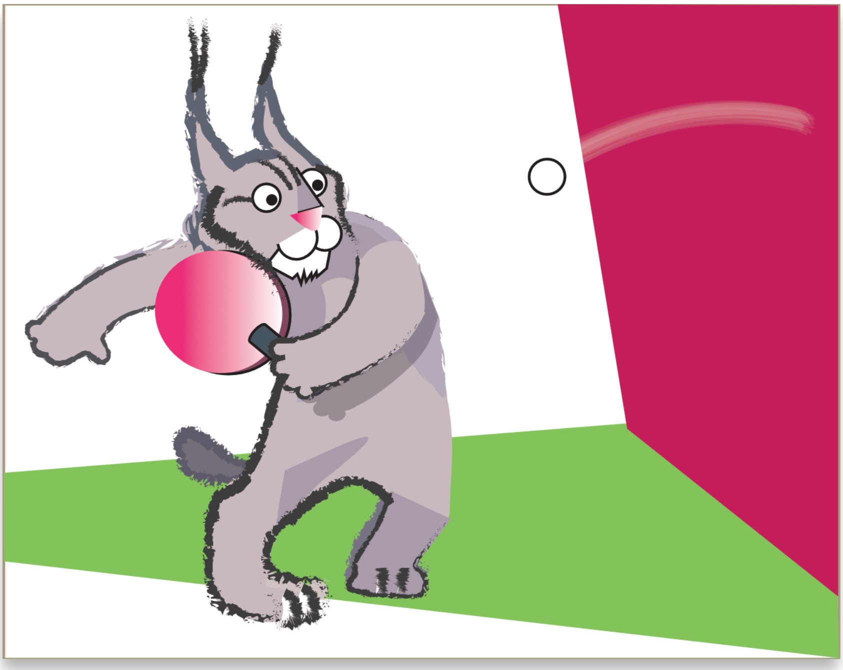
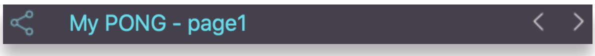
Project 6 - Pong!



Another classic game! In this version of Pong, you will play against the computer. You will control the paddle, and the ball will bounce on a wall on the other side of the page. This advanced project introduces **colour detection**, **collision detection**, and **mouse interaction**. Let's start! You are, after all, making a game that your friends can play!

Start a new project, and give it a new name now. Don't forget to save once in a while!



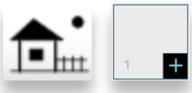
Create a ball

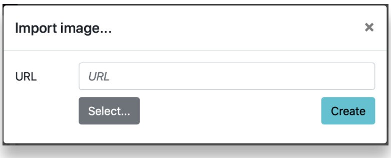
First, you need a ball. You can draw one using any paint program you may have, and save it as a PNG file *with transparency* around the ball. Or, you can download a ball clipart from a web site (see *Appendix D - Finding Things*). Look for a small file size, under 50 K.

If you don't use a PNG file with transparency, the ball will be round, but it will look like a white block when you place it on a nice background. Read more about this in *Appendix D*.

When you have a ball image, it's time to import it into your project:

- Open the Clipart Pane.
- Click anywhere in an empty spot.
- Click on the "**+**" to open the Import dialog box





- Click on **Select** to locate your your Ball file on your computer, then click on **Create**.
- There, you have it! Remember the number for the clipart.

A Ball-Turtle

If you don't have a turtle on the page, create one now. Choose **Turtle** in the + menu. Set its heading to 75 degrees. Type this in the Command Centre:

```
setheading 75
```

OR USE SETH 75

Give it the shape of a ball. You can either run this instruction in the Command Centre:

```
setshape 1
```

USE YOUR CLIPART NUMBER

or click on the Ball clipart to turn the mouse pointer into a hand, then click on the turtle. Click and Click, not Click and Drag!

Your clipart image may be too large. If the ball looks huge on the page, use an instruction such as this one to change its size:

```
setsize 10
```

LESS THAN 40 TO MAKE IT SMALLER,
MORE THAN 40 TO MAKE IT LARGER.

Get That Ball Moving!

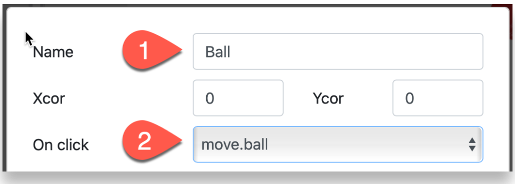
Create this procedure to make the ball glide in the play area. You will code the bouncing later.

```
to move.ball
ball,
forever [forward 10 wait 1]
end
```

Now right-click on the ball to open its dialog box.

1) name this turtle **Ball** (remember, no space, not even before or after the word), and

2) choose `move.ball` in the **On click** menu.



Name	Ball		
Xcor	0	Ycor	0
On click	move.ball		

Close the dialog box and click on the ball to test it. You can always come back to this procedure to change the speed of the ball. You know what number to change for speed, right?

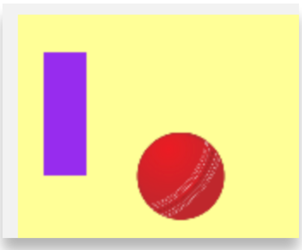
Create a paddle

A Paddle-turtle

The routine is similar to the ball routine. Use any paint program, or take a screenshot. This time, since the paddle is a full rectangle, the transparency around the paddle is not an issue.

- Create the clipart file.
- Load the clipart into an empty spot in the Clipart Pane. Remember you can also Copy the clipart file and Paste it into an empty spot.
- Create a new turtle.
- Give the paddle clipart to the turtle using `setshape` or Click and Click.
- Resize the paddle-turtle if needed.

At this point, you should have a ball and a paddle on the page.



Controlling the Paddle

This is the key element of the game: in order to play, the paddle must "follow" your mouse pointer. Right now, let's do this in a simple way - the Going Further section, at the end of the chapter, contains suggestions to improve the interaction.

Type this in the Command Centre:

```
show mousepos                  MEANS MOUSE POSITION, PRESS ENTER
-312.75 -497.671875
```

You will get a set of two numbers, the X and Y coordinates, based on the position of your mouse pointer on the screen. Your numbers will be different.

Now type this in the Command Centre:

```
forever [setpos mousepos]
```

Move the mouse pointer across the work area. The paddle should be following you, as long as you stay in the page.

You see, `mousepos` reports a set of two numbers, and `setpos` uses these numbers to set the turtle's position.

Is the ball following the mouse?

If the ball, not the paddle, is following the mouse, that's because the ball is the current turtle. Stop everything, click on the paddle to "make it current", i.e. the turtle that will listen to commands right now. Run the instruction again.

The current turtle is:

(a) the last turtle that you have created, or

(b) the last turtle that you have clicked on, or

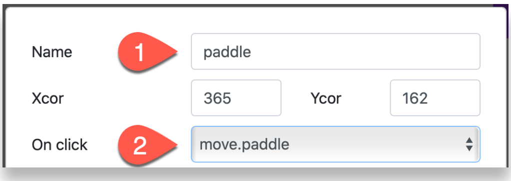
(c) the last turtle that you called using the "comma" method (`paddle,`).

Click on the **Stop All** button to stop the `forever` instruction.

Create this procedure to make the paddle always follow the mouse position while playing:

```
to move.paddle
paddle,
forever [setpos mousepos]
end
```

Now right-click on the paddle to open its dialog box. 1) name this turtle **Paddle**, and 2) choose `move.paddle` in the **On Click** menu.



Name 1 paddle

Xcor 365 Ycor 162

On click 2 move.paddle

Bounce off the paddle

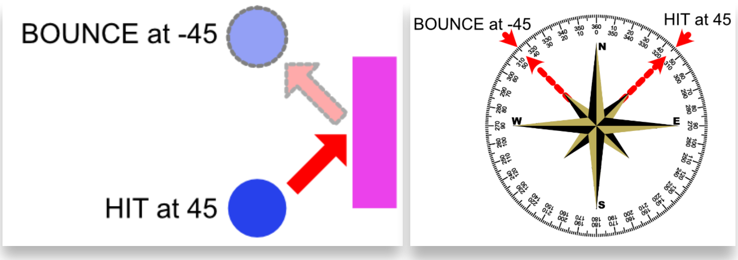
The ball and the paddle are both turtles. Lynx can detect collisions between turtles. When they collide, the ball should bounce.

There are many ways to code the bounce. It is all about the heading **before** the collision, and the heading **after** the collision. Here are five different possibilities that could happen to the ball after colliding:

```
rt 180
setheading minus heading
setheading heading + 180
setheading 360 - heading
setheading 180 + random 180
```

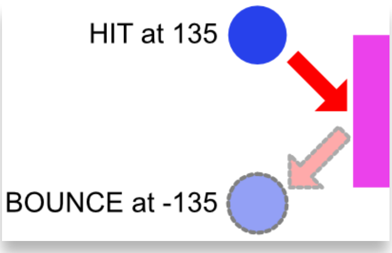
They are all worth trying and understanding, but we will focus on this one for the moment: `setheading minus heading`.

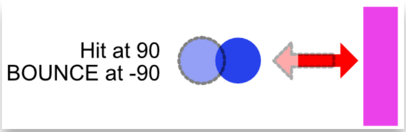
Consider this example: the ball is coming from the left and hits the paddle with a heading (on a compass) of 45 degrees:



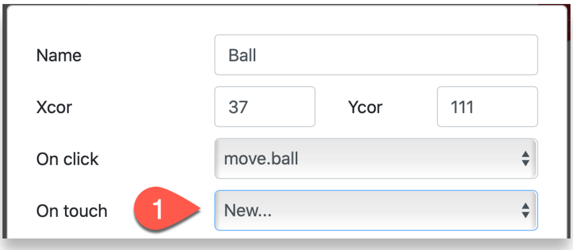
On a compass, **0** degrees is **North**. **45 degrees** is **North-East**. If you use negatives, you simply go counterclockwise: **minus 45 degrees** is **North-West** - it is just the **mirror** of 45 degrees.

And that works for other angles too!



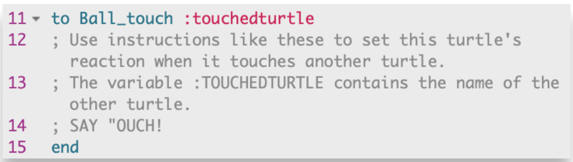


When should you execute the bounce procedure? When the ball hits the paddle! Right-click and open the ball's dialog box and choose **New** in the **On touch** menu:



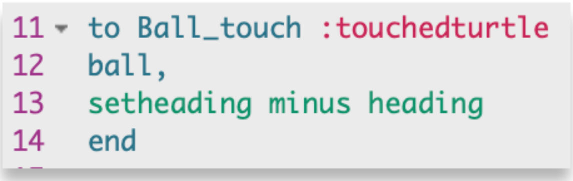
Name	Ball		
Xcor	37	Ycor	111
On click	move.ball		
On touch	New...		

This creates a procedure like this one in the Procedures Pane:



```
to Ball_touch :touchedturtle
; Use instructions like these to set this turtle's
  reaction when it touches another turtle.
; The variable :TOUCHEDTURTLE contains the name of the
  other turtle.
; SAY "OUCH!
end
```

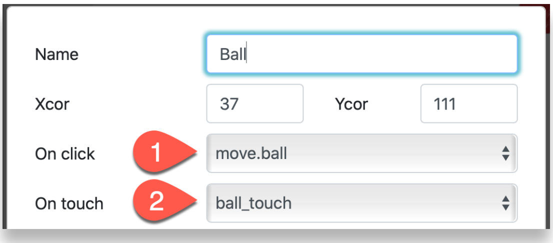
The grey text is just comments to help you understand Collisions. Read them and delete them. Then, fill in the procedure as follows:



```
to Ball_touch :touchedturtle
ball,
setheading minus heading
end
```

You don't have to worry about who is the touched turtle, it is always going to be the paddle *as it is the only other turtle in this game*.

The Ball's dialog box now looks like this, with both an **On click** instruction and an **On touch** instruction:



Name	Ball		
Xcor	37	Ycor	111
On click	move.ball		
On touch	ball_touch		

First big test

Set everything into action. Type this in the Command Centre:

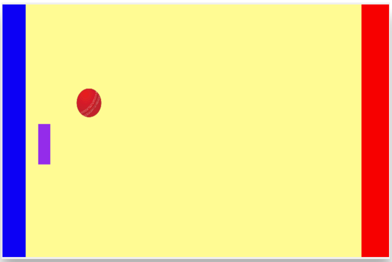
```
everyone [clickon]
```

`Everyone` means "every turtle on the page" and `clickon` means "act as if you got a mouse click".

The ball should start gliding. Move the mouse around the page, the paddle should track the mouse pointer. Place the paddle in the path of the ball. Did you get a bouncing action? If yes, it is working!

Build some walls

Here's a plan before we start building walls.



•The play area has a paddle and a ball.

•The ball will bounce on the right (red) wall, and each time, you get a point.

•The ball heads left, and if it hits the paddle, it stays in play. If you miss, the ball will hit the left (blue) wall, and you lose a point.

- Your score starts at 10. If you hit 20 points, you win the game. But if you get down to 0, it's game over!

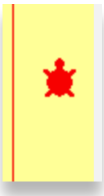
- Why not paint the background a nice colour? Use the Lynx Colour chart in *Appendix C* and remember the number. Then type this in the Command Centre:

- `setbg 42` OR WHATEVER NUMBER YOU CHOSE

Here's a quick way to draw the walls.

- Create a new turtle.
- Drag it near the right edge of the page.
- Type this in the Command Centre:

```
setcolour 'red'
pendown
forward 9999
```



- This will draw a thin red line as shown on the right.
- Now drag the turtle to the right of the red line, and execute this instruction:

`fill` FILLS THE AREA WITH THE TURTLE'S CURRENT COLOUR

- Type `setcolour 'black'` to set the turtle back to **black**, so you can see it again (can't see red on red).You should have a red wall now.

Do the same for the left blue wall.

- Drag the turtle near the **left** edge of the page, and execute these instructions in the Command Centre:

```
setcolour 'blue'
pendown
forward 9999
```

- Drag the turtle to the left of the blue line, and execute this instruction:

`fill`

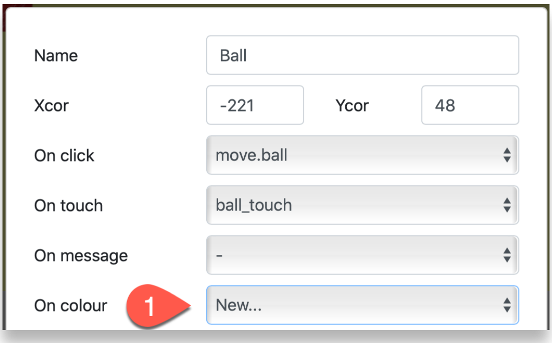
Do you have something like the image on the previous page?

Type `setcolour 'black'` to set the turtle back to **black**, so you can see it again. Right-click on this turtle and use the garbage can to delete it.

Count the good hits on the red wall

Colour detection

First, let's create the procedure that deals with colour detection. Open the ball's dialog box and choose **New** in the **On colour** menu.



Name	Ball
Xcor	-221
Ycor	48
On click	move.ball
On touch	ball_touch
On message	-
On colour	New...

This creates a procedure such as this one:

```
to Ball_oncolour :prevColour :newColour
  ; Use an instruction like this to do colour detection
    every time
  ; Moving from any colour to red, even red to red, will
    trigger the action.
  ; Pick your own colour name and instructions instead of
    [BACK 10 RIGHT 180]
  ; IF :NEWCOLOUR = "RED [BACK 10 RIGHT 180]
end
```

Again, the grey text is just comments. Read and delete them, then fill in this code:

```
to Ball_oncolour :prevColour :newColour
if :newColour = 'red' [hit]
if :newColour = 'blue' [miss]
end
```

`Hit` and `miss` are two procedures that **do not exist yet**. Don't worry, we're getting there. Before we do, let's examine `Ball_oncolour`: When the turtle moves, it always checks what colour it is currently on. That is the `:newColour`. If it touches the red wall, the value of the variable `:newColour` will be `'red'`.

So if the `:newColour` is `'red'`, you run the `hit` subprocedure.
If the `:newColour` is `'blue'`, you run the `miss` subprocedure.

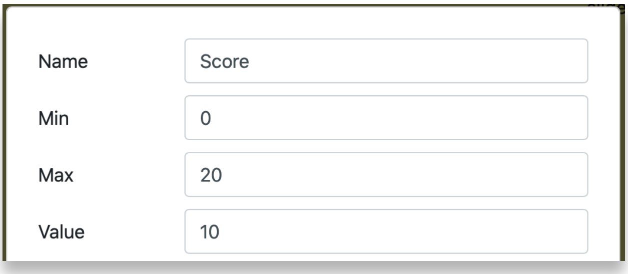
The turtle-ball will not react to any other colour!

A SLIDER TO COUNT THE HITS

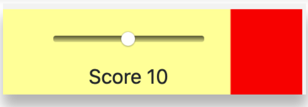
How about this: you can use a slider to count the score. The slider goes from 0 to 20, and the game starts at 10. You win a point on the red wall, and lose a point on the blue wall.

In the + menu, choose **Slider**.

Move the slider to a corner of the page (drag it by its name), and right-click on it to open its dialog box. Name it **Score**, and set the **MINimum and MAXimum** values and the **current** value as follows:



Name	Score
Min	0
Max	20
Value	10



WE HAVE A WINNER

Remember the instruction `if :newColour = 'red' [hit]`

Now we can work on the `hit` procedure. Create this procedure in the Procedure pane:

```
to hit
setheading minus heading
setscore score + 1
end
```

When the ball hits the red wall, it should bounce and increase the score. `Score`, the name of the slider, reports its current value. `Setscore` sets its new value, to the old value, plus one.

Count the misses on the blue wall

The miss procedure is very similar. Can you guess?

```
to miss
setheading minus heading
setscore score - 1
end
```

There is a problem ahead. You can try `everyone [clickon]` in the Command Centre. The ball will bounce off the paddle, and off both walls. However, if you reach 0 or 20, you will get an error message:

```
setscore doesn’t like 21 (or -1) as input
```

Naturally, because its limits are 0 and 20.

We need to change the `hit` and `miss` procedures. Add a line that checks the limit. The new line is highlighted:

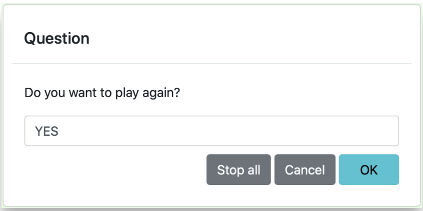
```
to hit
setheading minus heading
setscore score + 1
if score = 20 [announce 'You win!' play.again]
end

to miss
setheading minus heading
setscore score - 1
if score = 0 [announce 'Game over!' play.again]
end
```

Let’s check the case of the `hit` procedure. You hit the red wall. The ball bounces. The slider increases by one. Now check if the `score = 20`. If it is, announce the winner (that’s what `announce` does), and run the procedure `play.again` (it doesn’t exist yet).

The procedure `play.again` uses the primitives `question` and `answer`. Let’s look at them. Type this in the Command Centre:

```
question [Do you want to play again?]
```

Lynx displays a dialog box with that question. Type **YES** and click **OK**.


Question

Do you want to play again?

YES

Now type this in the Command Centre:

```
show answer             ANSWER REPORTS THE LAST ANSWER IN A DIALOG BOX
show answer = 'yes'
true                                        LYNX REPORTS TRUE OR FALSE
```

Create this procedure in the Procedures Pane:

```
to play.again
question [Do you want to play again?]
ifelse answer = 'yes'
 [start.game]
 [everyone [clickoff]]
end
```

You know about `question` and `answer`, but now look at `ifelse`. `Ifelse` checks if the answer was `'yes'`. If true, it runs the **first** list of instructions: `start.game`. If false, it runs the **second** list of instructions: `everyone [clickoff]`.

You don't have to format the code on three lines like that, but it makes all the instructions easier to read.

A Start button

You need a button to start the game. Remember how Play mode works. When you share this game, there will be no Command Centre and no Procedures Pane.

Create this procedure:

```
to start.game
ball,
setpos [0 0]
setheading 75                          CHOOSE WHATEVER # YOU LIKE
setscore 10                            CHOOSE WHATEVER # YOU LIKE
everyone [clickon]
end
```

At this point, you should be able to guess what each line does. Finally, create a button and set its **On click** instruction to `start.game`

All the procedures of this project

```
to move.ball
ball,
forever [forward 10 wait 1]
end

to move.paddle
paddle,
forever [setpos mousepos]
end

to Ball_touch :touchedturtle
; bounce when ball touches paddle
ball,
setheading minus heading
end

to Ball_oncolour :prevColour :newColour
; red wall on right, blue wall on left
if :newColour = 'red' [hit]
if :newColour = 'blue' [miss]
end

to hit
; bounce, then increase slider, max is 20
setheading minus heading
setscore score + 1
if score = 20 [announce 'You win!' play.again]
end

to miss
; bounce, then decrease slider, min is 0
setheading minus heading
setscore score - 1
if score = 0 [announce 'Game over!' play.again]
end
```

```
to play.again
question [Do you want to play again?]
ifelse answer = 'yes'
   [start.game]
   [everyone [clickoff]]
end

to start.game
ball,
setpos [0 0]
setheading 75
setscore 10
everyone [clickon]
end
```

Advanced ideas

Here is a long list of possible improvements, with some clues.

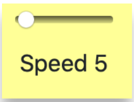
Add a sound effect!

You can add a "very short *bong*" sound effect that plays when the ball bounces (doesn't that give you clue about where to insert the sound effect in your code?)

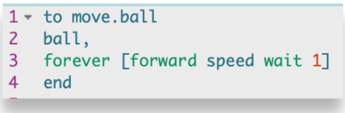
- Use a **wav** file, or record your own "toc" sound.
- Choose **Sound** in the + menu to open the **Import sound** dialog box.
- Once you have the sound icon on the page, click on it to test it.
- Right-click on the icon to open its dialog box, and set its name to a short, one-word name (like "`toc`"). While the dialog box is open, you can also take the time to make the icon invisible. It will still work.
- Add the name of the sound icon (in this case `toc` it is a command that plays the sound) in the `bounce` procedure.

Make Speed a Variable

Look at your procedures. Maybe you can figure out where you can change the speed of the ball, right? How about creating a slider and using its value for forward? Choose **Slider** in the + menu.



Right-click on the slider and name it **speed**.



```
to move.ball
ball,
forever [forward speed wait 1]
end
```

How about increasing the speed when the score reaches 15?

Spice up the game with random Bounces

The ball can bounce randomly on one of the walls. Instead of just using the same `bounce` procedure everywhere, you can have one `bounce.red` procedure that sets the heading to `-45 + random 90`. Use that special `bounce.red` procedure in the colour detection, for red.

Ceiling and Floor

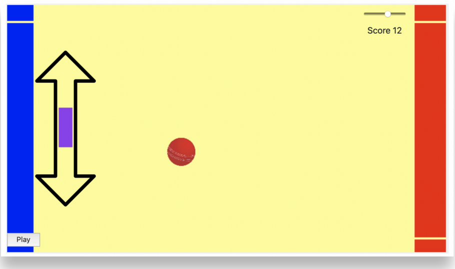
You can draw walls at the top and bottom of the page, and add more lines to your colour detection procedure:

```
to Ball_oncolour :prevColour :newColour
if :newColour = 'red' [hit]
if :newColour = 'blue' [miss]
if :newColour = ...
if :newColour = ...
end
```

You will need, however, a new type of **bouncing instruction**. The bouncing instruction for the red and blue walls will not do. You can try, but you will soon realize that it doesn't look natural. Explore bouncing methods such as

```
setheading heading - 90
setheading heading + 90
```

Vertical movement of the paddle



Right now, the paddle follows the mouse pointer everywhere on the page. How about if it glides only vertically, and stays on the left of the page?

In the `move.paddle` procedure, instead of using

```
forever [setpos mousepos]
```

use

```
forever [sety last mousepos]
```

`Mousepos` reports a list of two numbers, an **x** coordinate, and a **y** coordinate. `Last mousepos` reports the **last** item of that list, which is just the **y coordinate**. Now the paddle will only move up and down.

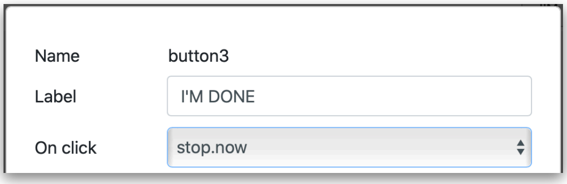
Don't forget to freeze the turtle so the player can't cheat: Open the paddle's dialog box and check the ☑ Frozen box. Do the same for the ball.

Stop the game!

A nice addition would be a STOP button, in case a player wants to end the game without finishing it. Write the procedure below and In the + menu, choose **Button.**

```
to stop.now
stopall
end
```

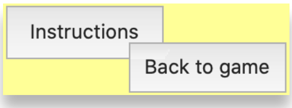
Give it this label: **I'M DONE**. and set its instruction to `stop.now`.



Name	button3
Label	I'M DONE
On click	stop.now

Instruction page

Create a second page, with some decor, and a text box with instructions for the player. Make sure you add navigation button to go to the instruction page, and return to the game.



Curriculum Links for Ontario

C3.1 - Solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves the analysis of data in order to inform and communicate decisions.

C3.2 - Read and alter existing code involving the analysis of data in order to inform and communicate decisions, and describe how changes to the code affect the outcomes and the efficiency of the code.