Simulating Movement through a Medium

Descriptions of sound waves often include diagrams intended to represent air molecules in a tube. Waves in mechanical, acoustic, and electric systems have similar characteristics. Waves in mechanical systems are easier to visualize and therefore provide a useful starting point for understanding these characteristics. A mechanical wave machine was developed at Bell Laboratories to demonstrate these properties. This introduction to wave properties can be viewed on YouTube at: https://youtu.be/DovunOxlY1k

Characteristics of Wave Motion

A computer program can also be used to simulate movement of a wave through a medium. Imagine a series of ping pong balls in a tube. Tapping on a ping pong ball at one end of the tube causes it to bump the ball next to it. As each ball bumps the next one, the movement is transmitted to the end of the tube.

The movement to the end of the tube and back simulates the transmission of a wave through a tube. Each individual ball remains in the same relative position in the tube. The movement is transmitted through collisions between each ball and an adjacent ball. This simulation illustrates two important characteristics of wave motion: period and frequency.

1. *Period*: The period of time required for the wave to travel the length of the tube and back.

   In this simulation, each collision between a ball and the adjacent ball takes one-tenth of a second. If there are ten balls in the tube, it will take one second for the wave to ripple to the end of the tube, and another second for the wave to return to the beginning of the tube. Therefore, the period in this instance is two seconds.

2. *Frequency*: The number of round trips that the wave can complete in a given time period.

   If it requires two seconds to complete a round trip, the wave will complete 30 round trips in one minute. Therefore, the frequency in this case is 30 round trips per minute.

Increasing the number of balls and the length of the tube will increase the time that it takes for a wave to make the round trip. In other words, the period increases as the length of the tube increases (assuming that additional balls fill the additional space in the tube). If it takes longer to make a round trip, the number of trips that it is possible to complete in a given time period decreases. In other words, as the period increases, the frequency decreases.

This simulation was developed in the educational programming language *Scratch* (https://scratch.mit.edu) to illustrate these properties. A slider can be used to adjust the number of balls from 3 to 20. The green start flag is clicked to transmit a wave through the medium.

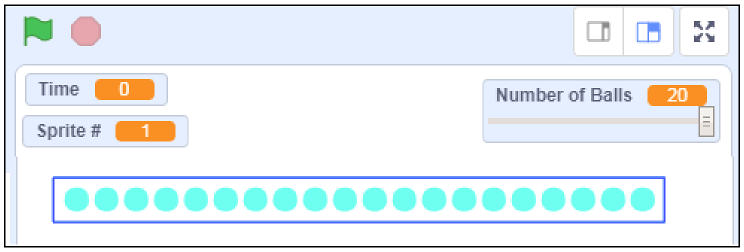


Figure 1. A wave simulation.

If you are not familiar with the Scratch programming environment, an introduction is provided here.

Scratch Graphics Editor

The Scratch programming environment provides a workspace known as the *Stage* that can be populated with actors known as *Sprites*. Sprites can wear different costumes as they move about the stage. A paint program can be used to create new costumes for sprites. In this simulation, the graphics editor was used to create a ball. Sprite 1 was given a costume consisting of this ball.

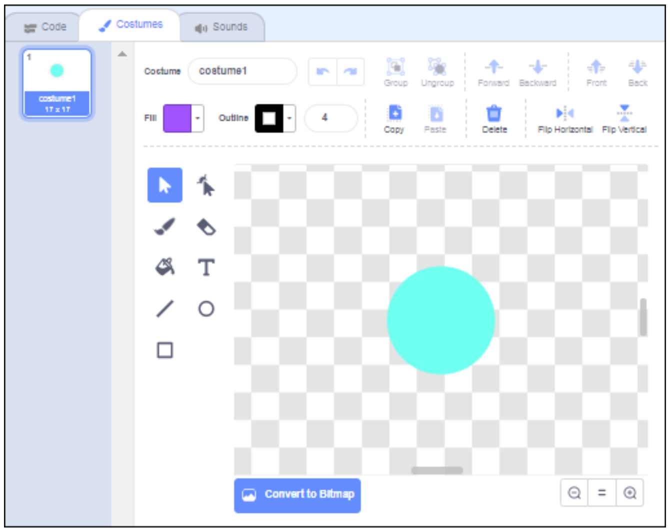
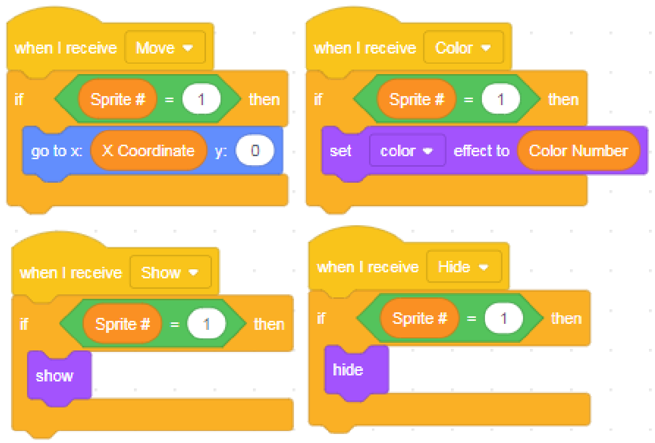


Figure 2. The Scratch paint editor.

Scripts that Guide Sprite Actions

Each sprite can be given directions through scripts that guide its actions. The scripts created for Sprite 1 provide directions that enable it to perform the following actions:

1. Move to a specified location on the X axis. (The Y position is set to 0.)
2. Set the color of the sprite.
3. Show the sprite.
4. Hide the sprite.



```
when I receive Move
if Sprite # = 1 then
    go to x: X Coordinate y: 0

when I receive Color
if Sprite # = 1 then
    set color effect to Color Number

when I receive Show
if Sprite # = 1 then
    show

when I receive Hide
if Sprite # = 1 then
    hide
```

Figure 3. Scripts that guide the actions of Sprite # 1.

Variables

The scripts created for Sprite 1 use three variables.

1. *Sprite #*: The variable *Sprite #* is used to track which Sprite is being addressed.
2. *Color Number*: The variable *Color Number* specifies a color numbered from 1 to 100. (The initial blue-green color is Color Number 10.)
3. *X Position*: The variable *X Position* specifies a location on the X axis.

Variables allow a master program to send messages to the sprite. For example, the *X Coordinate* variable can be used to tell the sprite to go to a specific location on the X axis and the *Color Number* variable can be used to change the color of the sprite.

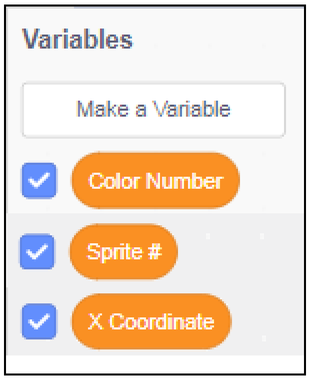


Figure 4. Variables used in the Sprite's scripts.

When a space is included in a variable name, Scratch only uses the part before the space. For example, Scratch treats *X* and *X Coordinate* as the same variable. However, inclusion of the word *Coordinate* makes the script clearer for humans who are reading it. (If it is necessary to differentiate two variables that begin with X, the words can be connected with an underscore. Scratch would treat *X Coordinate* and *X Count* as the same variable, but would treat *X_Coordinate* and *X_Count* as different variables.)

The Stage Workspace

After these three variables have been created, a procedure can be written to tell the sprite to bump the ball to its right. This procedure is placed in the script space for the *Stage*. This procedure applies equally to all of the sprites. In Scratch 3.0, the scripts for the Stage can be accessed by clicking the stage icon in the lower right-hand corner of the screen.

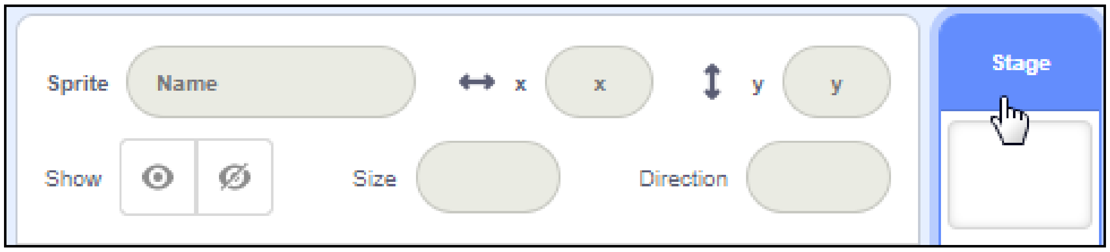


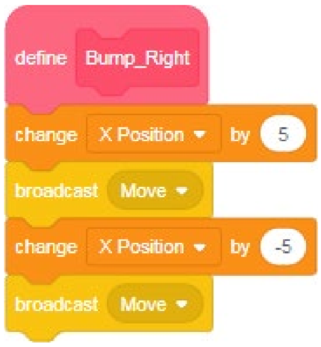
Figure 5. Click the Stage icon to access its script space.

Move Commands: The Bump Procedure

The *Bump_Right* procedure consists of the following steps:

1. Change the X Coordinate by + 5.
2. Broadcast the message *Move* to the sprite.
3. Change the X Coordinate by – 5.
4. Broadcast the message *Move* to the sprite.

These commands tell the sprite to move 5 steps to the right (bumping the sprite next to it) and then move 5 steps back to the left.



```
define Bump_Right
change X Position by 5
broadcast Move
change X Position by -5
broadcast Move
```

Figure 6. The Bump_Right procedure.

The *Broadcast* command sends a message to all sprites. When the variable *Sprite #* is set to 1, Sprite 1 knows that the message is addressed to it, and responds accordingly.

Duplicating Sprites and Their Scripts

After the *Bump_Right* procedure has been tested and verified to work with *Sprite # 1*, this sprite can be duplicated. Right-click the Sprite 1 icon in the sprite panel at the bottom of the stage to access the *Duplicate* option.

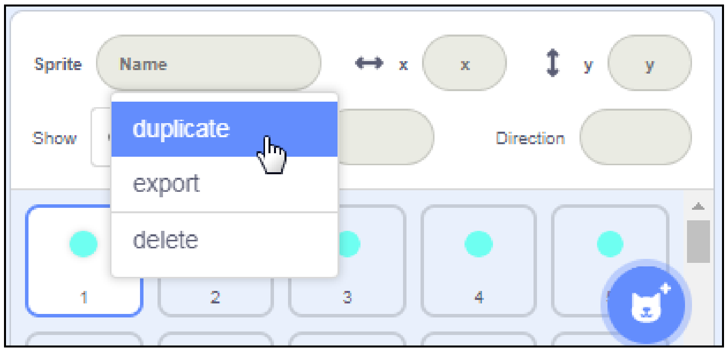
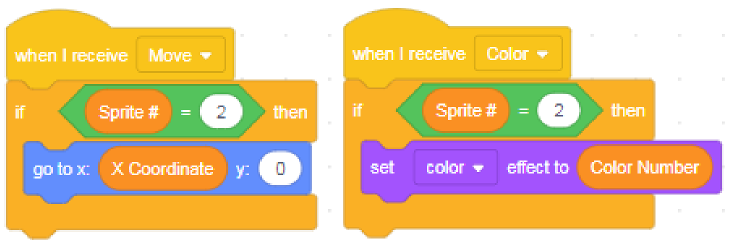


Figure 7. Duplicating Sprite # 1 and its scripts.

Sprite # 1 was duplicated to create a series of sprites wearing the same ball costume as the original sprite. The act of duplicating a sprite also duplicates all of its scripts. The script for each sprite will need to be edited to change the number in the “*If-Then Statement*” to the number of the sprite referenced. For example, the script for *Sprite # 2* would be changed to “*If Sprite # = 2*”.



```
when I receive Move
if Sprite # = 2 then
    go to x: X Coordinate y: 0
```

```
when I receive Color
if Sprite # = 2 then
    set color effect to Color Number
```

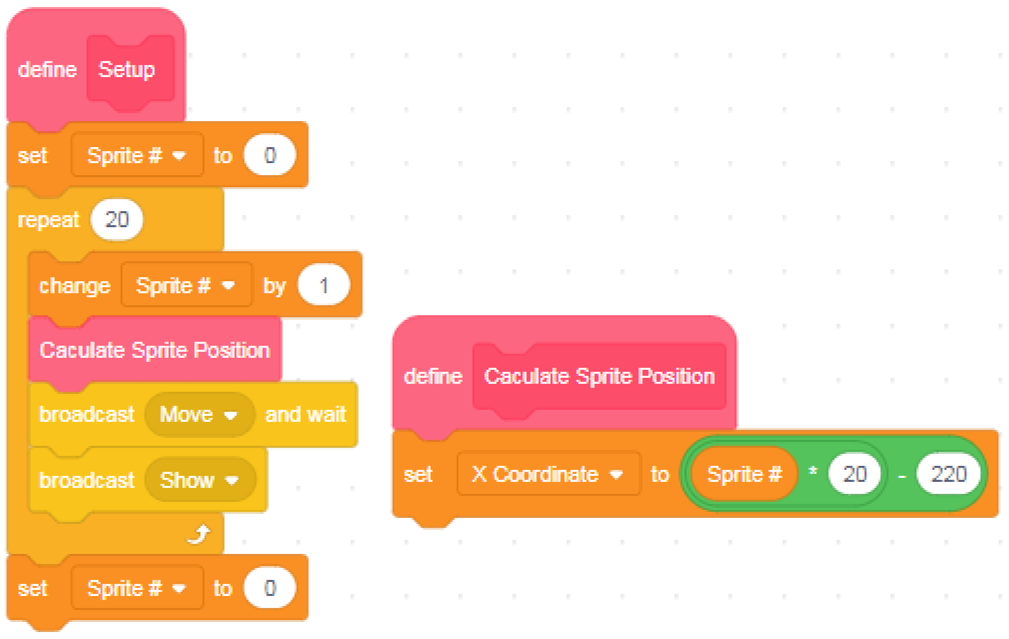
Figure 7. Revising the scripts for Sprite # 2.

Aligning the Sprites

Once all 20 sprites have been created, a procedure is needed to line them up in a row. The *Setup* procedure below accomplishes this:

1. Set the initial *Sprite #* to 0.
2. Repeat 20 times
   a. Increase the *Sprite #* by 1
   b. Calculate the *X Coordinate* associated with the sprite.
   c. Broadcast *Move*
   d. Broadcast *Show*

This procedure moves each of the sprites to the appropriate location on the X axis.



```
define Setup
set Sprite # to 0
repeat 20
    change Sprite # by 1
    Caculate Sprite Position
    broadcast Move and wait
    broadcast Show
set Sprite # to 0
```

```
define Caculate Sprite Position
set X Coordinate to Sprite # * 20 - 220
```

Figure 8. The Setup procedure aligns the sprites in a row.

The *Setup* procedure calls on another procedure, *Calculate Sprite Position*, that calculates the position of each sprite on the X axis. The formula that yields the X Coordinate associated with each sprite consists of the following:

220 – (Sprite # * 20)

For example, the calculation would be performed in this way for Sprite # 1;

220 – (1 * 20) = 200

Table 1 shows the calculations made to identify the X Coordinates associated with the first five sprites.

Table 1. Calculating the X Coordinates associated with each sprite.

Sprite #	1	2	3	4	5
X Coordinate	- 200	- 180	- 160	- 140	- 120
Calculation	220 – (1 * 20)	220 – (1 * 20)	220 – (1 * 20)	220 – (1 * 20)	220 – (1 * 20)

The Wave Procedure

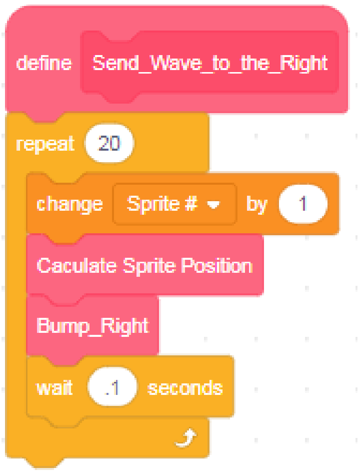
After the sprites are lined up in a row, the following procedure will send a wave to the right.

Send a Wave to the Right

Repeat 20 Times

1. Increase the Sprite # by 1.
2. Calculate the Sprite Position
3. Bump the sprite to the right of the current sprite.
4. Wait 0.1 seconds (one-tenth of a second).

Each sprite bumps the ball next to it. After this has been repeated 20 times, the ball at the end of the row will be reached. The end ball then hits the end of the tube and rebounds.



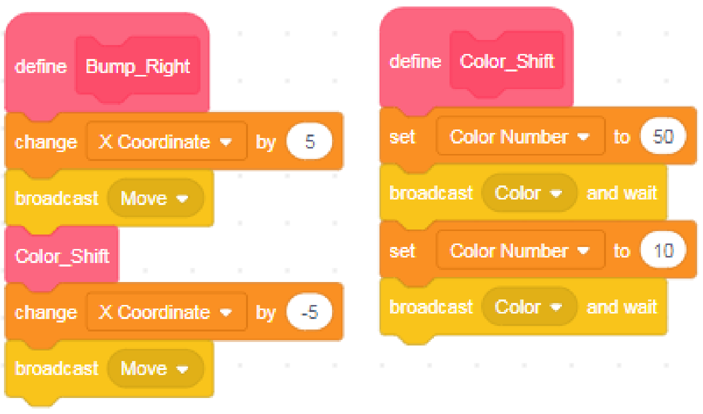
```
define Send_Wave_to_the_Right
repeat 20
    change Sprite # by 1
    Caculate Sprite Position
    Bump_Right
    wait .1 seconds
```

Figure 9. This procedure sends a wave to the right.

Changing the Color of a Sprite

Assuming that the wave procedure is working properly, other enhancements can be added. For example, the ball that is currently moving can be highlighted by shifting its color as it moves. This can be accomplished by incorporating *Color_Shift* in the *Bump_Right* procedure.

The *Color_Shift* procedure changes the color of the ball as it moves and then changes it back again. A chart of the colors available is provided in the appendix.



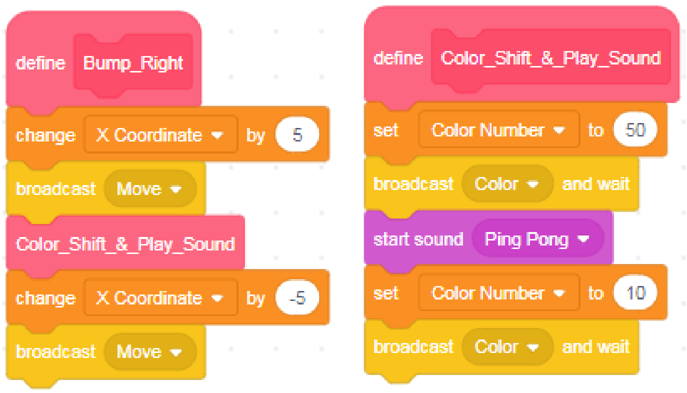
```
define Bump_Right
change X Coordinate by 5
broadcast Move
Color_Shift
change X Coordinate by -5
broadcast Move
```

```
define Color_Shift
set Color Number to 50
broadcast Color and wait
set Color Number to 10
broadcast Color and wait
```

Figure 10. The Color_Shift procedure changes the color of the sprite.

The Scratch Sound Editor

A further enhancement might include a sound that is played during the color shift. Scratch includes the capability to record a sound using the Scratch sound editor. The sound can then be played using the *Sound* command. The procedure name is updated to *Color_Shift_&_Play_Sound* to reflect the added functionality.



```
define Bump_Right
change X Coordinate by 5
broadcast Move
Color_Shift_&_Play_Sound
change X Coordinate by -5
broadcast Move
```

```
define Color_Shift_&_Play_Sound
set Color Number to 50
broadcast Color and wait
start sound Ping Pong
set Color Number to 10
broadcast Color and wait
```

Figure 11. The updated procedure also plays a sound.

Elapsed Time Counter

A time counter can also be added to the *Bump_Right* procedure. A new variable, *Time (seconds)*, is increased by 0.1 seconds each time this procedure is run. This makes it possible to display how much time has elapsed at each point in the procedure.

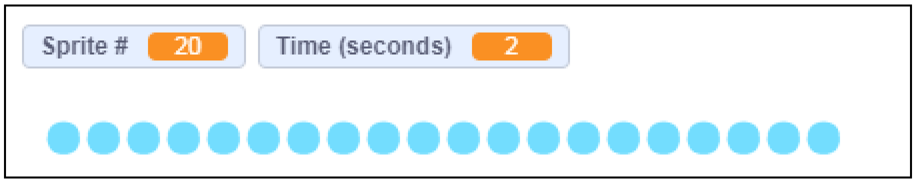


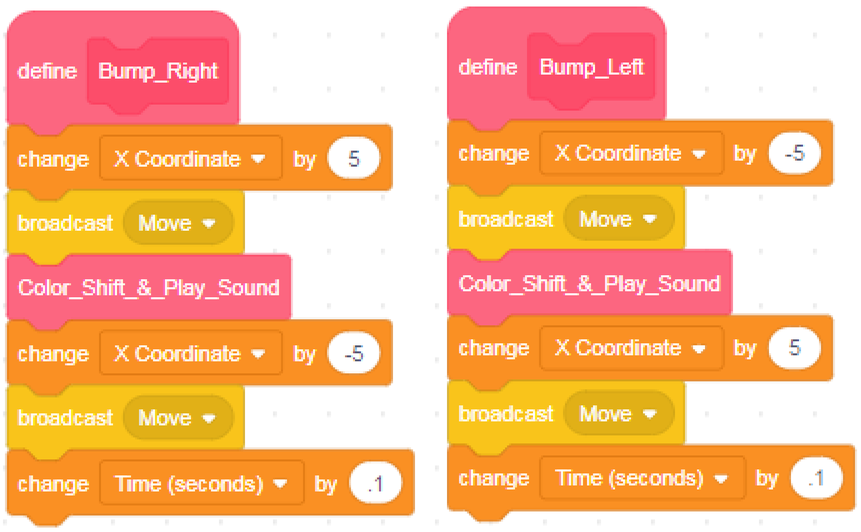
Figure 12. Tracking elapsed time (in seconds).

Completing the Round Trip

To send the wave back to the left, parallel procedures need to be created to send it the other direction:

• Send_Wave_to_Right	Send_Wave_to_Left
• Bump_Right	Bump_Left

For example, the *Bump_Left* procedure reverses the direction that the ball moves in the *Bump_Right* procredure.

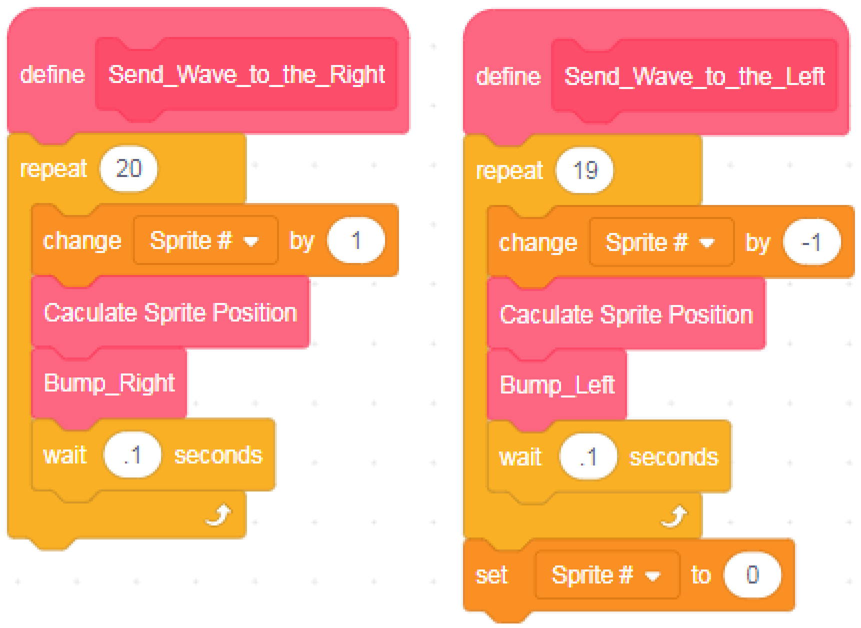


```
define Bump_Right
change X Coordinate by 5
broadcast Move
Color_Shift_&_Play_Sound
change X Coordinate by -5
broadcast Move
change Time (seconds) by .1
```

```
define Bump_Left
change X Coordinate by -5
broadcast Move
Color_Shift_&_Play_Sound
change X Coordinate by 5
broadcast Move
change Time (seconds) by .1
```

Figure 13. The Bump_Left procedure mirrors the Bump_Right procedure.

A similar process is applied to the related procedures to move the wave to the left instead of to the right.



```
define Send_Wave_to_the_Right
repeat 20
    change Sprite # by 1
    Caculate Sprite Position
    Bump_Right
    wait .1 seconds
```

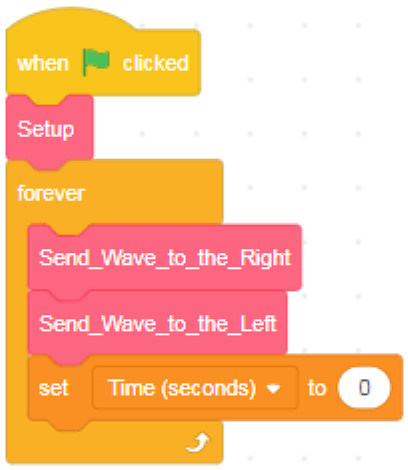
```
define Send_Wave_to_the_Left
repeat 19
    change Sprite # by -1
    Caculate Sprite Position
    Bump_Left
    wait .1 seconds
set Sprite # to 0
```

Figure 14. The Shift_Wave_to_the_Left procedure mirrors its counterpart.

The *Send_Wave_to_the_Left* procedure requires one less iteration than its counterpart to return the count to the original sprite. The *Sprite #* variable is then reset to 0. The loop (to the end of the row and back) is then ready to repeat again.

The Master Procedure

Once a parallel set of procedures has been developed to move the wave to the left as well as to the right, the master procedure uses a *Forever* loop to move the wave back and forth indefinitely.



```
when 🏳 clicked
Setup
forever
    Send_Wave_to_the_Right
    Send_Wave_to_the_Left
    set Time (seconds) to 0
```

Figure 15. The master procedure.

Adjusting the Number of Balls

This process works well if the number of balls is fixed at 20. However, one of the goals of the simulation is to demonstrate the effect that changing the number of balls has on frequency and period. (As the number of balls is increased the period increases and the frequency decreases.)

Addition of a new variable *Number of Balls* makes it possible to vary the number of balls used in the simulation. Variables can be displayed on the stage. Right-clicking the *Number of Balls* display provides access to an option to create a slider. The slider allows the user to adjust the number of balls that will be displayed. A related option allows the minimum and maximum value to be set (for example, from 5 to 20 balls).

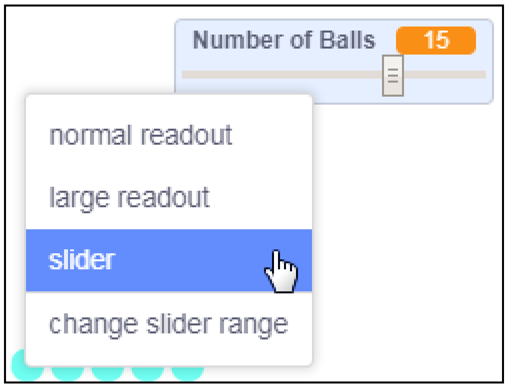
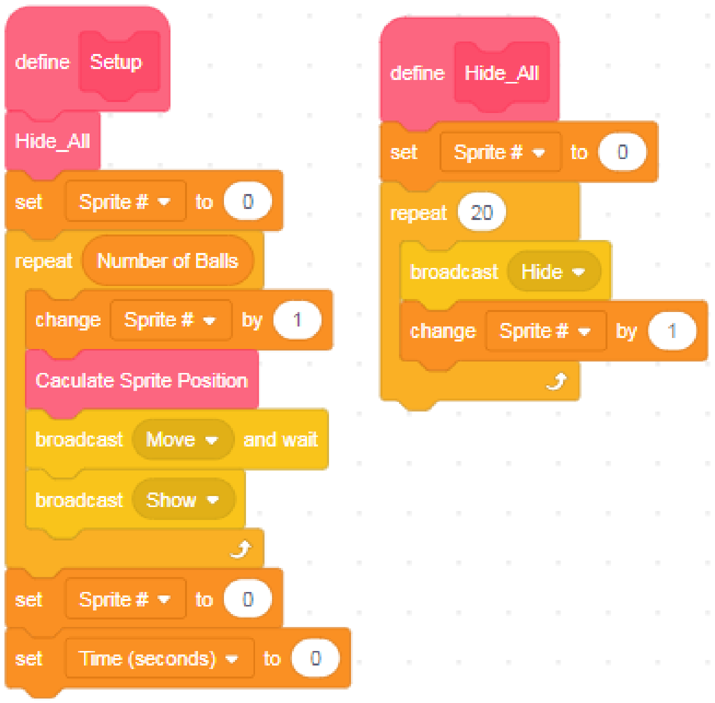


Figure 15. Creating a user-controlled slider.

The *Repeat* loop in the Setup procedure is then adjusted to line up and display the number of balls specified in the *Number of Balls* variable. A procedure *Hide_All* is used to hide all 20 balls. (The maximum number of balls that can be displayed is limited by the width of the stage).

As each ball is moved to its designated position, the *Broadcast Show* command makes each ball visible again as it is moved into place. If fewer than 20 balls are displayed, the other balls remain invisible.

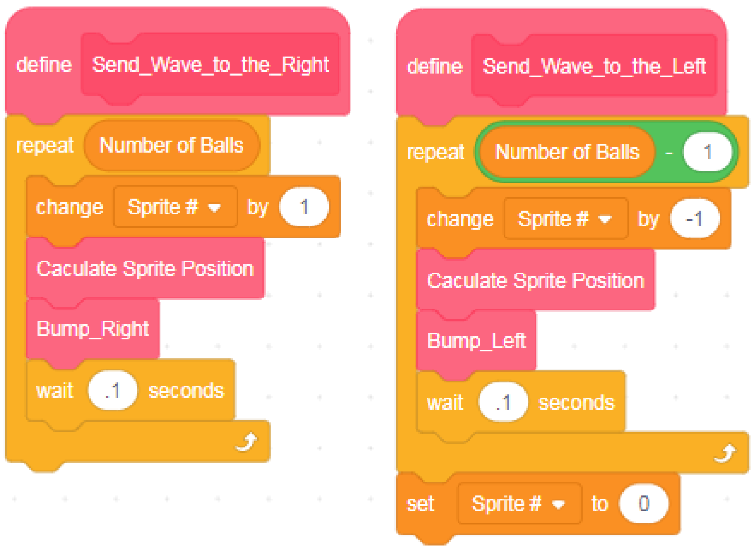


```
define Setup
Hide_All
set Sprite # to 0
repeat Number of Balls
    change Sprite # by 1
    Caculate Sprite Position
    broadcast Move and wait
    broadcast Show
set Sprite # to 0
set Time (seconds) to 0
```

```
define Hide_All
set Sprite # to 0
repeat 20
    broadcast Hide
    change Sprite # by 1
```

Figure 16. Updating the Setup procedure to reflect the number of balls chosen.

The *Number of Balls* variable is also incorporated into the *Send_Wave_to_the_Right* and *Send_Wave_to_the_Left* procedures.



```
define Send_Wave_to_the_Right
repeat Number of Balls
    change Sprite # by 1
    Caculate Sprite Position
    Bump_Right
    wait .1 seconds
```

```
define Send_Wave_to_the_Left
repeat Number of Balls - 1
    change Sprite # by -1
    Caculate Sprite Position
    Bump_Left
    wait .1 seconds
set Sprite # to 0
```

Figure 17. Updating the Send_Right and Send_Left procedures.

Creating a Tube to Contain the Balls

There are a number of other enhancements that can be added to the basic framework outlined above. For example, since the goal is to simulate balls in a tube, a box could be drawn around the balls to represent the tube in which the balls are colliding as they move back and forth.

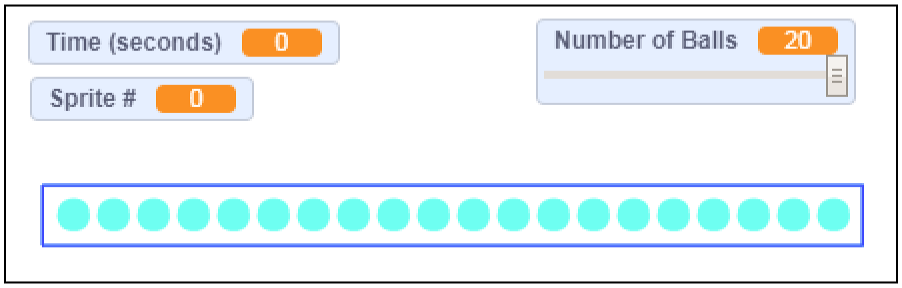


Figure 17. Drawing a box around the balls.

Remixing the Wave Simulation

The *Wave Simulation* project and associated code can be accessed at:

https://scratch.mit.edu/projects/322913865/

Scratch projects can be copied and remixed. Remixing involves revision of the code to add additional enhancement to a previously developed framework.

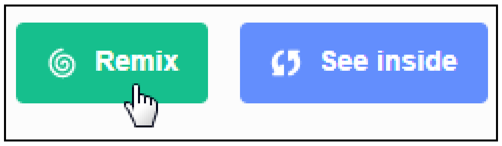


Figure 18. Remixing a project to adapt it or add new enhancements.

Computational Thinking

The processes outlined above can be described in terms of computational thinking. *Computational Thinking* refers to use of a computer to perform an action or solve a problem. Three important aspects of computational thinking involve *algorithms, abstraction, and analysis*.

1. *Algorithms*. Development of an algorithm involves identification of a pattern.

2. *Abstraction*. Once an algorithm has been identified for an activity, it can be implemented as a function – a named routine or block of code using parameters to specify different instances of the activity.

3. *Analysis*. Execution of a solution is followed by assessment and analysis of the result. Analysis and assessment can include correctness of the solution, elegance, and efficiency. In many instances, more than one solution will achieve the desired outcome, but solutions that are both efficient and easy to understand are more desirable.

For example, the core of the *Wave Simulation* program involves a procedure that causes a sprite to bump the adjacent sprite. Once code to accomplish this function was developed and tested, it was saved in the form of a procedure, *Bump_Right*. This code then became the basis of the *Send_Wave_to_the_Right* procedure, repeating the bump procedure to send a ripple down the entire row of balls.

By naming the *Bump_Right* procedure after it was tested, it was no longer necessary to remember all of the specific implementation details when it was used in the larger *Send_Wave_to_the_Right* procedure. By building and refining procedures incrementally in small steps, each piece of code could be verified before it was incorporated into the larger program.

The ability to build a library of functions in this way is a skill that contributes significantly to productivity and makes maintenance of code more efficient in the long term.

Scratch 3.0 Color Chart

The following numbers correspond to colors in the Scratch *Set Color Effect* command.

10 =	greenish-light cyan
20 =	light blue to a medium-dark blue
30 =	medium-dark blue
40 =	purplish blue
50 =	violet
60 =	light purple
70 =	purple / pink
80 =	pink
90 =	pink
100 =	pink
110 =	light orange
120 =	orange / yellow
130 =	yellow
140 =	yellow / green
150 =	light green
160 =	medium green
170 =	dark green
180 =	light green / cyan
190 =	cyan
200 =	light blue