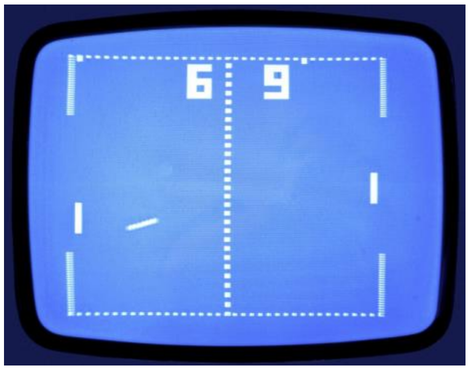
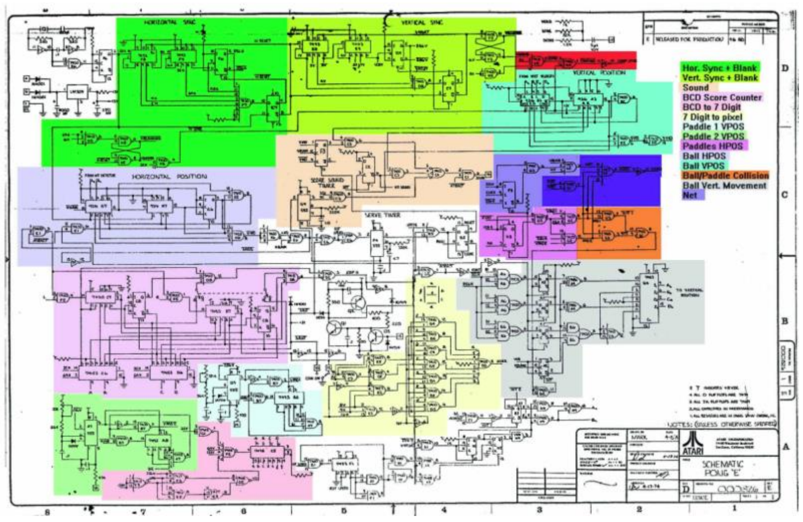
Arcade Games: Force and Motion

Glen Bull and Michael Littman

Pong was the first commercially successful video game. Pong was an electronic table tennis game released by Atari and helped establish the video game industry. Due to its cultural significance, Pong is now part of the Smithsonian's permanent collections.



When the original Pong game was developed by Atari, the game was created using discrete digital logic chips. In the color-coded schematic, the logic chips responsible for the vertical and horizontal positions of the game paddles are highlighted in green and purple. The logic chips that detect a collision between the ball and a paddle are highlighted in orange.

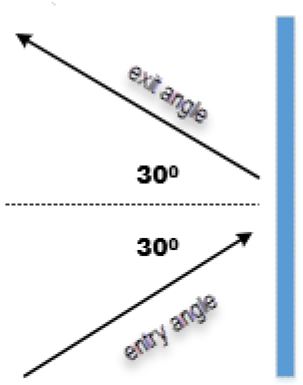


When the Atari Pong game was developed in 1972, computers were too expensive to devote a single computer to a game. Today small microcomputers like the Raspberry Pi microcomputer capable of implementing the Pong can be purchased for $25.

In this module, a Pong game will be created using *Snap!* Writing a program to recreate Pong provides opportunities to explore concepts related to force and motion. Key concepts include (1) motion is described by an object's speed and direction, (2) changes in motion are related to force and mass, and (3) friction is a force that opposes motion.

Ball Strikes the Paddle

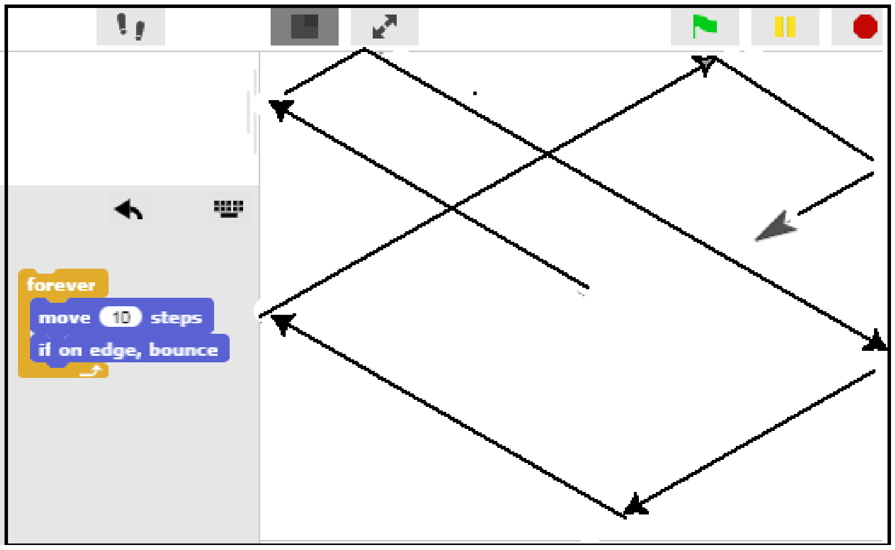
In this example, one sprite (labeled *Turtle*) will rebound when it strikes a second sprite (labeled *Paddle*). When an object such as a billiard ball strikes a surface, its angle of incidence (entry angle) when it strikes the surface is equal to its angle of reflection (exit angle) when it rebounds from the surface.



Snap! has a built-in command **If on Edge, Bounce** that automatically applies the correct angle of reflection when the turtle reaches the edge of the stage. This command is useful for preventing the turtle from inadvertently disappearing off the edge of the stage. The following commands will cause the turtle to travel across the stage, bouncing when it reaches an edge:

Forever [Move 10; If on Edge, Bounce]

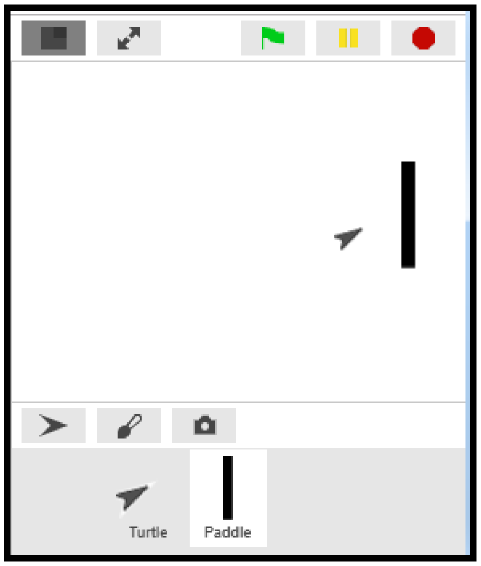
If the pen is down, the turtle will leave a trail that makes it possible to follow its path as it travels from one edge of the screen to the next.



```
forever
  move 10 steps
  if on edge, bounce
```

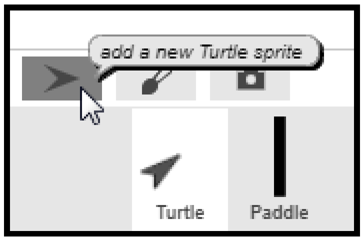
Creating the Game Paddle

In this recreation of the Pong game, the *Turtle* will travel across the screen until it strikes a second sprite (labeled *Paddle*).

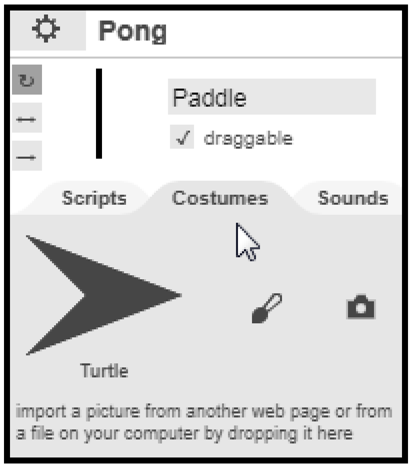


Creating the Paddle

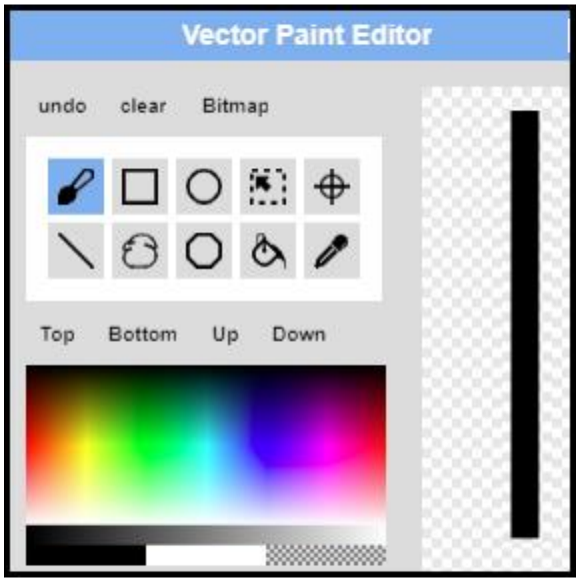
Click on the **Sprite** icon below the stage to create a new sprite.



Name the newly created sprite "Paddle" and then select the *Costumes* tab to access the *Paint Editor*.

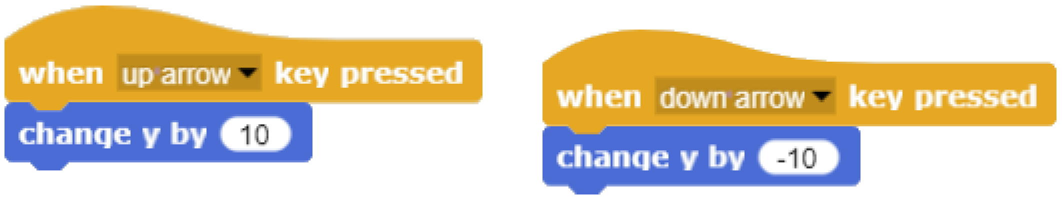

Pong
Paddle
draggable
Scripts
Costumes
Sounds
Turtle
import a picture from another web page or from a file on your computer by dropping it here

Use the *Paint Editor* to create a vertical bar that will become the game paddle. The *Paint Editor* has tools to adjust the size of the game paddle as needed.



Moving the Paddle

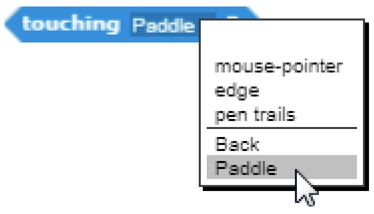
The game paddle sprite should be placed near the back wall of the stage. The *Up* and *Down* arrow keys on the key board can be used to move the game paddle up and down. The **When Up Arrow Key Pressed** and the **When Down Arrow Key Pressed** blocks (found in the *Control Palette*) can be to determine when the up and down arrow keys have been pressed.



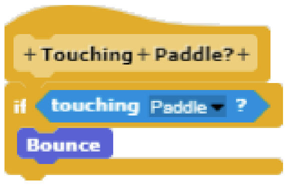
When the *up arrow* key is pressed, the **Change Y by 10** block is used to move the paddle upward. When the *down arrow* key is pressed, the **Change Y by -10** block is used to move the paddle downward.

Collision Detection

When the turtle strikes the paddle, it should rebound. To cause this to occur, a means of detecting when the turtle is touching the paddle is needed. One of the commands in the *Sensing* palette can be used to detect this. A drop-down menu can be used to select other objects on the stage.



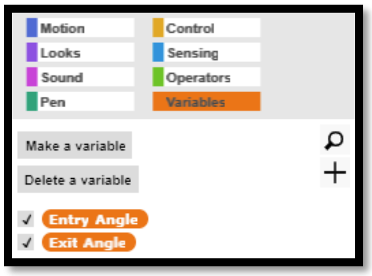
This command can be used to create a *Touching Paddle?* procedure. The *Touching Paddle?* procedure checks to see if the turtle is touching the paddle. When the turtle touches the paddle, the *Bounce* procedure is called.



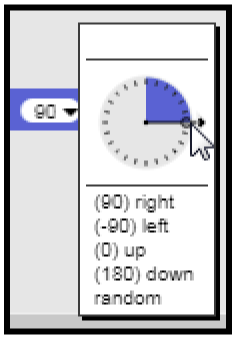
```
+Touching+Paddle?+
if touching Paddle ?
  Bounce
```

The Bounce Procedure

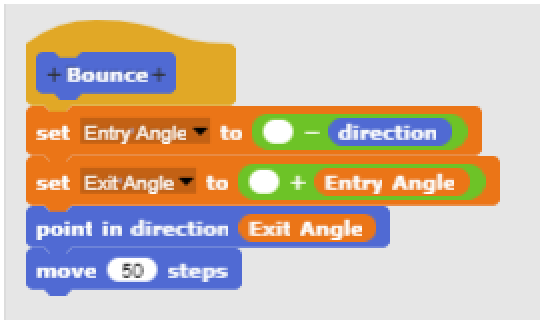
The *Bounce* procedure uses two variables: (1) Entry Angle and (2) Exit Angle. The *Entry Angle* refers to the angle of a moving object at the moment at which it strikes a surface. The *Exit Angle* refers the angle of the object at the moment at which it rebounds and exists the surface. (In geometry, the term “Angle of Incidence” is used to refer to the entry angle, and the term “Angle of Reflection” is used to refer to the exit angle.



The command *Direction* (located under the *Motion* commands palette) can be used to determine the direction in which the turtle is pointed. If the turtle is pointed straight up, the command *Direction* returns a value of 0 degrees. If the turtle is pointed straight down, the command direction returns a value of 180 degrees. If the turtle is pointed to the right, the command *Direction* returns a value of 90 degrees.



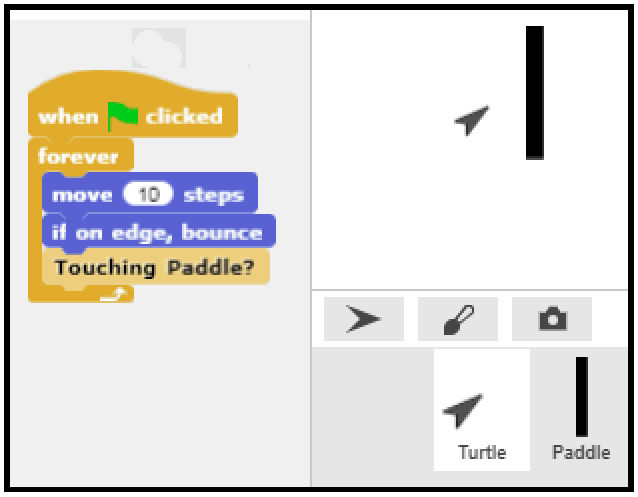
The turtle’s direction can be used to determine the *Entry Angle* as it travels toward the paddle. The Entry Angle, in turn, can be used to determine the *Exit Angle*. When the *Touching Paddle?* command detects that the turtle has touched the paddle, the *Bounce* procedure uses the *Exit Angle* to point the turtle in the right direction as it rebounds.


+ Bounce +
set Entry Angle to (− direction)
set Exit Angle to (+ Entry Angle)
point in direction Exit Angle
move 50 steps

The *Bounce* procedure essentially performs the same calculations when the paddle is touched as the built-in *If on Edge, Bounce* command uses when the edge of the stage is reached. By running the procedure with the pen down, it is possible to determine if the entry angle when the turtle strikes the paddle is the same as the exit angle when the turtle reaches the edge of the stage and rebounds. If both exit angles (for the edge of the stage and the paddle) are the same, then the variable *Exit Angle* has been correctly calculated.

The Master Procedure

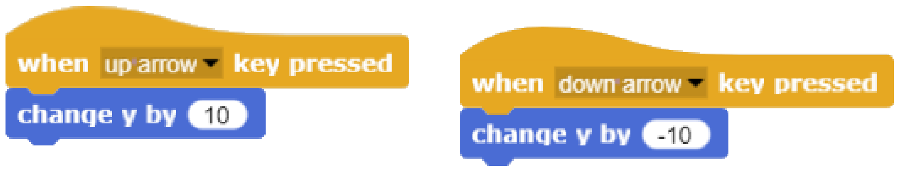
The master procedure (located in the turtle's script space) should look like this. The turtle should be turned to a starting position in which it is facing the paddle at an angle, as shown in the illustration.


when clicked
forever
move 10 steps
if on edge, bounce
Touching Paddle?

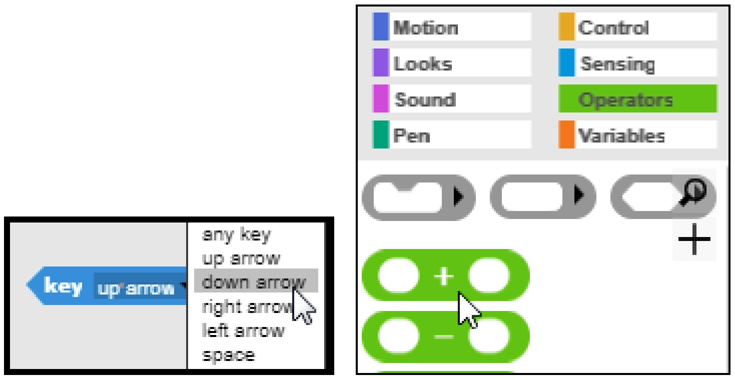
When the *Green* go flag is clicked, the master procedure will move the turtle across the screen until it touches the paddle or the edge of the screen. When the turtle touches the paddle, the *Bounce* procedure will cause it to rebound.

Increasing the Responsiveness of the Paddle Control

The blocks that detect a keypress check the keyboard approximately three times a second. In most cases, this rate is adequate. If the arrow keys are used to control the movement of the paddle, the responsiveness of the paddle can be improved by using an enhanced version of the procedure that checks the keyboard more frequently.



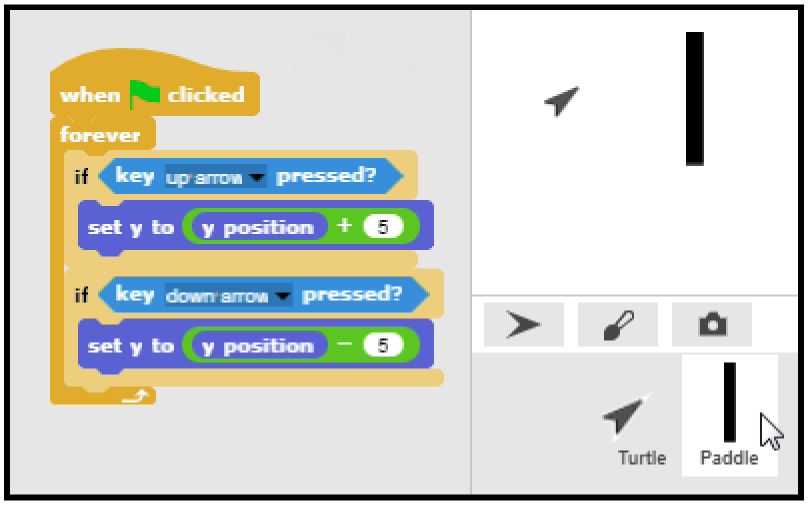
In the enhanced version, the *Key* command (located under the *Sensing* command palette) is still used to detect a keypress.



However, the keypress detection block is embedded in a procedure specifically written to control the paddle. The logic of the enhanced procedure that moves the paddle states that "If the *Up Arrow* key is pressed, then set the Y (vertical) position of the paddle to its current position plus 5 more steps." This effectively moves the paddle up 5 turtle steps each time that the *Up Arrow* key is pressed. (The *Addition* code block is found under the *Operators* palette.)

If *Key Up Arrow* is Pressed?
Then Set Y to Y Position + 5

Similar logic is used to move the paddle down when the *Down Arrow* key is pressed. Note that this script must be placed in the script space for the paddle sprite (and not the turtle sprite). This script space can be accessed by clicking the Paddle icon below the stage.

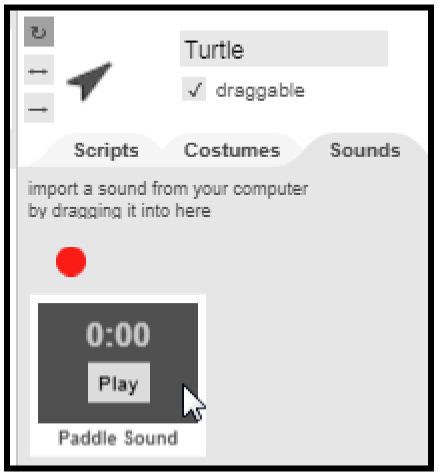


Additional Enhancements - Sound

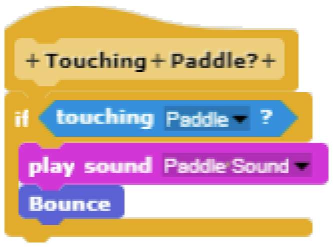
The basic program logic and procedures are now in place. Several enhancements will increase the play value. The Atari Pong game is instantly recognizable because of the electronic blip that occurs when the ball strikes the paddle.

Pong Sounds Folder

Select the *Sounds* tab of the turtle. Then drag the sound labelled *Paddle Sound* from the *Pong Sounds Folder* into sounds space of the turtle sprite.



Once the paddle sound is available, the *Touching Paddle?* script can be updated to play the Pong paddle sound each time the turtle sprite collides with the paddle.



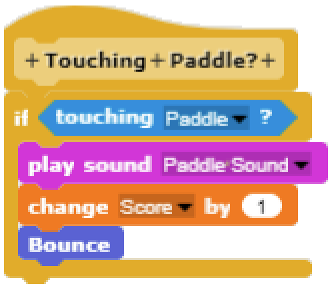
```
+Touching+Paddle?+
if touching Paddle ?
    play sound Paddle Sound
    Bounce
```

Enhancements - Scoring

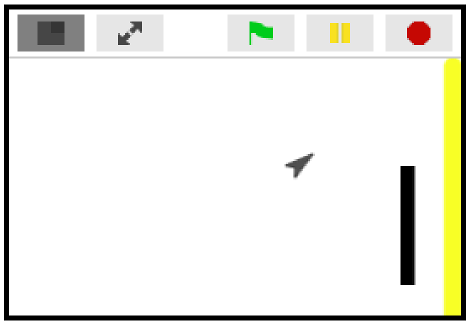
A method for maintaining the score is another important enhancement. A variable named *Score* could be added to track the score. The score could be increased by 1 point each time the player successfully block the turtle with the paddle by adding the code:

Change Score by 1

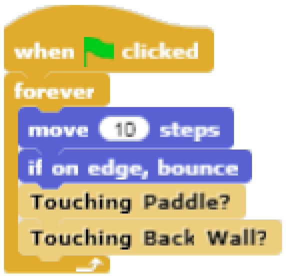
to the Touching Paddle? script. This variable would need to be initialized by setting its value to 0 at the beginning of the game.

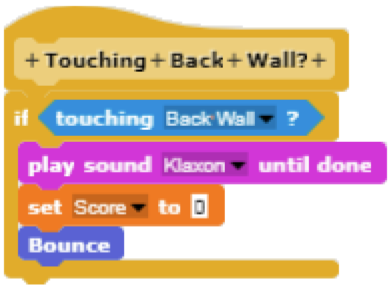

```
+Touching+Paddle?+
if touching Paddle ?
    play sound Paddle Sound
    change Score by 1
    Bounce
```

A third sprite, labeled “Back Wall” could be added to detect when the paddle does not block the turtle.



A *Touching Back Wall?* procedure could detect when the turtle touches the back wall.

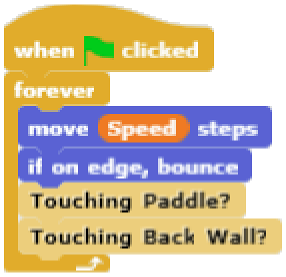

```
when clicked
forever
    move 10 steps
    if on edge, bounce
    Touching Paddle?
    Touching Back Wall?
```


```
+Touching+Back+Wall?+
if touching Back Wall ?
    play sound Klaxon until done
    set Score to 0
    Bounce
```

When the turtle escapes the paddle and touches the back wall, the *Touching Back Wall?* procedure could reset the score to zero (or end the game with a *Game Over* display).

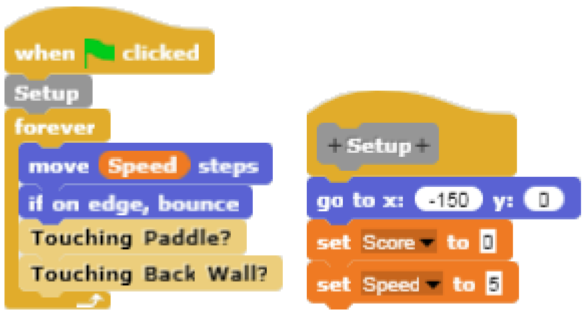
Speed

The number of steps that the turtle moves during each iteration of the *Forever* loop determines how fast it moves across the screen. Creation of a variable named *Speed* will make it possible to adjust the speed of the turtle during game play.


```
when clicked
forever
    move Speed steps
    if on edge, bounce
    Touching Paddle?
    Touching Back Wall?
```

For example, in one variation of the recreated Pong game, the speed is increased each time the player successfully deflects the turtle five times in a row. Eventually the turtle may move so fast that the player cannot block it in time. The score at that point becomes the player's high score.

At this point it may be useful to create a *Setup* procedure to establish the turtle's initial starting location, speed, and score.



```
when [flag] clicked
Setup
forever
    move (Speed) steps
    if on edge, bounce
    Touching Paddle?
    Touching Back Wall?
```

```
+ Setup +
go to x: -150 y: 0
set Score to 0
set Speed to 5
```

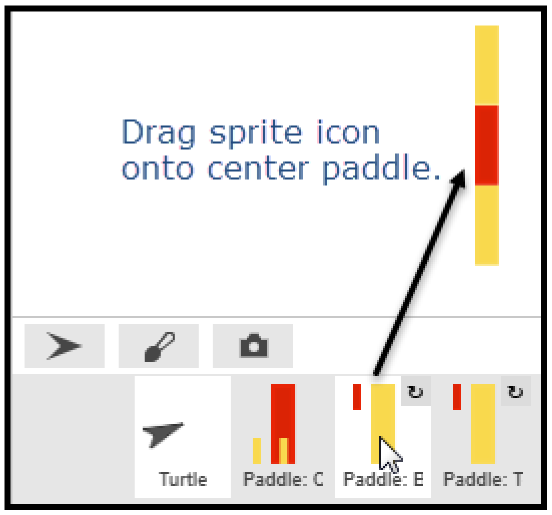
Paddle Surface

Until this point, the assumption has been that no energy is lost in the collision between the turtle and the paddle surface. In actuality, some energy is always absorbed in a collision of this kind. A softer, spongy surface will absorb more energy than a softer surface.

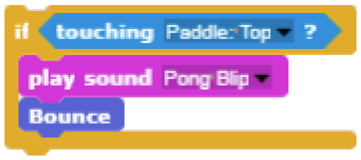
To reward players for accuracy, a composite paddle can be constructed with harder surfaces on the edges and a softer, more absorbant surface in the center. In the illustration, the harder surfaces on the edges are depicted in yellow while the softer, absorbant surface is depicted in red.

To construct a composite paddle, create two more paddle segment sprites. One sprite will be come the top paddle segment and the other sprite will become the bottom paddle segment. Make sure that there are no scripts in the script areas of these two additional sprites. Then line up the top and bottom paddle segments so that they are aligned with the center paddle segment.

Once the paddle segments are aligned, drag the top paddle segment icon (below the stage) onto the center paddle segment on the stage. Repeat the process for the bottom paddle segment. Once this is done, all three segments will move together as a single unit.

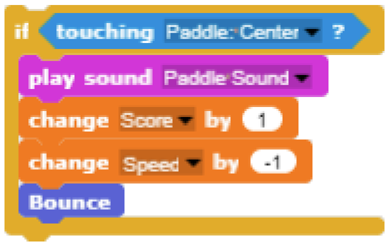


In this illustration, the top paddle sprite has been named *Paddle: Top* and the bottom paddle sprite has been named *Paddle: Bottom*. When the turtle strikes the top or bottom paddle segments, the program plays a sound and turtle rebounds, but the score is not increased.



```
if <touching [Paddle: Top v] ?>
    play sound [Pong Blip v]
    Bounce
```

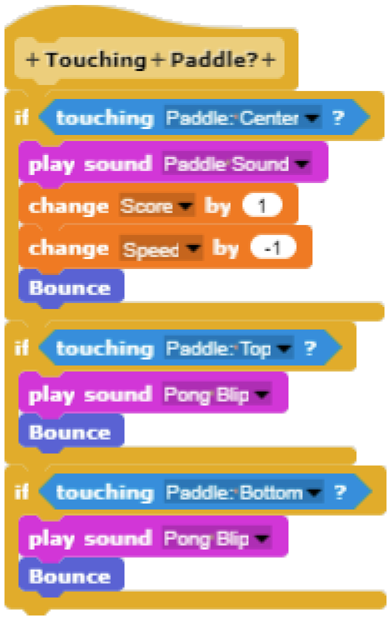
If the turtle collides with the center paddle segment, the player is rewarded for accuracy. The score is increased by one and the speed is decreased, making the game easier to play.



```
if <touching [Paddle: Center v] ?>
    play sound [Paddle Sound v]
    change [Score v] by (1)
    change [Speed v] by (-1)
    Bounce
```

The updated *Touching Paddle?* procedure:

1. Checks to see if the center paddle segment has been touched, and increases the score and decreases the speed before rebounding if this occurs.
2. Checks to see if the top paddle segment has been touched, but does not increases the score or change the speed before rebounding if this occurs.
3. Checks to see if the bottom paddle segment has been touched, but does not increases the the score or change the speed before rebounding if this occurs.



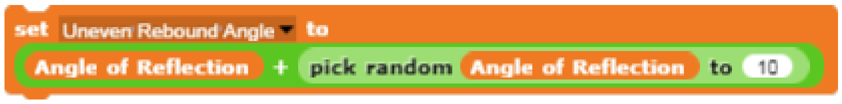
```
+ Touching + Paddle? +
if <touching [Paddle: Center v] ?>
    play sound [Paddle Sound v]
    change [Score v] by (1)
    change [Speed v] by (-1)
    Bounce
if <touching [Paddle: Top v] ?>
    play sound [Pong Blip v]
    Bounce
if <touching [Paddle: Bottom v] ?>
    play sound [Pong Blip v]
    Bounce
```

There are other variants that could be implemented. For example, assuming that the outer segments are constructed from brittle material, a counter could be used to count the number of times each segment on the edge of the paddle collides with the turtle. After an edge segment collides with the turtle a specified number of times, it could break off, leaving the player with a smaller paddle.

Angle of Reflection

Thus far an even surface has been assumed. Consequently, the turtle rebounds in the same predictable each time. The angle of reflection is always equal to the angle of incidences. For a more interesting, less predictable game, an uneven paddle surface could be created. This would result in some variation in the angle at which the turtle rebounds.

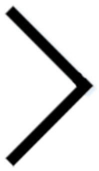
To accomplish this, a variable named *Uneven Rebound Angle* could be created that was set to the Angle of Reflection plus a random number between -10 and +10. Pointing the turtle in the direction of the Uneven Rebound Angle created in this way would introduce some variation in the angle at which the turtle rebounds from the paddle.



```
set Uneven Rebound Angle to
  Angle of Reflection + pick random Angle of Reflection to 10
```

Retroreflection

Paddles with different shapes would cause the turtle to rebound at different angles. For example, a "V" shaped paddle would cause the turtle to rebound at an angle of reflection 180 degrees from the angle of incidence.



Light rays have similar properties. An optical device that reflects light back to its original source is known as a retroreflector.

Retroreflection could be simulated in *Snap!* using mathematical calculations similar to those used to create the Pong game.