In-Class Assignment 13 – *Rolling Ball*

In this assignment, you will write a quick program that moves an object left and right across the screen using uniform motion. It will reflect off of the edges of the screen, and animation will be used to give it the illusion of rolling motion.

*Physics Goals*: Uniform motion, reflections, relative position and trigonometric functions.

*Programming Goals*:

*Instructions:*

1. *Make the ball move.* The ball will move like the Wheel did in one dimension, but instead it will move in two dimensions..
   1. In the update() method of the Ball class, modify both of the ball’s position properties by adding in the product of the velocity components in their respective directions and the amount of time that has passed since the last update.
   2. Call the update() method from the engine() function. If you run your program, it should not do anything yet, but it shouldn’t throw any errors.
   3. Change the initial velocity of the wheel to be 1 m/s in the x-direction and something similar in the y-direction. Your wheel should move now.
2. *Make the wheel reflect.* Simple reflections at the side of the screen or at a pre-defined surface position are an essential feature of many games. Here, you will create a set of reflections for the wheel so that it will continuously move
   1. Determine if the wheel should reflect off of the wall. Fill in the hits\_edge\_of() call of the Wheel class. Is should return False unless the edge of the field overlaps some part of the ball, in which case it should return True. This has to work for both the top and the bottom walls.
   2. In the engine() function, call the hits\_edge\_of() method with this. (If you put a print statement in your hits\_edge\_of() method, you can check if it’s working before you work on the bounces\_off() method, next).
   3. Reflect the wheel off of the edge. When the ball bounces\_off() a Field object, you’ll need to change the wheel’s position and velocity as you did previously, but this time you’ll do it in a method using the object’s properties directly and you’ll need to do it for both the top of the field and the bottom of the field.
   4. In the engine() function, call the bounces\_off() method of the ball whenever the ball hits the edge of the field.

In-Class Assignment 9 – *Colliding Wheels*

Reflections against stationary objects are

*Physics Goals*: Collision detection, collision resolution

*Programming Goals*: Functions, Methods, Objects, Reusing Code

*Instructions:*

1. *Reuse code from the rolling wheel*. When you’ve already built something in a previous program, you can cannibalize the code for the next program. In this case, you’re going to program two wheels that roll around on the screen, so you’ll want to
   1. Copy your Wheel class from the previous class and use it in this one.
   2. Do the same with the reflect() function.
2. *Create two different wheels that move on the screen*. Every collision requires two objects. You’ll need two objects that start on opposite sides of the screen and move towards each other.
   1. Initialize two wheels with two different names in the initialization section of the main() function.
   2. Send both wheels to the view() function by placing them in the list wheels at the end of the initialization section of the main() function. They should now appear on the screen, but not update.
   3. Send both wheels to the engine() function as a list. Then, index the list to make two individual objects in the engine(), and update each individually.

Your wheels should now roll around on the screen, but pass through each other.

1. *Modify the wheel class*. When a heavy thing hits a stationary light thing, the heavy thing keeps moving in the same direction. When a light thing hits a stationary heavy thing, it bounces off of it.
   1. Add a mass property to the class. Modify the \_\_init\_\_() method to ask for the mass when the object is created.
   2. Change the animate() method of the Wheel class to so that the radius of the wheel’s hub is proportional to its mass.
2. *Create the collision code*. This includes both collision detection and collision resolution. Collision detection determines if there has been a collision, and should be part of the Wheel class, and collision resolution determines what to do with the two objects that have collided.
   1. Create a collides\_with() method for the Wheel class. It should take another wheel as an argument and it should return True whenever the distance between the centers of the two wheels are close enough that they will hit and returns False otherwise.
   2. In the engine() function, check if the two wheels hit using the collides\_with() method of one with the other as then argument. This should be done after the wheels’ positions have updated. Place this in a print statement for now, and you will be able to tell if it works by monitoring the output.
   3. Create a collide() function that takes two objects that have hit each other and changes their velocities in accordance with the rules of collision.
   4. In the engine() function, remove the print statement, and replace it with an if-statement that sends the objects to the collide() function when the wheels hit each other.

In-Class Assignment 10 – *1D Bowling*

This worksheet will introduce you to activity by making a simple state machine that is controlled by the user through clickable on-screen buttons.

*Physics Goals*: Forces, Impulses, and Friction

*Programming Goals*: Inheritance

1. Use the Wheel class and the reflect() and collide() functions from the previous assignment. Change the name of the wheel class to Translatable.
2. Create a Ball class using inheritance. Since this class will require almost the same code as the wheel did, create it with inheritance.
   1. To create a class using another class, define the name of the class as

class Ball(Translatable):

* 1. Then, create a new \_\_init\_\_() method under it. For this application, you can copy and paste the \_\_init\_\_() method from the Translable class.
  2. In the main() function, create a ball, add it to the update() function, and draw it in the view() function.

1. Your bowling pins will be tall rectangles that otherwise act the way the Ball does – translating freely on the surface of the
   1. Use inheritance to create a Pin class.
   2. Consider each method() in turn, starting with the \_\_init\_\_() method. Is there anything that has to be different about this because you now have a rectangular object instead of a circular one? How do the methods() need to be modified to If so change it. Things to think about:
      1. The size of the rectangle has two components rather than one.
      2. The width is the full width and not the diameter.
      3. The x and y positions of the Rectangle should be its center, but the pg.draw.rectangle() function draws the object from the upper left hand corner.
2. Add friction.

In-Class Assignment 11 –

This worksheet will introduce you to computer logic by building an interactive set of indicators that light up when different buttons are pressed on the screen.

*Physics Goals*:

*Programming Goals*:

1. Create a switch. In this step, you’ll simulate a switch by using a Button object to control a Light object. When the button is pressed it will switch the light from being on to being off, and vice versa.
2. Create a button labeled “A” using the same method as we used in the previous worksheet. It should be somewhere on the bottom half of the screen. There will eventually be three switches, so keep that in mind.
3. Add the button to the buttons list by placing its variable name in between the brackets.
4. Create an indicator that will be controlled by the switch in same way you did this in the last worksheet.
5. Add the light to the lights list by placing its variable name in between the brackets.
6. In the main() function, use an if-statement to activate the toggle() method in the light. This will switch it from on to off and back again.
7. Create an indicator for a logical combination. Make an indicator light that turns on when the A button is inactive and turns off when it’s active.
8. Create a light somewhere on the top of the screen (you will eventually have four, plan for that).
9. Add the light to the lights list by placing its variable name in between the brackets.
10. Use an if-statement in the main loop to turn the light on and off using its activate() and deactivate() methods based upon the state of the indicator connected to the “A” button.
11. Use the screenprint() function to label the light “-A”.
12. Create two more switch combinations. These use both a button and a light. Label one of them “B” and the other “C”.
13. Create two more indicators. These should light up when:
14. Either button B or button C is active, and
15. Both button B and button C are active.

Include labels for each.

1. Create a final indicator. This one will light up when A is active and neither B nor C is active. Include a label.
2. Complete the following chart using your program:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | C | -A | B ˅ C | B ˄ C | A ˄ -(B ˅ C) |
| T | T | T |  |  |  |  |
| T | T | F |  |  |  |  |
| T | F | T |  |  |  |  |
| T | F | F |  |  |  |  |
| F | T | T |  |  |  |  |
| F | T | F |  |  |  |  |
| F | F | T |  |  |  |  |
| F | F | F |  |  |  |  |

Is it all right?

1. Draw a wiring diagram for the logical operations in the background.
2. Call the background() function from the main() function.
3. Use the pg.draw.line() function to draw lines on the screen connecting the buttons to the logic and the logic to the lights. The logic can be drawn with the gameclass functions draw\_and(), draw\_or(), and draw\_not().

In-Class Assignment 6 – *Stopwatch*

This worksheet will introduce you to working with time by allowing you to build your first object class. The stopwatch has two different states (on and off, running or stopped), and a value that changes only when the stopwatch is on. You should be able to reset the stopwatch as well.

*Physics Goals*: Vector position, time

*Programming Goals*:

1. Begin by creating a time property in the Timer class and updating it in the update() method.
   1. Initialize the time property in the \_\_init\_\_() method.
   2. Every time the update() method is called, increase the time property by the value time sent to the method from main().
   3. Put a print statement in the view() method. This will tell you if your time is updating correctly.
2. Now print that number to the screen. You’ll need to choose a position for it, and to make a value for it that’s a string.
   1. Create a string from the time property and print it instead of printing the time property directly.
   2. Create some properties for the x and y coordinates of the upper lefthand corner of your output.
   3. Use the screenprint() function from the gameclass to print to the screen. This should print to the upper left hand corner of the screen.
   4. Change the position of the timer by changing the (0,0) tuple in the timer definition to another number.
3. Create a button that will turn the timer on and off labelled “Start/Stop”. This will require both making a button and modifying the timer to include a property that keeps track of its activity state, and change the state when the start() method is called. The update() method in the timer should only update when the button has been pressed, and you’ll need to send the button to the view() and control functions.
4. Create a lap button that will pause the display.
   1. You will need to create a button in main() labelled “Lap/Clear”, and then send the button to the view() and control() functions
   2. You will need to create properties for a display time value and a lap on/off Boolean in the \_\_init\_\_() method of your timer so that your lap() method can pause the display while the
   3. Modify the pause() and update() methods so that the lap button pauses the display.
   4. Modify the control function so that when the lap button is clicked, it calls the pause() method.
5. Modify the pause() method to clear the stopwatch if the lap button is pressed when the stopwatch is off.
6. In the format\_time() function, create a more realistic display string, such as hh:mm:ss.ff, rather than just showing the total number of milliseconds.

In-Class Assignment 7 – *Jumping Jack Man*

For homework, you’ve sketched out the silhouettes of your jumping jack man.

In this assignment, you will animate him without user control.

*Physics Goals*:

*Programming Goals*:

1. Draw the jumping jack man at rest (arms low)
2. Draw the jumping jack man while transitioning (arms middle)
3. Draw the jumping jack man in the top position (arms high)
4. Create a set of object properties to track the state of the avatar
   1. Create properties in the \_\_init\_\_() method to:
      1. Track the state of the avatar
      2. Keep track of the time the avatar has been in that state
      3. Keep track of the length of time the state should persist before switching
   2. Increment the time in the state in the update() method
   3. In the update() method, switch to the next state if the time in that state is longer than the length of the state.
      1. Use a print statement to provide output so that you can check that it’s working
5. Use selection rules to draw the avatar in those states
   1. Use an if-elif-else structure to choose which jumping jack man to draw.
   2. If you haven’t already, modify the update function to return to the rest state every time the jumping jack man has completed a jumping jack.