitational force on the ball. Fill in this line using the variables for the mass of the ball and Earth's gravitational field.

3. Line 23 is the net force on the ball. This is computed by summing all forces on the ball. But the only force on the ball in this case is the gravitational force. Fill in line 23 with the variable representing the gravitational force on the ball.

4. Line 24 updates the velocity of the ball and line 25 updates the position. Fill in each of these lines with the appropriate calculation for updating the velocity and position of the ball. Refer to the previous chapter on the fancart if you forget how to do this.

5. Run your program and make sure it shows a bouncing ball.

What is the purpose of lines 26 and 27?

If line 26 was changed to `if(ball.pos.y < ground.pos.y):` , what would occur and why is this worse than the original version of line 26? (You should comment out line 26 and type this new code in order to check your answer.)

Is the gravitational force on the ball constant or does it change? Explain your answer.

6. The Moon has a gravitational field that is $1/6$ that of Earth. Change $\vec{g}$ to model the motion of a bouncing ball on the Moon and re-run your program.

What is the primary difference in the motion of a ball dropped on the Moon and a ball dropped from the same height on Earth? In other words, if you were to see an animation of each ball, side by side, how would you know which animation is of the ball on the Moon?

Moon Lander

We will now model the motion of a lunar module that is landing on the moon.

Figure 8.1: Apollo 16 LM *Orion*

7. Start a new program and type the following code into GlowScript.

```
GlowScript 2.1 VPython

scene.range=20
```

```
ground = box(pos=vector(0,-10.05,0), size=vector(40.0,1,1), color=color.
    white)
spaceship = box(pos=vector(0,8,0), size=vector(2,5,2), color=color.yellow)

spaceship.m = 1
spaceship.v = vector(0,0,0)
g=1/6*vector(0,-10,0)

dt = 0.01
t = 0

scale=5.0
FgravArrow = arrow(pos=spaceship.pos, axis=scale*spaceship.m*g, color=
    color.red)

while 1:
    rate(100)
#     Fgrav=
#     Fnet=
#     spaceship.v =
#     spaceship.pos =
    if(spaceship.pos.y-spaceship.height/2<ground.pos.y+ground.height/2):
            print("spaceship has landed")
            break
    t = t+dt
    FgravArrow.pos=spaceship.pos
    FgravArrow.axis=scale*Fgrav
```

8. Fill in lines 20-23 with the appropriate expressions.

9. We are now going to add a force of thrust due to rocket engines. Before the `while` loop, define a thrust force.

```
Fthrust=vector(0,4,0)
```

10. After defining the thrust vector, create another arrow that will represent the thrust force. Call it `FthrustArrow` as shown.

```
FthrustArrow = arrow(pos=spaceship.pos, axis=scale*Fthrust, color=color.
    cyan)
```

11. In the while loop, change the net force so that it is the sum of the gravitational force and the thrust of the rocket engine.

```
Fnet=Fgrav+Fthrust
```

12. Also, in the while loop, update the thrust arrow's position and axis.

```
FthrustArrow.pos=spaceship.pos
FthrustArrow.axis=scale*Fthrust
```

13. Run your program and verify that the motion of the spaceship is what we expect from Newton's second law.

> Change the thrust to 10/6 N (in the $+y$ direction. Describe the motion. Is this consistent with Newton's second law?

Let's use the keyboard to turn on and off the engine. In this case, "on" means that the vertical thrust is $(0,4,0)$ and "off" means that the thrust is zero, $(0,0,0)$.

14. Create a new program. Copy and paste your last program into this new file, and set the value of `Fthrust` to zero, $(0,0,0)$. Run your program and verify that the lunar module accelerates downward and stops when reaching the Moon's surface. (Make sure that $g = (0,-10/6,0)$ N/kg.

15. Near the top of your program at approximately line 2, add the following function.

```
##add keyboard control
def process(event):
    global Fthrust
    if event.type=='keydown':
        k = event.which
        if k == 38: #up arrow turns on the vertical thruster
            Fthrust=vector(0,4,0)
    elif event.type=='keyup': #releasing the key turns off the thruster
        Fthrust=vector(0,0,0)

    FthrustArrow.axis=scale*Fthrust

scene.bind('keydown keyup', process)
```

> What key is used to turn on the thruster? What causes the thruster to turn off?

16. Run your program and verify that it works. Use the up arrow key to control the thruster. Land the lunar module as gently as possible on the Moon's surface.

Analysis

C Complete this exercise and do the following.

1. Print the speed of the spaceship and the clock reading when it lands.

B Do everything for **C** and the following.

1. If the speed of the spaceship is greater than a minimum requirement (like 1 m/s), print "You lose."
2. If the speed of the spaceship is less than this minimum, print "You win."

A Do everything for **B** with the following modifications and additions.

1. Create an engine that fires in the $+x$ direction (the engine is on the left so the arrow points to the right) when the right arrow key is pressed.
2. Create an engine that fires in the $-x$ direction (the engine is on the right so the arrow points to the left) when the left arrow key is pressed.
3. Place a target on the ground.
4. Check that the lunar module lands on the target.
5. Check that the x-velocity is very small (perhaps less than 1 m/s for example) when the spaceship hits the target and print "You win" if and only if the spaceship has a very small x-velocity.
6. Since you don't want to waste fuel, assign points based on the time elapsed and cause the player to lose if the clock reading exceeds some amount. If you want, you can create a timer that starts at some value like 20 s and counts down to zero.

9   GAME – Tank Wars

Apparatus

Computer
GlowScript – www.glowscript.org

Goal

The purpose of this activity is to create a Tank Wars game where you move a tank and adjust the launch angle and launch speed to hit a target.

Procedure

1. Create a new file in Glowscript and copy and paste the following code template from our Trinket site. For your reference, the program is printed below, but it will be much faster to copy and paste from our Trinket site.

```
1   GlowScript 2.1 VPython
2
3   def control(event):
4       global theta, muzzlespeed, bulletsList
5       if event.type=='keydown':
6           k = event.which
7           if k == 38:
8               theta=theta+dtheta
9               if theta>rad(90):
10                  theta=rad(90)
11              turret.axis=L*vector(cos(theta),sin(theta),0)
12              angleBar.axis=(5*theta)*vector(1,0,0)
13          elif k == 40:
14              theta=theta-dtheta
15              if theta<0:
16                  theta=0
17              turret.axis=L*vector(cos(theta),sin(theta),0)
18              angleBar.axis=(5*theta)*vector(1,0,0)
19          elif k == 37:
20              muzzlespeed=muzzlespeed-dspeed
21              if muzzlespeed<1:
22                  muzzlespeed=1
23              speedBar.axis=(muzzlespeed/2+0.5)*vector(1,0,0)
24          elif k == 39:
25              muzzlespeed=muzzlespeed+dspeed
26              if muzzlespeed>20:
27                  muzzlespeed=20
28              speedBar.axis=(muzzlespeed/2+0.5)*vector(1,0,0)
29          elif k==32:
```

```
30                bullet=sphere(pos=turret.pos+turret.axis, radius=0.5, color=
                     color.white)
31                bullet.v=muzzlespeed*vector(cos(theta),sin(theta),0)
32                bulletsList.append(bullet)

33
34    scene.bind('keydown', control)

35
36    def rad(degrees): #converts an angle in degrees to an angle in radians
37        radians=degrees*pi/180
38        return radians

39
40    scene.range=20
41    scene.width=600
42    scene.height=400

43
44    #create objects
45    ground = box(pos=vector(0,-15,0), size=vector(60,2,2), color=color.green)
46    tank = box(pos=vector(-18,-13,0), size=vector(2,2,2), color=color.yellow)
47    turret = cylinder(pos=tank.pos, axis=vector(0,0,0), radius=0.5, color=tank
         .color)
48    turret.pos.y=turret.pos.y+tank.height/2
49    angleBar = cylinder(pos=vector(-18,-19,0), axis=vector(1,0,0), radius=1,
         color=color.magenta)
50    speedBar = cylinder(pos=vector(5,-19,0), axis=vector(1,0,0), radius=1,
         color=color.cyan)

51
52    #turret
53    theta=rad(45)
54    dtheta=rad(1)
55    L=3
56    turret.axis=L*vector(cos(theta),sin(theta),0)

57
58    #bullets
59    bulletsList=[]
60    m=1
61    muzzlespeed=15
62    dspeed=1

63
64    #Bar
65    angleBar.axis=(5*theta)*vector(1,0,0)
66    speedBar.axis=(muzzlespeed/2+0.5)*vector(1,0,0)

67
68    #motion
69    g=vector(0,-10,0)
70    dt = 0.01
71    t = 0

72
73    while True:
74        rate(100)

75
76        for thisbullet in bulletsList:
77            if(thisbullet.pos.y<ground.pos.y+ground.height/2):
78                thisbullet.Fnet=vector(0,0,0)
79                thisbullet.v=vector(0,0,0)
```

```
80          else :
81              thisbullet . Fnet=m∗g
82          thisbullet . v=thisbullet . v+thisbullet . Fnet /m∗dt
83          thisbullet . pos=thisbullet . pos+thisbullet . v∗dt
84
85      t=t+dt
```

2. Run the program above. Then, study it and use it to answer the following questions.

What line updates the velocity of the bullet?

What line updates the position of the bullet?

Is the world in this simulation Earth? Cite a particular line number in the code in order to support your answer.

What does the `rad()` function do?

What does the variable L tell you?

What is the variable name for the initial speed of the bullet? How much does the launch speed of the bullet increase or decrease when you press the right or left arrow key only once? (And what is this variable called?)

What is the variable name for the angle the bullet is launched at? How much does the angle increase or decrease when you press the up or down arrow key once? Is the unit radians or degrees?

After a bullet hits the ground, what is the net force on the bullet and what is its velocity? Cite the particular line numbers that support your answer.

When the bullet is in the air, what is the net force on the bullet? Cite the particular line number that supports your answer.

What key strokes are used to change the launch angle?

What key strokes are used to change the launch speed?

What is the maximum and minimum launch speed allowed? Which line numbers tell you this?

What is the maximum and minimum launch angle allowed? Which line numbers tell you this?

We are now going to add a few features to the game.

Creating a Target

3. Create a target that is a box named `target` that is the size of the tank and place it somewhere else on the terrain. Run your program to see the target. (You can think of this as the opposing player's tank, perhaps.)

   We will need to check for collisions between a bullet and the target. It is convenient to create a function that does the math to see if the bullet (sphere) and target (box) overlap. If they do overlap, it returns `True`. If they do not overlap, the it returns `False`.

4. At the top of your program near where the `rad()` function is defined, add the following function. You do not have to type the commented lines. You will need to double check your typing to make sure you do not have any typos. The condition of the `if` statement is rather long, so check it for accuracy. Again, this code snippet is on our Trinket site if you want to copy and paste.

```
#determines whether a sphere and box intersect or not
#returns boolean
def collisionSphereAndBox(sphereObj, boxObj):
    if((sphereObj.pos.x-sphereObj.radius<boxObj.pos.x+boxObj.length/2 and
        sphereObj.pos.x+sphereObj.radius>boxObj.pos.x-boxObj.length/2) and
        (sphereObj.pos.y-sphereObj.radius<boxObj.pos.y+boxObj.height/2
        and sphereObj.pos.y+sphereObj.radius>boxObj.pos.y-boxObj.height/2)
        ):
            result=True
    else:
            result=False
    return result
```

5. At the `if` statement at line 77 in the original template above, where the program checks whether the bullet hits the ground, add an `elif` statement (between the if and else) to check whether the bullet collides with the target. It should look like this.

```
        elif collisionSphereAndBox(thisbullet,target):
            thisbullet.Fnet=vector(0,0,0)
            thisbullet.v=vector(0,0,0)
            print("direct hit!")
```

6. Run your program. Fire a bullet that hits the target and observe the console after the collision. You should see it printing "direct hit!" over and over.

```
 Why does it continuously print "direct hit!" instead of printing it just one time?



```

We would like to effectively reset the game to its initial state. To do this we will need to:

(a) Erase all of the bullets in the scene.

(b) Reset the bulletsList to an empty list.

7. Inside the `elif` statement that you just created, after you print "direct hit" add the following lines:

```
scene.waitfor("click")

for thisbullet in bulletsList:
    thisbullet.visible=False
bulletsList=[]

break
```

These lines will pause the game until the user clicks the scene. Then it will make each bullet invisible. Then, it creates an empty list of bullets. Finally, it breaks out of the bullet loop.

Technically you might want to do other things after a target is hit, like increment a score or reset the launch speed and launch angle back to their initial values for example. Or, you might blow up the target.

8. Run your program. Verify that it works as expected.

Analysis

C Complete this exercise and do the following.

1. Print the launch speed of the bullet, the launch angle of the bullet, and the x-distance traveled by the bullet (called the range) when the bullet hits the target or ground. Note that the range is not the same as the x-position of the bullet because the bullet isn't launched from the origin.

2. Change the maximum speed of the bullet to 30.

3. Set the initial launch angle to 60 degrees (do this in the code) and find the launch speed necessary to hit the target if the target is at $x = 18$. Set the initial launch speed to this value in the code so that when your code is run for the first time, the projectile will be launched at 60 degrees and will hit the target which is at $x = 18$.

B Do everything for **C** and the following.

1. Add a key stroke that will move the tank left and right, but do not allow the tank to go past the center of the screen or off the left side of the screen.

A Do everything for **B** with the following modifications and additions.

1. Create a barrier (a box) that sits between the tank and the target. If a bullet hits the barrier, it stops. Remember to use your `collisionSphereAndBox()` function to check for a collision between the bullet and barrier.

2. Create additional gameplay features like a scoring system and a max number of bullets.

3. Create different levels of the game. Create a variable called `level` that is an integer from 1–5. In the first level, there is no barrier. In the second level, there is a barrier. For other levels, place the tank at a different height or the target at a different height, for example. After getting to the fifth level, either end the game or go back to the first level.

10   PROGRAM – Modeling motion with friction

Apparatus

computer
GlowScript

Goal

In this activity, you will learn how to add a frictional force to a simulation.

Introduction

We are going to model the motion of a puck in air hockey that is slowed by a frictional force between the puck and the table. Though in practice, a puck in air hockey may be more influenced by air drag than friction with the table, we will treat the frictional force as *sliding friction*. Sliding friction occurs when one objects slides against the surface of another object. In the simplest situations, slide friction is:

$$\vec{f}_{slide} \quad = \quad \mu_k F_\perp (-\hat{v})$$

where μ_k is the coefficient of kinetic friction, $F_\perp$ is the perpendicular component of the contact force, and $\hat{v}$ is the unit vector pointing in the direction of the velocity of the object. In the case of a puck sliding on a level air hockey table, $F_\perp = mg$, the weight of the puck. Notice the negative sign for the direction of the frictional force. This means that sliding friction is always opposite the velocity of the object relative to the surface it is in contact with.

The coefficient of kinetic friction μ_k is a constant that depends on the materials in contact. For example, wood sliding on glass, wood sliding on concrete, and wood sliding on sand paper have different values of μ_k. A higher coefficient of friction means that there is a greater frictional force. Smaller coefficient of friction is less friction, and zero coefficient of friction is what we call "frictionless" (which doesn't exist in practice though friction might be negligible if it is small enough).

Rolling friction

A rolling ball also experiences a frictional force that decreases its center-of-mass velocity. If you push an object on wheels across carpet, you can see the carpet bunch up in front of the wheel. This is what happens at the microscopic scale for any surface. Rolling friction can be characterized as $\vec{f}_{roll} = \mu_{roll} mg(-\hat{v})$, where μ_{roll} depends on the properties of the surfaces in contact.

Procedure

In this program, we will model the motion of a golf ball rolling on a level green. This code template can be copied from our course web site on Trinket.

1. Begin by typing the following template for a golf ball rolling in the x-direction on a green.

```
1   GlowScript  2.1  VPython
2
3   scene.range=20
4   scene.width=400
5   scene.height=400
6
7   ground = box(pos=vector(0,0,0), size=vector(40,40,1), color=color.green)
8   ball = sphere(pos=vector(-18,0,0), radius=0.5, color=color.white)
9   ground.pos.z=ground.pos.z-ground.width/2-ball.radius
10  hole = cylinder(pos=vector(15,0,ground.pos.z+ground.width/2),axis=vector
        (0,0,1), radius=3*ball.radius, color=vector(0.8,0.8,0.8))
11  hole.pos.z=hole.pos.z-mag(hole.axis)*0.9
12
13  #ball, friction, and grav
14  ball.m=0.045
15  g=10
16  mu=0.1
17
18  #speed
19  initialspeed=5
20
21  #velocity vector
22  ball.v=initialspeed*vector(1,0,0)
23  scale=5/initialspeed
24  varrow = arrow(pos=ball.pos, axis=scale*ball.v, shaftwidth=0.5, color=
        color.yellow)
25
26
27  #clock
28  dt=0.01
29  t=0
30
31  scene.waitfor("click")
32
33  while 1:
34          rate(100)
35          vhat=ball.v/mag(ball.v)
36          Fnet=mu*ball.m*g*(-vhat)
37  #          ball.v=
38  #          ball.pos=
39
40          varrow.pos=ball.pos
41          varrow.axis=scale*ball.v
42
43          t=t+dt
```

2. Fill in lines 38 and 39 and run the program.

What does line 36 do?

If you wanted to add other forces to the simulation, which line would you change?

What does the arrow represent?

What variable represents the coefficient of friction?

3. Try different values of the initial speed until you can get the ball to stop in the hole.

4. Now change the coefficient of friction to either a smaller or larger value and find the new initial speed needed to get the ball into the cup.

Motion on a Hill

Suppose an object like a golf ball travels across a hill of constant slope. Let's define $+y$ in the plane of the hill and directly uphill. Define $+x$ to the right, in the plane of the hill, perpendicular to the $+y$ axis. Then, with this coordinate system, $+z$, is perpendicular to the hill toward the sky (but not directly outward from Earth due to the inclination of the hill). Suppose that the hill is inclined at an angle α relative to vertical (as established by hanging a weight on a string).

Sketch a top view and side view of the hill. Show the $+x$, $+y$, and $+z$ directions in each view.

The motion of the ball is similar to projectile motion, but its acceleration is modified by the angle of inclination of the hill. (And of course there is rolling friction.) In this case,

$$F_{grav,y} \quad = \quad -mg\sin(\alpha)$$

and

$$F_{grav,z} \quad = \quad -mg\cos(\alpha)$$

Since the ball has no acceleration in the z direction, the perpendicular component of the force by the ground on the ball is equal in magnitude to $F_{grav,z}$, but in the $+z$ direction. Thus,

$$F_{ground,z} \quad = \quad mg\cos(\alpha)$$

Then the rolling frictional force is:

$$
\begin{aligned}
\vec{f}_{roll} \quad &= \quad \mu_{roll}F_{\perp}(-\hat{v}) \\
&= \quad \mu_{roll}mg\cos(\alpha)(-\hat{v})
\end{aligned}
$$

The net force on the ball is the sum of the gravitational force and force by the ground (both parallel and perpendicular to the ground). Thus,

$$
\begin{aligned}
\vec{F}_{net} \quad &= \quad \vec{F}_{grav} + \vec{F}_{roll} + \vec{F}_{ground,z} \\
&= \quad (0, -mg\sin(\alpha), mg\cos(\alpha)) + \mu_{roll}mg\cos(\alpha)(-\hat{v}) + (0, 0, mg\cos(\alpha))
\end{aligned}
$$

For simplicity, I will combine $\vec{F}_{grav}$ and $\vec{F}_{ground,z}$ to give:

$$\vec{F}_{net} \quad = \quad (0, -mg\sin(\alpha), 0) + \mu_{roll}mg\cos(\alpha)(-\hat{v})$$

where g is the magnitude of the gravitational field of Earth at its surface. You can think of this conceptually as the component of the gravitational force parallel to the hill plus the frictional force:

$$\vec{F}_{net} \quad = \quad \vec{F}_{grav,\|} + \vec{F}_{roll}$$

If you strike the golf ball at an angle θ relative to the x-axis with a speed v_i, then its velocity in the plane of the hill is:

$$\vec{v} \quad = \quad v_i(\cos(\theta), \sin(\theta), 0)$$

Using the net force and initial velocity of the ball, you can model the motion of a golf ball that includes both "break" (due to the slope of the hill) and rolling friction.

5. Create a new program.

6. Copy and paste the following template from our course web site.

```
1  GlowScript 2.1 VPython
2
3  def rad(degrees): #converts an angle in degrees to an angle in radians
4      radians=degrees*pi/180
5      return radians
6
7  scene.range=20
8  scene.width=400
9  scene.height=400
10
11 ground = box(pos=vector(0,0,0), size=vector(40,40,1), color=color.green)
12 ball = sphere(pos=vector(-18,0,0), radius=0.5, color=color.white,
       make_trail=True)
13 ground.pos.z=ground.pos.z-ground.width/2-ball.radius
14 hole = cylinder(pos=vector(15,0,ground.pos.z+ground.width/2),axis=vector
       (0,0,1), radius=3*ball.radius, color=vector(0.8,0.8,0.8))
15 hole.pos.z=hole.pos.z-mag(hole.axis)*0.9
16
17 #ball, friction, and grav
18 ball.m=0.045
19 g=10
20 mu=0.2
21 alpha=rad(5)
22
23 #speed and angle
24 initialspeed=12
25 theta=15
26
27 #velocity vector
28 #ball.v=
29 scale=5/initialspeed
30 varrow = arrow(pos=ball.pos, axis=scale*ball.v, shaftwidth=0.5, color=
       color.yellow)
31
32
33 #clock
34 dt=0.01
35 t=0
36
37 scene.waitfor("click")
38
39 while 1:
40         rate(100)
41         vhat=ball.v/mag(ball.v)
42 #          Fgrav=
43 #           Ffriction=
44         Fnet=Fgrav+Ffriction
45         ball.v=ball.v+Fnet/ball.m*dt
46         ball.pos=ball.pos+ball.v*dt
47
48         varrow.pos=ball.pos
49         varrow.axis=scale*ball.v
50
```

```
51                    t=t+dt
```

7. Fill in lines 28, 42, and 43 and remove the comment marks. Run the program.

<div style="border:1px solid black; padding:10px;">
What variable will you change in order to make the hill steeper or less steep?
</div>

<div style="border:1px solid black; padding:10px;">
What variable will you change to adjust the coefficient of friction?
</div>

<div style="border:1px solid black; padding:10px;">
What variable will you change to adjust the initial speed of the ball?
</div>

<div style="border:1px solid black; padding:10px;">
What variable will you change in order to strike the golf ball at a greater or less angle (relative to the $+x$ axis).
</div>

Lab Report

C Complete the experiment and report your answers for the following questions.

1. Go back to your first program of a golf ball rolling on a level green.
2. Place the cup the location $x = 15$ m, $y = 15$ m, near the top right corner of the green.
3. Find the initial velocity vector needed to get the ball into the cup, if $mu = 0.1$.

B Do all parts for **C** and do the following.

1. Go back to your program of a golf ball rolling on a hill.
2. Add keyboard interaction so that by using the up and down arrows, you can change the angle at which you strike the ball, and by using the left and right arrows, you can change the initial speed of the ball. Then, use the spacebar to putt the ball. (Use your tank wars program for a reminder on how to write the code.) Use the initial position of the hole at $x = 15$ m, along the x-axis in order to test out your code.
3. Alert the player and stop the ball if the ball goes into the hole.
4. Alert the player and stop the ball if the ball goes off screen.

A Do all parts for **C** and **B** and do the following.

1. Use your program from Part B.
2. Change properties of the green (coefficient of friction or slope of the hill or location of the hole) each time the player sinks a putt and allow the player to play again. Reset the ball to its starting location when the ball misses the hole and stops or if it goes off the screen.

11   GAME – Asteroids

Apparatus

Computer
Glowscript – www.glowscript.org

Goal

The purpose of this activity is to modify an asteroids game so that when a large asteroid explodes into two smaller asteroids, the center of mass continues with the same velocity, as you observed in the experiment of the colliding pucks.

Procedure

Playing Asteroids

1. Play the game *Asteroids*. A link is available from our course web site. Note the actions of the up, down, left, and right arrow keys and how they affect the motion of the spaceship.

2. Copy the template from our course web site, and paste it into a new file in GlowScript.

3. Run the program. Note how the up, down, left, and right keystrokes affect the motion of the spaceship. It's different than in the original Asteroids game. Also, note that hitting an asteroid with a bullet does not make it break up into pieces. (You will add this functionality.)

4. Answer the following questions.

    (a) What are the three functions used for generating random numbers and what is the purpose of each of these functions?

    (b) What is the magnitude of the thrust of the engine when it is firing?

    (c) What line number calculates the net force on the spaceship?

    (d) What line numbers update the velocity and position of the spaceship?

    (e) What line numbers update the positions of the asteroids?

(f) What line numbers update the positions of the bullets?

(g) What line number makes the asteroid disappear when it is hit by a bullet?

(h) How many asteroids are created when the function `createAsteroids()` is called and where do these asteroids come from?

(i) The velocities of the asteroids are randomized. What is the maximum possible velocity of an asteroid? What is the minimum possible velocity of an asteroid? (Note that the directions are randomized as well. By "maximum velocity" I am referring to the maximum absolute value of the x and y components of the velocity vector. By "minimum velocity", I am referring to the minimum absolute value of the x and y components of the velocity vector.)

(j) If you decrease the mass of the spaceship (i.e. change it from 1 to a smaller number like 0.5), how would it affect the spaceship's motion when the engine is firing? Answer the question, then test your answer to see if it is correct, and then comment on whether your observation matched your prediction.

Adding Asteroid Explsions

In the last chapter, you learned that the center of mass of a two-body system is constant. Whether it is a collision of two objects or an exploding fireworks shell, during the collision or explosion the center of mass velocity of the system remains constant.

In the classic Asteroids game, when a large asteroid is shot with a bullet, it breaks into two fragments. Since the center of mass of the system must remain constant then

$$\vec{v}_{ast} = \frac{m_1 \vec{v}_1 + m_2 \vec{v}_2}{m_1 + m_2}$$

where 1 and 2 refer to the two fragments. Define the total mass of the fragments as $M = m_1 + m_2$, which is the mass of the asteroid before the explosion. Then,

$$\vec{v}_{ast} = \frac{m_1 \vec{v}_1 + m_2 \vec{v}_2}{M}$$

In our game, let's assume that the asteroid breaks into two equal mass fragments that are each $1/2$ the total mass of the asteroid. Then, $m_1 = m_2 = 1/2M$. Thus,

$$\vec{v}_{ast} = \frac{1/2M\vec{v}_1 + 1/2M\vec{v}_2}{M}$$

$$\vec{v}_{ast} = \frac{\vec{v}_1 + \vec{v}_2}{2}$$

In other words, since the fragments have equal masses, then the asteroid's velocity is the arithmetic mean of the velocities of the fragments (i.e. the sum of their velocities divided by 2).

In our game, we will randomly assign the velocity of fragment 1 so that it will shoot off with a random speed in a random direction. Then we will calculate the velocity of fragment 2 so that

$$\vec{v}_2 = 2\vec{v}_{ast} - \vec{v}_1$$

5. Above the `while` loop, find the line where the asteroidList is created. Near this point in the program, create a list called `fragmentList`. To do this, type the following line:

```
fragmentList=[]
```

6. Near the top of the program where other functions are defined (e.g. after the `createAsteroids()` function is defined), write a function that creates the fragments when an asteroid is hit. Type the following code and replace the commented line with the correct calculation for the velocity of fragment 2.

```
def createFragments(asteroid):
    fragment1=sphere(pos=asteroid.pos, radius=0.5, color=color.magenta)
    fragment2=sphere(pos=asteroid.pos, radius=0.5, color=color.magenta)
    fragment1.m=0.5
    fragment2.m=0.5
    fragment1.v=vector(0,0,0)
    fragment1.v.x=choice(1,-1)*randint(1,5)
    fragment1.v.y=choice(1,-1)*randint(1,5)
#    fragment2.v=
    fragmentList.append(fragment1)
    fragmentList.append(fragment2)
```

7. In the main `while` loop, there is an `if` statement that checks for a collision between a bullet and asteroid. The line reads:

```
if(collisionSpheres(thisbullet,thisasteroid)and thisbullet.visible==True):
```

Inside this `if` statement, call the `createFragments` function using:

```
createFragments(thisasteroid)
```

This creates the explosion whenever the bullet hits an asteroid.

8. You will need to update the positions of the fragments (i.e. make them move) inside the main `while` loop. It's easiest to copy and paste the section of code for the asteroids and change the variables names to the appropriate names for the fragments. You can also copy and paste this code from our course web site. Here's an example of what it should look like:

```
    #update positions of fragments
    for thisfragment in fragmentList:
        if thisfragment.visible==True:
            thisfragment.pos=thisfragment.pos+thisfragment.v*dt
            #check for collision with spaceship
            if(collisionConeSphere(spaceship,thisfragment)):
                spaceship.visible=False
                fire.visible=False
            #wrap at edge of screen
            if thisfragment.pos.x>20 or thisfragment.pos.x<-20:
                thisfragment.pos=thisfragment.pos-thisfragment.v*dt
                thisfragment.pos.x=-thisfragment.pos.x
            if thisfragment.pos.y>20 or thisfragment.pos.y<-20:
                thisfragment.pos=thisfragment.pos-thisfragment.v*dt
                thisfragment.pos.y=-thisfragment.pos.y
            #check for collision with bullets
            for thisbullet in bulletsList:
                if(collisionSpheres(thisbullet,thisfragment)and thisbullet
                    .visible==True):
                    thisfragment.visible=False
                    thisbullet.visible=False
```

9. The section of code that counts how many asteroids are left also needs to count how many fragments are left. After the `for` loop that counts the asteroids that are left, add a second for `loop` that counts the remaining fragments.

```
    for thisfragment in fragmentList:
        if thisfragment.visible:
            Nleft=Nleft+1
```

10. That should do it. Run your code and fix any errors.

11. Note that we could have written our code more efficiently. Whenever you find that you are copying and pasting the same code over and over, you might want to consider putting it into a function and calling the function. For example, we have to wrap the motion of the spaceship, asteroids, and fragments. We can create a `wrap()` function that takes an argument of the object (like spaceship) and checks to see if it is off screen. If it is off screen, it wraps the position of the object. If you notice ways like this to reduce the number of lines of code, feel free to make those changes.

Analysis

C Complete this exercise and do the following.

    1. Add a point system that tallies points whenever an asteroid is shot.

    2. Print the total number of points at the end of the game when the spaceship collides with an asteroid.

B Do everything for **C** and the following.

    1. Add a keystroke for the "down" arrow key that rotates the spaceship $180°$ (π radians). For the syntax on how to do this, see the left and right arrow keys.

    2. Place the entire `while spaceship.visible==True:` loop inside another `while True:` loop so that after the spaceship collides with an asteroid the program will not end, but rather it will begin again. If you select the entire `while spaceship.visible==True:` loop and hit the TAB key, then the selected code will automatically indent.

    3. You will need to reinitialize your variables for the velocity, thrust, and net force on the spaceship, along with your lists, in order to reset the game. You can copy code like the following and paste it inside the `while True:` statement but at the end. In other words, when the inner loops ends (by virtue of the spaceship not being visible), then this code block below will run before the outer loops starts again.

```python
scene.waitfor("click")

spaceship.visible=True
spaceship.v=vector(0,0,0)
thrust=0
Fnet=vector(0,0,0)

#bullets
for b in bulletsList:
    b.visible=False
bulletsList=[]

#asteroids
Nleft=0 #counter for number of asteroids left in the scene
for ast in asteroidList:
    ast.visible=False
asteroidList=[]

createAsteroids()

#fragments
for f in fragmentList:
    f.visible=False
fragmentList=[]
```

A Do everything for **B** with the following modifications and additions.

    1. Think of an improvement that will require knowledge of physics. Here are some ideas:

       (a) asteroids collide with each other

       (b) bullets have mass. Every time a bullet is fired, it exerts a backwards force on the spaceship and causes the spaceship to lose mass.

       (c) the asteroid fragments should have have the volume of the asteroids and thus should have the appropriate radius.

(d) the asteroid can break into three fragments instead of two.

(e) the fragments can have different masses thought the sum is equal to the mass of the asteroid

2. Talk to Dr. T the improvement you want to make.

3. Implement your idea.