Scratch tutorial

Version 2020

juin 19, 2020

Contents:

CHAPITRE 1

Introduction

Scratch is a **programming language** which uses blocks. Each **block** is an instruction for the the computer to do something. To create a **Scratch progam**, you assemble colored blocks into a sequence.

1.1 Create an account

You can create a Scratch program directly in a web navigator by going to this address

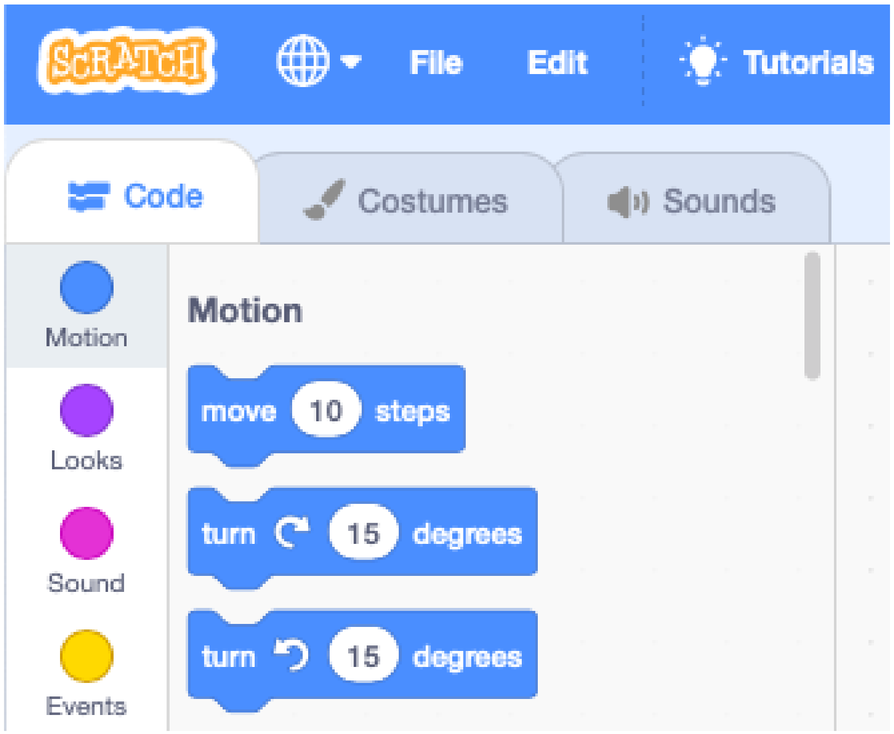
https://scratch.mit.edu/

If you want to save and share your programs you have to create your own account. Don't use your real name when creating an account.

1.2 Switch the language

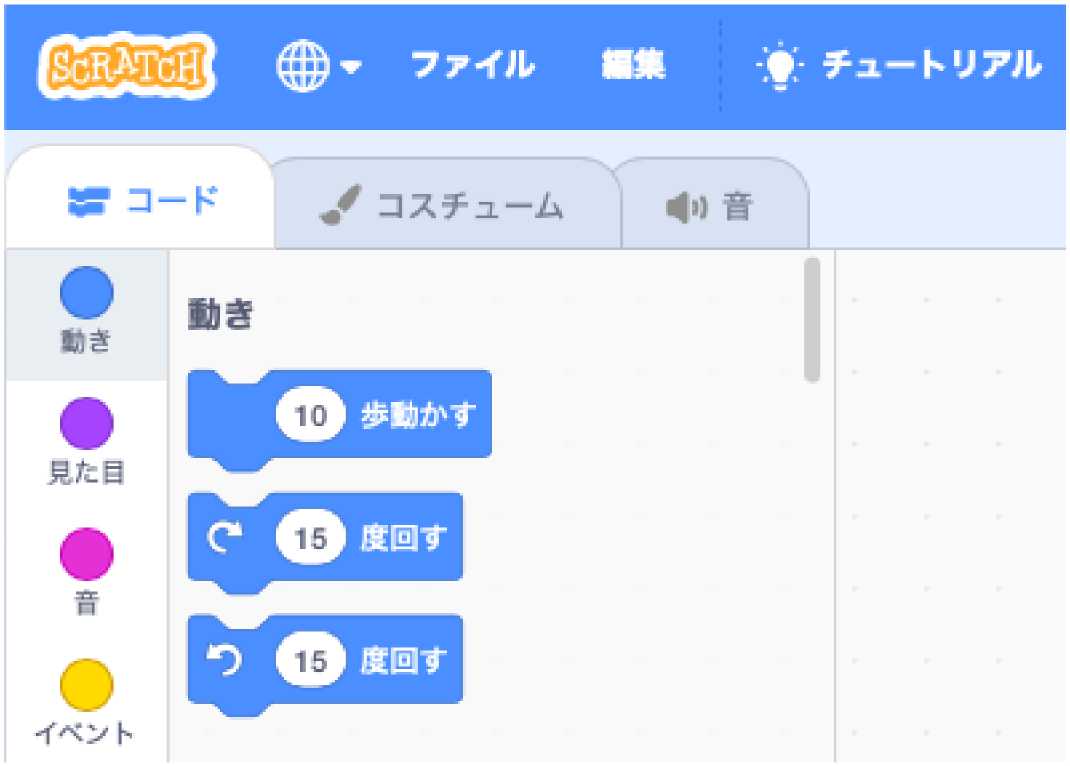
To change the language of the Scratch menus and of the programming blocks, click on the globe symbol and select your desired language.

In **English** you get this interface :



In **Japanese** you will get the interface below. Not only the Scratch menus and tabs are translated to the new language, also the language inside the program blocks changes.

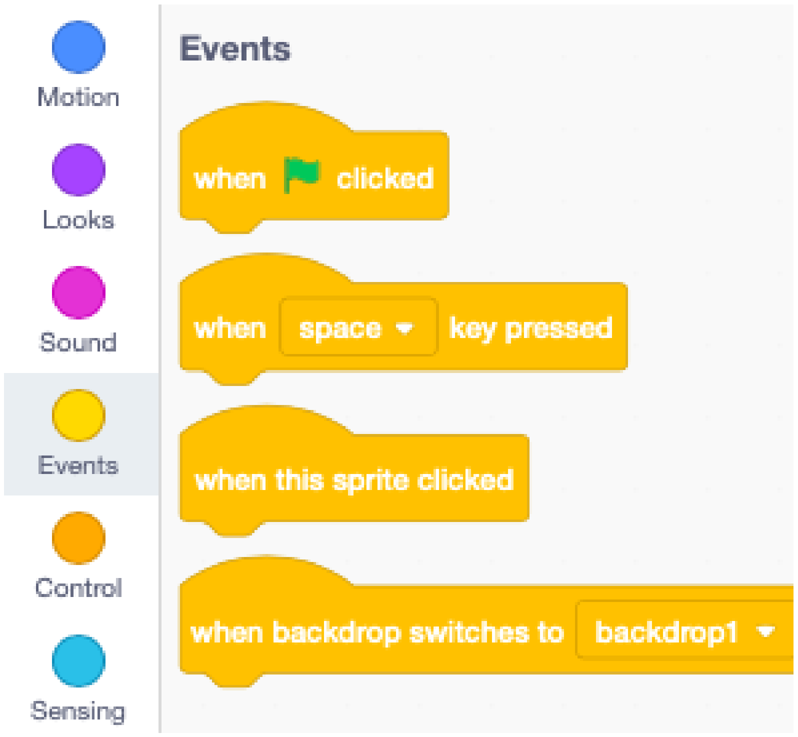
For example the block *move 10 steps* in English becomes *10* in Japanese.



1.3 The programming blocks

In the **Code** tab you find 9 different block categories. Each category has a different color. The **Motion** blocks are blue. This category contains all the blocks to move a sprite. The coloring helps you to recognize the blocks more easily.

The blocks in the **Events** category are yellow. You find for example the block *when clicked*. This type of block is called a **hat block** and is used at the top of a progamming stack. It allows to attach other blocks only below, not above.

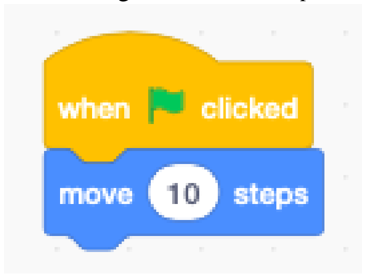


To create a program you drag blocks from the palette to the script canvas. Two or more blocks interlock to form a **stack**.

1.4 Your first program

Create your first program by doing the following :

— select the **Events** category
— drag the *when clicked* block to the canvas
— select the **Motion** category
— drag the *move 10 steps* block to the canvas and attach it



You can execute the program either by

— clicking on the code blocks

— clicking on the green flag

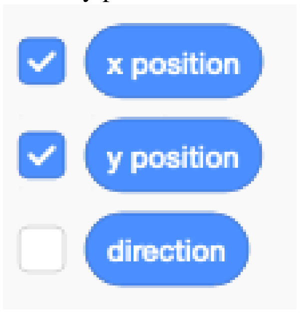
Executing the code makes the cat move 10 steps forward. A step is the base unit of the screen (also called a pixel) ; the whole stage is 480 pixels wide.

1.5 The coordinate system

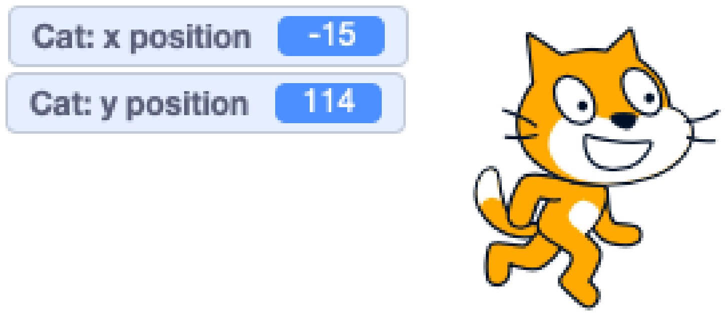
At the end of the **Motion** palette you can check

— x position

— y position



This will display the coordinates of the cat on the stage. In the example below, the cat is at position x=-15 and y=114.



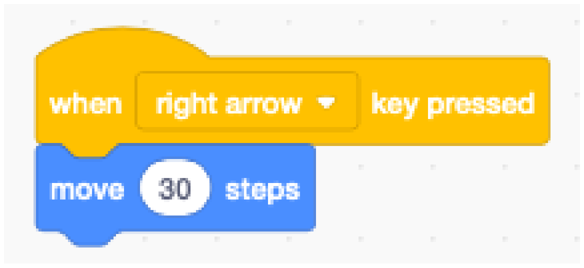
Try now to move the cat with the mouse.

— the x position goes from -240 to 240

— the y position goes from -180 to 180

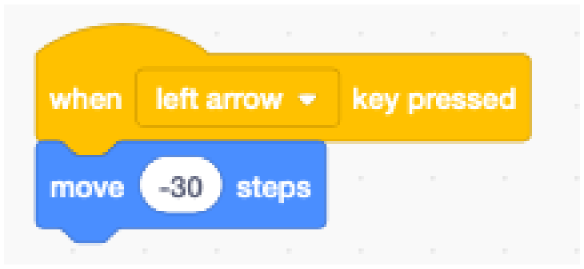
1.6 Move a sprite

You can use the arrow keys to move the cat sprite to the left or to the right. Create this code which moves the cat sprite 30 steps when you press the *right arrow* key.



```
when right arrow key pressed
move 30 steps
```

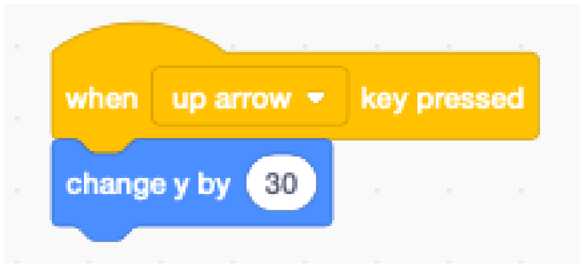
Create another code stack which moves the cat sprite 30 steps to the left, when you press the *left arrow* key.



```
when left arrow key pressed
move -30 steps
```

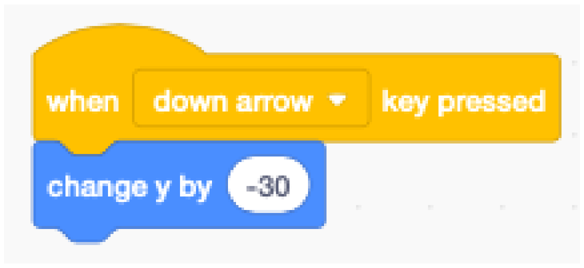
1.7 Move along the y axis

You can also move the sprite up along the y-axis using this



```
when up arrow key pressed
change y by 30
```

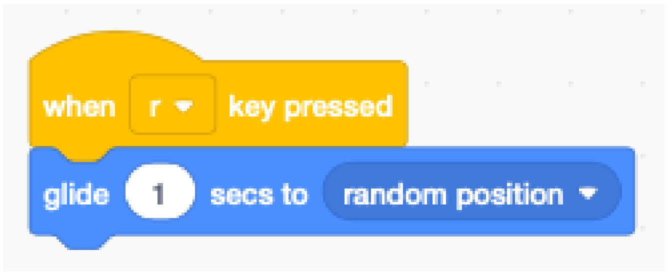
or move the sprite down, using that



```
when down arrow key pressed
change y by -30
```

1.8 Glide to a random position

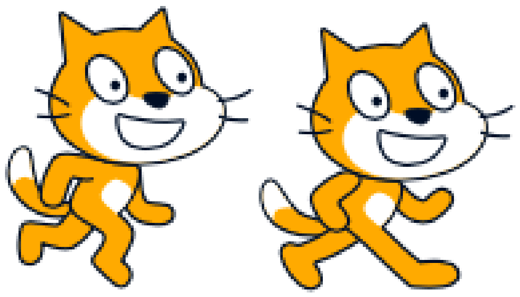
There is a block to move the sprite to a random position. Try this :



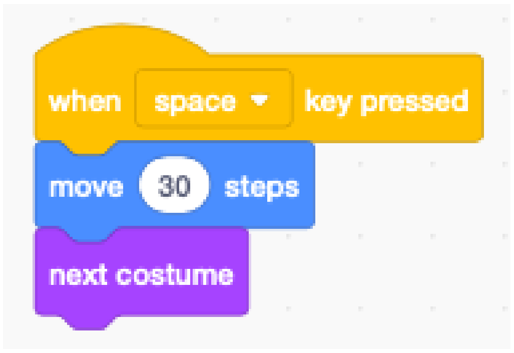
```
when [r ▾] key pressed
glide (1) secs to (random position ▾)
```

1.9 Animate walking

A sprite can have more than one appearance. It's called a **costume**. The cat has 2 costumes which are called *costume1* and *costume2*. It's feet are in a different walking position. These are the two cat costumes.

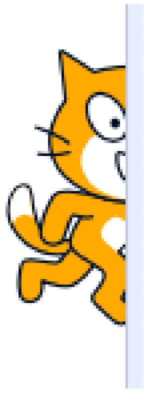


By alternating between these two costumes and advancing at the same time, we get the illusion of walking.



```
when [space ▾] key pressed
move (30) steps
next costume
```

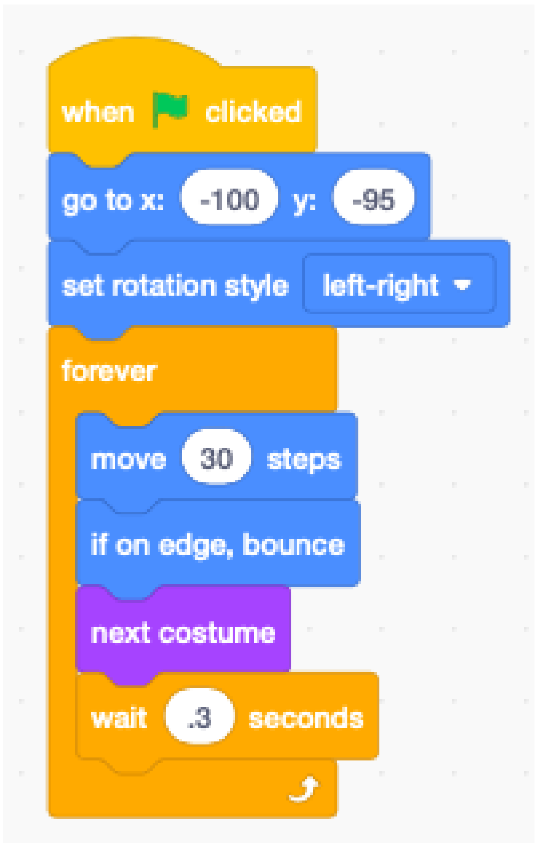
When the cat reaches the right edge, it cannot move any further.



You can add the block **if on edge, bounce** to make the cat go the other way.

1.10 Continous animation

You can animate a sprite continously by using a loop. But now we have to slow it down with a **wait 0.3 seconds** block.



```
when 🏴 clicked
go to x: -100 y: -95
set rotation style left-right ▾
forever
    move 30 steps
    if on edge, bounce
    next costume
    wait .3 seconds
```

When you start the program by clicking on the green flag, the cat starts to pace back and forth continously.

1.11 Add a backdrop

You can also add a backdrop to your program.

https://scratch.mit.edu/projects/380796710

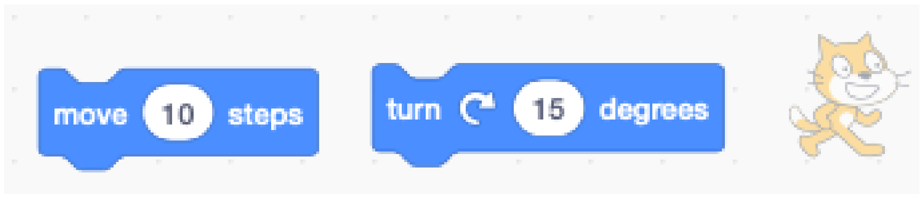
CHAPITRE 2

Techniques

This section shows useful programming techniques.

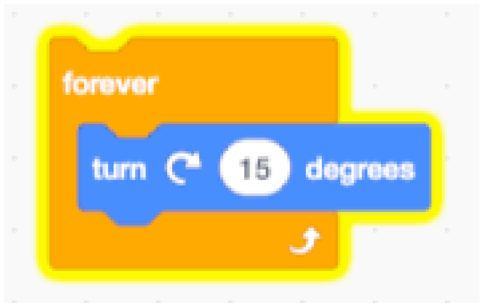
2.1 Click on blocks to execute

You can click on blocks in the palette and on blocks on the canvas and execute these blocks. For exemple you can click on these **move** and **turn** blocks to test them quickly. This is very useful to understand which processes are currently running.



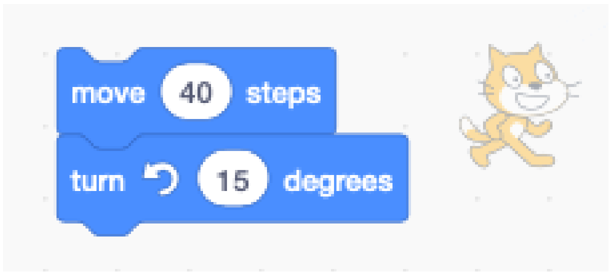
2.2 Running block is yellow

When your block is running, it has a yellow outline. You can start the **forever** block by clicking on it. When you click a second time, it stops.



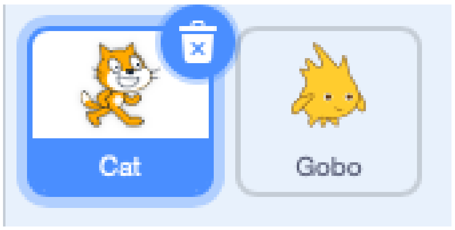
2.3 Stacked blocks

You can stack blocks to create program sequences (or stacks). Clicking on any of the blocks, will execute the whole stack.



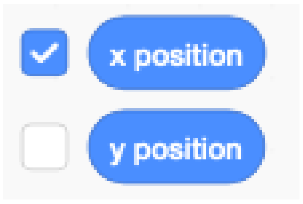
2.4 Capitalize sprite names

Capitalize sprite names. They are classes from which you can clone objects.

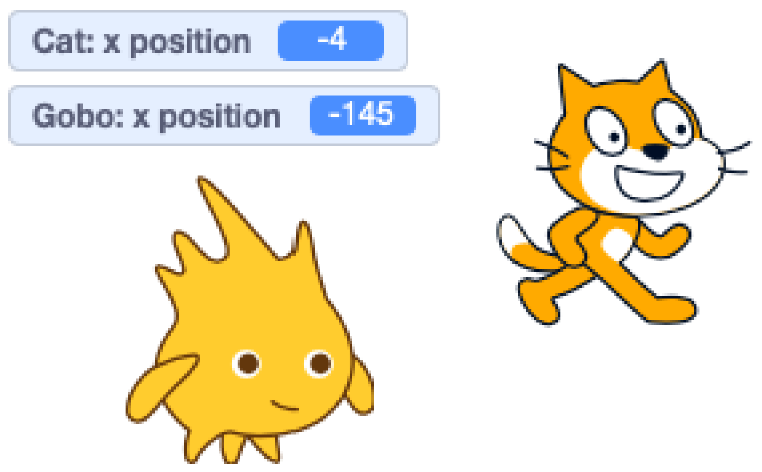


2.5 Sprites have variables

Sprites have built-in variables, which describe their state. You can visualize these **reporter** blocks by clicking on their check box.

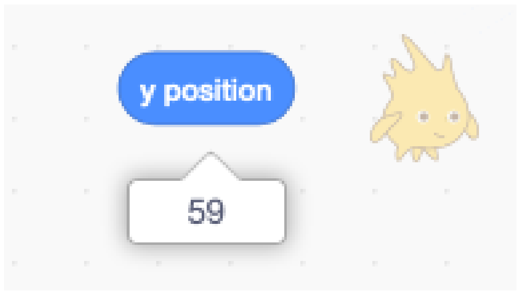


This makes the values be displayed on the stage.



2.6 Click on reporters

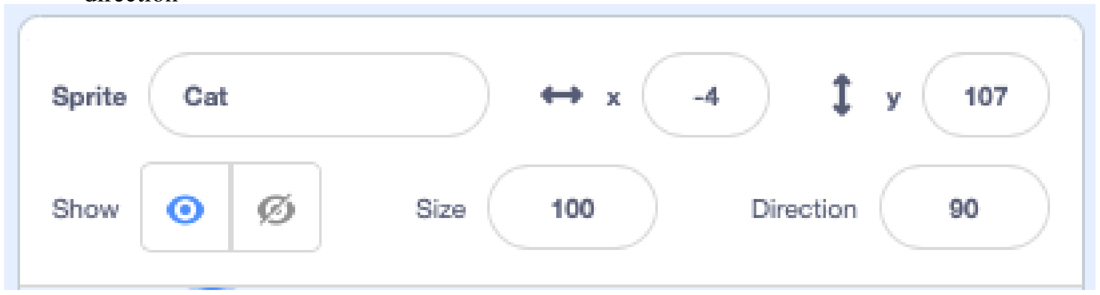
Click on a reporter block to display it's value.



2.7 Sprite variables

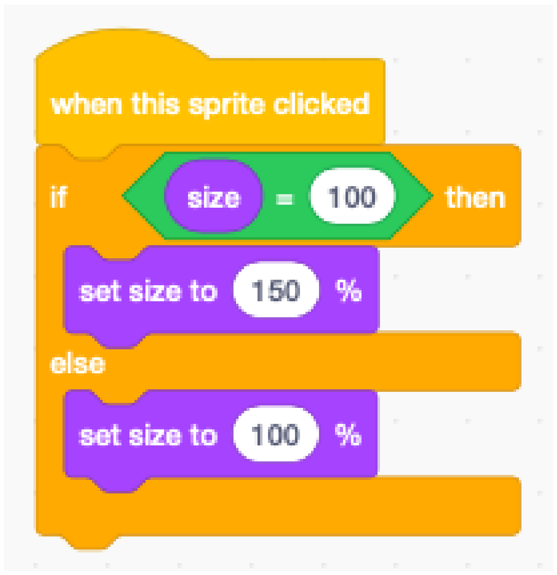
Each sprite (and clone) has these variables :

- position (x, y)
- visible
- size
- direction



2.8 Toggle sprite size

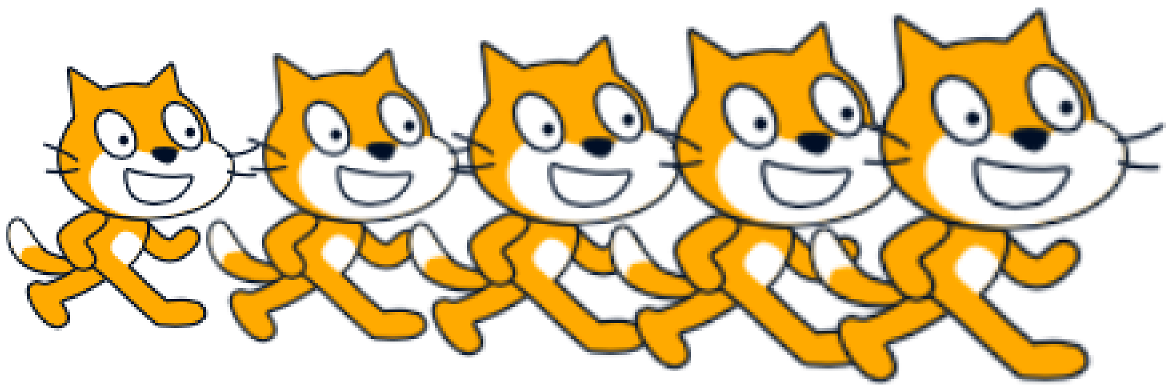
When toggling a value, you need a variable to store the current state. You can use existing variables. In the following example the size variable is used to toggle.



```
when this sprite clicked
if size = 100 then
    set size to 150 %
else
    set size to 100 %
```

2.9 Change size by

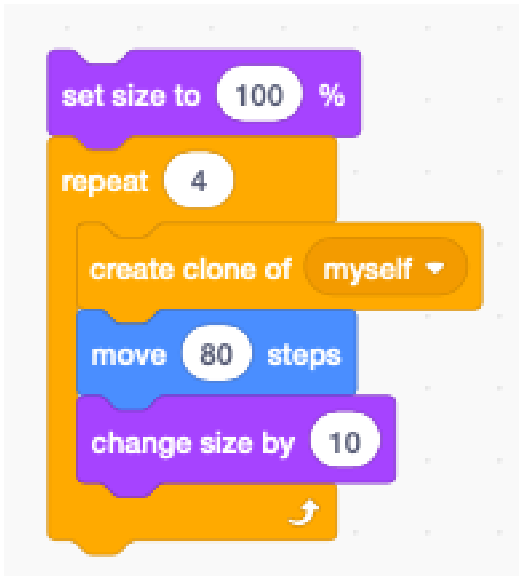
We can show the effect of the **change size by** block by applying it in a loop on multiple clones of the cat.



Inside the loop we :

— create a clone
— move by 80 points
— change the size by 10

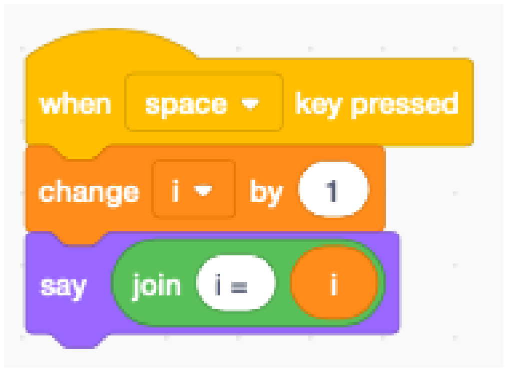
The first 4 cats are clones, the last one is the orignal.



```
set size to (100) %
repeat (4)
    create clone of (myself ▾)
    move (80) steps
    change size by (10)
end
```

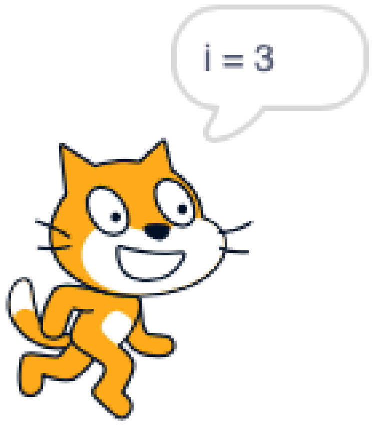
2.10 Increment a variable

The block **change by** can be used to increment a variable. The exemple below increments the variable **i** by 1 when pressing space.



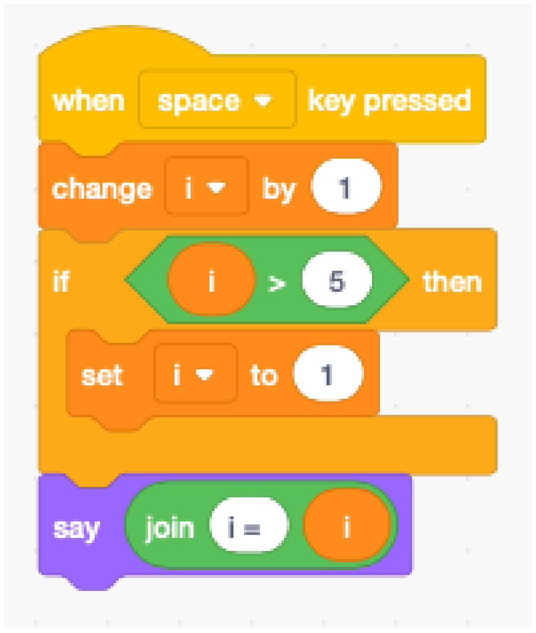
```
when [space ▾] key pressed
change [i ▾] by (1)
say (join (i =) (i))
```

We display the variable in a speech bubble.



2.11 Reset the counter

Each time we press the space bar the counter **i** increments. We can add an **if** block to reset the counter when reaching a certain value.

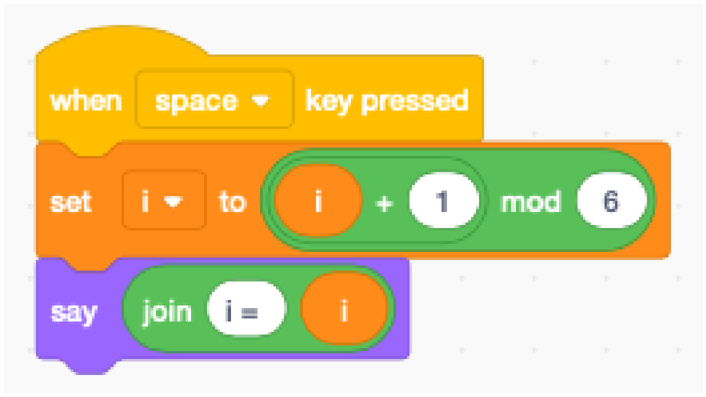


```
when space key pressed
change i by 1
if i > 5 then
    set i to 1
say join i = i
```

2.12 Use modulo to wrap

Another way to wrap around is the function **modulo** or **mod** which returns the remainder of a division.

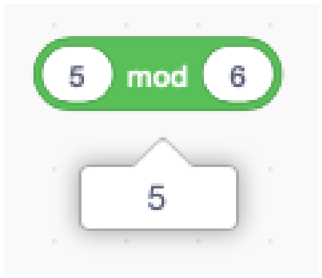
This is shorter :



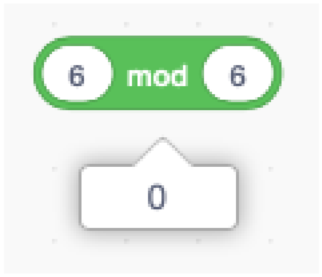
```
when space key pressed
set i to i + 1 mod 6
say join i = i
```

How does it work ? You can place only the **mod** block on the programming canvas and test it by entering numbers and click with the mouse to evaluate the expression.

You will see **5 mod 6** is 5 (the reminder of 5 divided by 6).

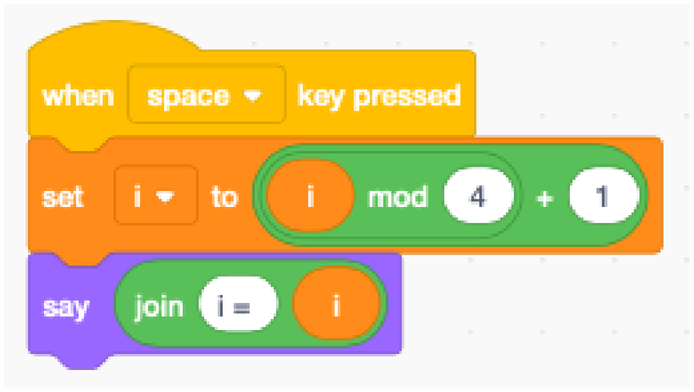


You will see **6 mod 6** is 0 (the reminder of 6 divided by 6).



The counter i cycles through the 6 values 0, 1, 2, 3, 4, 5.

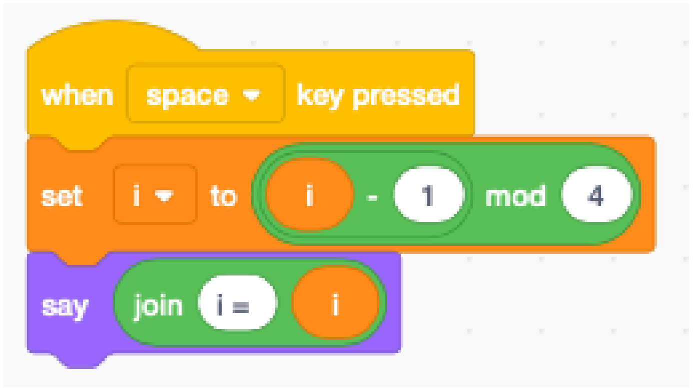
To start with 1 instead with 0 we can modifiy the expression to this.



The counter i cycles now through the 4 values 1, 2, 3, 4.

2.13 Cycle the other way

We can also decrement and cycle back when reaching 0.



```
when [space ▾] key pressed
set [i ▾] to (((i) - (1)) mod (4))
say (join (i =) (i))
```

Here the counter **i** cycles through the range 3, 2, 1, 0.

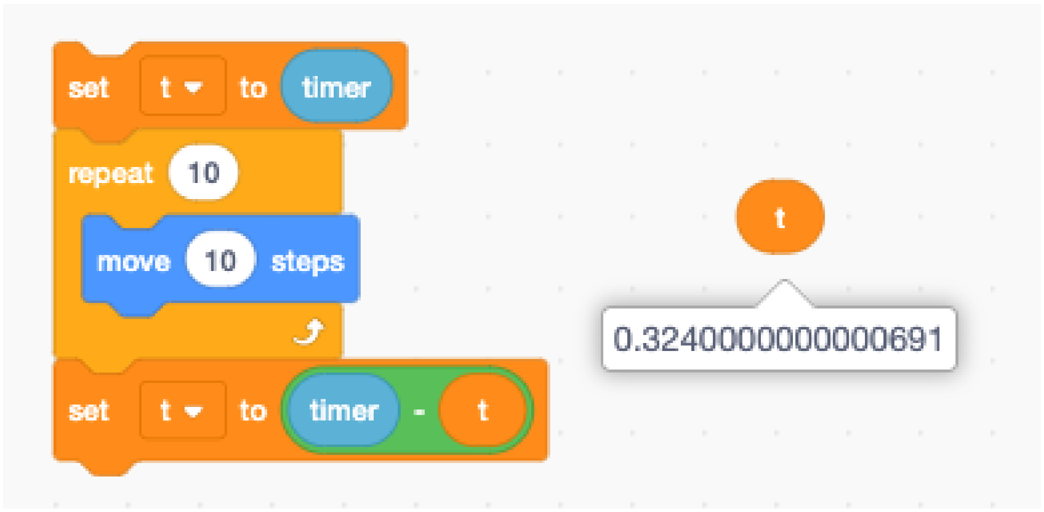
2.14 Timing

Sometimes we need to know how long it takes to execute a piece of code. We can use the **timer** block for this. To measure the time we use the variable **t**.

— Start : memorize the start time in **t**
— End : substract the start time **t** from the current time

2.14.1 The animation loop

Loops in Scratch are slowed down to allow simple animation. Let's measure the time to repeat a **move** block 10 times. The total time is 0.32 seconds. Thus the loop time is 32 ms which results roughly in 30 frames per second.

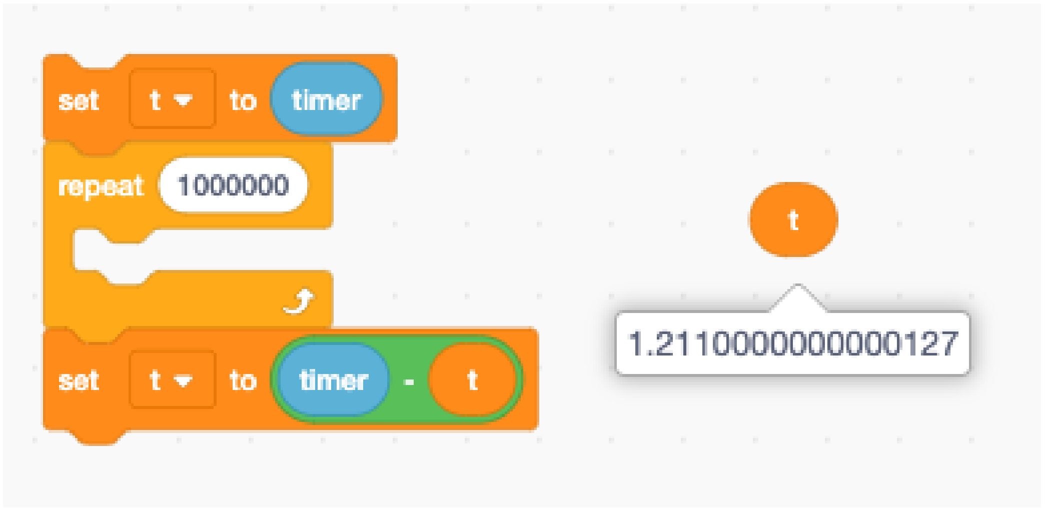


```
set [t ▾] to (timer)
repeat (10)
    move (10) steps
end
set [t ▾] to ((timer) - (t))
```

```
(t)
0.32400000000000691
```

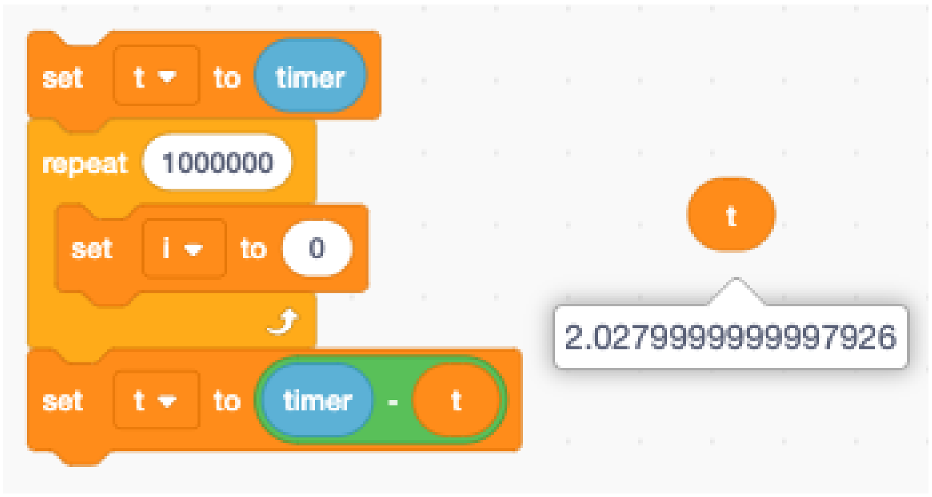
2.14.2 Empty loop

How long does it take to repeat an empty loop 10 times ? The time is too small to be measured. Even if we increase to 1000 times, the measured time still shows 0. We need to repeat the loop 1 million times, to be abe to measure something. It takes a total time of 1.2 seconds. Thus executing the empty loop takes only 1.2 us.

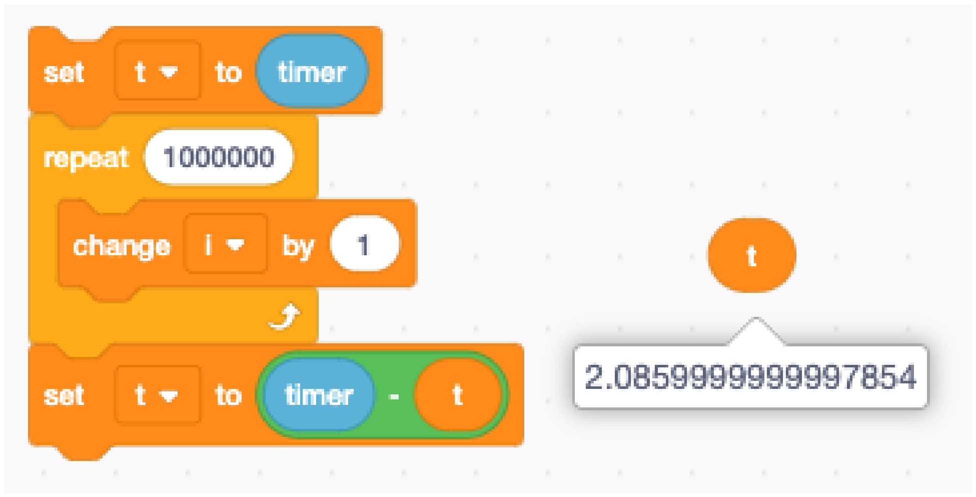


2.14.3 Simple assignment

When adding a **set to** block the time increases to 2 us. We conclude that the **set to** block alone takes 0.8 us.

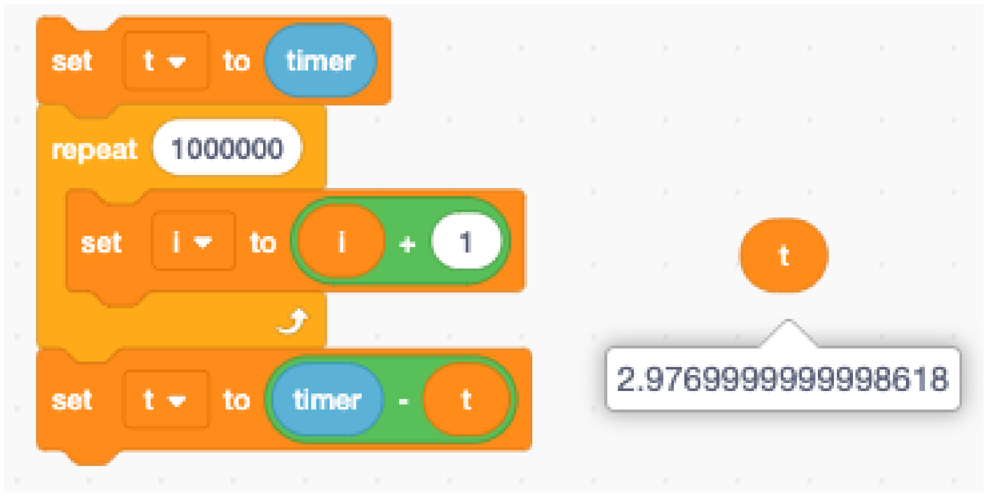


The **change by** block takes the same amount of time, roughly 0.8 us.



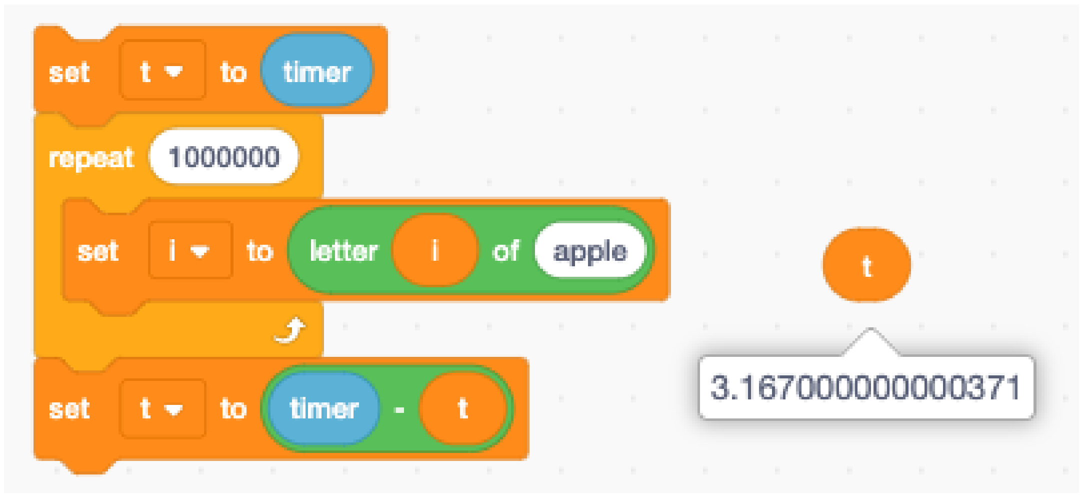
2.14.4 Math operation

When putting the **add** block inside the **set to** block the loop execution time becomes 3 us. Thus we conclude the **add** block takes 1 us.



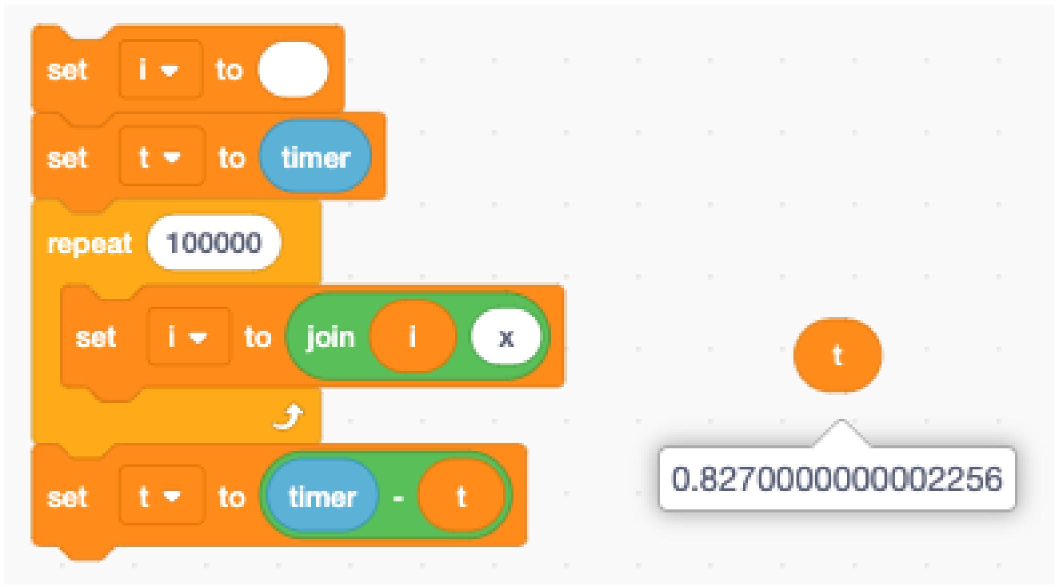
2.14.5 String operation

A simple string operation takes roughly the same time as a math operation. Accessing an indexed letter in a string takes 1.2 us.



```
set t to timer
repeat 1000000
    set l to letter i of apple
set t to timer - t
```

The **join** block however needs to copy strings. As the strings get longer, this operation takes more and more time. We decrease the repetition count to 100'000. In this task we add the letter "x" to the string variable. The string length varies from 1 to 100'000. The average string length is 50'000. The average **join** operation takes now 8 us.

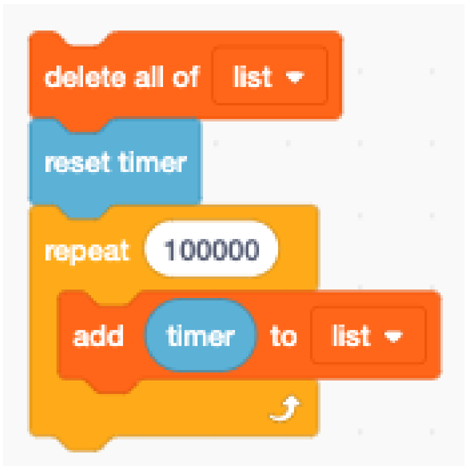


```
set i to
set t to timer
repeat 100000
    set i to join i x
set t to timer - t
```

2.14.6 Resolution of the timer

What is the resolution of the timer ? Does it have micro-second resolution ?

In fact now. The Scratch **timer** is a VERY low resolution timer. To measure its resolution, we record all timer values in a list.

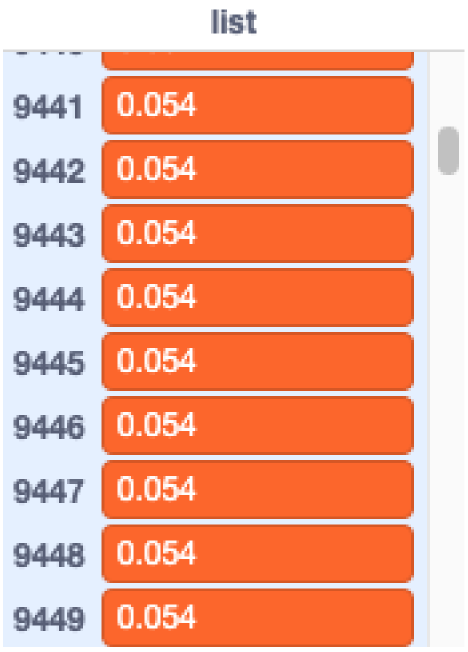


```
delete all of list
reset timer
repeat 100000
    add timer to list
```

When recording 100'000 times values we get the following values :

— 0
— 0.054
— 0.101
— 0.139
— 0.184
— 0.227
— 0.263
— 0.3

The increments are 54, 47, 38, 45, 43, 36, 37 ms. This is a very crude timer, and we have to take this into account when we make measurements.



list

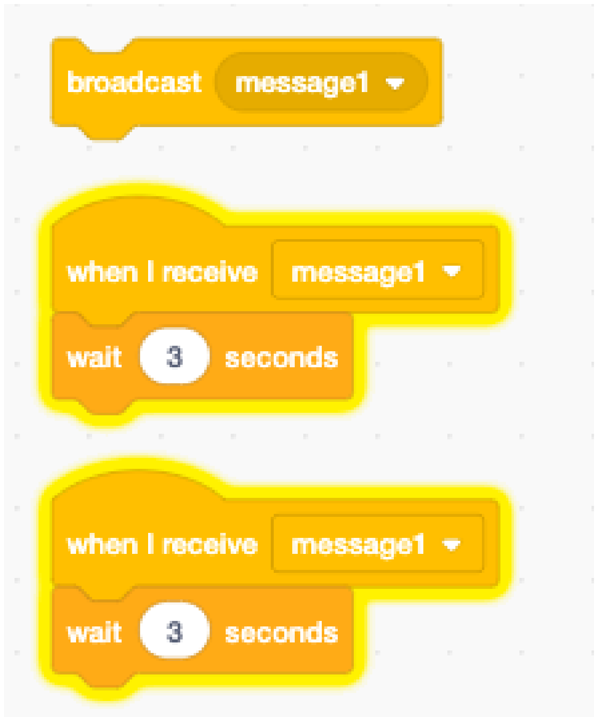
9441	0.054
9442	0.054
9443	0.054
9444	0.054
9445	0.054
9446	0.054
9447	0.054
9448	0.054
9449	0.054

2.15 Broadcast messages

Sending messages allows to start arbitrary pieces of code. If there are multiple receivers, they apperantly all start at the same time. In reality, however, control is rapidly being swiching between the different processes.

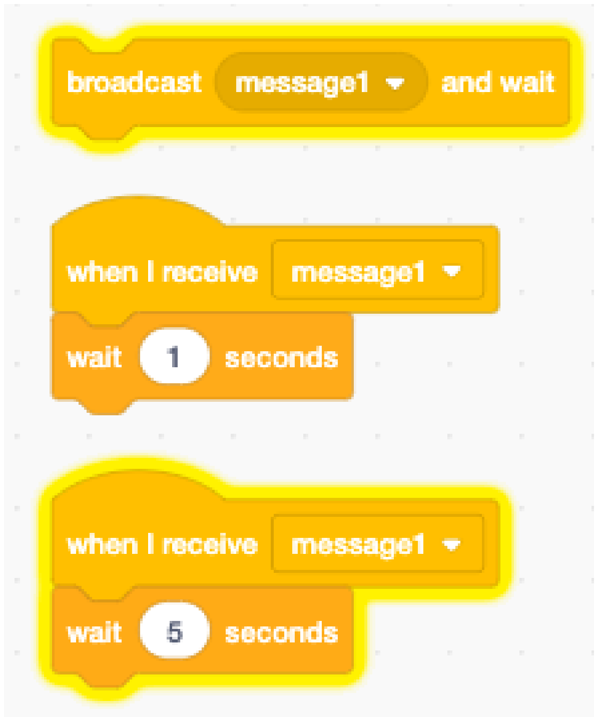
2.15.1 Simultaneous receivers

Here we see the broadcast of **message 1** which has already ended, and the 2 receiving blocks which are still active.



2.15.2 Broadcast and wait

The **broadcast and wait** block waits until the last message has been finished. Here we see the first stack finished and the second stack still being executed.



```
broadcast (message1 ▾) and wait

when I receive [message1 ▾]
wait (1) seconds

when I receive [message1 ▾]
wait (5) seconds
```

2.15.3 Multi-threading

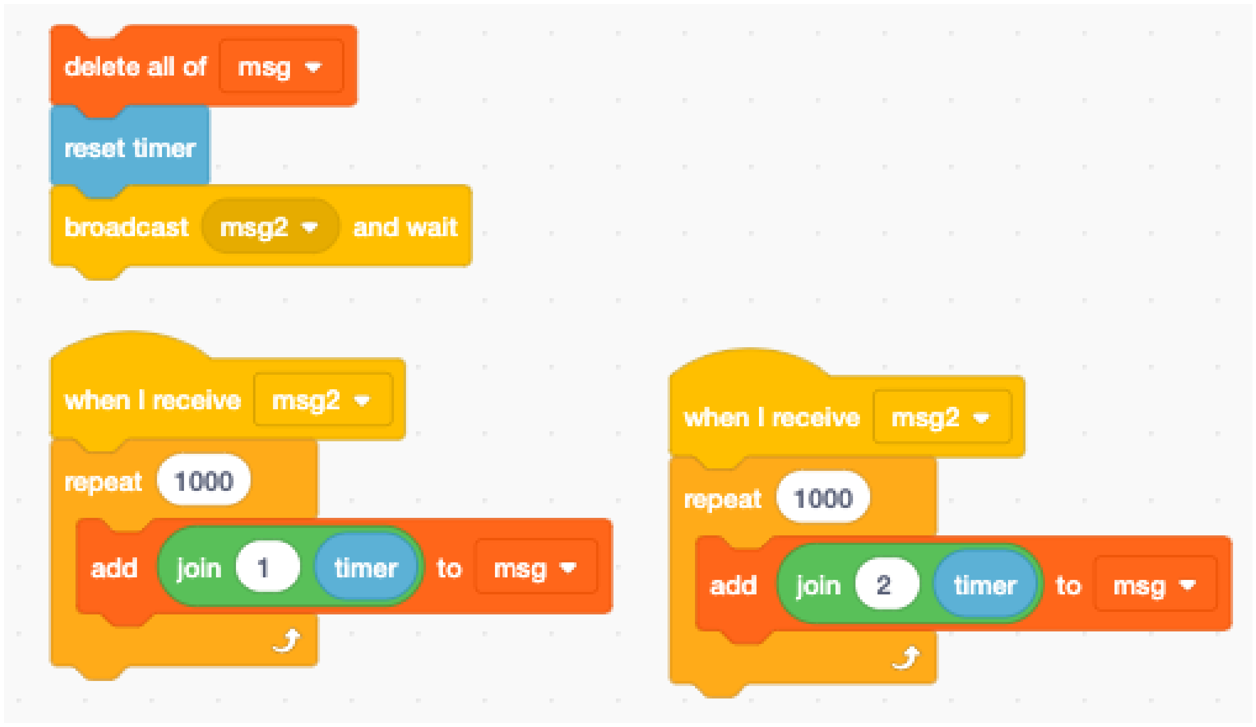
How does it work when two stacks execute simultaneously ? Let's try to understand what happens. We will create a **msg** list, where each process makes an entry when it executes. Both processes (1 and 2) run concurrently and execute a loop 1000 times.

The entry is :

— process number (1 or 2)

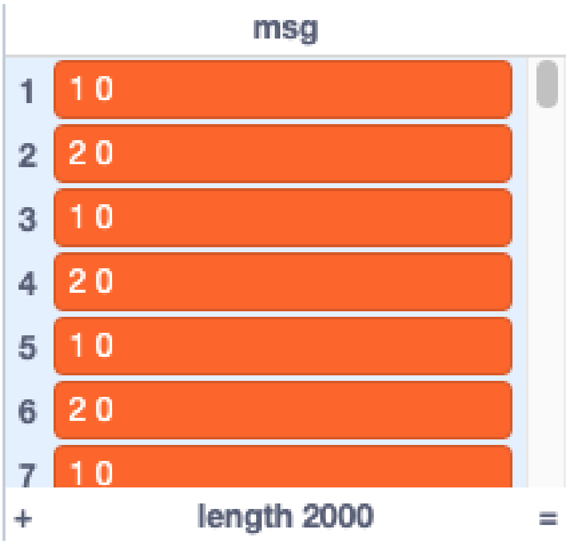
— timer

At the start the timer is reset.



```
delete all of msg
reset timer
broadcast msg2 and wait

when I receive msg2
repeat 1000
    add join 1 timer to msg

when I receive msg2
repeat 1000
    add join 2 timer to msg
```

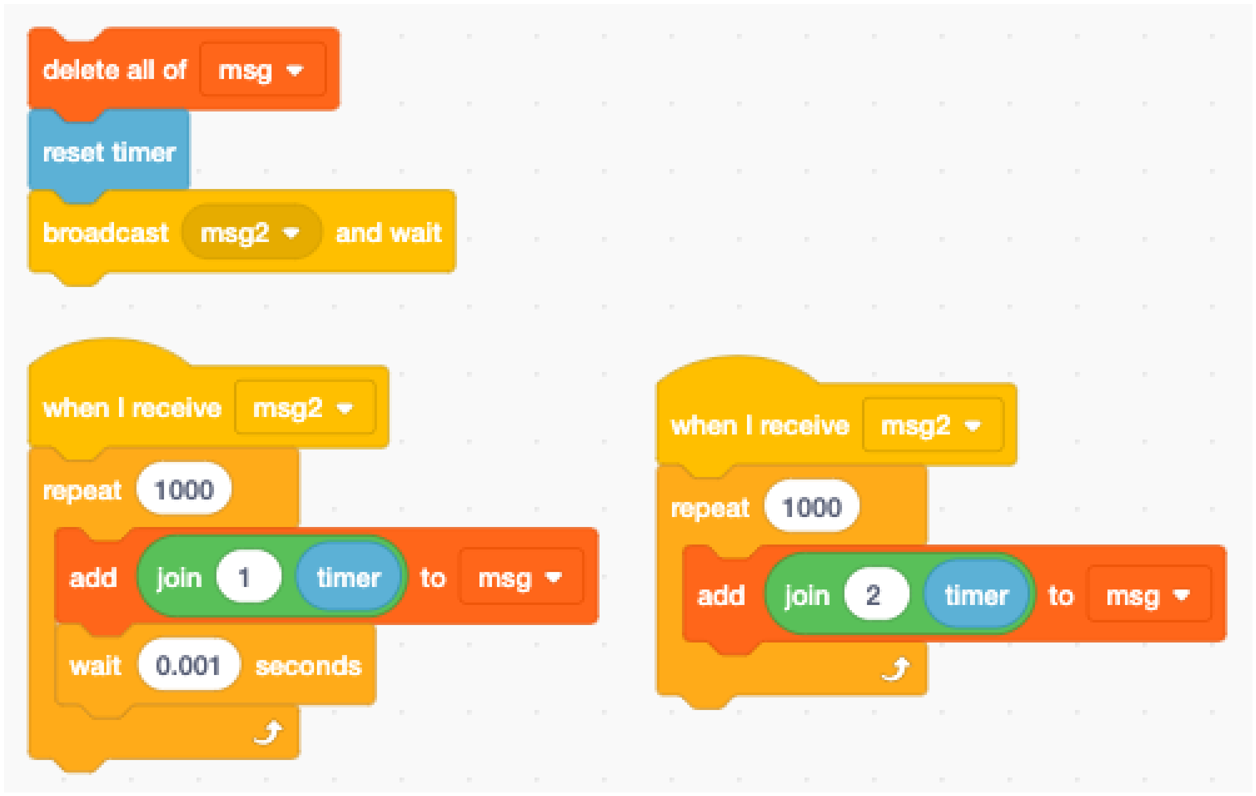
The result is that for each of the 1000 iterations the 2 processes alternate. Each one gets a turn during each iteration.



```
msg
1  1 0
2  2 0
3  1 0
4  2 0
5  1 0
6  2 0
7  1 0
+  length 2000  =
```

2.15.4 Wait inside a thread

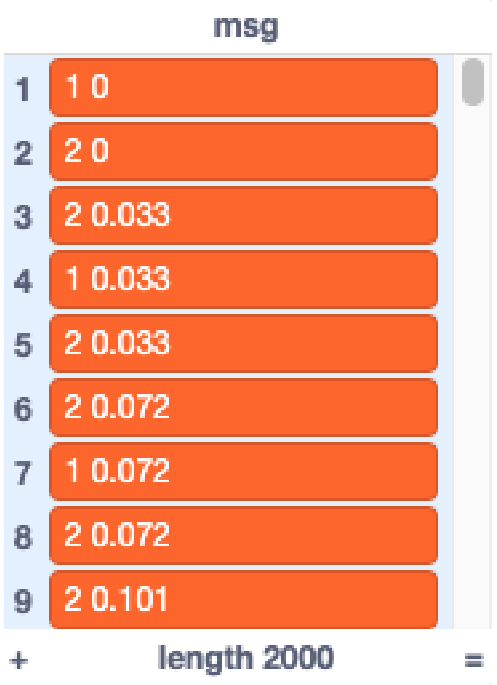
What happens when we introduce an extra wait in one of the processes ? Let's say we introduce a 1 ms wait in process 1. Will process 2 get more time to run ?



```
delete all of [msg]
reset timer
broadcast [msg2] and wait

when I receive [msg2]
repeat (1000)
    add (join (1) (timer)) to [msg]
    wait (0.001) seconds

when I receive [msg2]
repeat (1000)
    add (join (2) (timer)) to [msg]
```

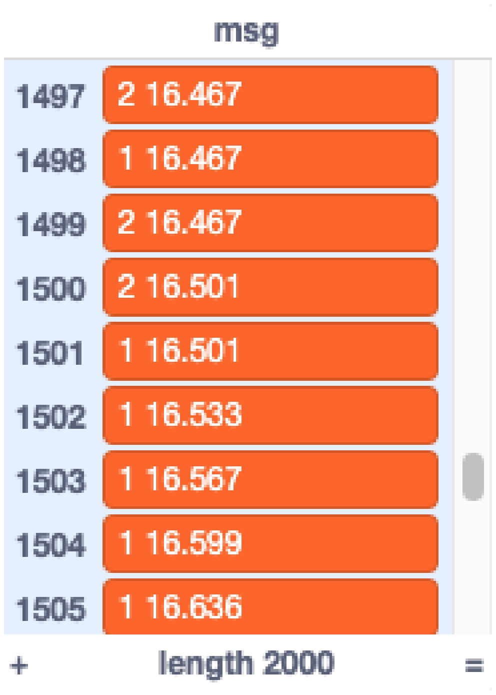
The result is quit curious. Process 1 and process 2 still alternate, but now process 2 get's a second chance to run during the wait time of process 1. The overall result is that process 2 runs twice as often as process 1.

The wait block has a 1 ms wait time, but in reality the waiting is aligned with the 33 ms internal screen refresh timer.



msg	
1	1 0
2	2 0
3	2 0.033
4	1 0.033
5	2 0.033
6	2 0.072
7	1 0.072
8	2 0.072
9	2 0.101

The two processess alternate until index 1500, at that time process 2 has done the 1000 iterations and terminates. Process 1 continues for another 1000 iteration roughly every 33 ms. The total time is 33 seconds.


msg

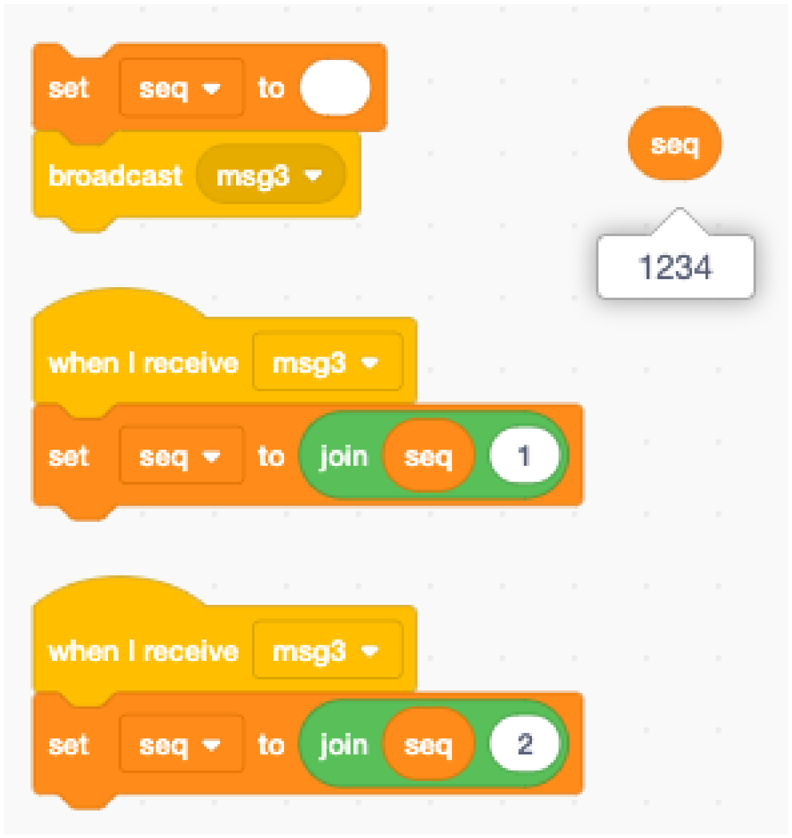
1497	2 16.467
1498	1 16.467
1499	2 16.467
1500	2 16.501
1501	1 16.501
1502	1 16.533
1503	1 16.567
1504	1 16.599
1505	1 16.636

2.15.5 Execution order

In what order are the processes executed ? A little bit of experimentation shows the processes are executed in their order of creation. This order cannot be changed. Copy-paste a previous process creates a new one at the last position.

Each process adds it's process number to the variable **seq**. We see

- process 1 and 2, which are in the same sprite
- process 3 in a different sprite
- process 4 in the stage


set seq to
broadcast msg3
when I receive msg3
set seq to join seq 1
when I receive msg3
set seq to join seq 2
seq
1234

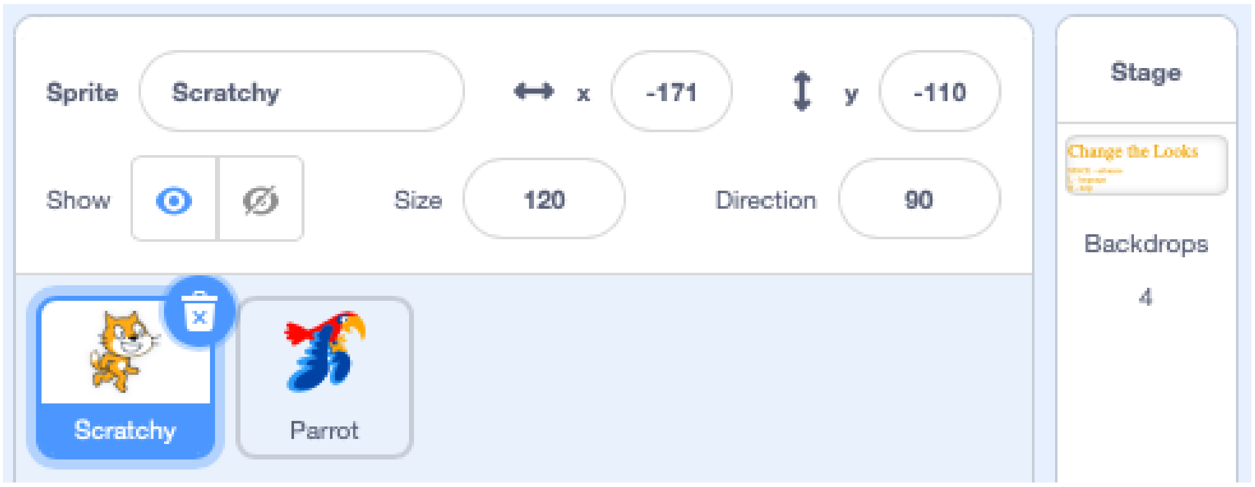
CHAPITRE 3

Looks

In this section we focus on how the graphical elements (sprites) look and how you can modify their look.

3.1 Sprites

Sprites are the graphical objects which you place into the stage.



They have a

- name (Scratchy)
- position (x=-171, y=-110)
- visbility (show)
- size (120%)
- direction (90°)

3.2 Backdrops

Backdrops are like sprites, but they are fix in size and cannot be repositioned.

As bitmaps they have the size 360 x 480 pixels. However in the editor there is twice the pixel density. In full-screen mode you can see the pixels.

3.3 Changing looks

https://scratch.mit.edu/projects/395424129

3.4 Draw a spiral

CHAPITRE 4

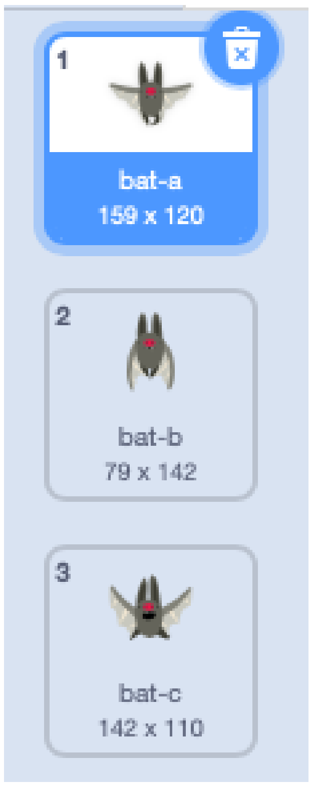
Animation

With Scratch you can easily animate sprites. Many sprites already have multiple costume. You can give the impression of walking or flying by simply alternating between two different costumes.

4.1 Flying sprites

https://scratch.mit.edu/projects/391871367

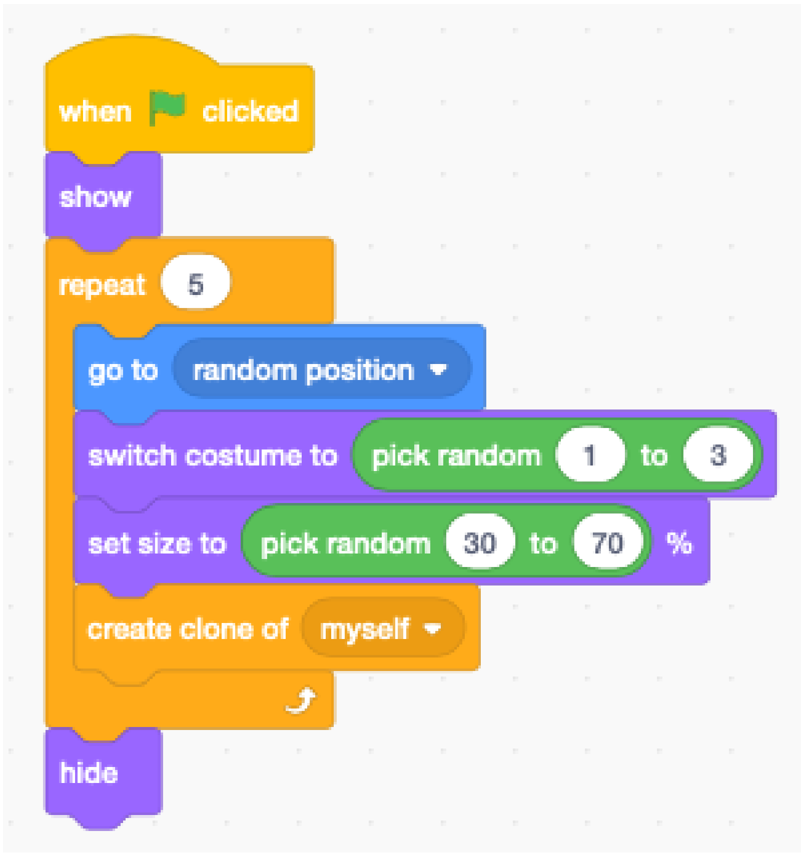
The **Bat** sprite has 4 costumes. You can delete the 4th one (sleeping) and just keep the 3 flying costumes.



Want multiple bats in the sky. Therefore we are going to make clones. Note that we do not anymate the original. We only use it to make clones. At the end we hide the original.

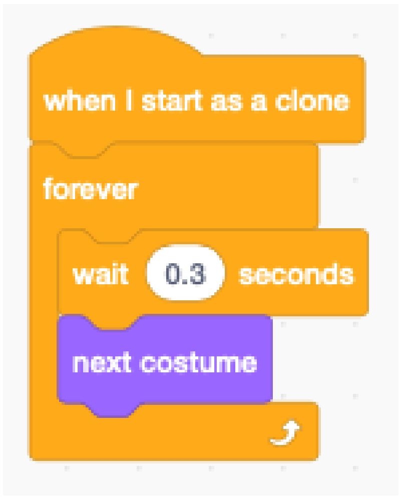
Inside I loop we

- go to a random position
- choose a randome size
- choose a random costume
- make a clone



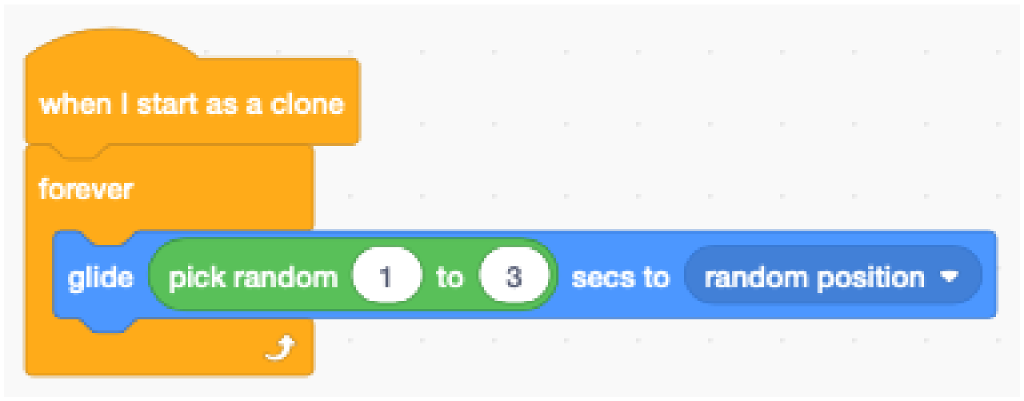
```
when 🏴 clicked
show
repeat (5)
    go to (random position ▾)
    switch costume to (pick random (1) to (3))
    set size to (pick random (30) to (70)) %
    create clone of (myself ▾)
hide
```

In order to give the impresion to fly, we change the costume every 0.3 seconds.



```
when I start as a clone
forever
    wait (0.3) seconds
    next costume
```

To move the bats around on the screen, we let them glide to random positions, every 1-3 seconds.



```
when I start as a clone
forever
    glide (pick random (1) to (3)) secs to (random position ▾)
```

4.2 Moving chickens

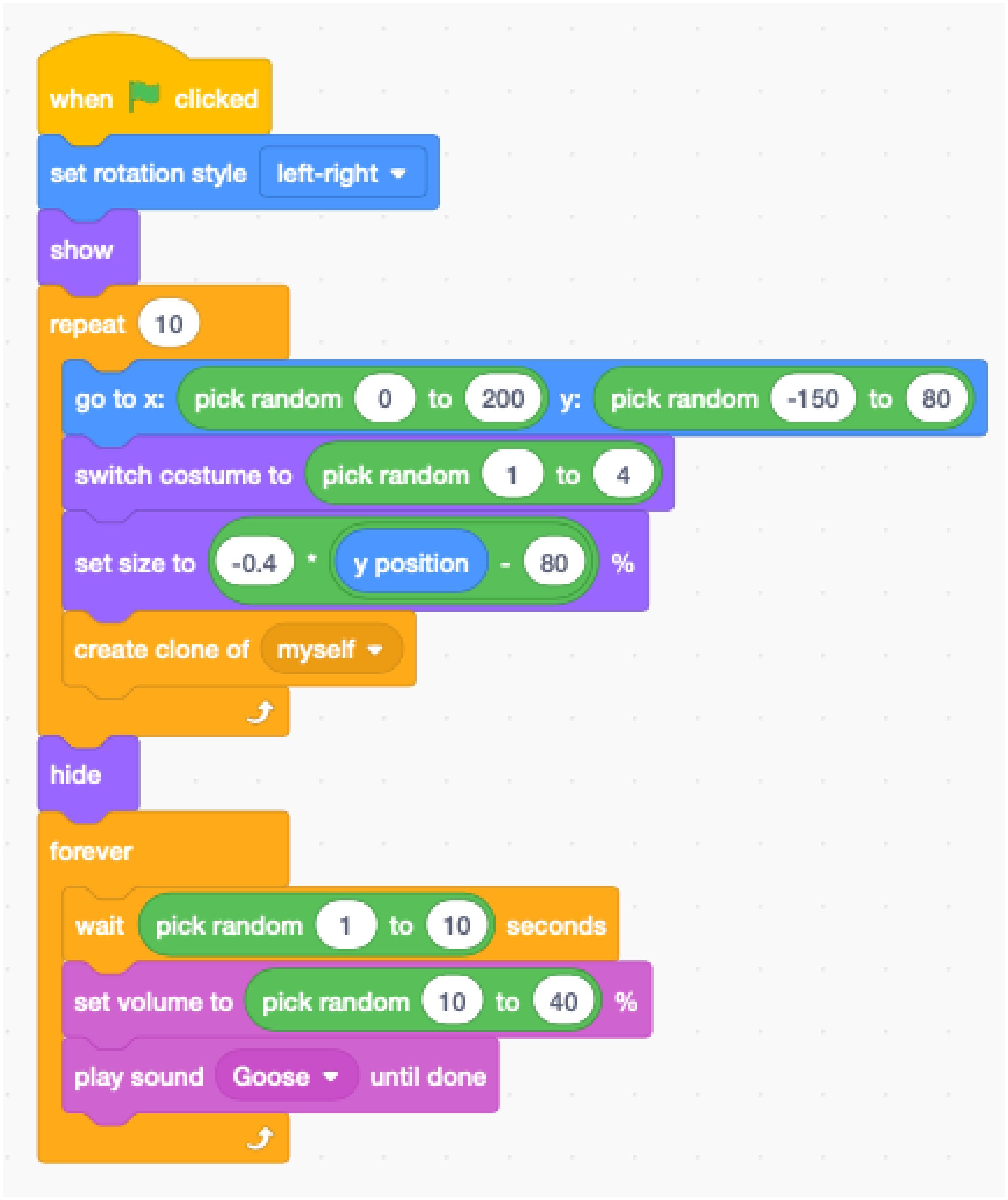
The following example shows how to animate chickens.

https://scratch.mit.edu/projects/395241257

At the start we set the rotation style to **left-right** because we do not want the hens to be upside down.

Inside the loop we :

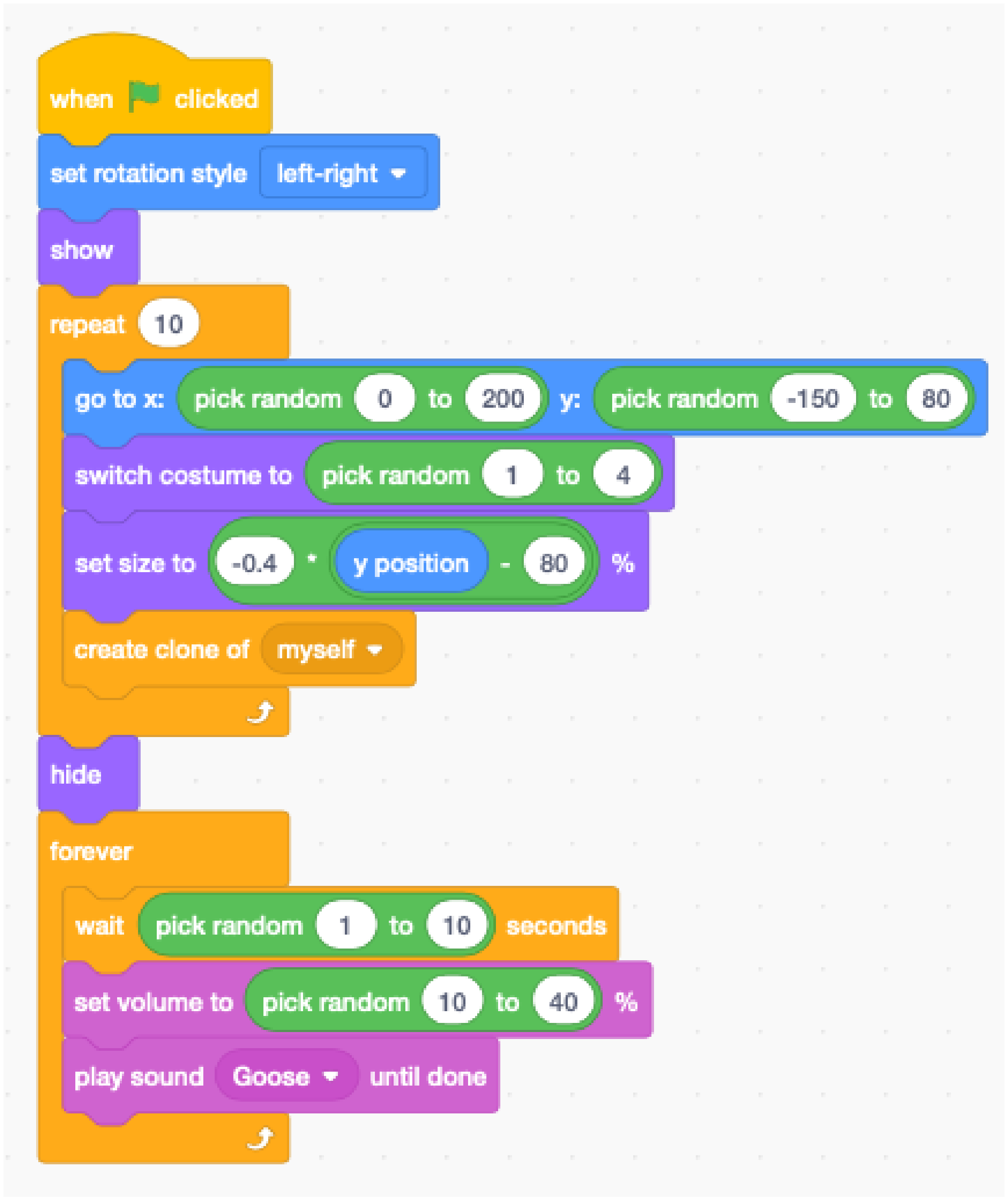
- go to a random position (avoding the cabbin and the far back)
- switch to a random costume
- set a size which depends on y
- create a clone



```
when ⚑ clicked
set rotation style left-right ▾
show
repeat 10
    go to x: pick random 0 to 200 y: pick random -150 to 80
    switch costume to pick random 1 to 4
    set size to -0.4 * y position - 80 %
    create clone of myself ▾
hide
forever
    wait pick random 1 to 10 seconds
    set volume to pick random 10 to 40 %
    play sound Goose ▾ until done
```

For each cloned hen we do this in a loop :

— switch to the next costume
— wait 1-2 seconds
— randomly move to the left or to the right
— randomly point to the left or to the right



```
when 🏁 clicked
set rotation style left-right ▾
show
repeat 10
    go to x: pick random 0 to 200 y: pick random -150 to 80
    switch costume to pick random 1 to 4
    set size to -0.4 * y position - 80 %
    create clone of myself ▾
hide
forever
    wait pick random 1 to 10 seconds
    set volume to pick random 10 to 40 %
    play sound Goose ▾ until done
```

4.3 Animate fish

https://scratch.mit.edu/projects/395268942

4.4 Animate a play

Wouldn't it be interesting to use Scratch to create a theater play. Let's take the first scene from **Alice in Wonderland** from the Gutenberg project.

https://www.gutenberg.org/files/35688/35688-h/35688-h.htm

4.5 Scene 1 : Alice's home

```
Alice’s home. Lewis Carroll is discovered, playing chess. Golden-haired Alice, in a␣
↪little blue dress, a black kitten in her arms, stands watching him.
Alice
That’s a funny game, uncle. What did you do then?
Carroll
A red pawn took a white pawn; this way. You see, Alice, the chessboard is divided␣
↪into sixty-four squares,
red and white, and the white army tries to win and the red army tries to win. It’s␣
↪like a battle!
Alice
With soldiers?
Carroll
Yes, here are the Kings and Queens they are fighting for. That’s the Red Queen and␣
↪here’s the White Queen.
Alice
How funny they look!
Carroll
See the crowns on their heads, and look at their big feet.
Alice
It’s a foot apiece, that’s what it is! Do they hump along like this?
Carroll
Here! You’re spoiling the game. I must keep them all in their right squares.
Alice
I want to be a queen!
Carroll
Here you are [he points to a small white pawn] here you are in your little stiff␣
↪skirt!
Alice
How do you do, Alice!
Carroll
And now you are going to move here.
Alice
Let me move myself.
Carroll
When you have traveled all along the board this way and haven’t been taken by the␣
↪enemy you may be a queen.
Alice
Why do people always play with kings and queens? Mother has them in her playing cards␣
↪too. Look!
[Alice goes to the mantel and takes a pack of playing cards from the ledge.]
Here’s the King of Hearts and here’s his wife; she’s the Queen of Hearts--isn’t she␣
↪cross-looking? wants to bite one’s head off.
[Carroll moves a pawn.]
You’re playing against yourself, aren’t you?
Carroll
That’s one way of keeping in practice, Alice; I have friends in the university who␣
↪want to beat me.
```

(suite sur la page suivante)

(suite de la page précédente)

```
Alice
But if you play against yourself I should think you'd want to cheat!
Carroll
Does a nice little girl like you cheat when she plays against herself?
Alice
Oh! I never do! I'd scold myself hard. I always pretend I'm two people too. It's lots␣
↪of fun, isn't it?
Sometimes when I'm all alone I walk up to the looking glass and talk to the other␣
↪Alice.
She's so silly, that Alice; she can't do anything by herself. She just mocks me all␣
↪the time.
When I laugh, she laughs, when I point my finger at her, she points her finger at me,␣
↪and when I stick my tongue out at her she sticks her tongue out at me!
Kitty has a twin too, haven't you darling?
[Alice goes to the mirror to show Kitty her twin.]
Carroll
I'll have to write a book some day about Alice--Alice in wonderland, "Child of the␣
↪pure unclouded brow and dreaming eyes of wonder!" or, Alice through the looking␣
↪glass!
Alice
Don't you wish sometimes you could go into looking-glass house? See!
[Alice stands on an armchair and looks into the mirror.]
There's the room you can see through the glass; it's just the same as our living-room␣
↪here, only the things go the other way.
I can see all of it--all but the bit just behind the fireplace. Oh! I do wish I could␣
↪see that bit!
I want so much to know if they've a fire there. You never can tell, you know, unless␣
↪our fire smokes.
Then smoke comes up in that room too--but that may be just to make it look as if they␣
↪had a fire--just to pretend they had.
The books are something like our books,[Pg 5] only the words go the wrong way. Won't␣
↪there ever be any way of our getting through, uncle?
Carroll
Do you think Kitty would find looking-glass milk digestible?
Alice
It doesn't sound awful good, does it; but I might leave her at home. She's been into␣
↪an awful lot of mischief today.
She found sister's knitting and chased the ball all over the garden where sister was␣
↪playing croquet with the neighbors.
And I ran and ran after the naughty little thing until I was all out of breath and so␣
↪tired! I am tired.
[She yawns and makes herself comfortable in the armchair.]
Carroll
[Replaces the playing cards on the mantel and consults his watch.]
Take a nap. Yes, you have time before tea.
Alice
[Half asleep.]
We're going to have mock turtle soup for supper! I heard mamma tell the cook not to␣
↪pepper it too much.
Carroll
What a funny little rabbit it is, nibbling all the time!
[He leans gently over the back of her chair, and seeing that she is going to sleep␣
↪puts out the lamp light and leaves the room.
A red glow from the fireplace illumines Alice.]
[Dream music. A bluish light reveals the Red Chess Queen and the White Chess Queen in␣
↪the mirror.]
Red Queen
```

(suite sur la page suivante)

(suite de la page précédente)

```
[Points to Alice and says in a mysterious voice.]
There she is, let's call her over.
White Queen
Do you think she'll come?
Red Queen
I'll call softly, Alice!
White Queen
Hist, Alice.
Red Queen
Alice!
White Queen
Hush--if she wakes and catches us--
Both Queens
Alice, come through into looking-glass house!
[Their hands beckon her.]
Alice
[Rises, and talks sleepily. The Queens disappear.
Alice climbs from the arm of the chair to the back of another and so on up to the mantel ledge, where she picks her way daintily between the vases.]
I--don't--know--how--I--can--get--through. I've tried--before--but the glass was hard--and I was afraid of cutting--my fingers--
[She feels the glass and is amazed to find it like gauze.]
Why, it's soft like gauze; it's turning into a sort of mist; why, it's easy to get through! Why--why--I'm going through!
[She disappears.]
```

Download : `alice.txt`

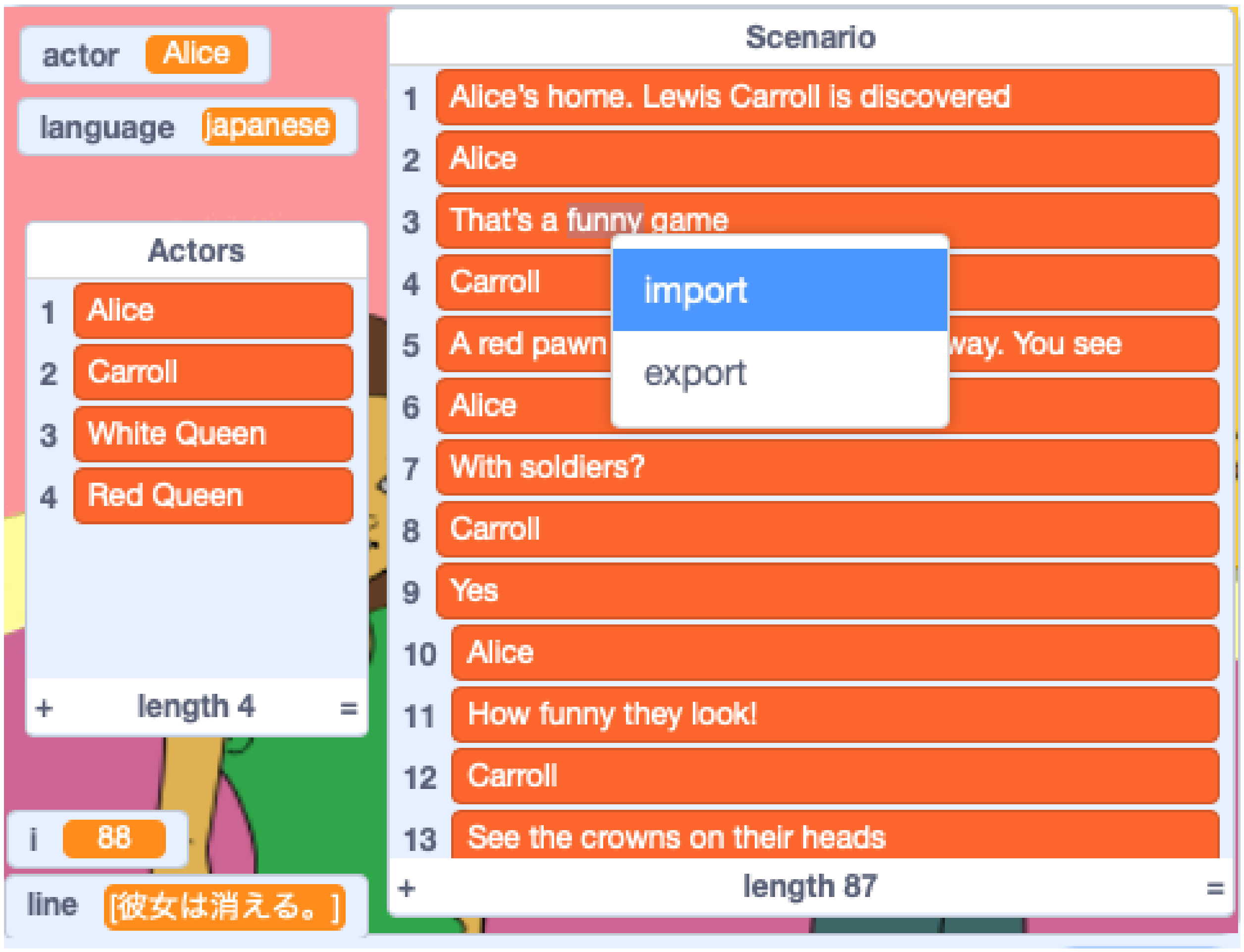
4.6 Scenario

The most important point here is to work with lists. We have two main lists :

- Scenario
- Actors

Fortunately we can import and export lists. Let's prepare the dialog in a text editor.

- remove the emply lines
- break dialog into lines of less then 128 characters (which is the limit for text-to-speech)
- place the actor's name on a single line
- followed by one more multiple lines of dialog for that actor


actor Alice

language japanese

Actors

1 Alice
2 Carroll
3 White Queen
4 Red Queen

+ length 4 =

i 88

line [彼女は消える。]

Scenario

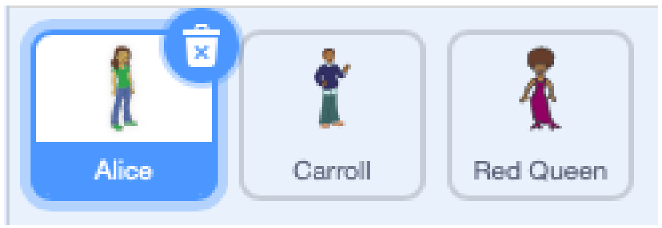
1 Alice's home. Lewis Carroll is discovered
2 Alice
3 That's a funny game
4 Carroll
5 A red pawn … way. You see
6 Alice
7 With soldiers?
8 Carroll
9 Yes
10 Alice
11 How funny they look!
12 Carroll
13 See the crowns on their heads

+ length 87 =

So far we have 3 actors :

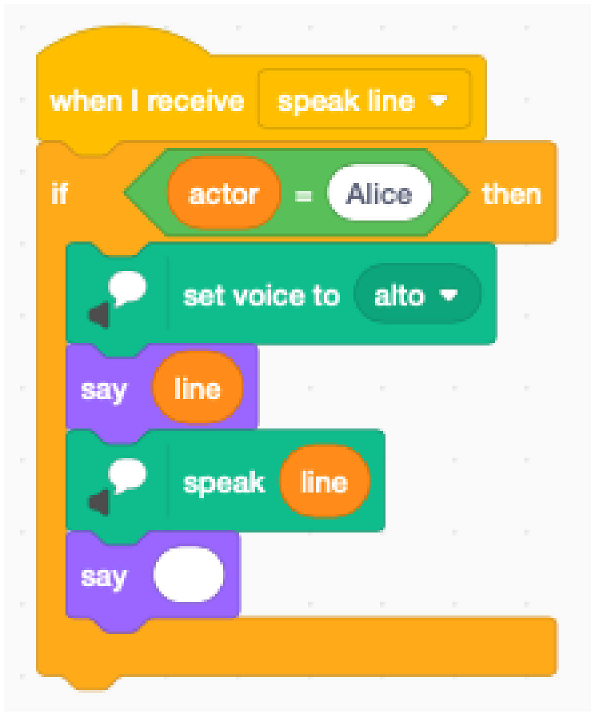
- Alice
- Carroll
- Red Queen

We use Capitalized names. These names will also figure in the **Actors** list.



Only sprites can show text. We use the broadcast message **read line** to

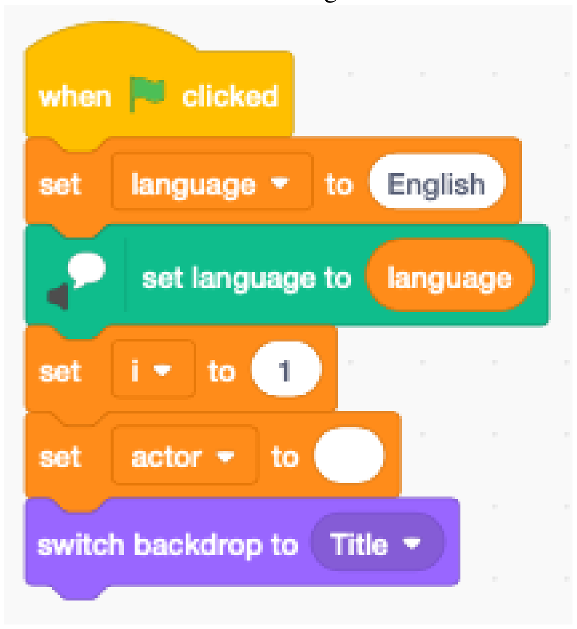
- set the voice for that actor
- show the dialog text in a bubble
- pronounce it with text-to-speach
- turn off the speech bubble



```
when I receive [speak line ▾]
if <(actor) = (Alice)> then
    set voice to (alto ▾)
    say (line)
    speak (line)
    say ()
```

At the start we

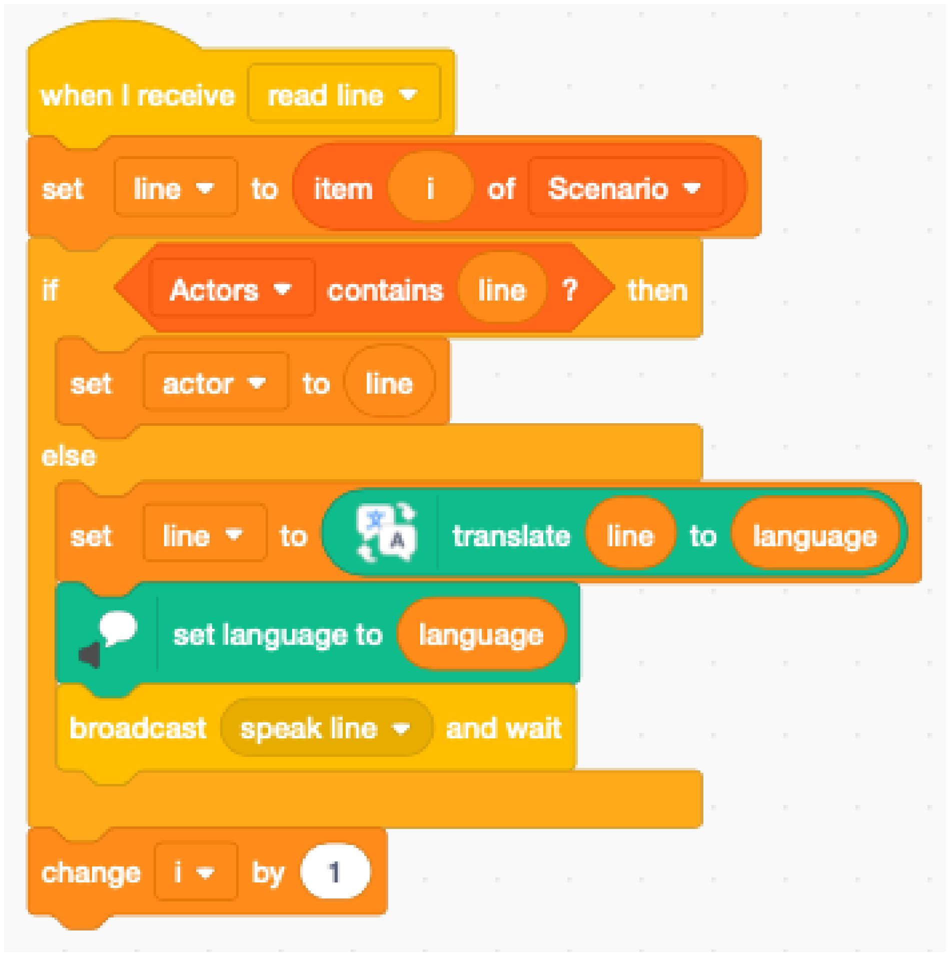
- set the language to English.
- set the index to 1
- set the actor to empty
- switch to the title background



```
when 🏳 clicked
set [language ▾] to (English)
set language to (language)
set [i ▾] to (1)
set [actor ▾] to ()
switch backdrop to (Title ▾)
```

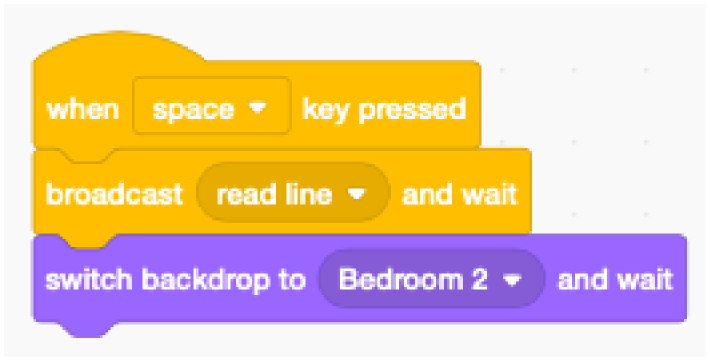
To read a single line, we load the current **Scenario** line into the variable **line**. If the line contains exactly the name of an actor in the **Actors** list, we set the **actor** variable and proceed.

Otherwise we translate the line of dialog and send the **speak line** broadcast.



```
when I receive [read line ▾]
set [line ▾] to (item (i) of [Scenario ▾])
if <[Actors ▾] contains (line) ?> then
    set [actor ▾] to (line)
else
    set [line ▾] to (translate (line) to (language))
    set language to (language)
    broadcast [speak line ▾] and wait
end
change [i ▾] by (1)
```

Pressing the SPACE bar will perform just the current line of dialog.



```
when [space ▾] key pressed
broadcast [read line ▾] and wait
switch backdrop to [Bedroom 2 ▾] and wait
```

Pressing A (All) will perform the whole play from the beginning to the end. .. image : : alice_all.png

https://scratch.mit.edu/projects/391344110

CHAPITRE 5

Cloud data

With Scratch you can save data on the cloud. This only works with the online version, not the offline version.

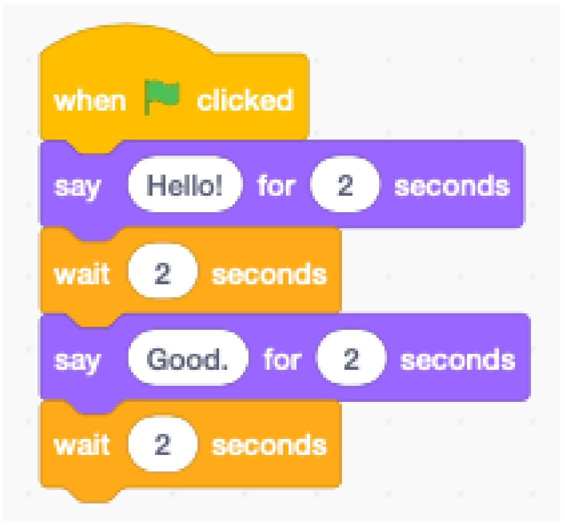
CHAPITRE 6

Dialog

https://scratch.mit.edu/projects/381954573

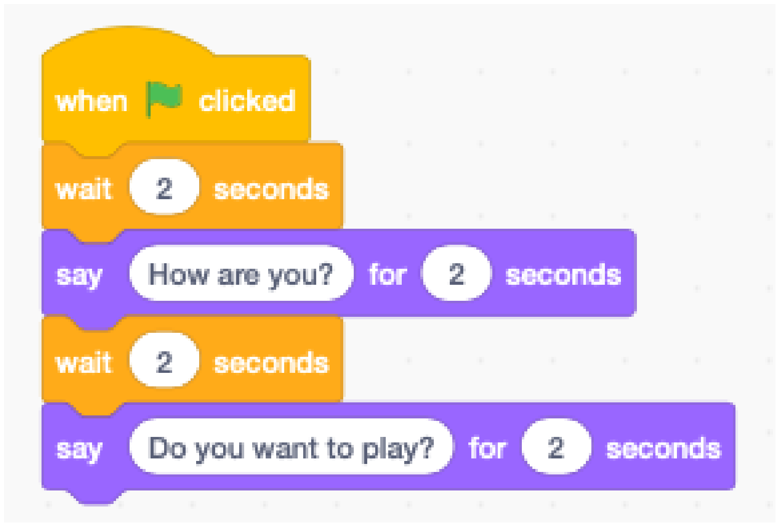
6.1 Use simple wait blocks

The simplest way to make a dialog between two sprites, is to use wait blocks. Here we have Scratchy the cat and Gobo the blowfish talk to each other.



```
when flag clicked
say Hello! for 2 seconds
wait 2 seconds
say Good. for 2 seconds
wait 2 seconds
```

Each sprite needs to respect the other sprite's timing.



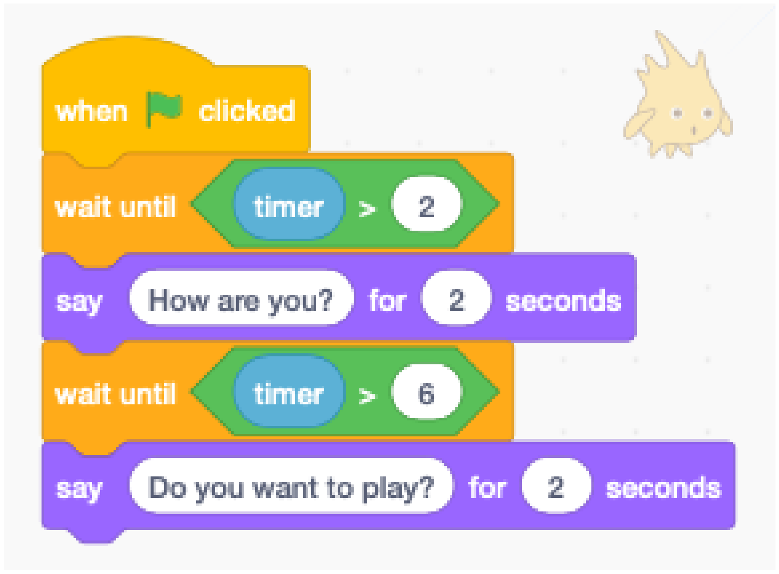
```
when [flag] clicked
wait 2 seconds
say How are you? for 2 seconds
wait 2 seconds
say Do you want to play? for 2 seconds
```

6.2 Use the timer

https://scratch.mit.edu/projects/390653673

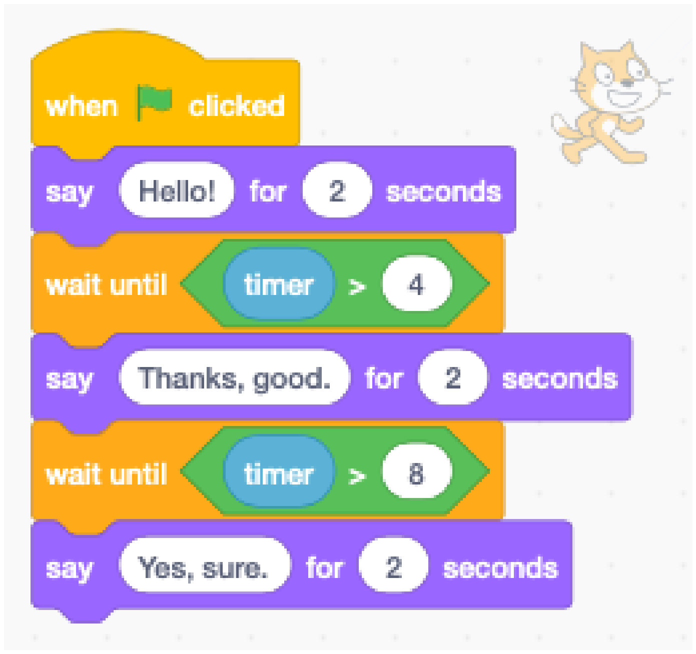
A much better way is to use the timer. Each Scratch project has a timer. When the program starts, the timer starts incrementing. It is possible to reset the timer. Here we use the timer to know the onset of a new part of dialog.

The Scratchy starts talking at 0, 4, and 8 seconds.



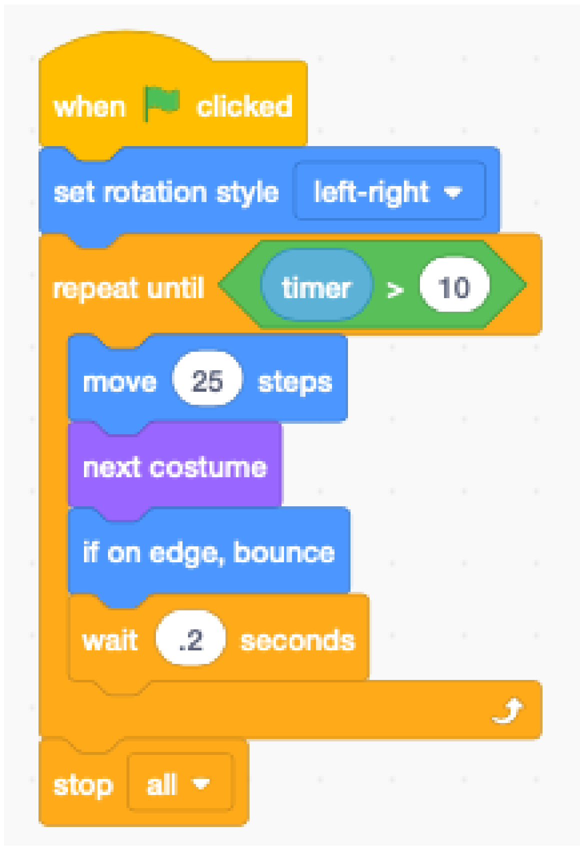
```
when [flag] clicked
wait until timer > 2
say How are you? for 2 seconds
wait until timer > 6
say Do you want to play? for 2 seconds
```

Gobo starts talking at 2 and 6 seconds.



```
when flag clicked
say Hello! for 2 seconds
wait until timer > 4
say Thanks, good. for 2 seconds
wait until timer > 8
say Yes, sure. for 2 seconds
```

Simultaneously the sprite can do an activity, such as walking. Here the Scratchy paces from left to write from 0 to 10 seconds.



```
when flag clicked
set rotation style left-right
repeat until timer > 10
    move 25 steps
    next costume
    if on edge, bounce
    wait .2 seconds
stop all
```

6.3 Use a list for dialog

If you have a lot of dialog, it's best to use a list.

https://scratch.mit.edu/projects/390664649

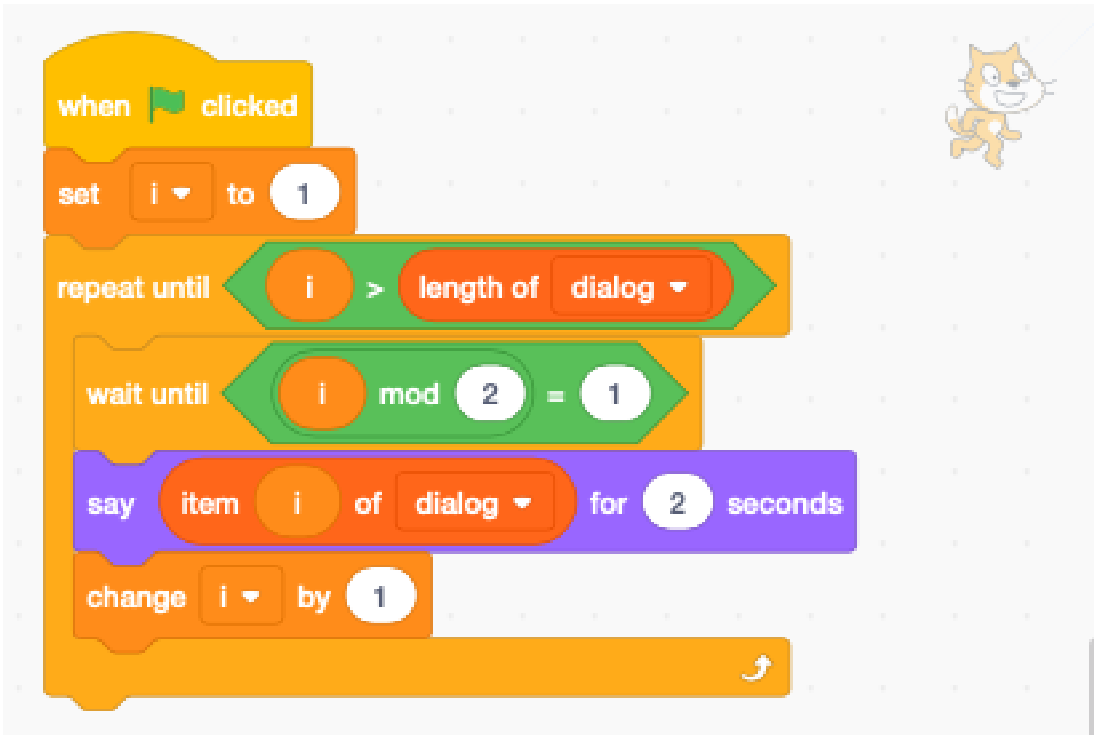
It's easy to change the dialog. Just change the list items.

To start with, you need to create
- a list **dialog** to contain the dialog items
- a variable **i** to be used as list index

We iterate until the list index **i** is larger than the list length.

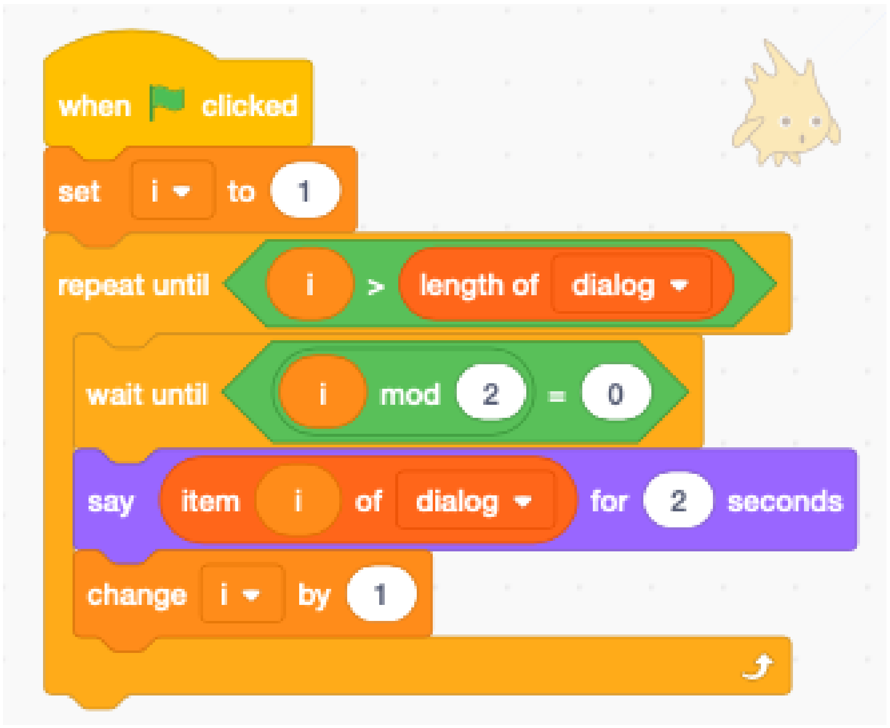
The **mod** operator returns the rest of the division by 2.
- **i mod 2 = 1** means i is **odd**
- **i mod 2 = 0** means i is **even**

For Scratchy we wait until the list index **i** is odd.



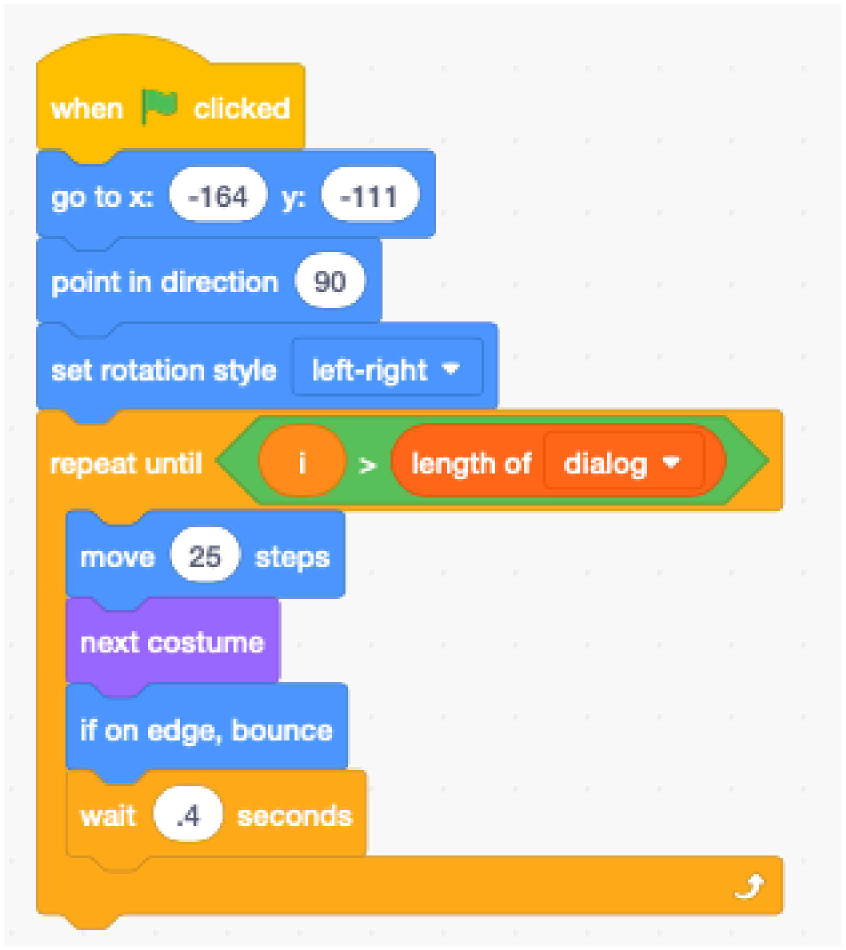
```
when flag clicked
set i to 1
repeat until i > length of dialog
    wait until i mod 2 = 1
    say item i of dialog for 2 seconds
    change i by 1
```

For Gobo we wait until the list index **i** is even.



```
when [flag] clicked
set i to 1
repeat until < (i) > (length of dialog) >
    wait until < ((i) mod (2)) = (0) >
    say (item (i) of dialog) for (2) seconds
    change i by 1
```

Again, we make Scratchy walk, while he is talking. This time we set the inital position and the initial direction.



```
when [flag] clicked
go to x: -164 y: -111
point in direction 90
set rotation style left-right
repeat until < i > length of dialog >
    move 25 steps
    next costume
    if on edge, bounce
    wait .4 seconds
```

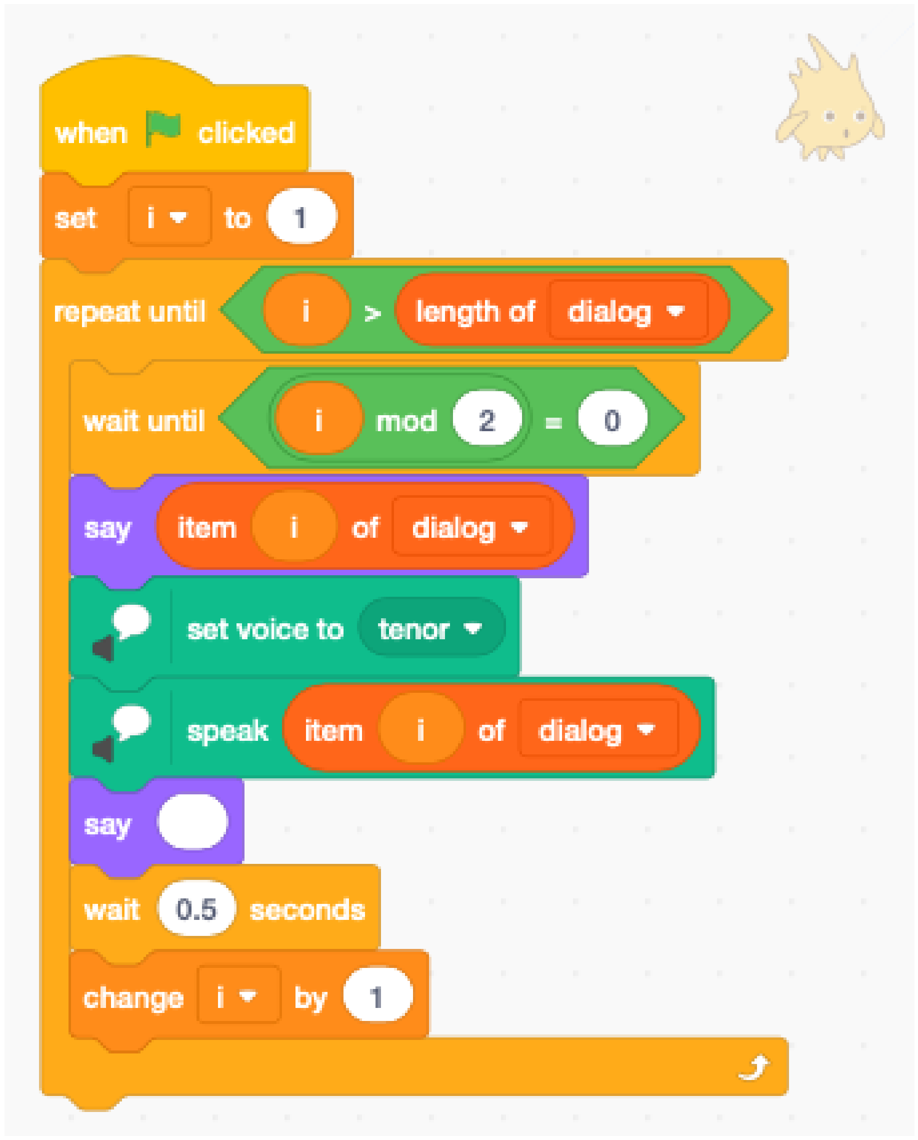
6.4 Spoken dialog

In the bottom of the block palette, click the extensions button and select the **Text to Speech** extension.

https://scratch.mit.edu/projects/390682479

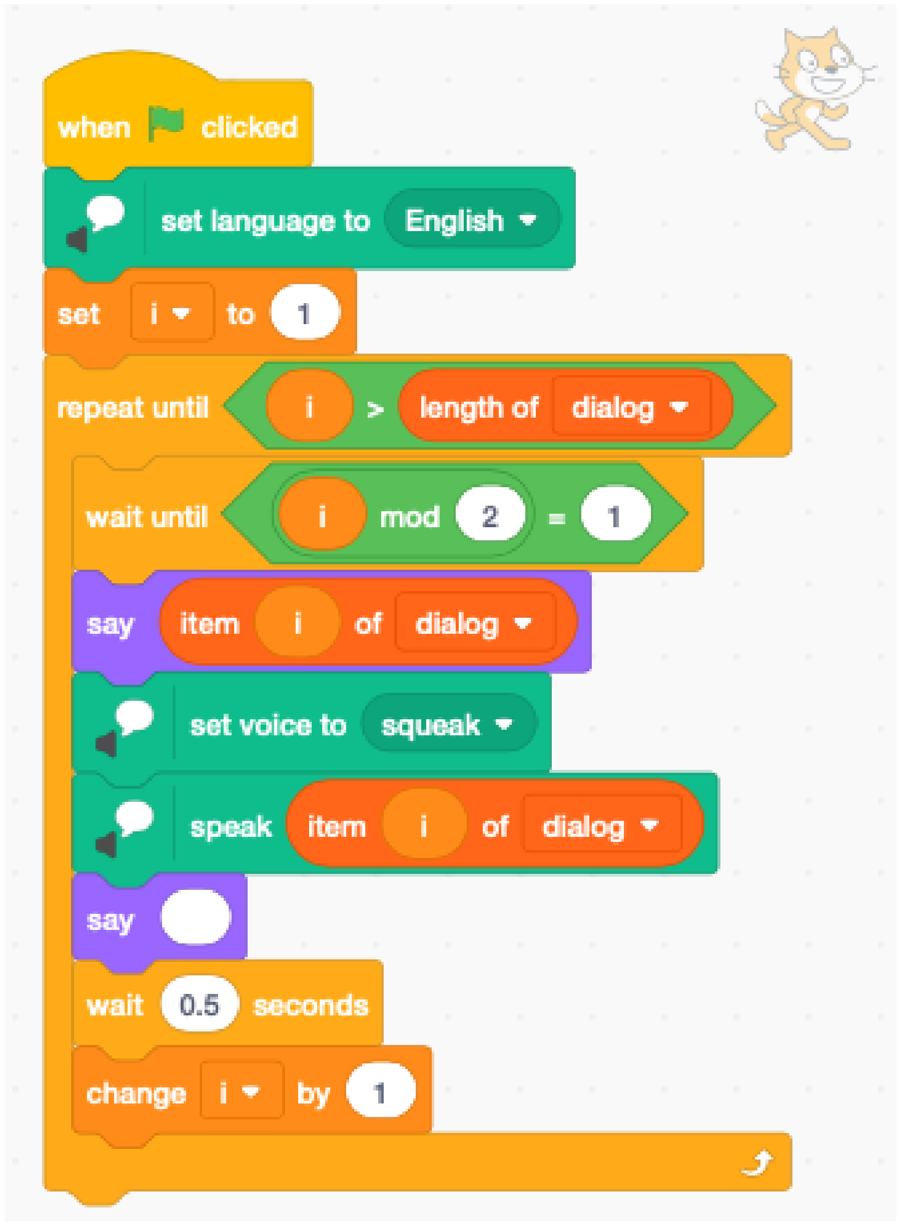
Now you can have the sprites pronounce the dialog.

We give a **tenor** voice to Gobo. The duration of the speech is now dermining the display of the bubble. To turn off the bubble, you have to say an empty text.



```
when 🏳 clicked
set i to 1
repeat until i > length of dialog
    wait until i mod 2 = 0
    say item i of dialog
    set voice to tenor
    speak item i of dialog
    say
    wait 0.5 seconds
    change i by 1
```

Scratchy is using an **squeaky** voice.



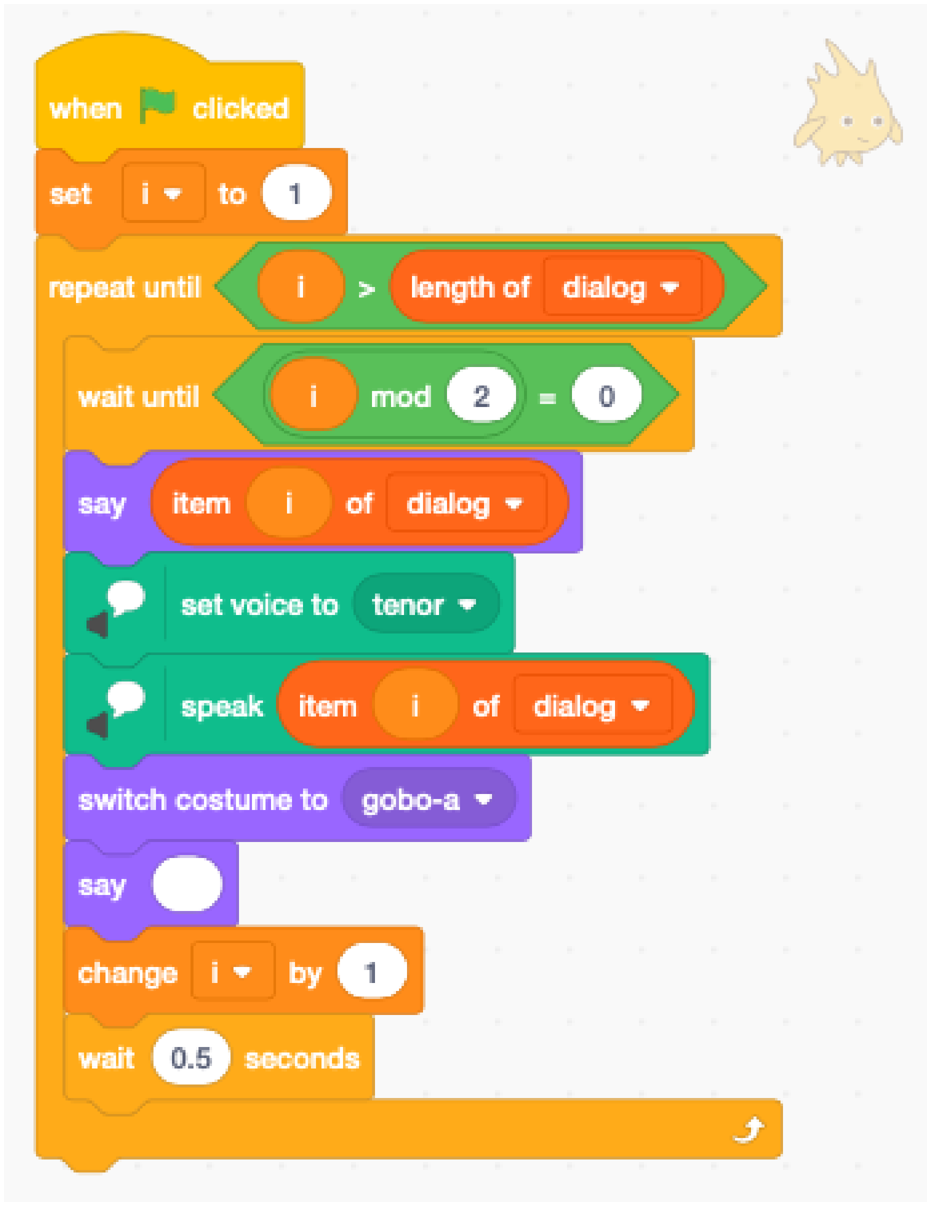
```
when 🏁 clicked
set language to English ▾
set i ▾ to 1
repeat until < (i) > (length of dialog ▾) >
    wait until < ((i) mod 2) = 1 >
    say (item (i) of dialog ▾)
    set voice to squeak ▾
    speak (item (i) of dialog ▾)
    say ()
    wait 0.5 seconds
    change i ▾ by 1
```

6.5 Add mouth movement

Gobo has 3 costumes with different mouth positions
— gobo-a : closed
— gobo-b : half-open
— gobo-c : open
Since they have a discussion about tennis, we add a sports ground as a back drop.

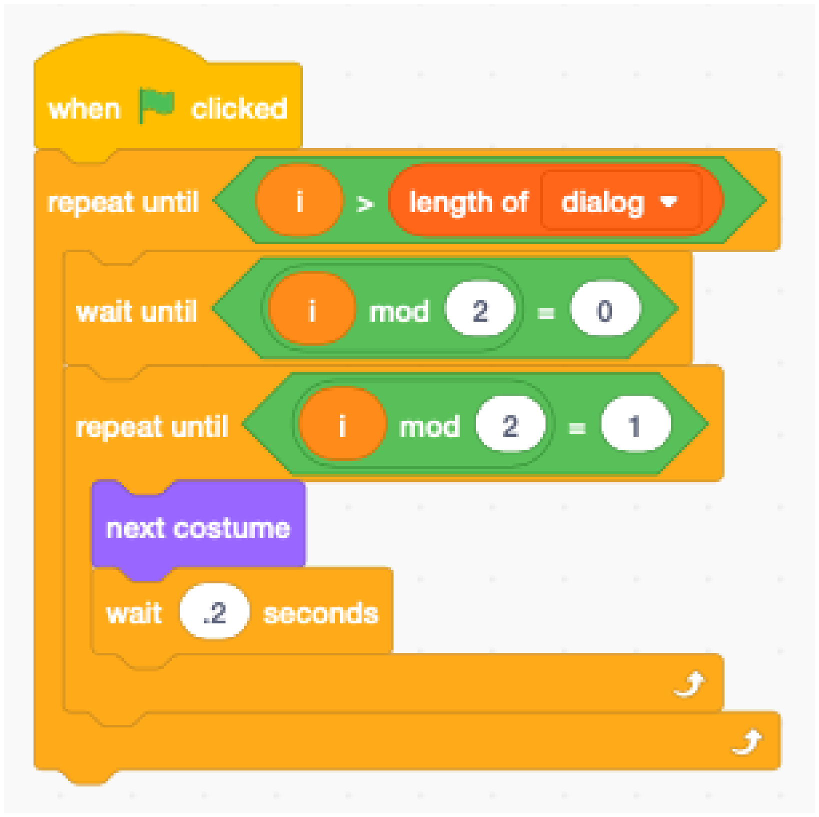
https://scratch.mit.edu/projects/390691994

Inside the code for speaking we add a block to close the mouth after speaking.



```
when [flag] clicked
set i to 1
repeat until i > length of dialog
    wait until i mod 2 = 0
    say item i of dialog
    set voice to tenor
    speak item i of dialog
    switch costume to gobo-a
    say
    change i by 1
    wait 0.5 seconds
```

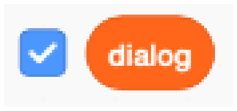
We add a second stack, which is active during the time the index **i** is even. It cycles through the costumes every 0.2 second.



```
when flag clicked
repeat until < (i) > (length of [dialog v]) >
    wait until < ((i) mod (2)) = (0) >
    repeat until < ((i) mod (2)) = (1) >
        next costume
        wait (.2) seconds
    end
end
```

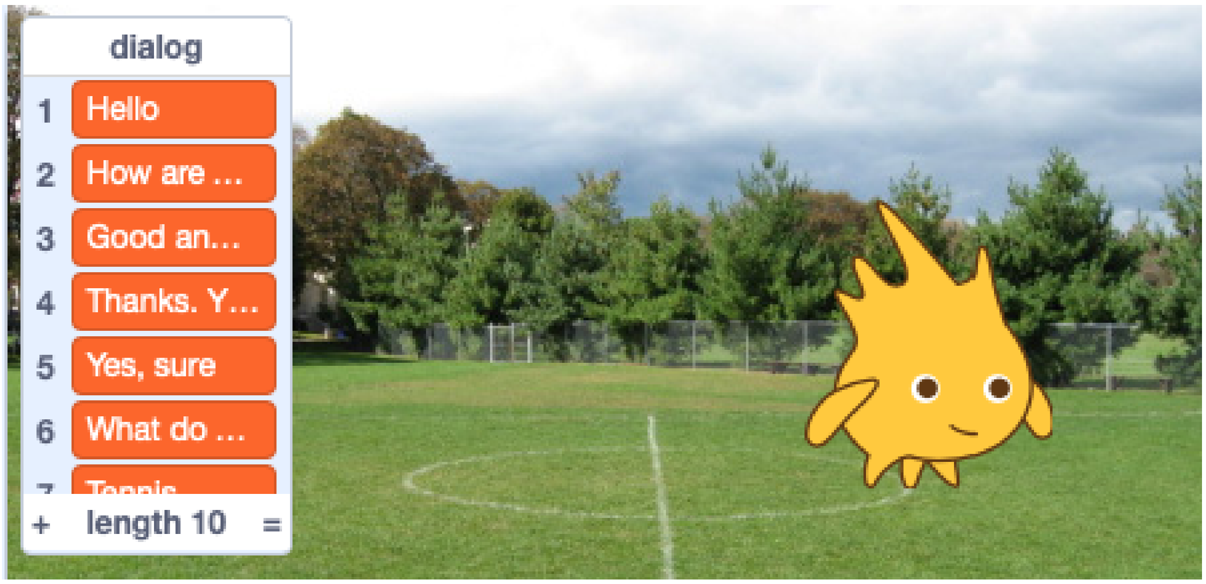
6.6 Enter the dialog

To enter the dialog into the list, you have the make the list visible on the stage. Check the box next to the list reporter in the palette.



Now you can
- modify
- add
- delete

dialog items in the **dialog** list.


dialog
1 Hello
2 How are …
3 Good an…
4 Thanks. Y…
5 Yes, sure
6 What do …
7 Tennis
+ length 10 =

CHAPITRE 7

Menu

In this section we handle

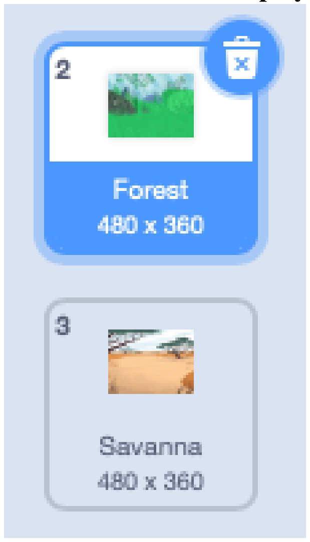
- backdrops
- buttons
- background music

https://scratch.mit.edu/projects/391162659

7.1 Backdrops

For this project we select two backdrops

- *Forest* for the **home** screen
- *Savanna* for the **play** screen

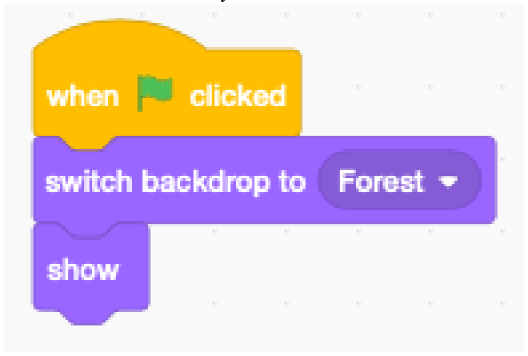


The home screen has a play button. The play screen has a home button.

7.2 The home screen

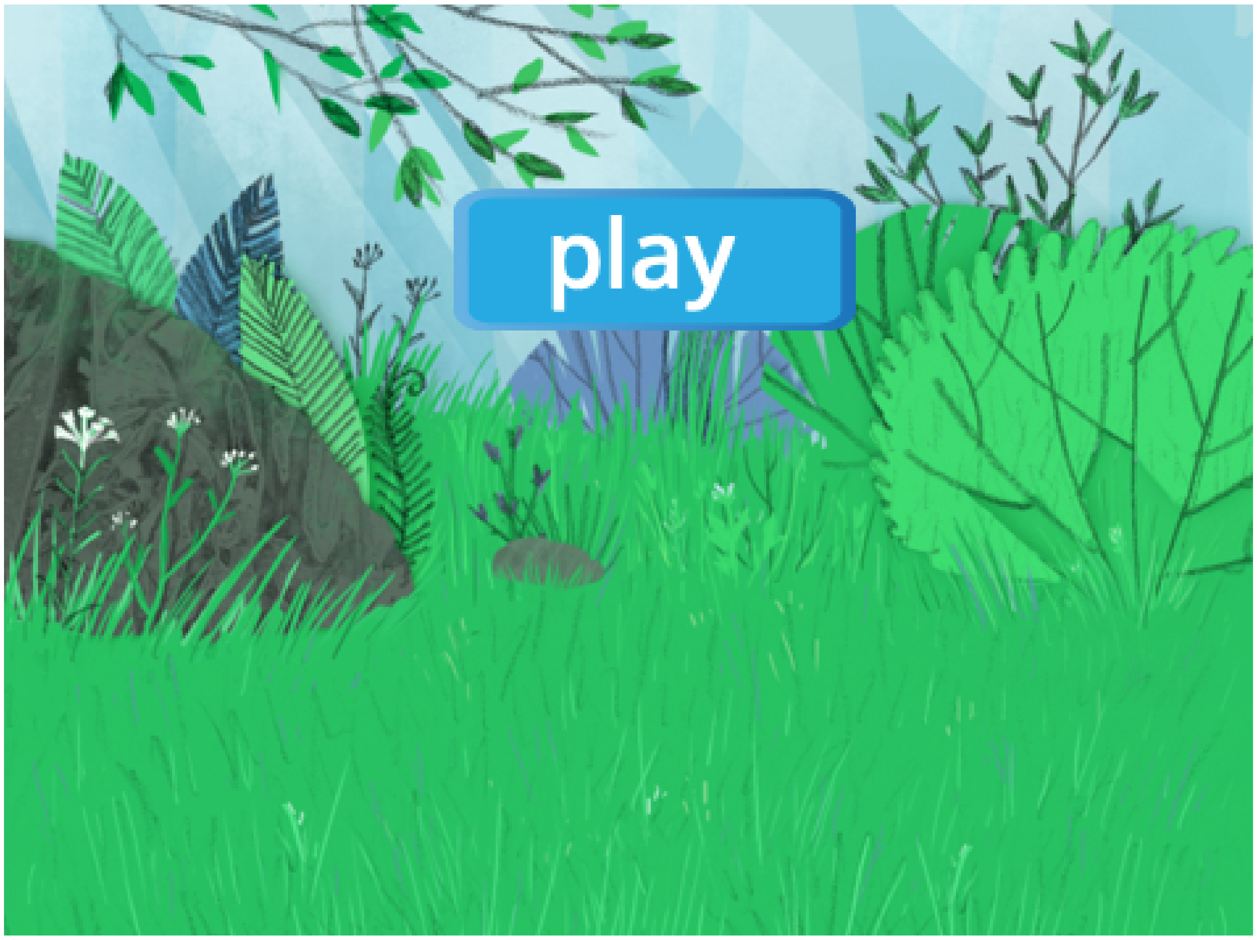
When the program starts we

- switch to the *Forest* background
- show the *Play* button



```
when [flag] clicked
switch backdrop to (Forest ▾)
show
```

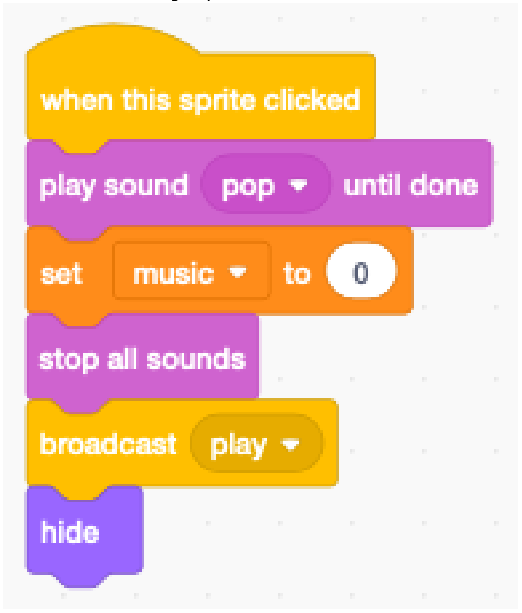
We could add more buttons for options, high-score, etc.



7.3 The play button

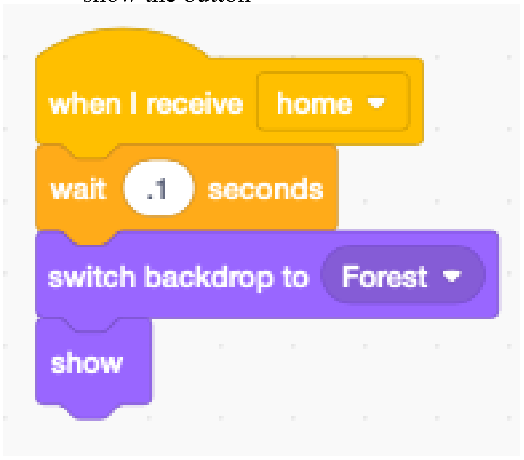
When the play button is clicked we

— play a pop sound
— set the boolean variable *music* to zero (to stop the music loop)
— stop all sounds
— broadcast the message *play*
— hide the *play* button



```
when this sprite clicked
play sound pop until done
set music to 0
stop all sounds
broadcast play
hide
```

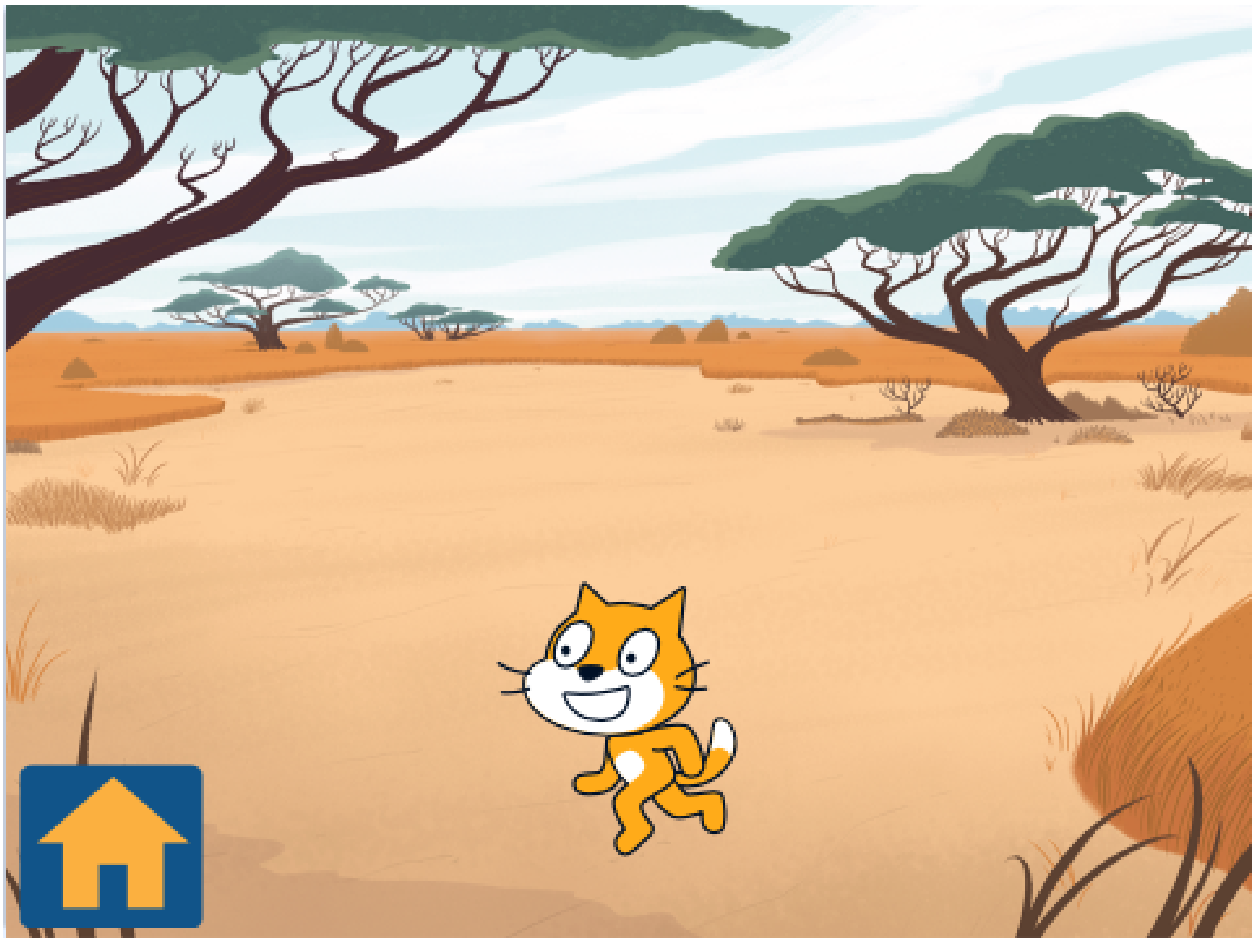
When the play button receives the *home* message it
— waits 0.1 seconds (to be sure the old music stops)
— switch to the *home* background to *Forest*
— show the button



```
when I receive home
wait .1 seconds
switch backdrop to Forest
show
```

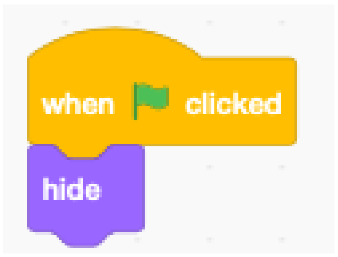
7.4 The play screen

The play screen has a home button and a player (Scratchy).



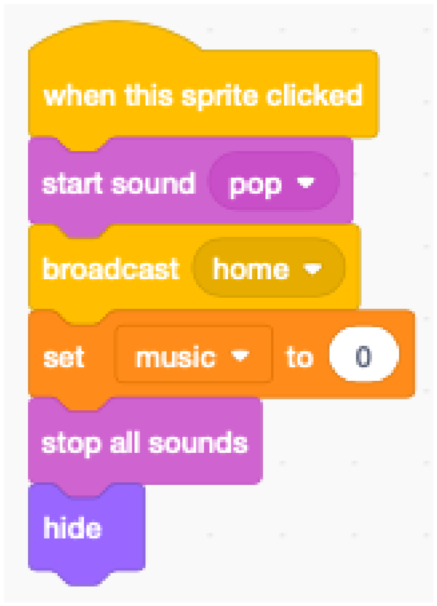
7.5 The home button

The program always starts with the **home** screen, so initially the home button is hidden.



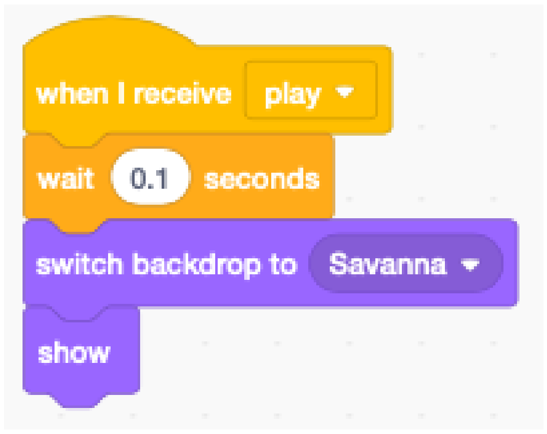
```
when flag clicked
hide
```

When the *home* clicked is clicked it stops the current music and broadcasts the *home* message.



```
when this sprite clicked
start sound pop
broadcast home
set music to 0
stop all sounds
hide
```

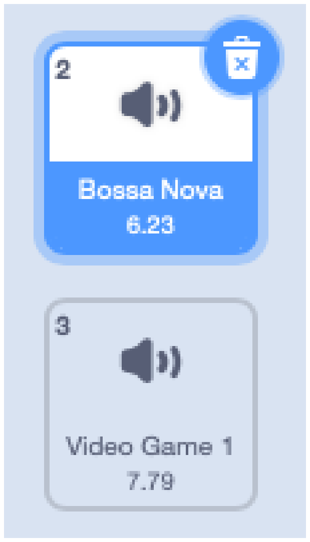
When it receives the *play* message it shows play screen and home button.



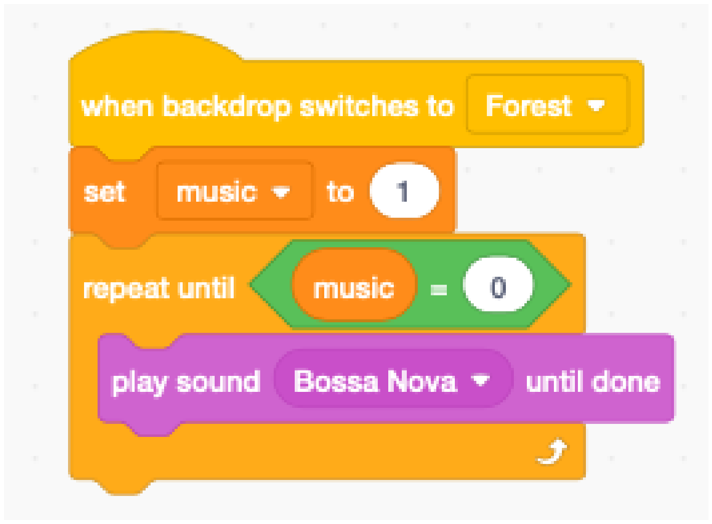
```
when I receive play
wait 0.1 seconds
switch backdrop to Savanna
show
```

7.6 Background music

The background music is associated with the stage. We choose :
- *Bossa Nova* for the home screen
- *Video Game 1* for the play screen

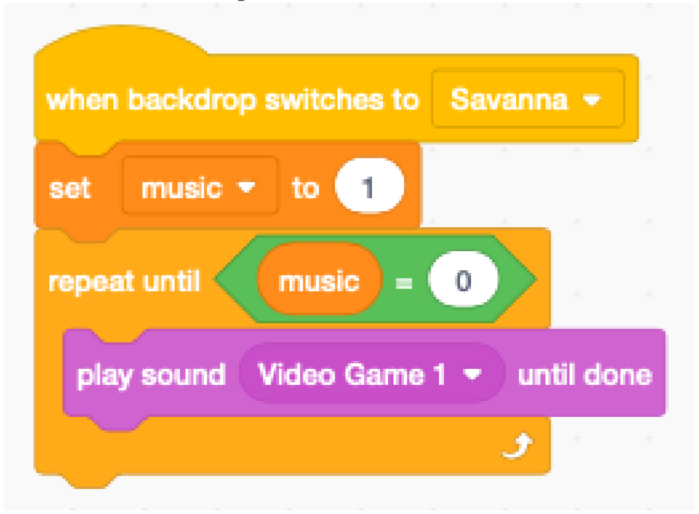


When the backdrop switches, a new music is played



```
when backdrop switches to [Forest ▾]
set [music ▾] to (1)
repeat until <(music) = (0)>
    play sound [Bossa Nova ▾] until done
```

— the boolean variable **music** is set to 1
— the sound is repeated until the boolean **music** is reset to 0



```
when backdrop switches to [Savanna ▾]
set [music ▾] to (1)
repeat until <(music) = (0)>
    play sound [Video Game 1 ▾] until done
```

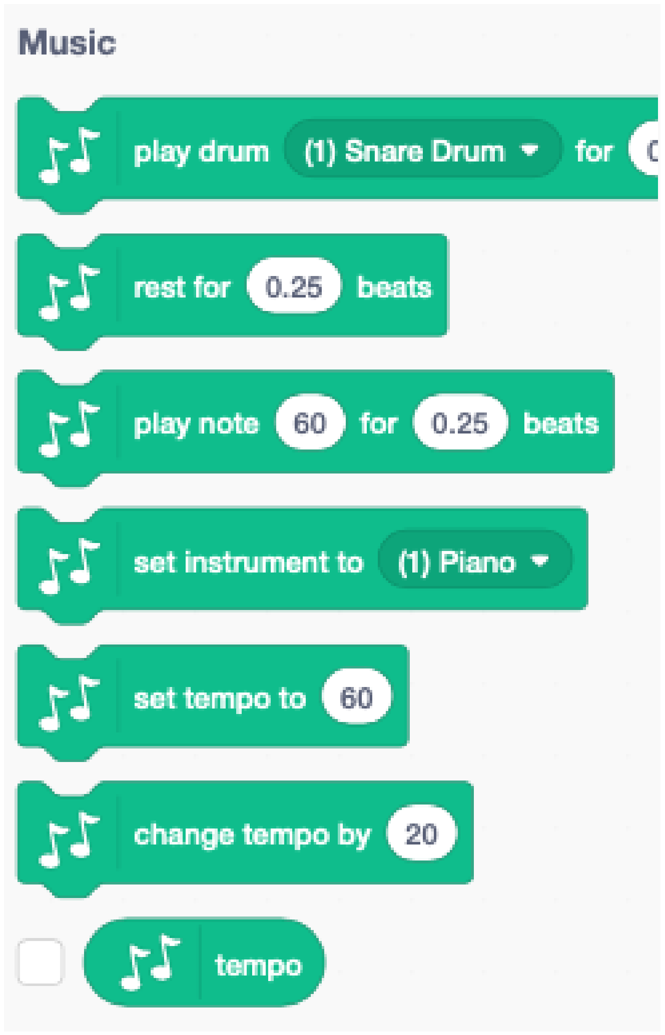
CHAPITRE 8

Music

With Scratch you can play music.

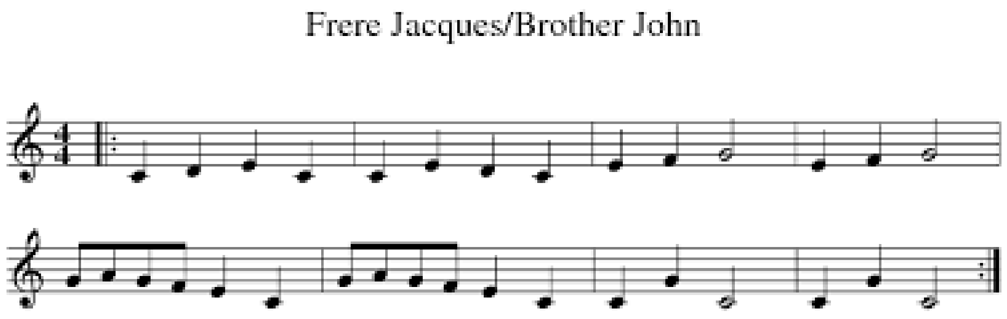
https://scratch.mit.edu/projects/395043889

You can play music in Scratch, but you have to load the extension **Music**. This will add the following new blocks :


Music
play drum (1) Snare Drum for
rest for 0.25 beats
play note 60 for 0.25 beats
set instrument to (1) Piano
set tempo to 60
change tempo by 20
tempo

8.1 Play a melody

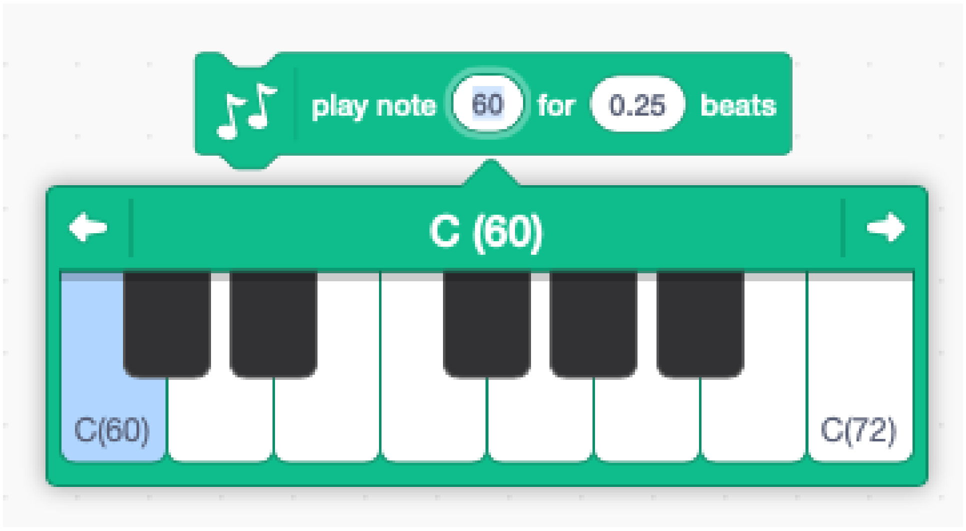
Lets play the music French folk song **Brother John**.



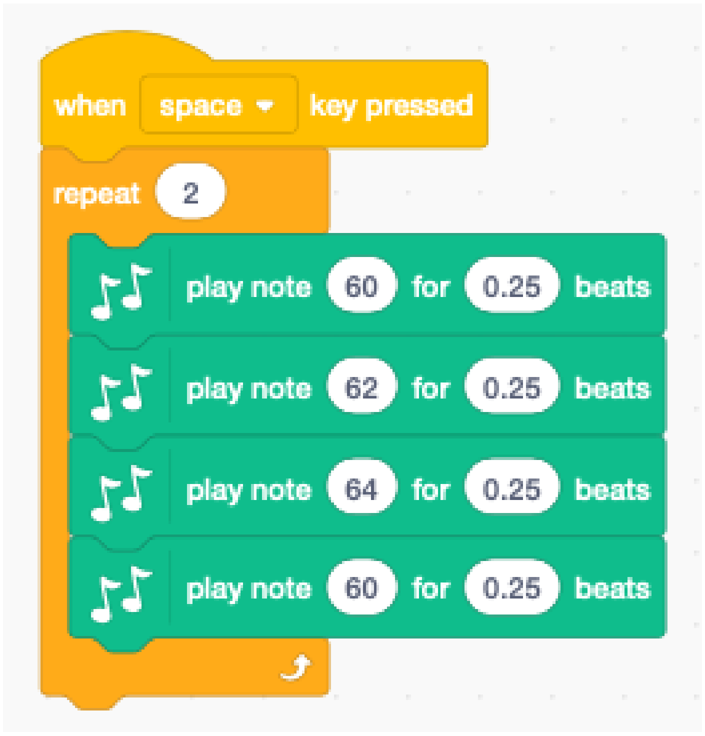
During the first measure we have 4 quarter notes.

- C (60)
- D (62)
- E (64)

— C (60)
Select the block **play note** and select the note C (60) on the pop-up keyboard which appears. You can leave the duration at 0.25 beats.

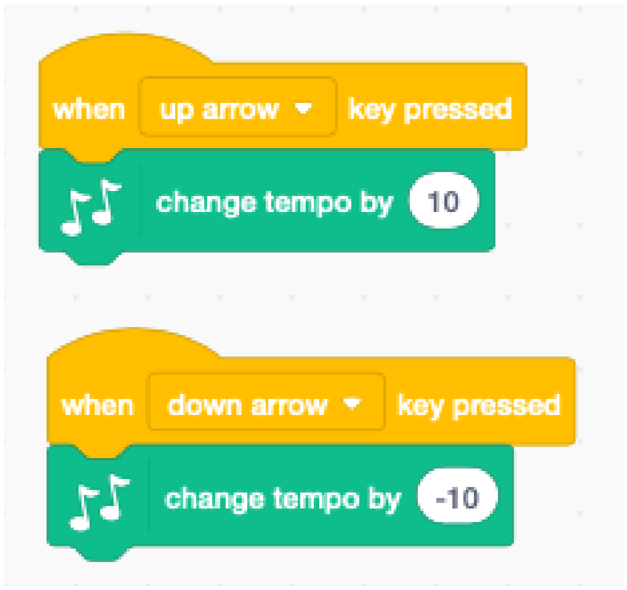


The same 4 notes repeat for a second measure. We can use a loop to repeat.


```
when space key pressed
repeat 2
    play note 60 for 0.25 beats
    play note 62 for 0.25 beats
    play note 64 for 0.25 beats
    play note 60 for 0.25 beats
```

8.2 Change the tempo

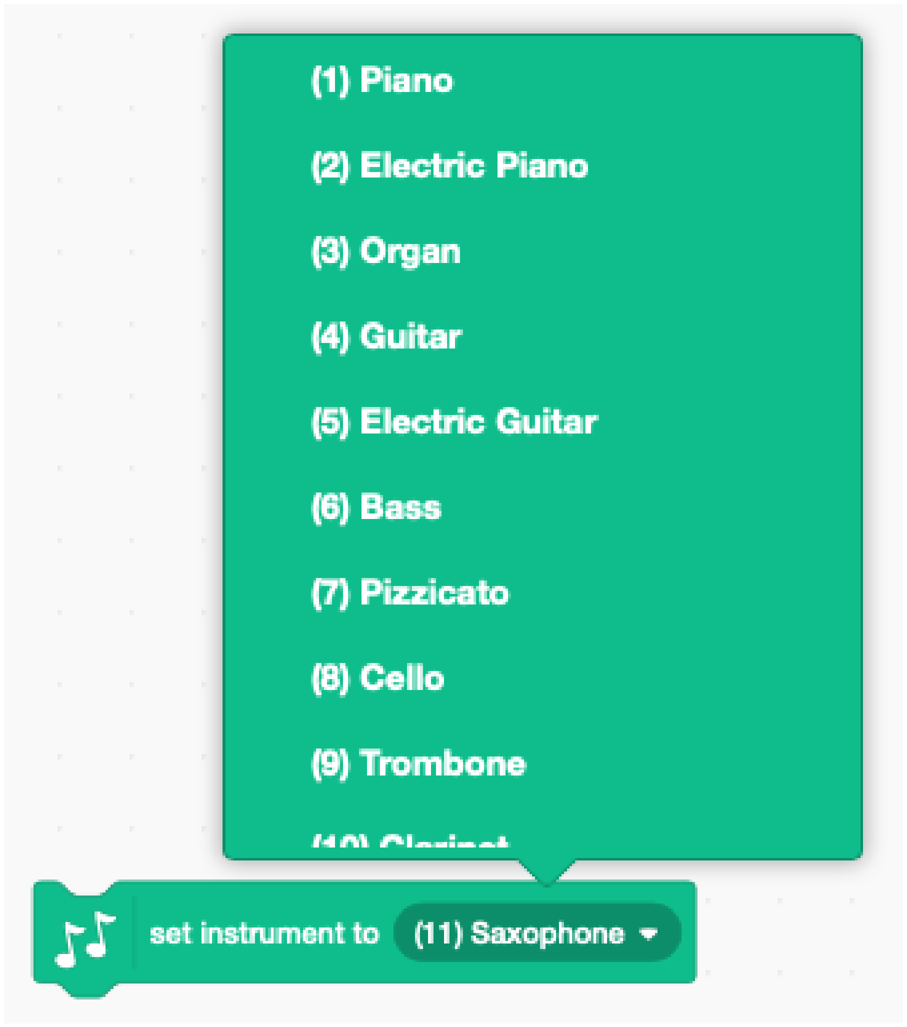
The original tempo is 60 bpm (beats per minute). Let's add a way to change the tempo with the up/down keys.



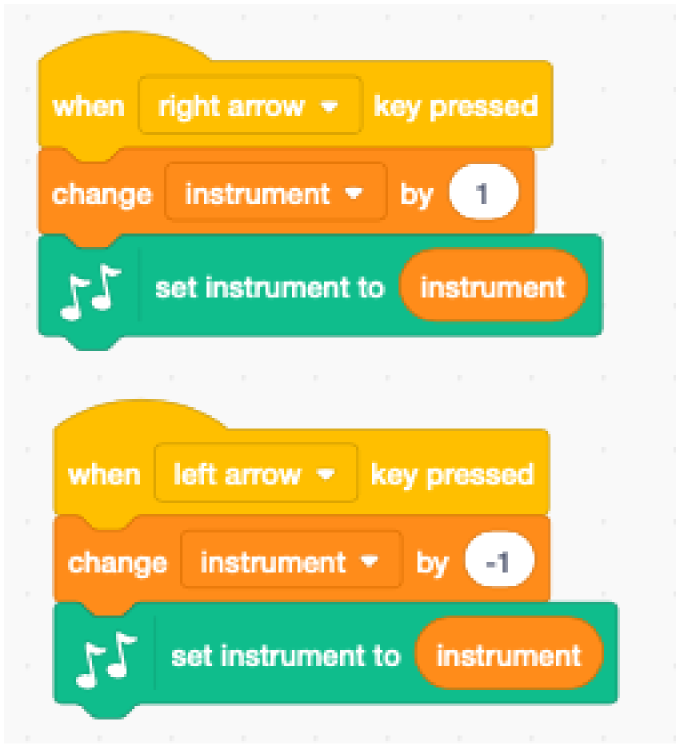
```
when up arrow key pressed
change tempo by 10

when down arrow key pressed
change tempo by -10
```

8.3 Change the instrument

The default instrument is a piano (1). The block **set instrument to** allows to chose among 21 different instruments.



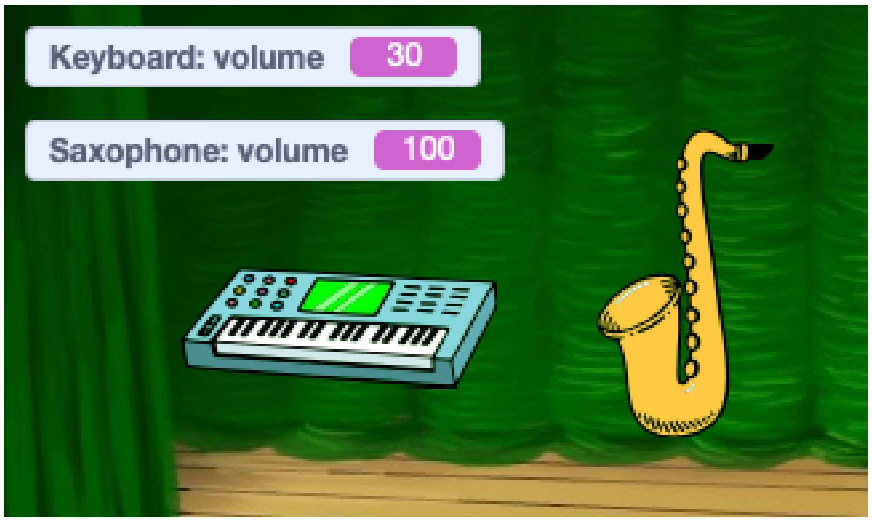
Create a variable called **instrument**. Let's add a way to change the instrument with the left/right keys.



```
when [right arrow] key pressed
change [instrument] by (1)
set instrument to (instrument)

when [left arrow] key pressed
change [instrument] by (-1)
set instrument to (instrument)
```

8.4 Set the volume

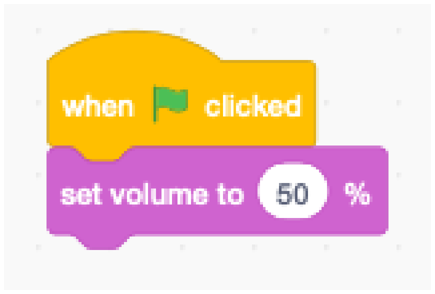
The volume can be set separately for each sprite. For example the keyboard can have the volume at 30%, while the saxophone has the volume at 100%.



Keyboard: volume 30

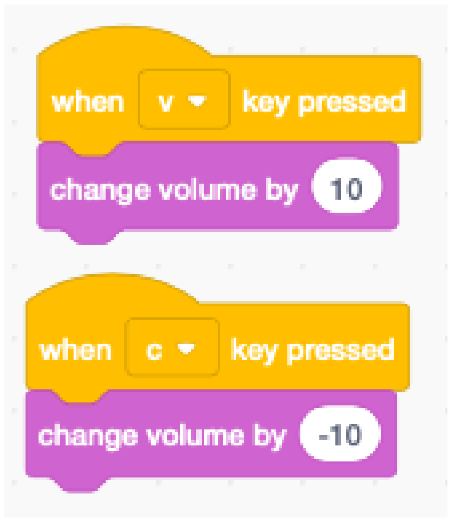
Saxophone: volume 100

At the start we set the volume to 50%.



```
when 🏳 clicked
set volume to (50) %
```

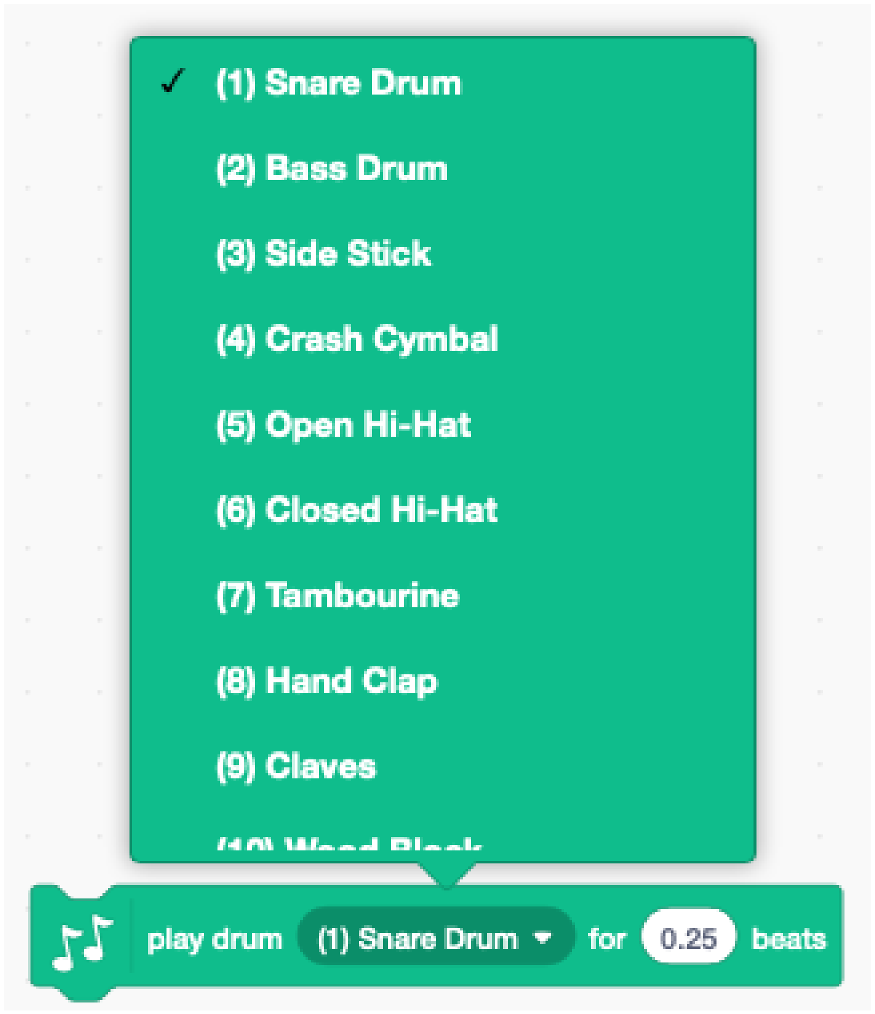
We use the two C/V buttons to set the volume.



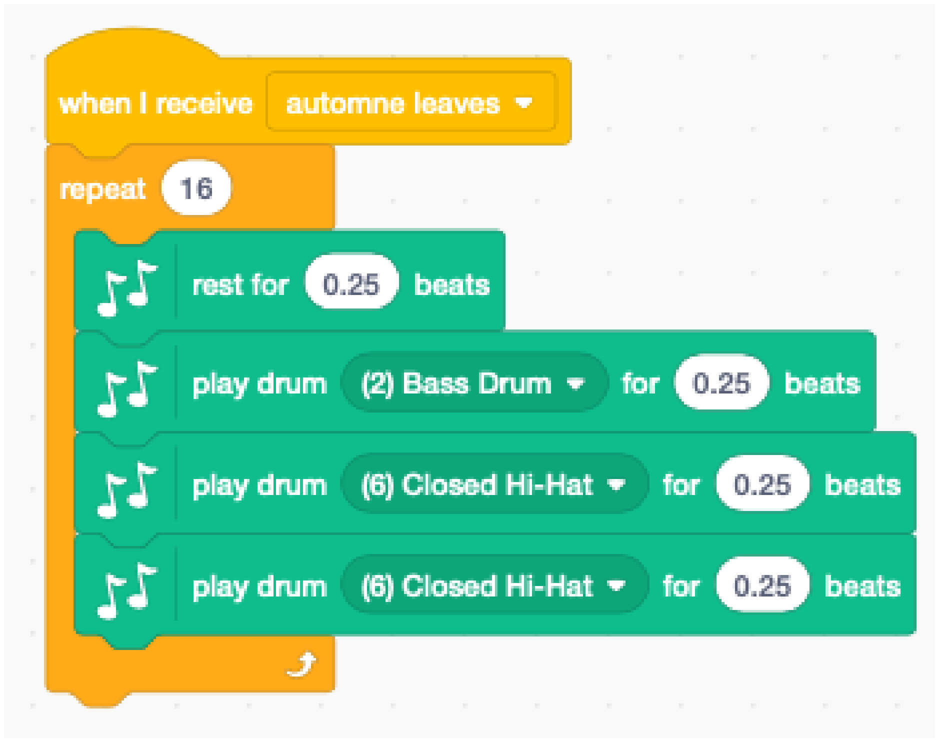
```
when [v ▾] key pressed
change volume by (10)

when [c ▾] key pressed
change volume by (-10)
```

8.5 Play a drum

The block **play drum** allows to play one of 18 different drums.



We can combine 4 quarter beats in a loop and repeat 16 times.



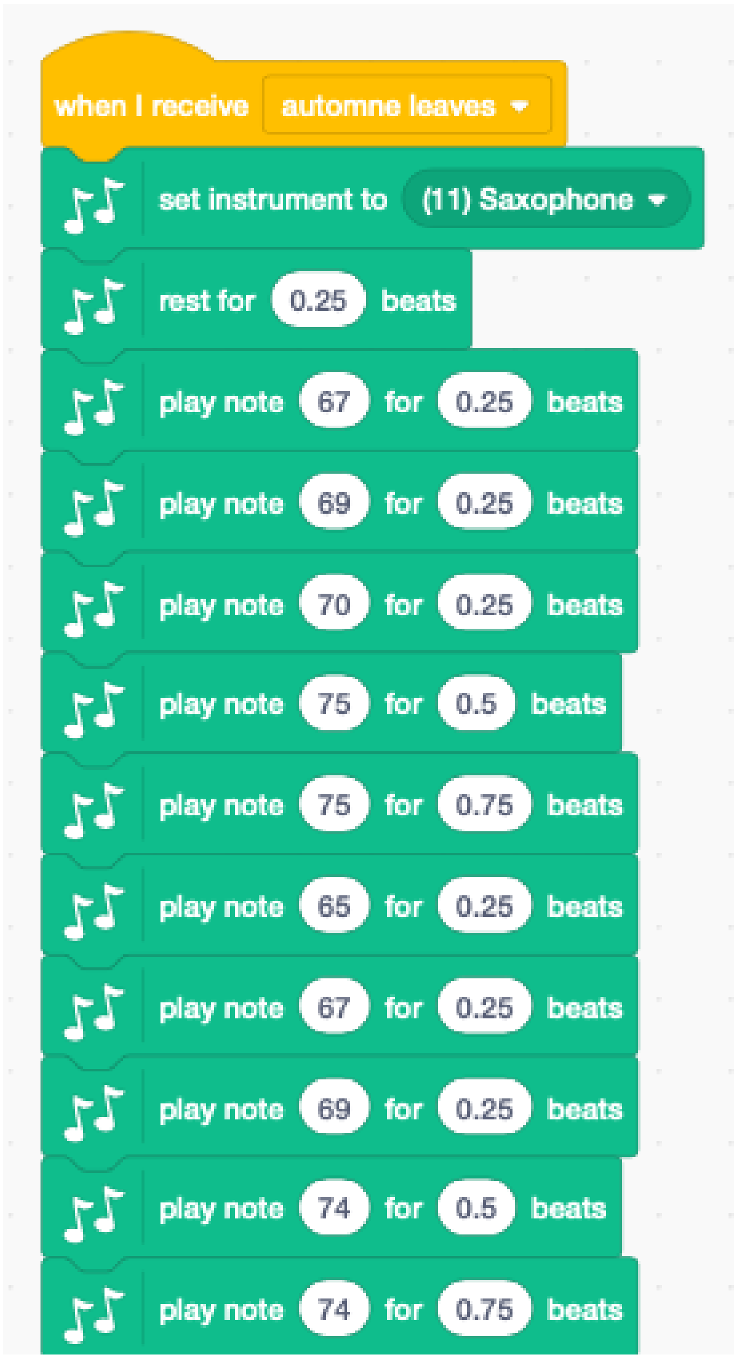
```
when I receive [automne leaves ▾]
repeat (16)
    rest for (0.25) beats
    play drum ((2) Bass Drum ▾) for (0.25) beats
    play drum ((6) Closed Hi-Hat ▾) for (0.25) beats
    play drum ((6) Closed Hi-Hat ▾) for (0.25) beats
end
```

This creates a base rythm for our music.

8.6 Play a Jazz classic

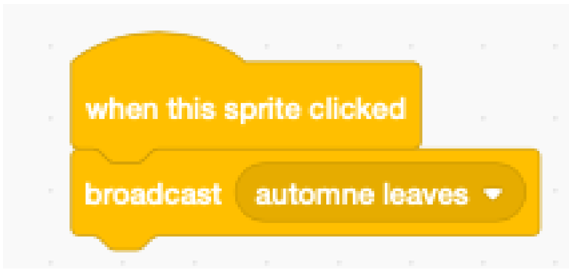
Together with the percussion, we can play a saxophone melody. The melody has a length of 16 measures and is the beginning of **Autumn leaves**, a jazz classic.

We use the **automn leaves** message to start the melody and the percussion at the same time.



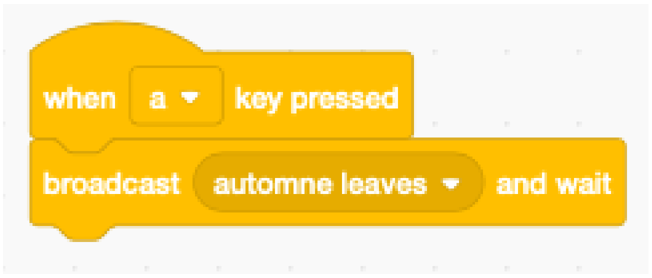
```
when I receive [automne leaves ▾]
set instrument to [(11) Saxophone ▾]
rest for (0.25) beats
play note (67) for (0.25) beats
play note (69) for (0.25) beats
play note (70) for (0.25) beats
play note (75) for (0.5) beats
play note (75) for (0.75) beats
play note (65) for (0.25) beats
play note (67) for (0.25) beats
play note (69) for (0.25) beats
play note (74) for (0.5) beats
play note (74) for (0.75) beats
```

We can start the music when clicking the saxophone sprite.



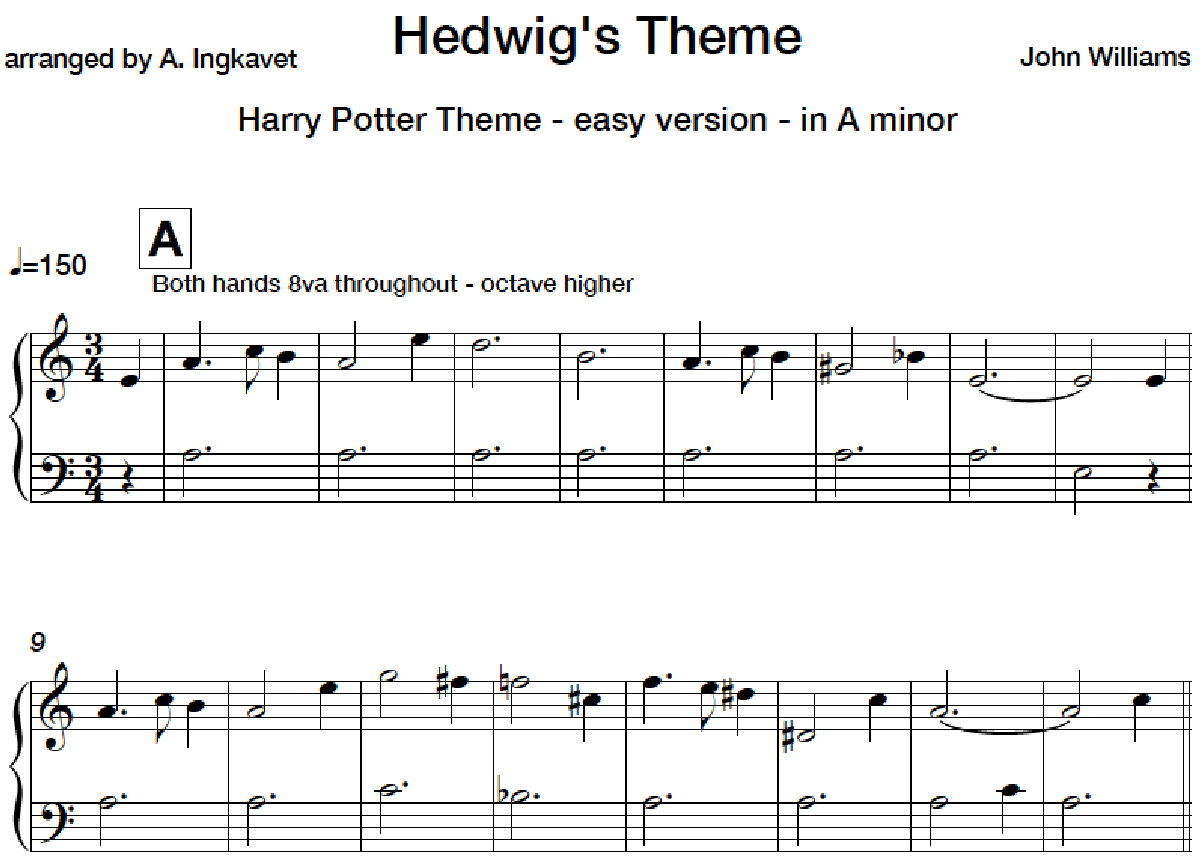
```
when this sprite clicked
broadcast [automne leaves ▾]
```

We can also start when pressing the A key.

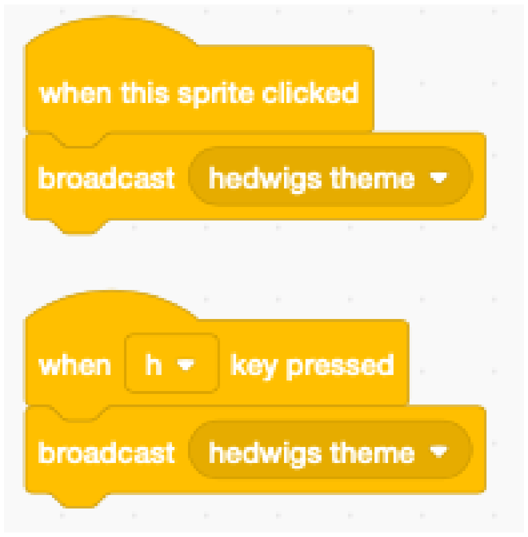


8.7 Use a list for the score

A more flexible way to enter the music is to use a list. Let's see how we can encode the following music score of **Hedwig's Theme** in a list.


Hedwig's Theme
arranged by A. Ingkavet
John Williams
Harry Potter Theme - easy version - in A minor
♩=150
A
Both hands 8va throughout - octave higher
9

First we send the message **hedwigs theme** when the guitar sprite is clicked or the H key is pressed.



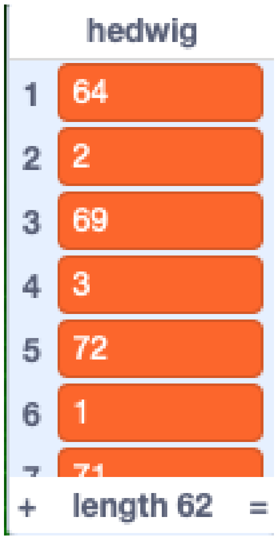
```
when this sprite clicked
broadcast [hedwigs theme ▾]

when [h ▾] key pressed
broadcast [hedwigs theme ▾]
```

The notes and duration of the score is kept in a list **hedwig**.

— odd indexes contain a note

— even indexes contain a duration

The total lenght of the list is 62.



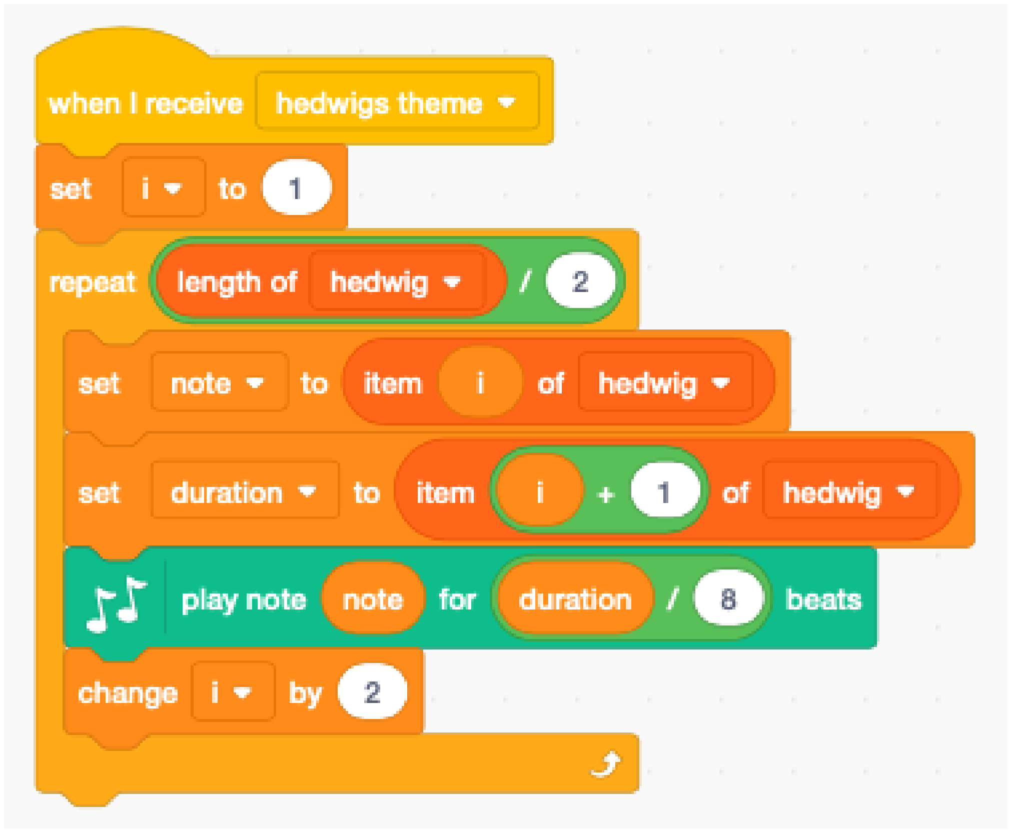
hedwig	
1	64
2	2
3	69
4	3
5	72
6	1
7	71
+	length 62

We can play the list and access each successive note and duration with an index pointer **i**.

— the i-th elemetn of the list is a note

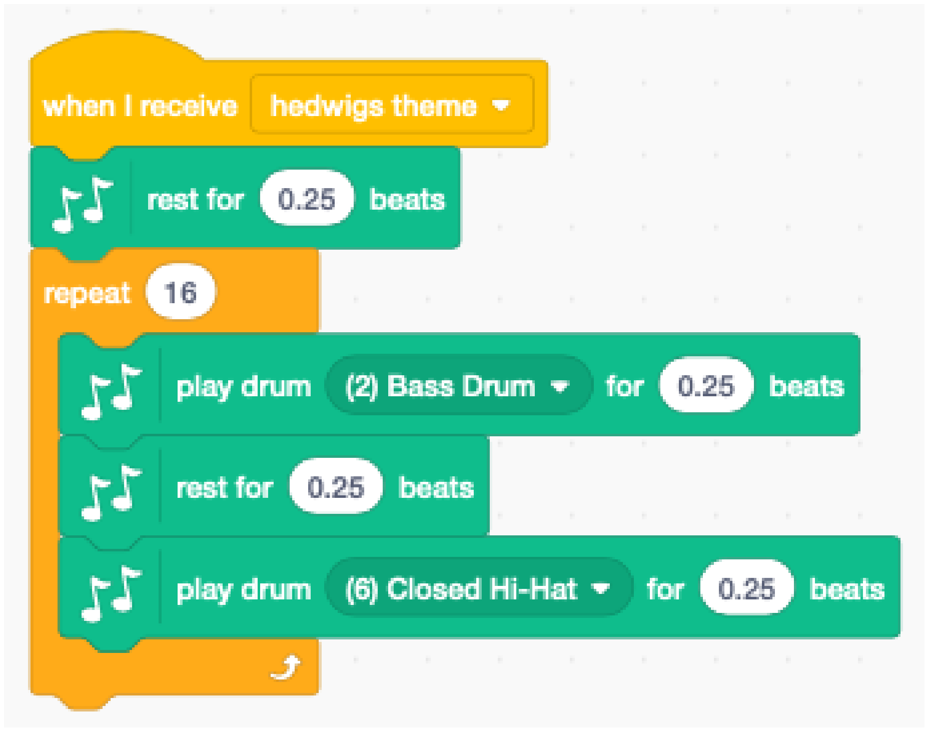
— the i+1-th element is the duration

To avoid decimal fractions we multiply the duration by 8. That means a duration of 0.125 beats is encoded as 1 A duration of 0.25 as 2 and 0.5 as 4.



```
when I receive [hedwigs theme]
set [i] to (1)
repeat ((length of [hedwig]) / (2))
    set [note] to (item (i) of [hedwig])
    set [duration] to (item ((i) + (1)) of [hedwig])
    play note (note) for ((duration) / (8)) beats
    change [i] by (2)
```

With this music as well, we add a little percussion loop.

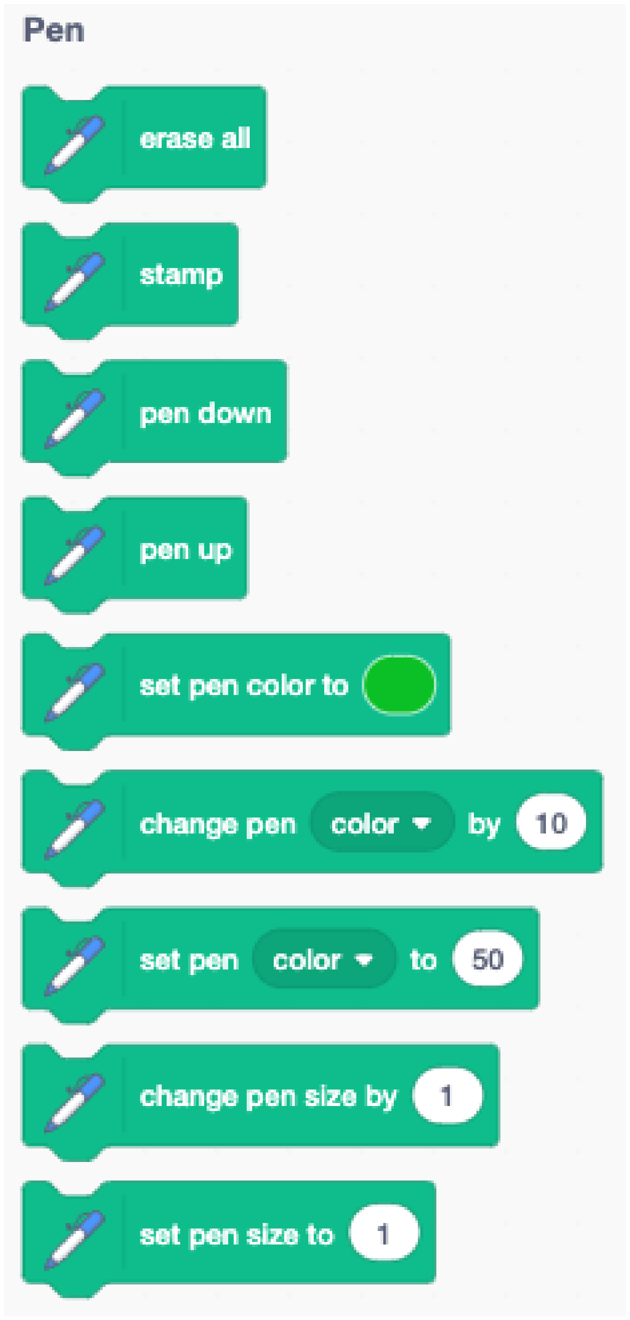


```
when I receive [hedwigs theme]
rest for (0.25) beats
repeat (16)
    play drum [(2) Bass Drum] for (0.25) beats
    rest for (0.25) beats
    play drum [(6) Closed Hi-Hat] for (0.25) beats
```

CHAPITRE 9

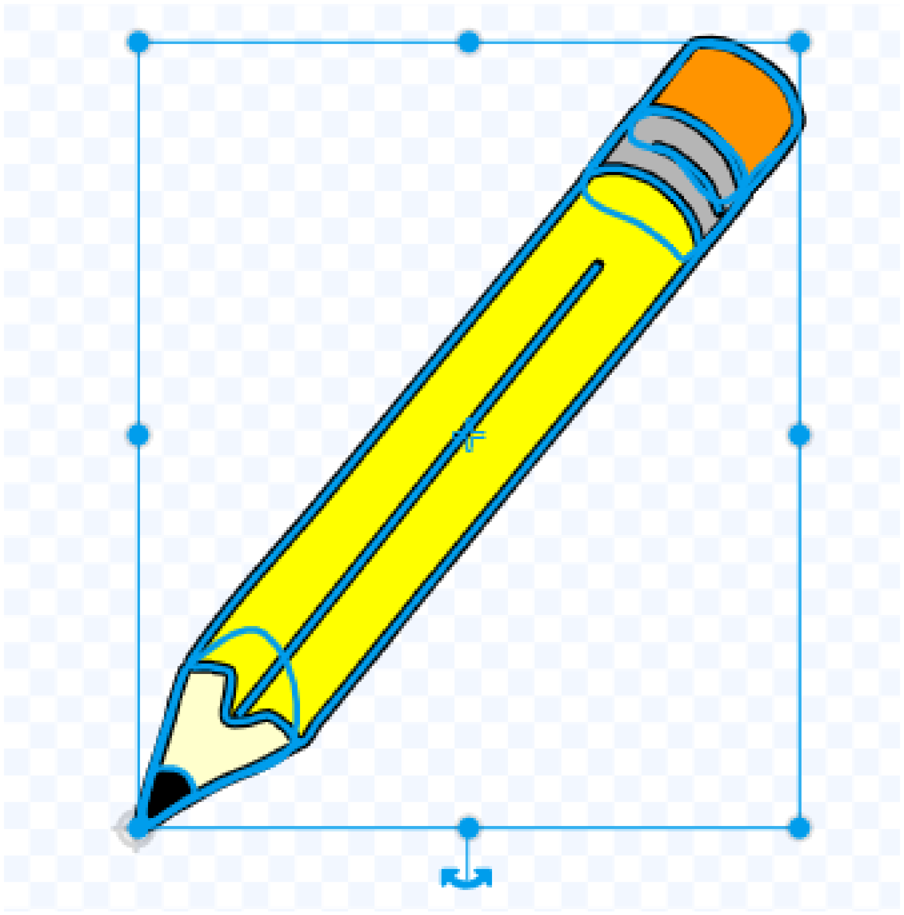
Pen

The pen extension is a drawing module. It adds 9 dark green blocks.



9.1 The pencil sprite

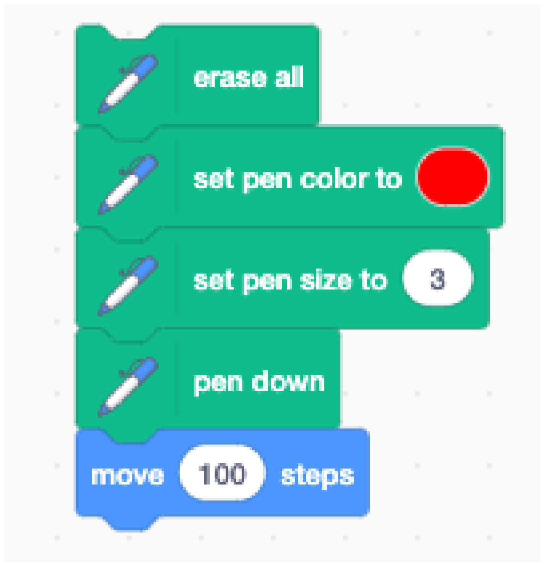
We are going to use the **Pencil** sprite. In order to make the line appear at the point of the pencil, you must go to the sprite editor, select the whole drawing (cmd+A) and place the tip of the pencil at the origin marker.



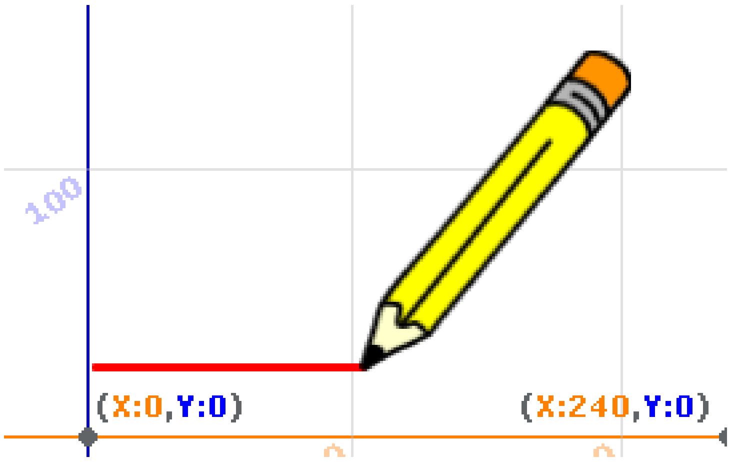
9.2 Draw a line

Now we can place the pencil somewhere on the stage and draw a line. To do so we :

- — erase all
- — set the pen color (red)
- — set the pen size (3 points)
- — put the pen down
- — move 100 steps



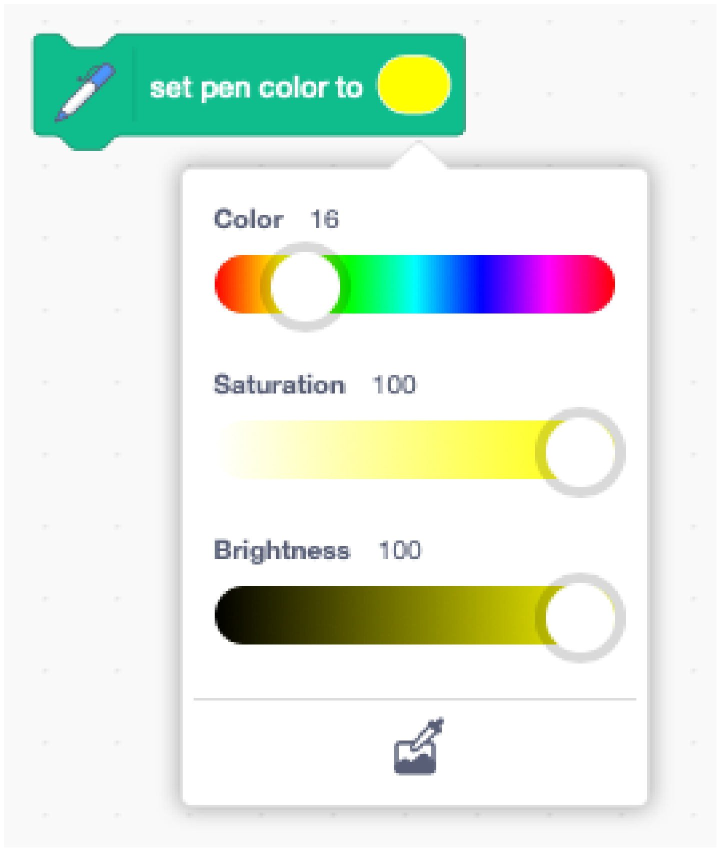
This is the result, drawn on the backdrop with xy axes.



9.3 RGB colors

The **set pen color to** () block has a drop-down menu to select the color via :

- color
- saturation
- brightness



It is also possible to provide the color with an integer variable. Most computers represent color with the 3 components :

- red
- green
- blue

This is also called the **RGB color** system. The three components are expressed with a byte-sized value. The intensity of each color component can go from 0 to 255.

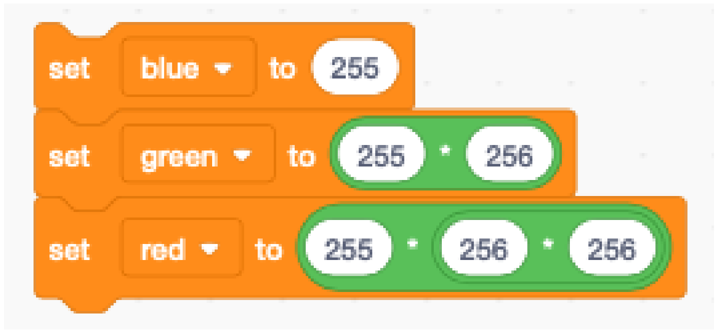
Each color component occupies one byte and the 3 components can be combined into a 3-byte integer. When expressed as hexadecimal digit, the RGB color number has this format :

```
0xRRGGBB
```

For example :

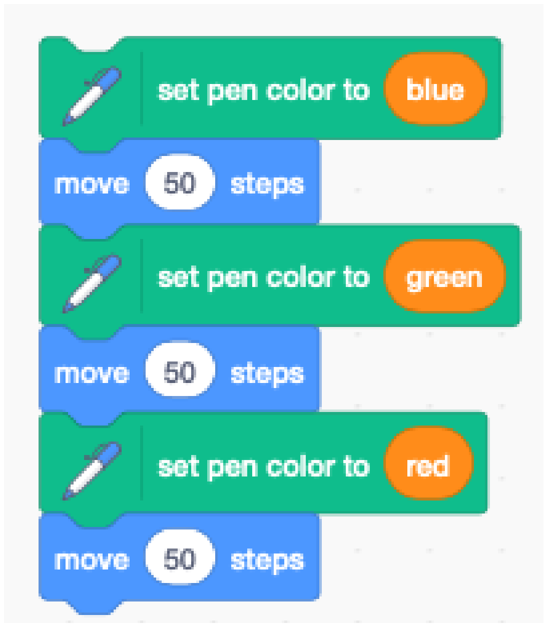
- `0xFF0000` is red
- `0x00FF00` is green
- `0x0000FF` is blue

We use 3 variables and express each base color at maximum intensity.



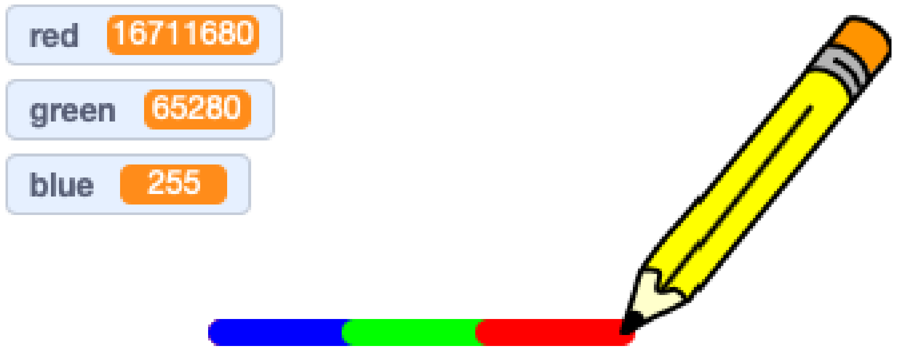
```
set blue to 255
set green to 255 * 256
set red to 255 * 256 * 256
```

Now we can used these variables to draw three line segments in these base colors. - blue is 255 - green is 255 shifted by 8 bits (multiplied by 256) - red is 255 shifted by 16 bits (multiplied by 256*256)



```
set pen color to (blue)
move (50) steps
set pen color to (green)
move (50) steps
set pen color to (red)
move (50) steps
```

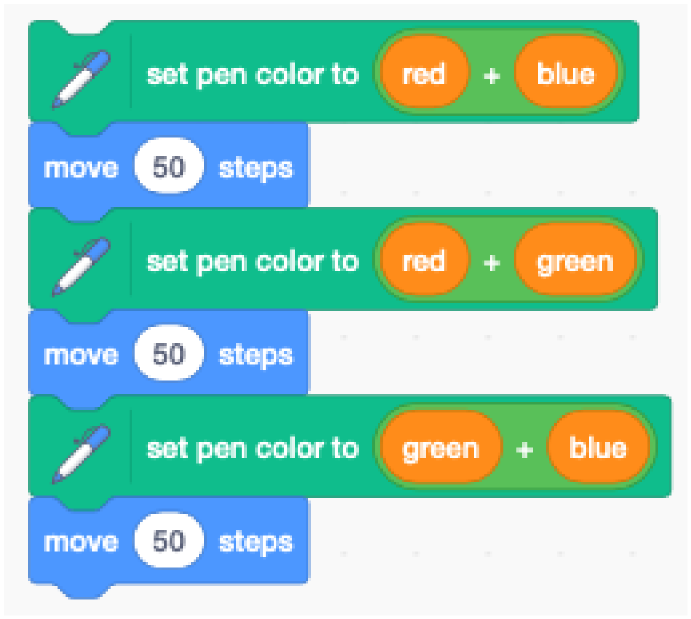
This is the result :



```
red 16711680
green 65280
blue 255
```

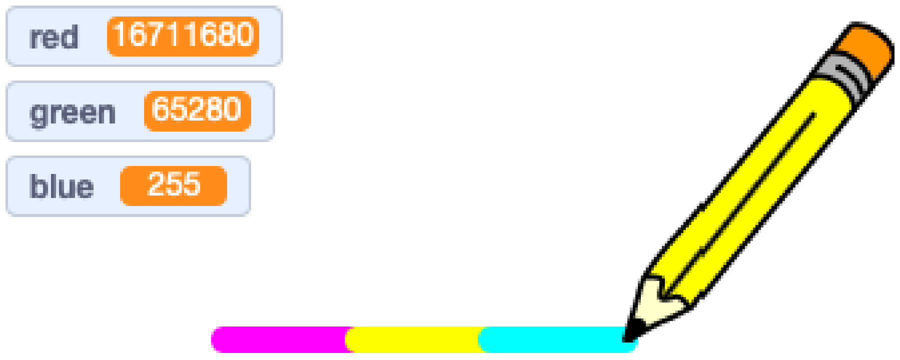
We can also add two base colors to get a new color.

- red and green = yellow
- red and blue = magenta
- blue and gren = cyan



```
set pen color to (red + blue)
move (50) steps
set pen color to (red + green)
move (50) steps
set pen color to (green + blue)
move (50) steps
```

This is the result :



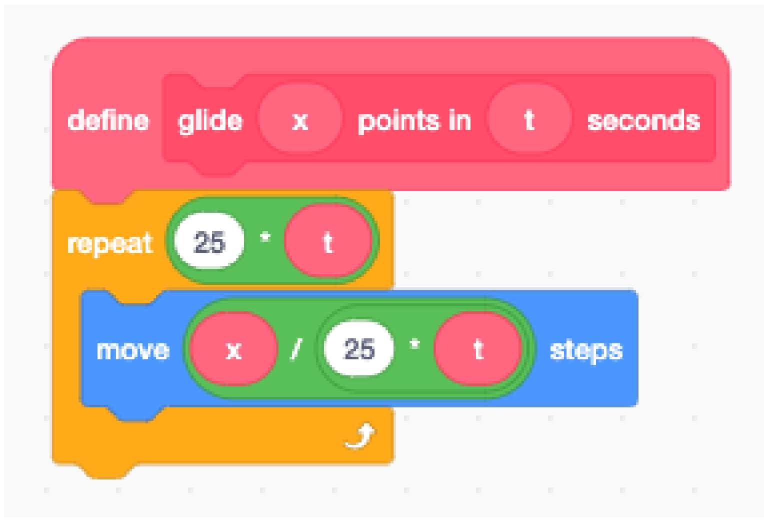
red	16711680
green	65280
blue	255

9.4 Move slowly

The **Motion** category has a **glide to x, y** block. We can create a similar block for gliding a certain distance.

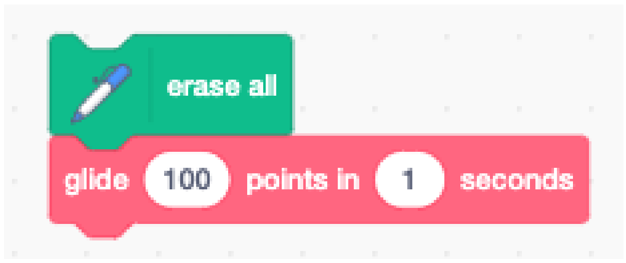
In Scratch a loop executes 25 times each second. To last **t** seconds, it must repeat `t*25` times.

The distance increment of each iteration is `1/(t*25)` of the total distance.



```
define glide (x) points in (t) seconds
repeat ((25) * (t))
    move ((x) / ((25) * (t))) steps
end
```

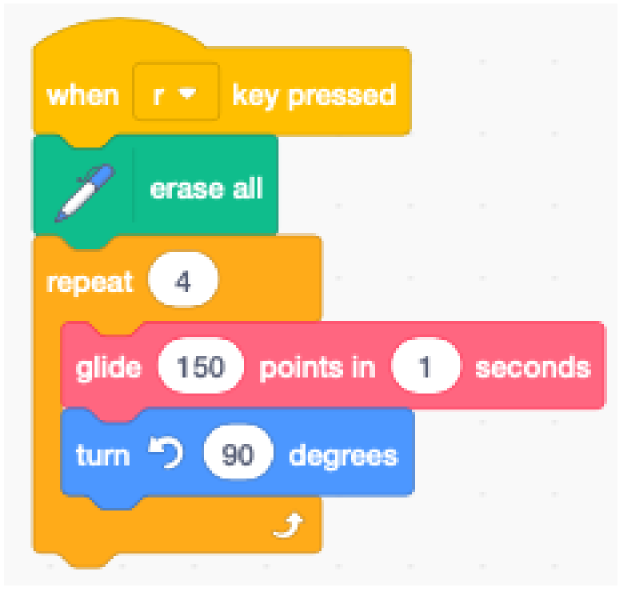
To test the gliding move we write :



```
erase all
glide (100) points in (1) seconds
```

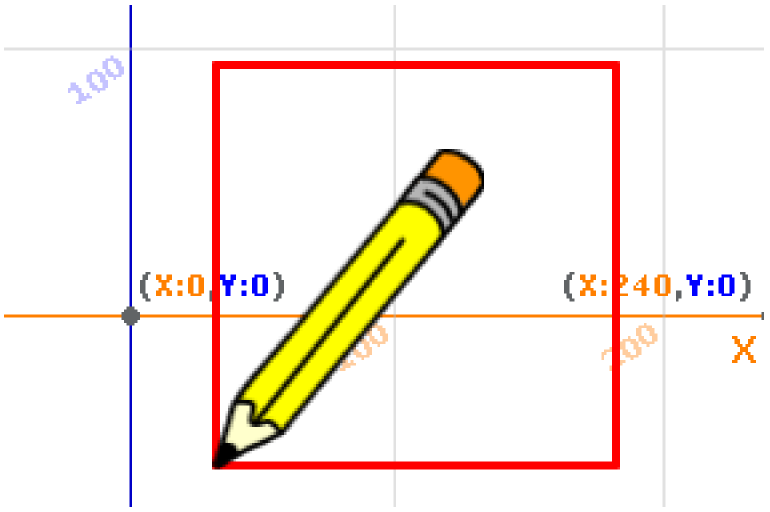
9.5 Draw a square

Now we combine the slow move and the slow turn block to draw a square.

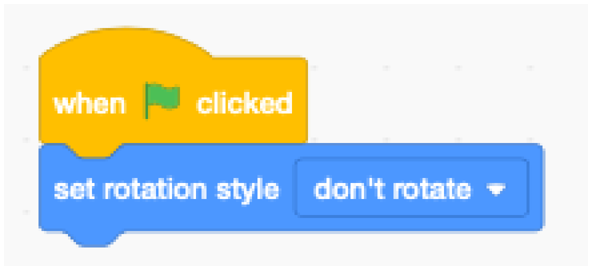


```
when [r] key pressed
erase all
repeat (4)
    glide (150) points in (1) seconds
    turn ↺ (90) degrees
end
```

This is the result.



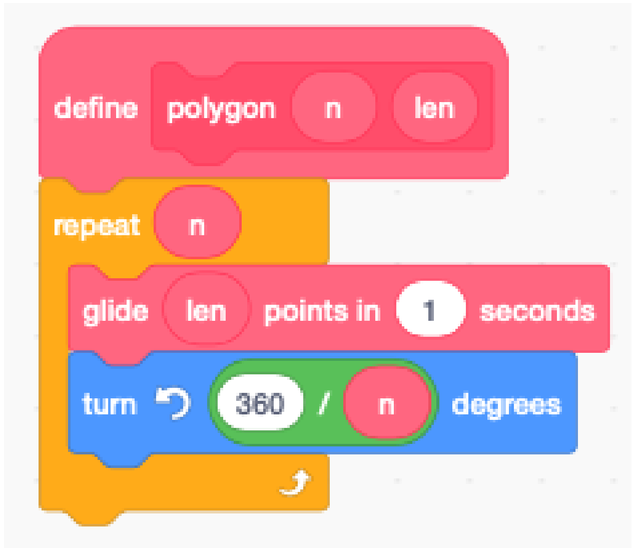
If we do not set the rotation style, the pen will turn by 90 degrees at each corner. That does not give a natural animation style. It's better to not rotate the pen. Therefore at the start we set the rotation style to **don't rotate**.



```
when flag clicked
set rotation style [don't rotate]
```

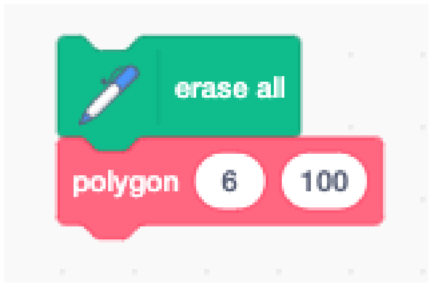
9.6 Draw a polygon

We can turn this into a function **polygon** which can draw any arbitrary regular polygon.



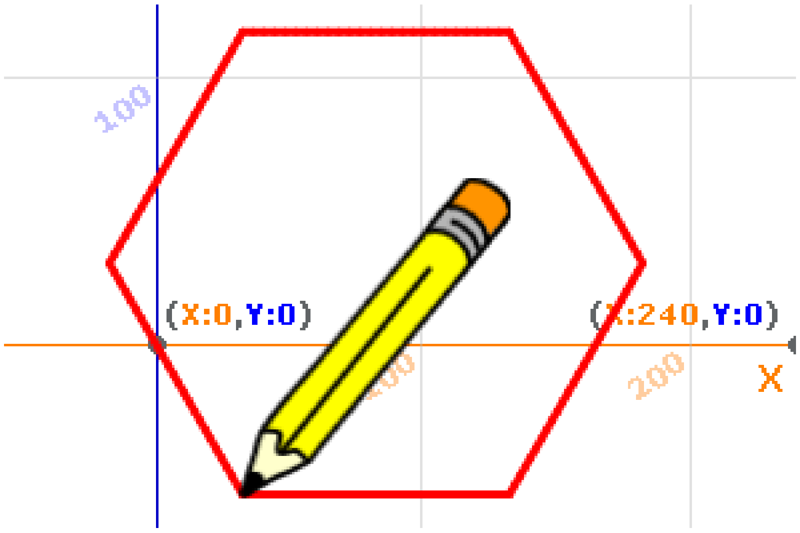
```
define polygon (n) (len)
repeat (n)
    glide (len) points in (1) seconds
    turn ↺ ((360) / (n)) degrees
```

This is the code to test for a hexagon.



```
erase all
polygon 6 100
```

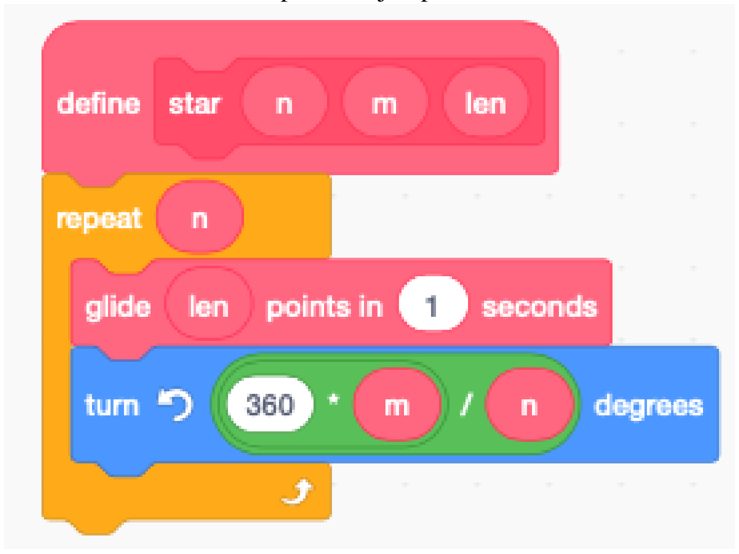
This is the result drawn to the stage.



9.7 Draw a star

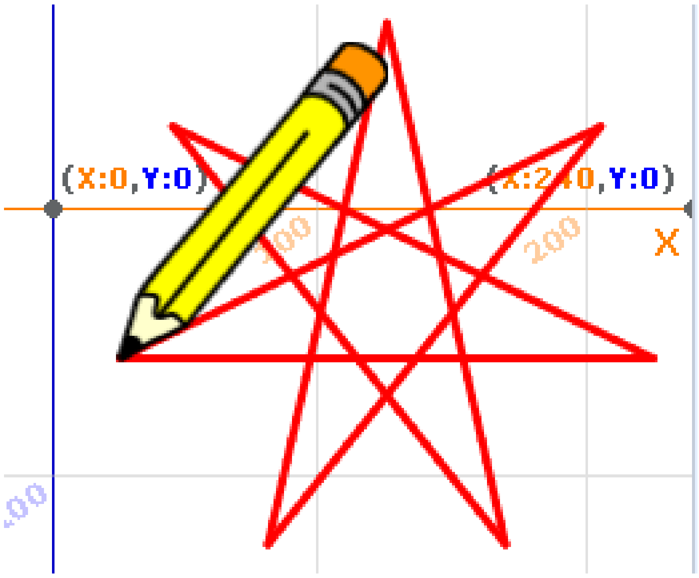
There is a simple algorithm to draw a star. It's almost like a polygon, but it has an extra parameter **m**.

— n : the number of points
— m : the number of points to jump over



```
define star n m len
repeat n
    glide len points in 1 seconds
    turn ↺ 360 * m / n degrees
```

This is the result for a **(7, 3)** star which is a 7-pointed star, jumping always to the 3rd point.



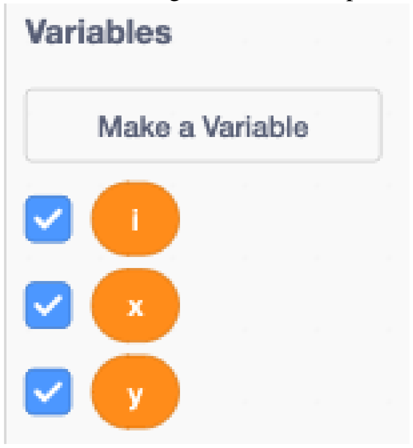
Here is the code :

https://scratch.mit.edu/projects/397107138

9.8 Store points in a list

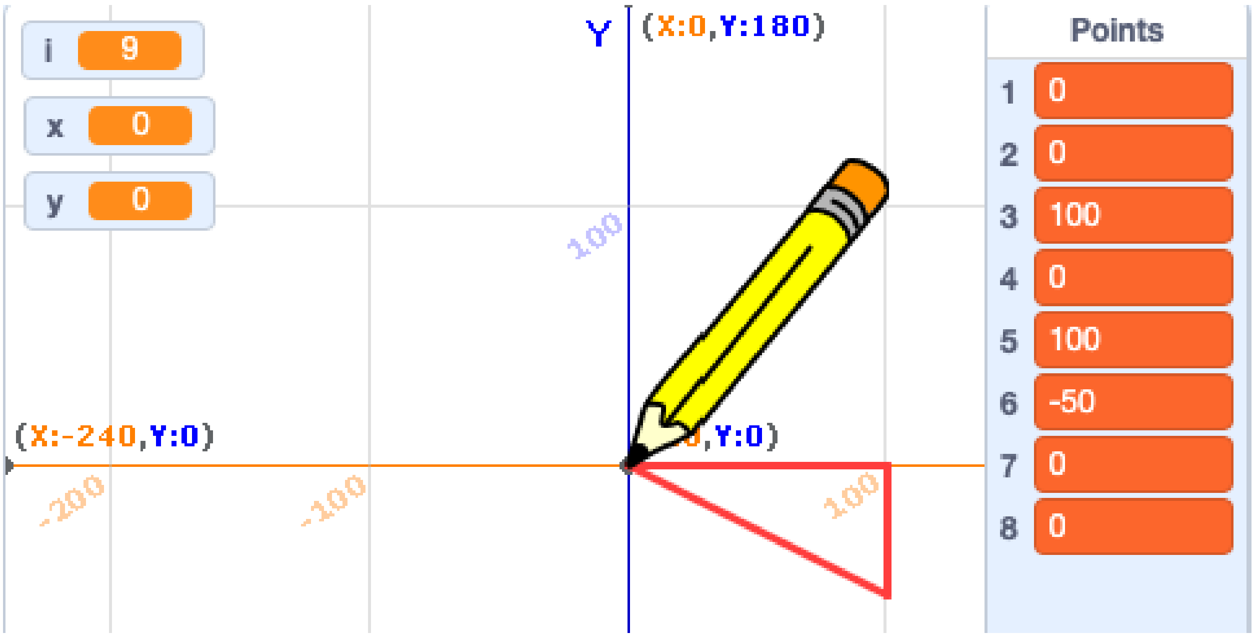
In the following section we look at a technique to store points in a list. For this we need to define 3 variables :

- **x, y** the 2D coordinates
- **i** an integer index which points to the current point in the list

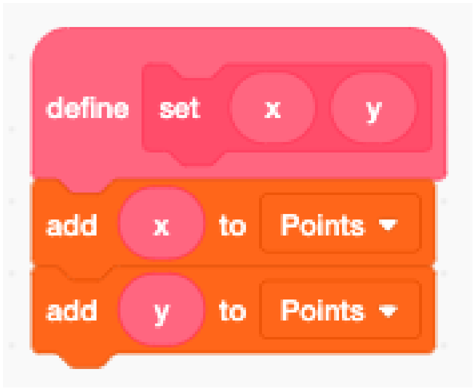


Then we create a list called **Points**. This list contains the coordinates of all the points of the shape.

The image below shows the **Points** list with the points (0, 0), (100, 0) and (100, -50). They represent a rectangular triangle. The first point is repeated at the end to close the shape.

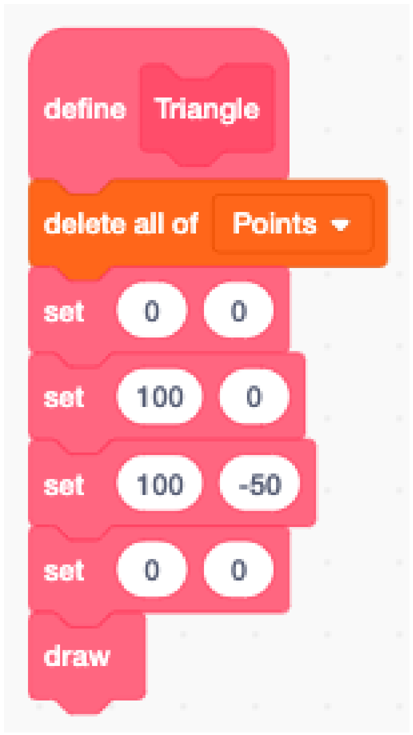


We define a fonction **set x, y** to add the point (x, y) to the **Points** list.



```
define set (x) (y)
add (x) to [Points ▾]
add (y) to [Points ▾]
```

The function **Triangle** sets the 4 points of the triangle shown above.

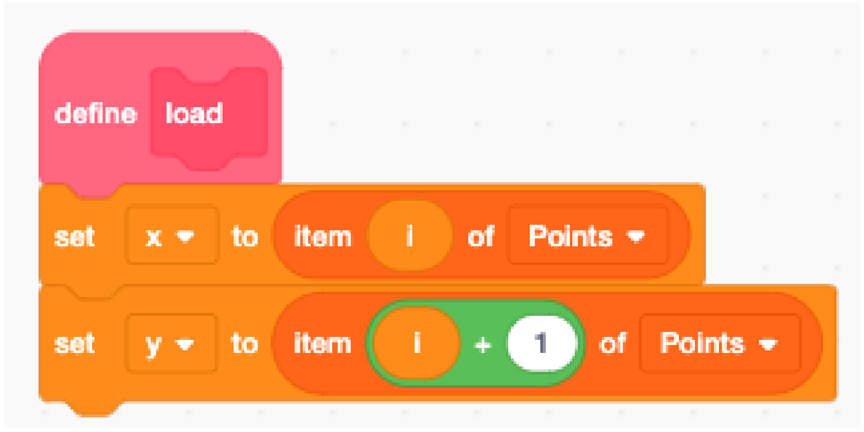


```
define Triangle
delete all of Points
set 0 0
set 100 0
set 100 -50
set 0 0
draw
```

9.9 Load and store

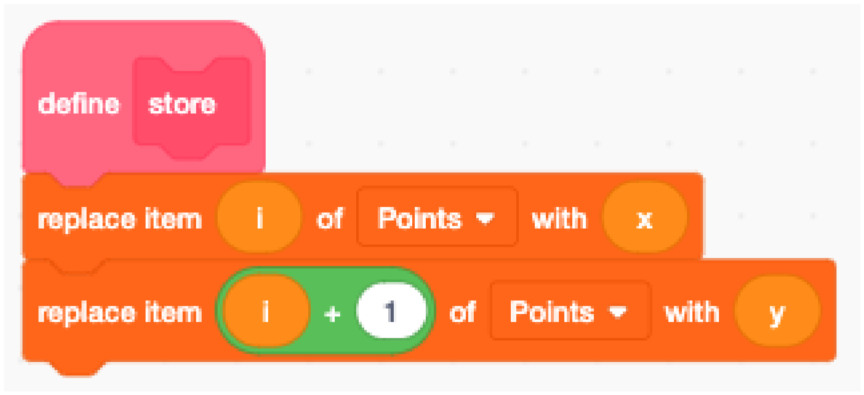
Operations are executed using two variables **x** and **y**. At any time, the index variable **i** points at a pair of coordinates in the list.

The **load** function loads the coordinates (x, y) from the **Points** list to the variables.



```
define load
set x to item i of Points
set y to item i + 1 of Points
```

The **store** function transfers the coordinates (x, y) from the variables back to the **Points** list.

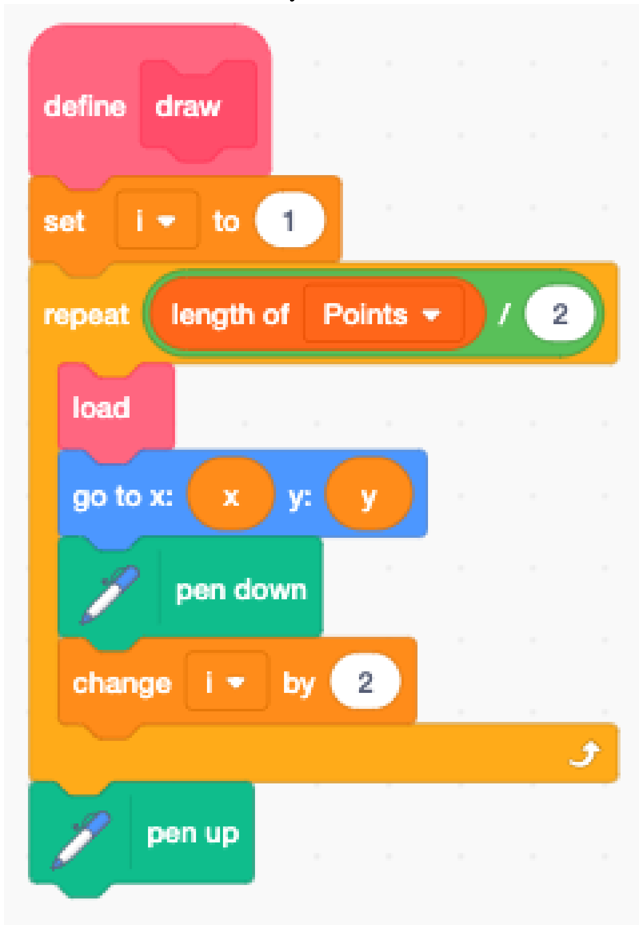


```
define store
replace item (i) of [Points ▾] with (x)
replace item ((i) + (1)) of [Points ▾] with (y)
```

9.10 Draw a shape

The **draw** function does the following

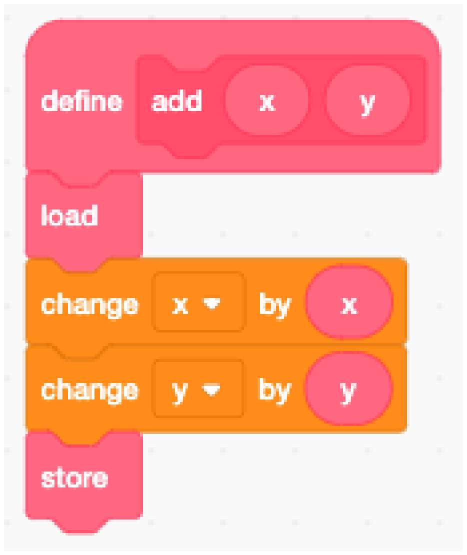
- set the index **i** to the first point
- repeat for each point
- load the current point to the variables **x** and **y**
- go to position (x, y)
- put the pen down
- increment index **i** by 2



```
define draw
set [i ▾] to (1)
repeat ((length of [Points ▾]) / (2))
    load
    go to x: (x) y: (y)
    pen down
    change [i ▾] by (2)
end
pen up
```

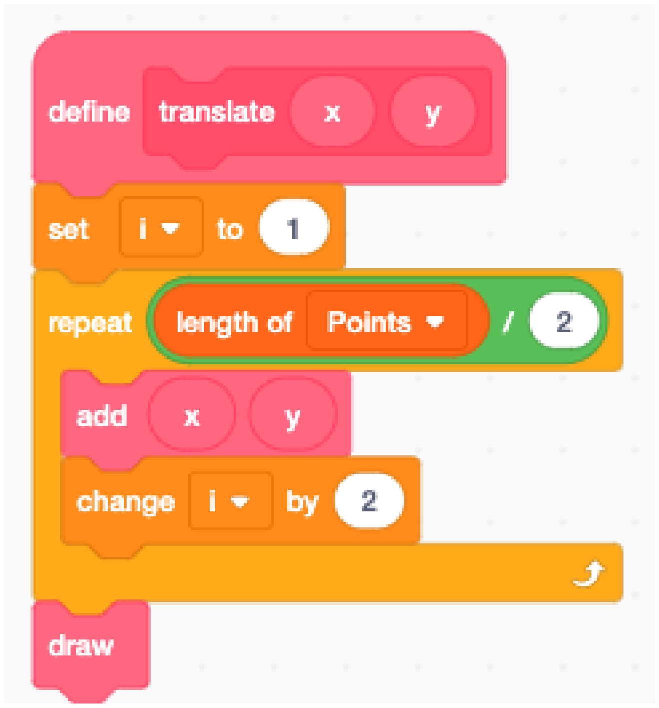
9.11 Translate a shape

The function **add** adds the vector (x, y) to the point currently pointed to by the index **i**.



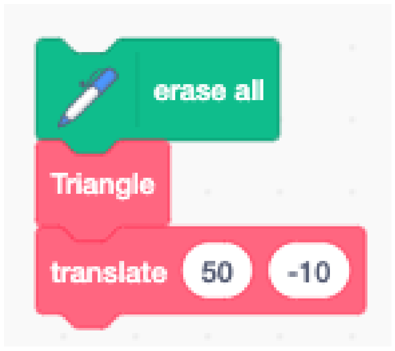
```
define add (x) (y)
load
change [x ▾] by (x)
change [y ▾] by (y)
store
```

The function **translate** does this for every point in the **Points** list. It also draws the shape.

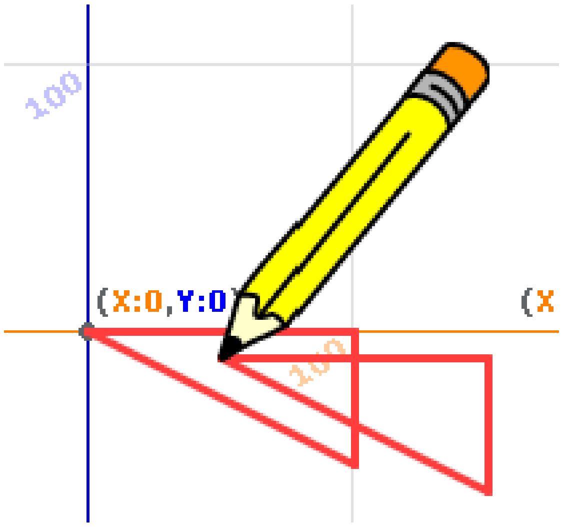


```
define translate (x) (y)
set [i ▾] to (1)
repeat ((length of [Points ▾]) / (2))
    add (x) (y)
    change [i ▾] by (2)
end
draw
```

Now we can apply the translation to the triangle.

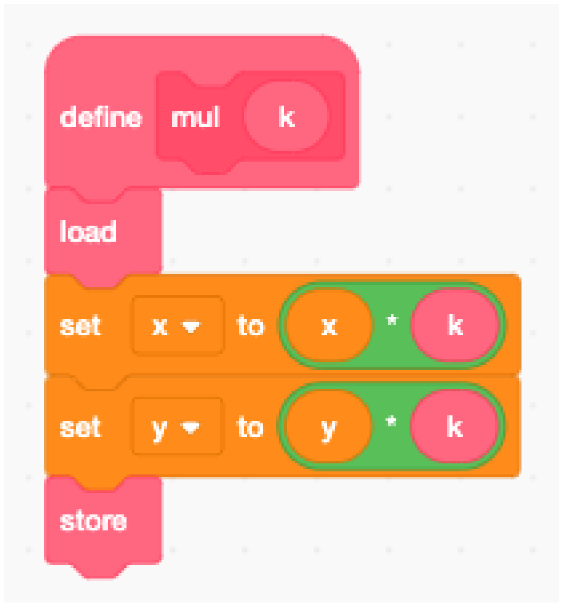


This is the result.



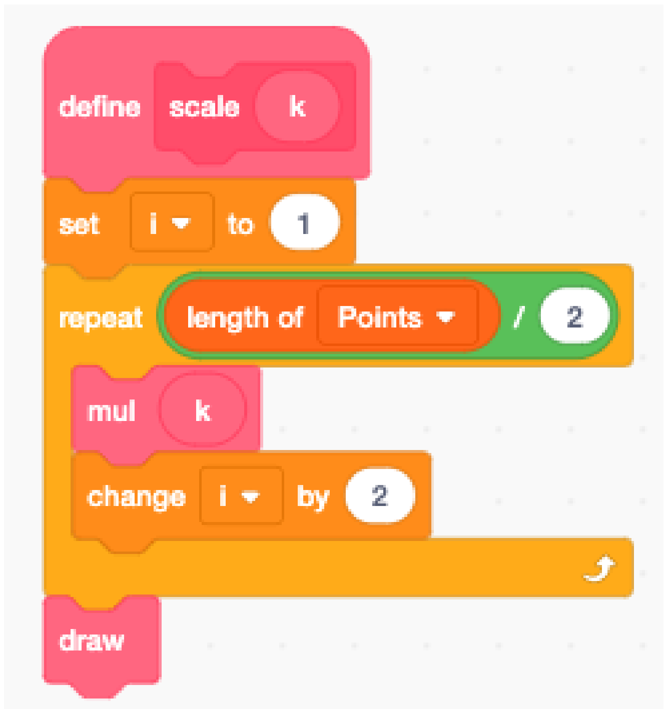
9.12 Scale a shape

The function **mul** multiplies the point currently pointed to by the index **i** with a scalar value **k**. A positive angle turns counter-clockwise.



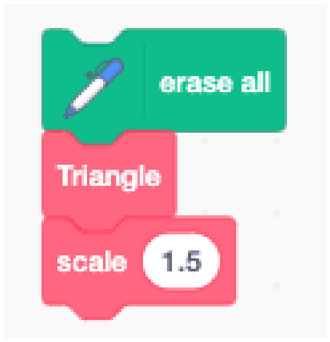
```
define mul (k)
load
set x to ((x) * (k))
set y to ((y) * (k))
store
```

The function **scale** does this for every point in the **Points** list. It also draws the shape.



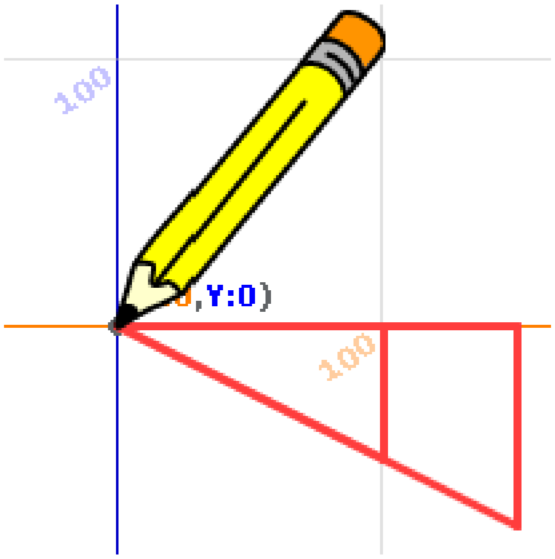
```
define scale (k)
set i to 1
repeat ((length of Points) / 2)
    mul (k)
    change i by 2
draw
```

Now we can apply the scaling to the triangle.



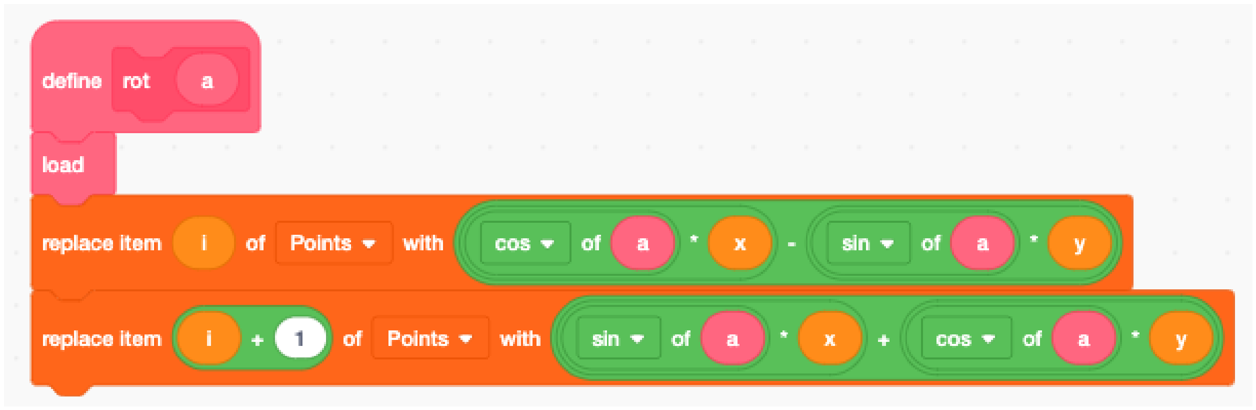
```
erase all
Triangle
scale (1.5)
```

This is the result.



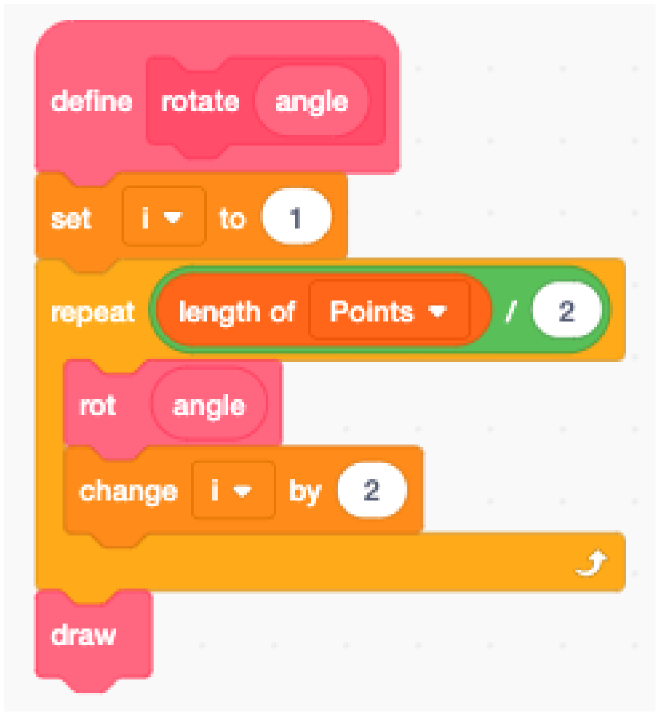
9.13 Rotate a shape

The function **rot** rotates the point currently pointed to by the index **i** by an angle **a**, with reference to the origin.



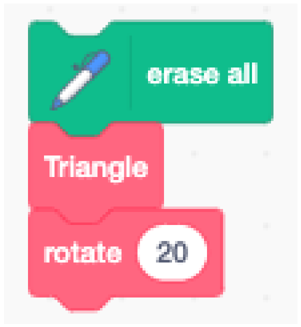
```
define rot (a)
load
replace item (i) of [Points] with (((cos of (a)) * (x)) - ((sin of (a)) * (y)))
replace item ((i) + (1)) of [Points] with (((sin of (a)) * (x)) + ((cos of (a)) * (y)))
```

The function **rotate** does this for every point in the **Points** list. It also draws the shape.



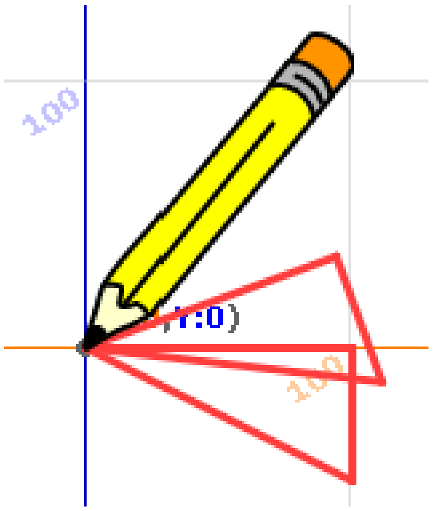
```
define rotate (angle)
set i to 1
repeat (length of Points / 2)
    rot (angle)
    change i by 2
draw
```

Now we can apply the rotation to the triangle.



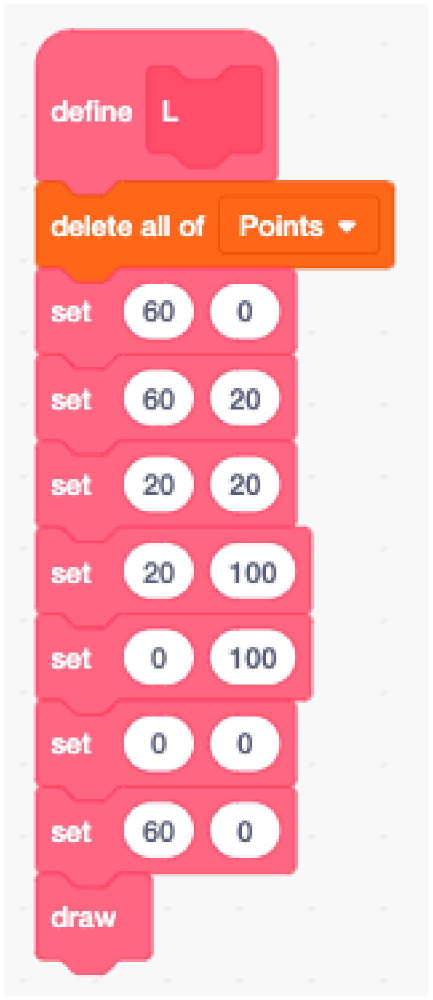
```
erase all
Triangle
rotate 20
```

This is the result.



9.14 Translate, scale and rotate

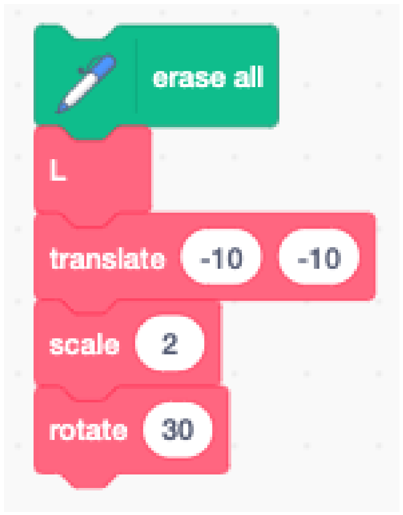
Now we define a slighly more complex L-shape.



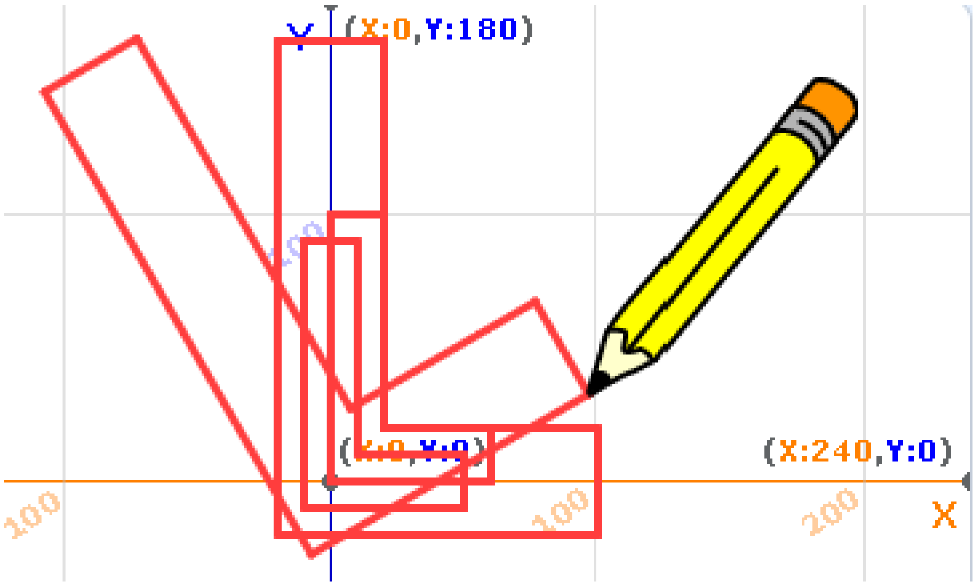
```
define L
delete all of Points
set 60 0
set 60 20
set 20 20
set 20 100
set 0 100
set 0 0
set 60 0
draw
```

We apply successively a

- translation
- scaling
- rotation


```
erase all
L
translate -10 -10
scale 2
rotate 30
```

This is the result :

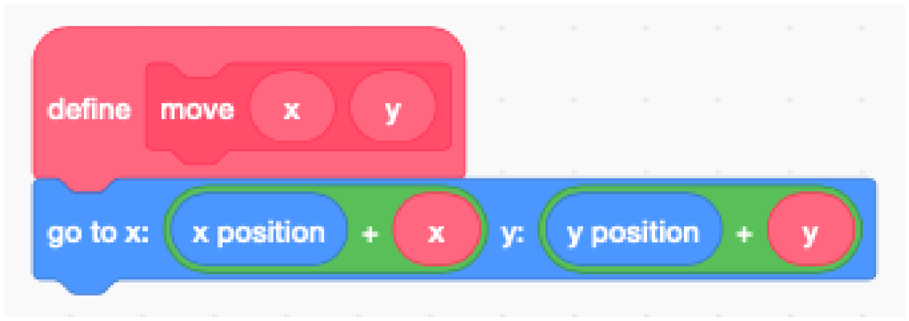


This is the project

https://scratch.mit.edu/projects/398912533

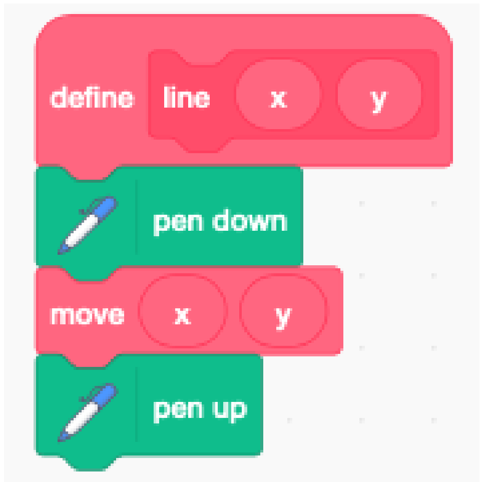
9.15 Move by (x, y)

There are two **change x/y by** blocks but no block which changes both at the same time. Let's create such a function.



```
define move (x) (y)
go to x: ((x position) + (x)) y: ((y position) + (y))
```

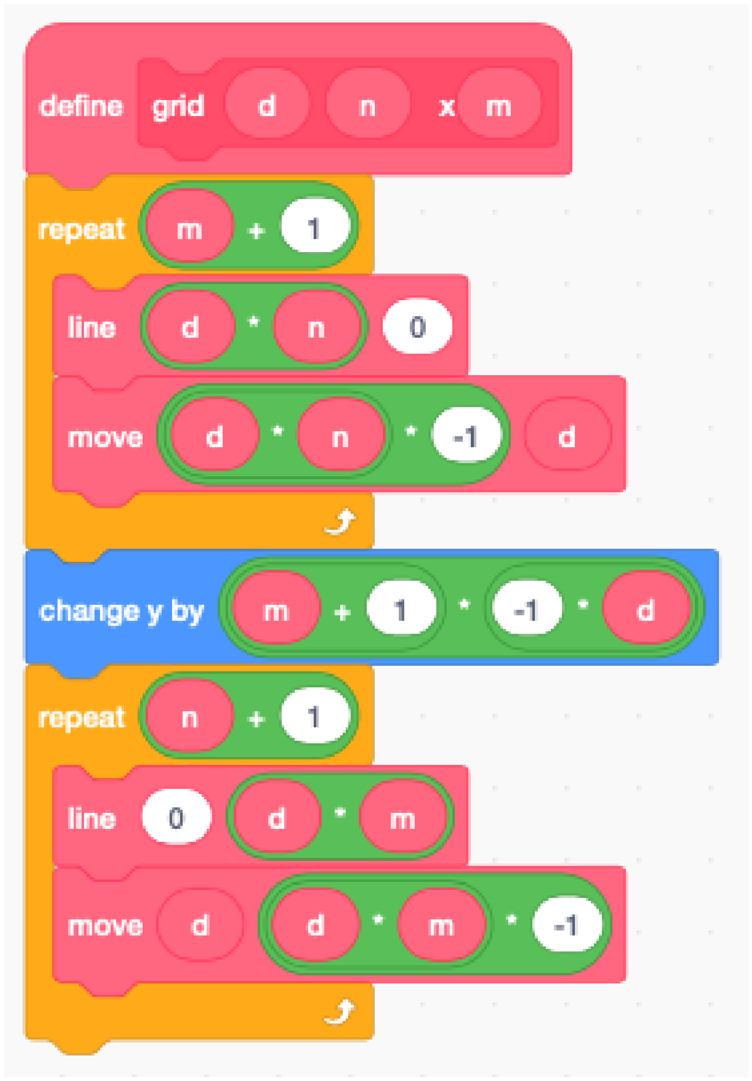
Now let's define one to draw a line by an amount (x, y).



```
define line (x) (y)
pen down
move (x) (y)
pen up
```

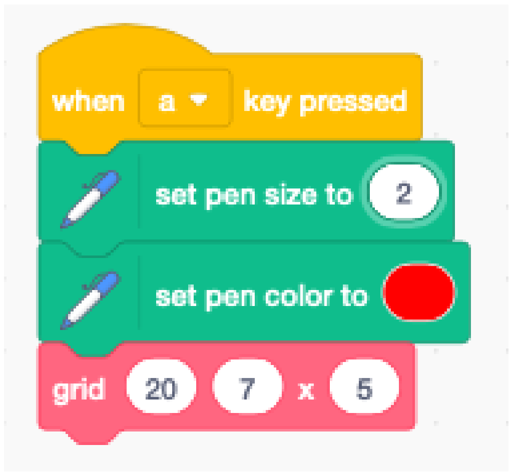
9.16 Draw a grid

Now we have everything to draw a grid.



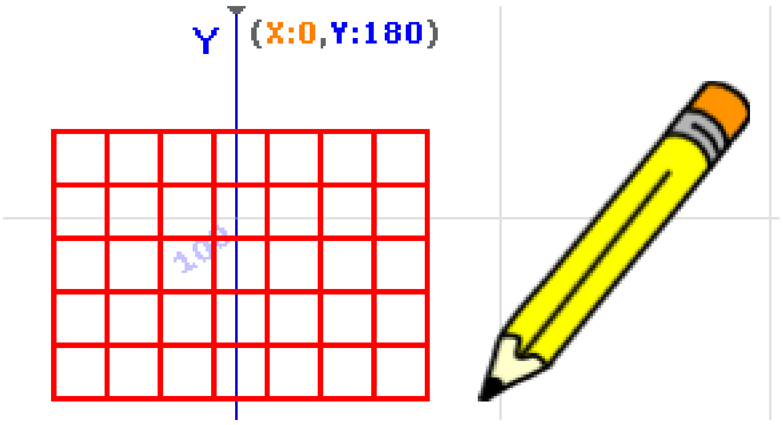
```
define grid (d) (n) x (m)
repeat ((m) + (1))
    line ((d) * (n)) (0)
    move (((d) * (n)) * (-1)) (d)
end
change y by ((((m) + (1)) * (-1)) * (d))
repeat ((n) + (1))
    line (0) ((d) * (m))
    move (d) (((d) * (m)) * (-1))
end
```

We place the pen to the starting position. Now draw a 7x5 grid with a line distance **d=20**.

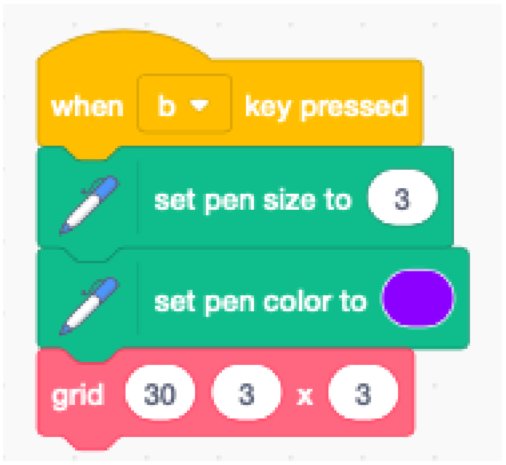


```
when [a ▾] key pressed
set pen size to (2)
set pen color to [🟥]
grid (20) (7) x (5)
```

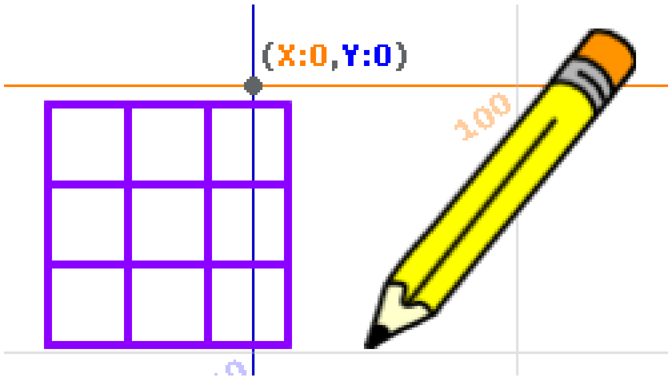
And this is the result.



We place the pen to another starting position, then change the color to purple and the thickness to 3. Now draw a 3xs grid with a line distance **d=30**.



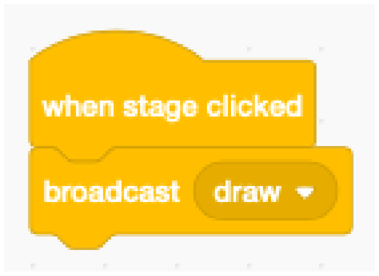
And this is the result.



https://scratch.mit.edu/projects/398983654

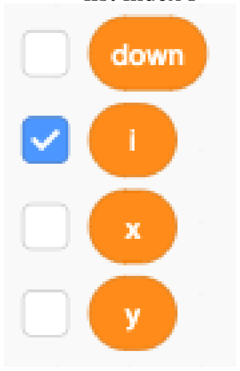
9.17 Record pen movement

In this example we use a list to memorize the pen movenent. Clicking anywhere on the stage, starts drawing the line with the pen sprite, and also recording of the x, y coordinates.

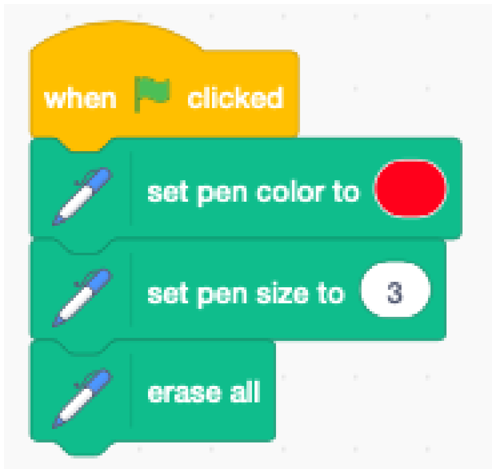


It uses the 4 variables :

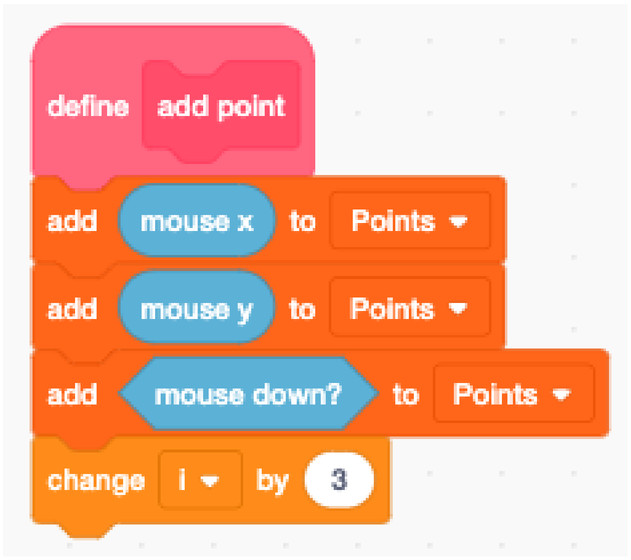
— mouse position **x, y**
— mouse **down** state (true or false)
— list index **i**



At the start, we set the color to red and the thickness to 3.

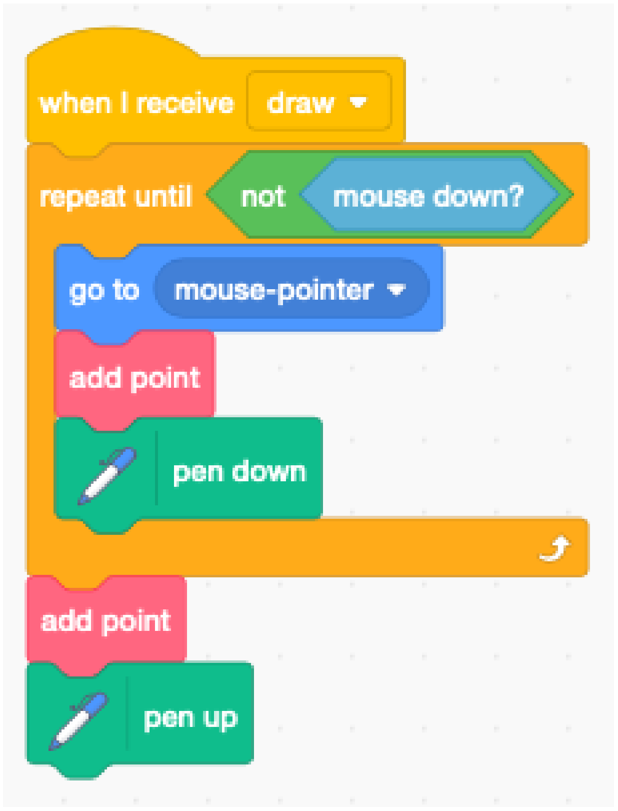


The function **add point** adds the current mouse position to the **Points** list.



```
define add point
add (mouse x) to [Points ▾]
add (mouse y) to [Points ▾]
add <mouse down?> to [Points ▾]
change [i ▾] by (3)
```

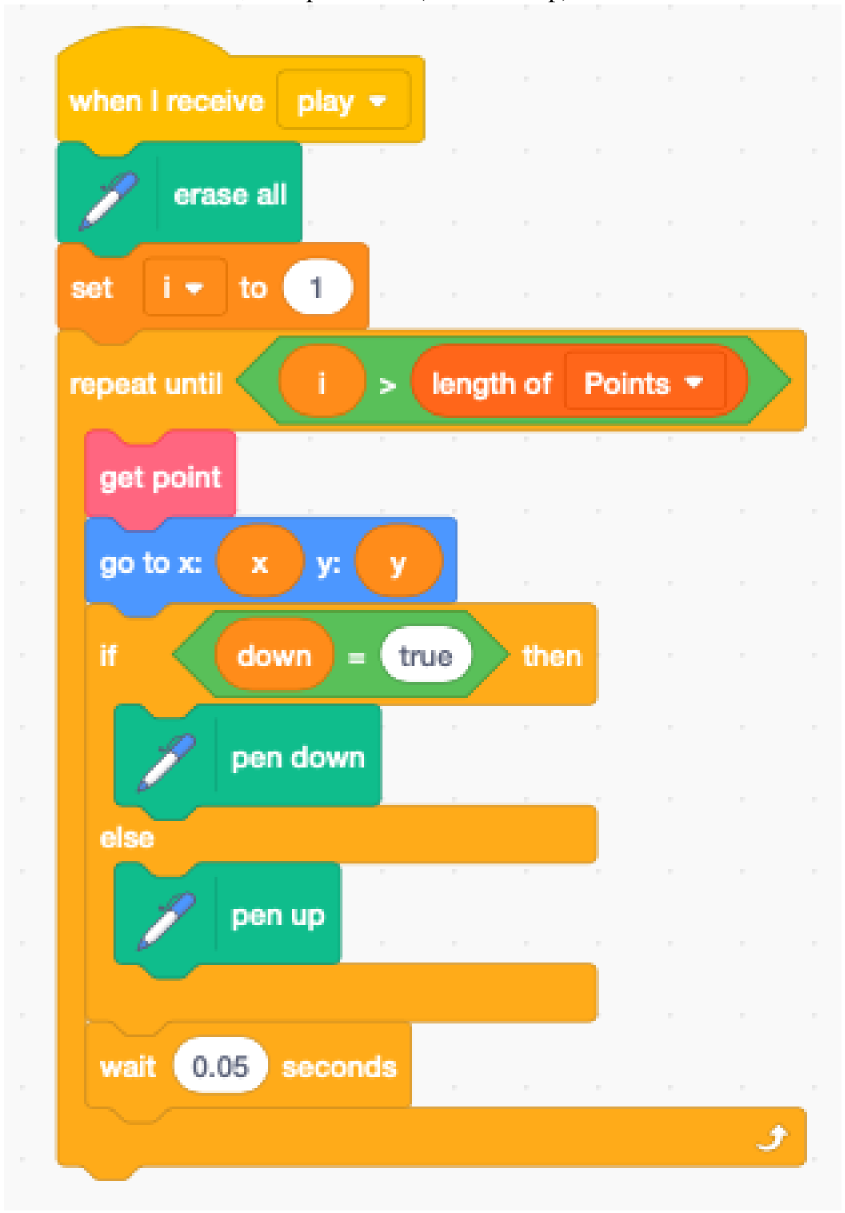
When the stage is clicked, the message **draw** is broadcast. This
— moves the pen to the mouse
— records the position
— puts the pen down
until the mouse is up.



```
when I receive [draw ▾]
repeat until <not <mouse down?>>
    go to (mouse-pointer ▾)
    add point
    pen down
end
add point
pen up
```

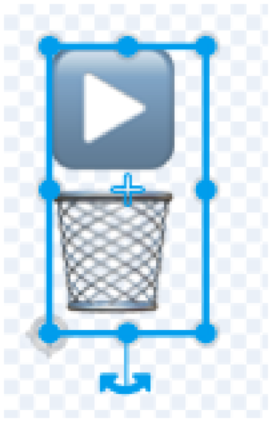
When the **Play** button is pressed the recorded points are redrawn.
— reset index **i** to 1
— repeat until the end of the list
— read the 3 variables **x, y, down** from the *Point* list

— to to **x, y**
— if **down** is true, then pen down (otherwise up)



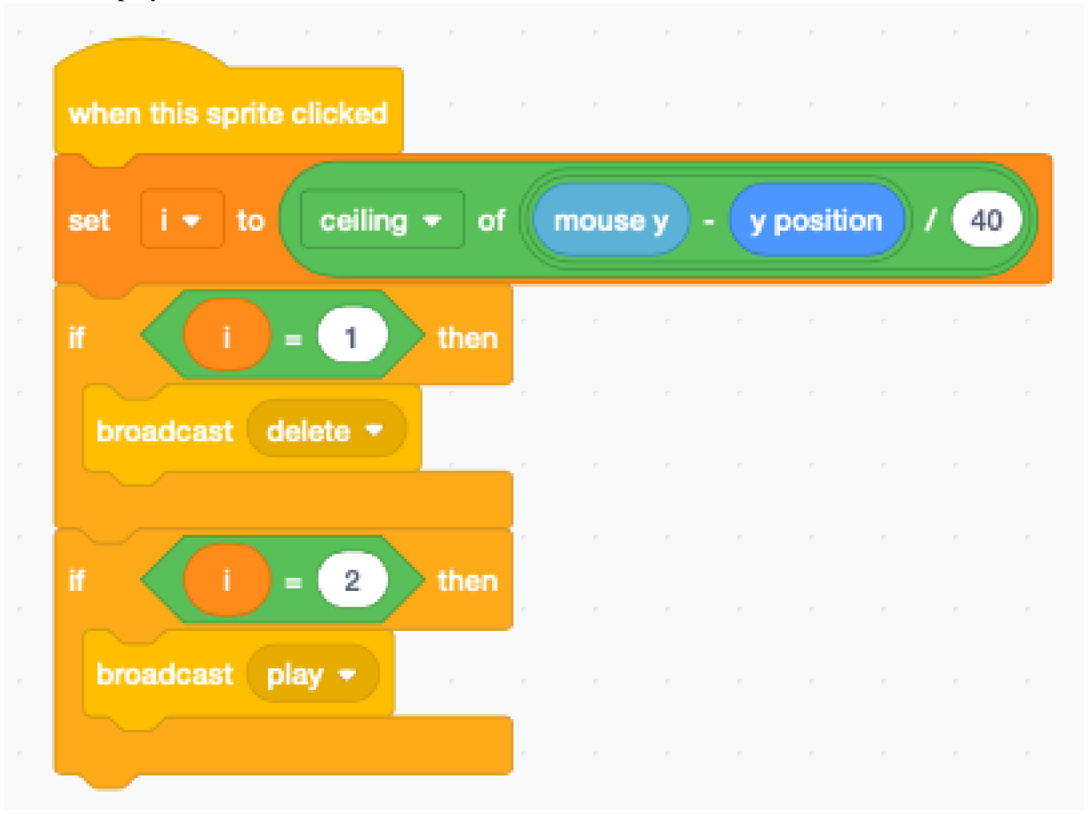
```
when I receive [play ▾]
erase all
set [i ▾] to (1)
repeat until <(i) > (length of [Points ▾])>
    get point
    go to x: (x) y: (y)
    if <(down) = (true)> then
        pen down
    else
        pen up
    wait (0.05) seconds
```

The **Button** sprite uses 2 emoticons. Pay attention to leave a space after the first icon, in order to make the frame larger. On the Android OS the icons are larger as on the iOS, and the icons are cut off on the right side. If you don't add the extras empty space after the emoticon, the icon will be cropped.



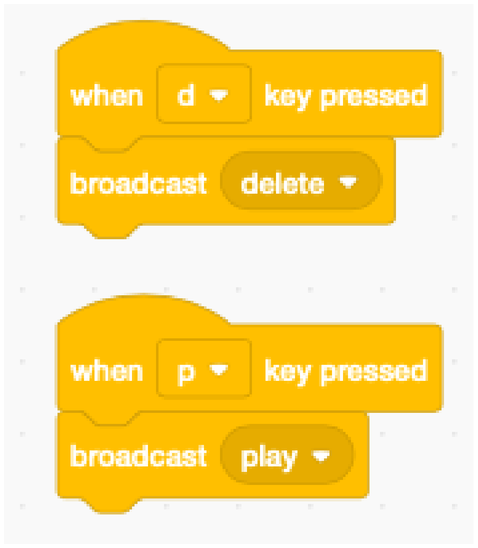
Make the button hight 40 points. Using **ceiling** we can calculate the index **i** of the button.

— 1 : delete
— 2 : play



```
when this sprite clicked
set i to (ceiling of ((mouse y - y position) / 40))
if <(i) = 1> then
    broadcast delete
if <(i) = 2> then
    broadcast play
```

To use keyboard shortcuts on the computer, we add this



```
when d key pressed
broadcast delete

when p key pressed
broadcast play
```

Try it out

https://scratch.mit.edu/projects/404148380

CHAPITRE 10

Rooms

In some games the player can move to different rooms.

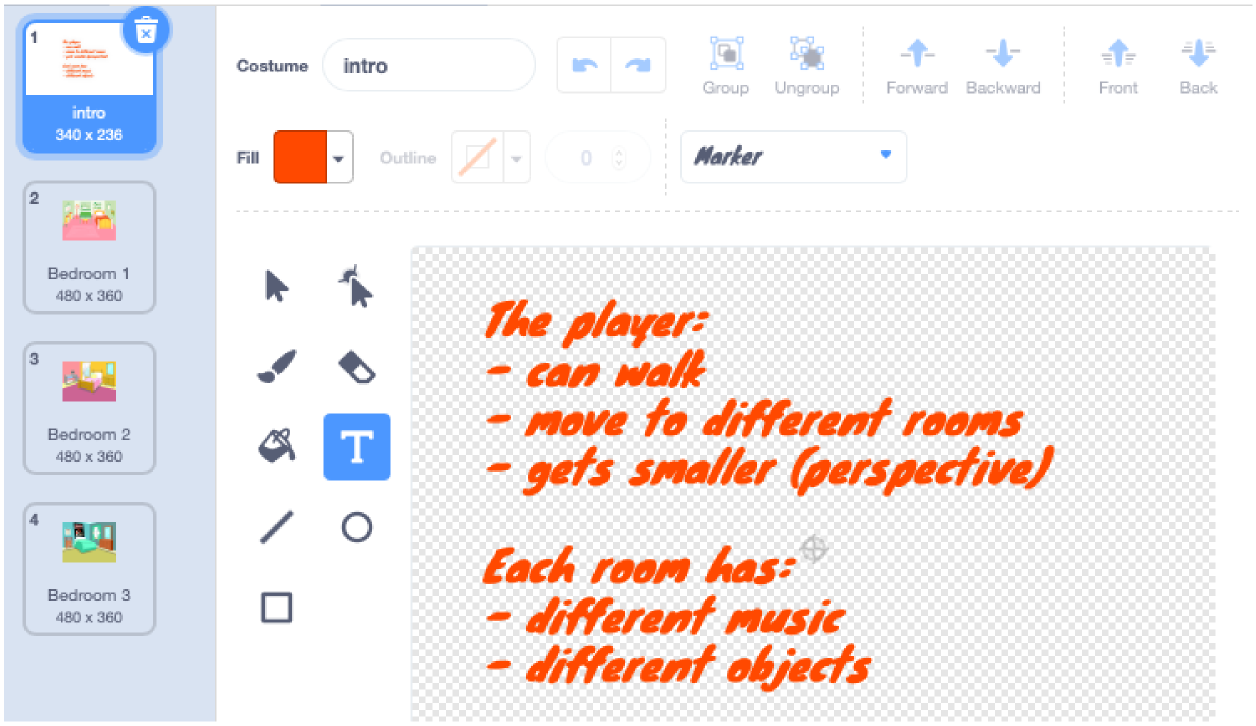
https://scratch.mit.edu/projects/391583147

10.1 The backdrops

For this project we use 4 backgrounds :

— one intro screen
— three different bedrooms

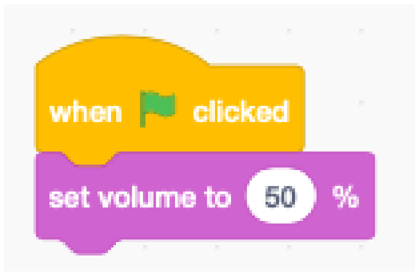
The player starts on the intro screen and can visit all three rooms. Each room has it's own music and object.



The player:
- can walk
- move to different rooms
- gets smaller (perspective)

Each room has:
- different music
- different objects

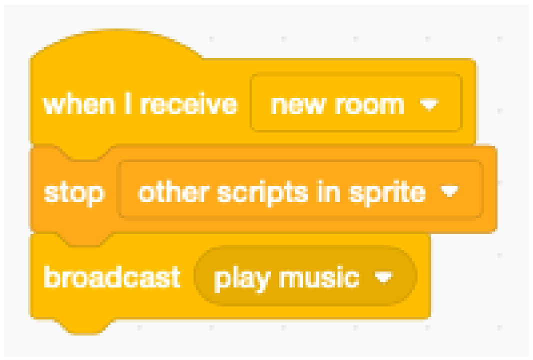
When the stage starts, it sets the background volume level.



```
when [flag] clicked
set volume to (50) %
```

Switching rooms is done from the player. It's the player who decides to enter new rooms. In that case he sends a **new room** message.

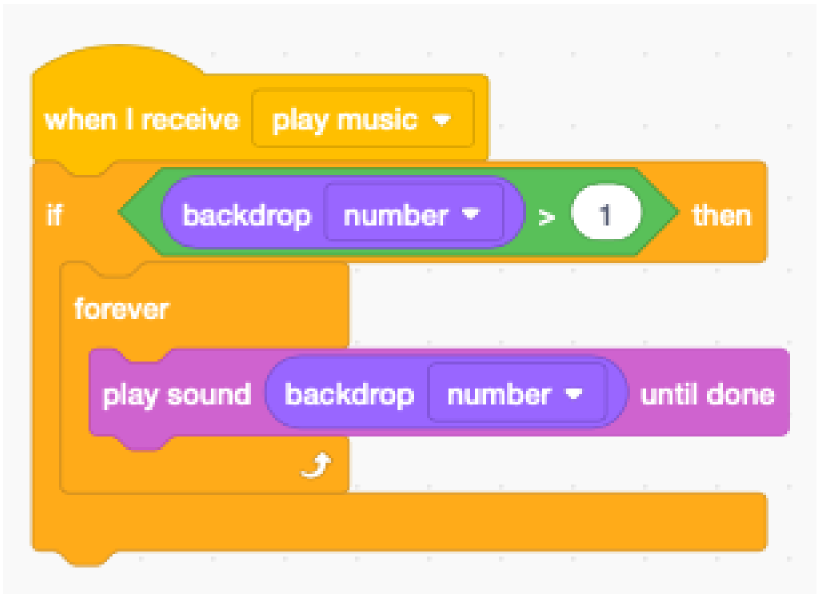
When the stage recives the **new room** message, it stops the other stage scripts, which stops the current music. It then sends a new message **play music**.



```
when I receive [new room]
stop [other scripts in sprite]
broadcast [play music]
```

When the stage receveives the message **play music** it checks if the backdrop is the intro screen (1). There is no music with the intro.

Otherwise it enters a **forever** loop to play the sound whos number corresponds to the current backdrop



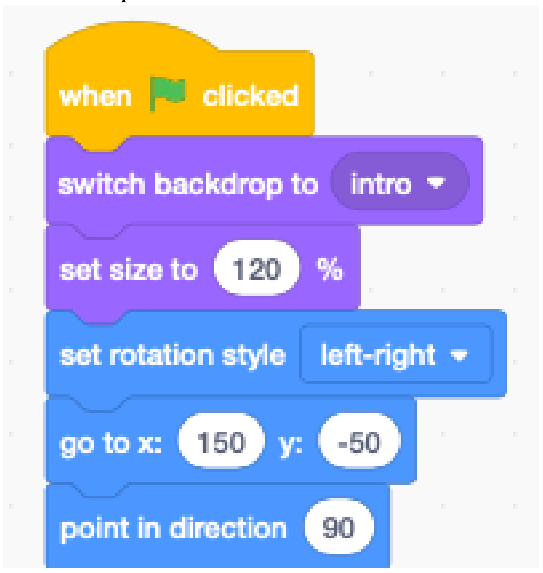
```
when I receive [play music ▾]
if <(backdrop [number ▾]) > (1)> then
    forever
        play sound (backdrop [number ▾]) until done
    ⤴
```

The only way to interrupt this **forever** loop is to use the block ** stop othe scripts in sprite**.

10.2 The player

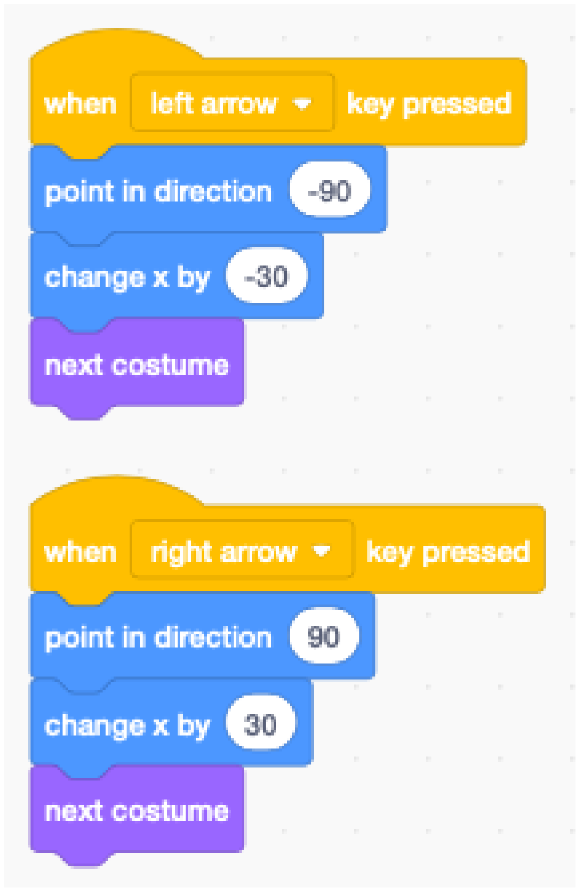
The cat is the main character. It is moved via the arrow keys.

When starting the program we set
  — intro backdrop
  — cat position, size and direction



```
when ⚑ clicked
switch backdrop to (intro ▾)
set size to (120) %
set rotation style [left-right ▾]
go to x: (150) y: (-50)
point in direction (90)
```

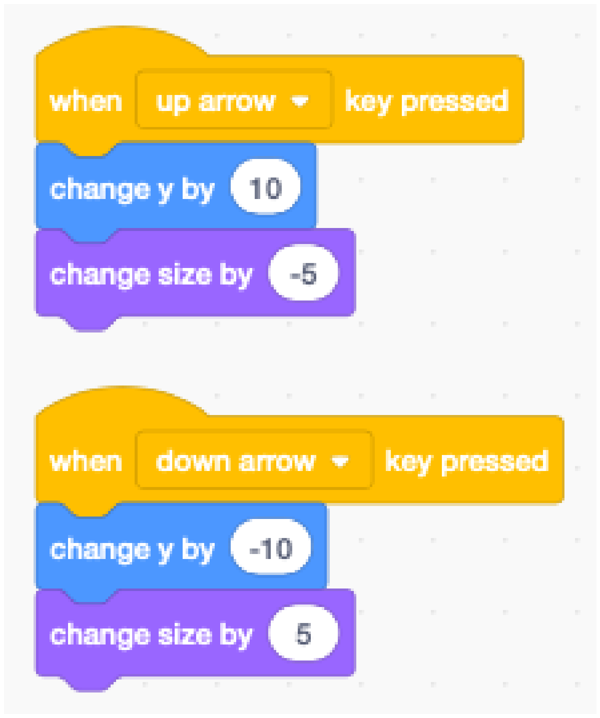
The player can move left and write.



```
when left arrow key pressed
point in direction -90
change x by -30
next costume

when right arrow key pressed
point in direction 90
change x by 30
next costume
```

10.3 Perspective view

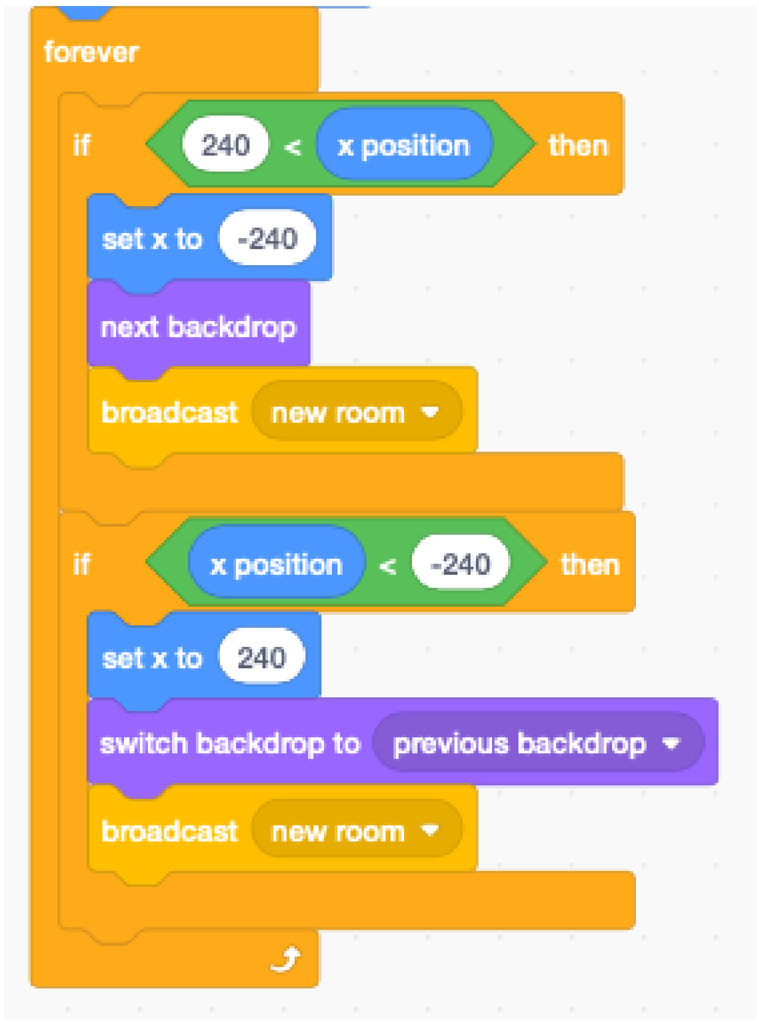
The cat can also go deeper into the room. Because it moves away from the viewer, we make it smaller.



```
when up arrow key pressed
change y by 10
change size by -5

when down arrow key pressed
change y by -10
change size by 5
```

10.4 Changing rooms

Whenever the cat reaches the left or right border, it enters a new room and reappears on the other side.



```
forever
    if 240 < x position then
        set x to -240
        next backdrop
        broadcast new room
    if x position < -240 then
        set x to 240
        switch backdrop to previous backdrop
        broadcast new room
```

CHAPITRE 11

Scenario

Many Scratch projects involve story-telling such as :

— tutorials
— animations
— comics
— music videos

The common trait is a linear progression. There is a beginning and an end. In this section we create a framework to easily order and manipulate

— dialog
— sprite movement
— background
— sound

The basic idea is to use a **script** or scenario. It is a list where each line describes what happenes at that moment of the story.

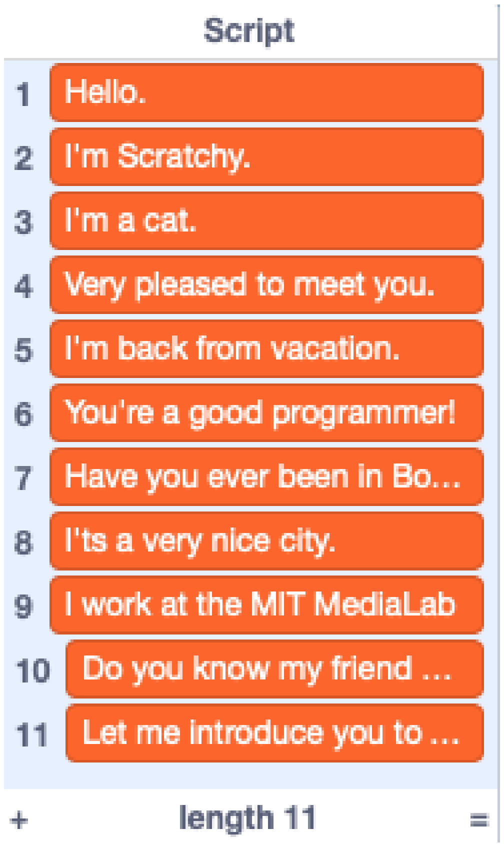
It contains the

— dialog of all actors
— when and what they do
— where they are

https://scratch.mit.edu/projects/399904114

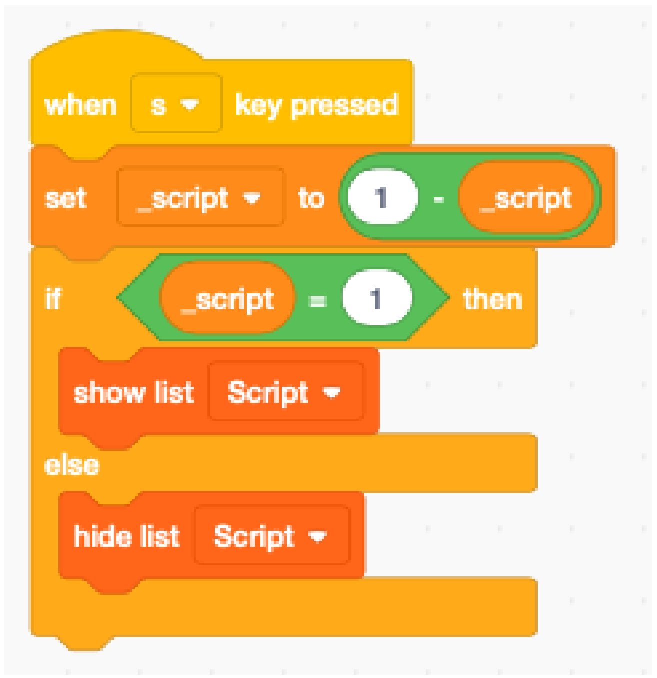
11.1 The Script list

The main data structure of this program is the **Script** list. It contains all lines of dialog.


Script
1 Hello.
2 I'm Scratchy.
3 I'm a cat.
4 Very pleased to meet you.
5 I'm back from vacation.
6 You're a good programmer!
7 Have you ever been in Bo…
8 I'ts a very nice city.
9 I work at the MIT MediaLab
10 Do you know my friend …
11 Let me introduce you to …
+ length 11 =

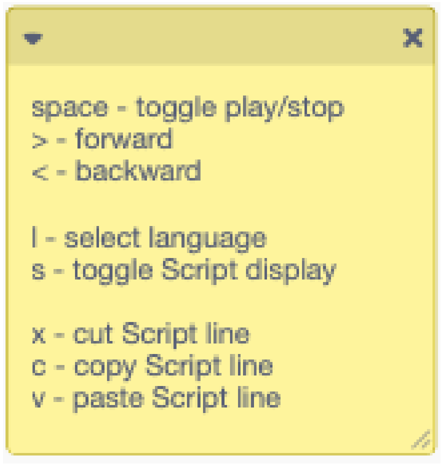
In order to create the scenario, it is easiest to show the **Script** and edit it by :
- + add a new item
- x delete the item
- (return) add new item

Pressing the **S** toggles the boolean **_script** variable and shows or hides the **Script** list.


```
when s key pressed
set _script to (1 - _script)
if _script = 1 then
    show list Script
else
    hide list Script
```

11.2 Add a comment

The canvas contains a comment to give an overview of what the program can do and how to use the program.

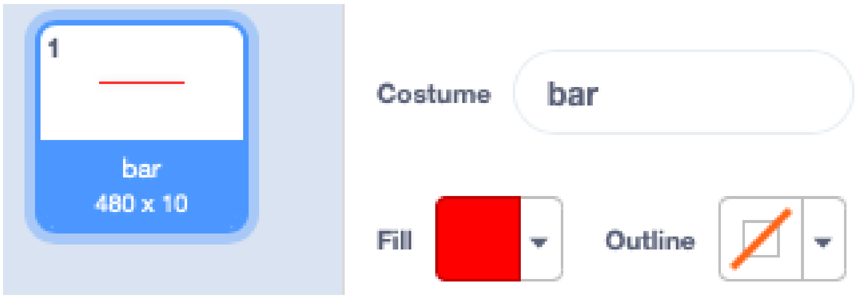


```
space - toggle play/stop
> - forward
< - backward

l - select language
s - toggle Script display

x - cut Script line
c - copy Script line
v - paste Script line
```

11.3 The progress bar

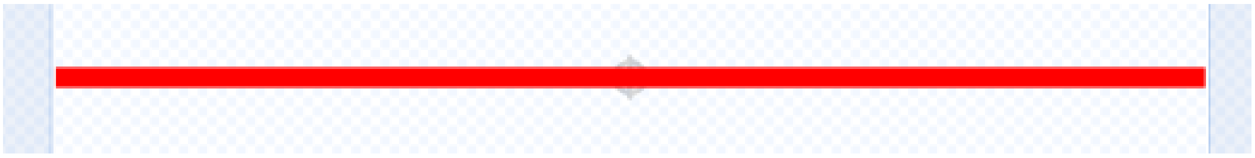
To indicate the progression, we will use a progress bar, like the one used in Youtube.

We call the costume **bar**, make it red like in Youtube, without an outline. The important thing is to make it exactly the lenght of the screen (480 pixel) and 10 pixel high.

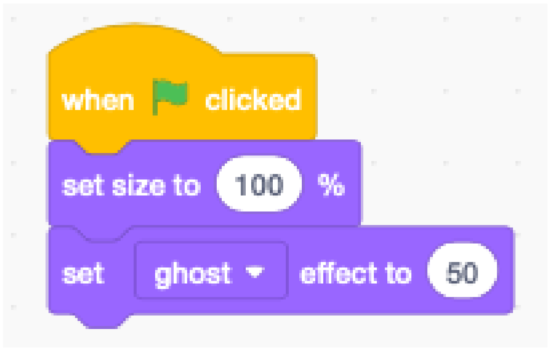


Costume: bar
Fill: 🟥
Outline: none

It must be placed so that the origin is exaclty in the center.

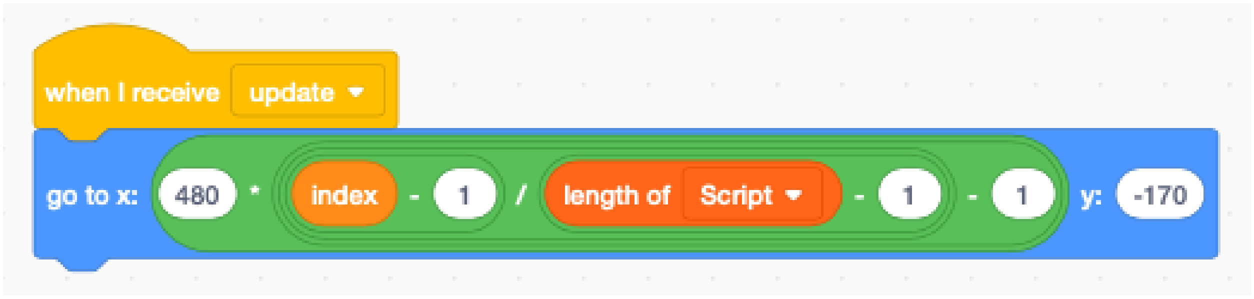


At start we set the size to 100% and the ghost effect to 50% transparency.



```
when [flag] clicked
set size to (100) %
set [ghost] effect to (50)
```

When the script **index** moves we update the porgress par. It covers the complete area of the screen.



```
when I receive [update]
go to x: ((480) * (((index) - (1)) / ((length of [Script]) - (1)))) - (1) y: (-170)
```

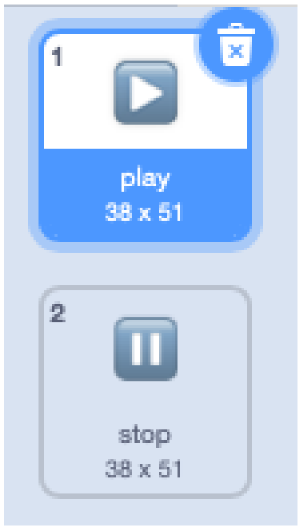
11.4 The play/stop button

Besides the progress bar, the play/stop button is the major user interface element. When the program loads, it starts playing from the beginning.

We create a new Sprite called **Bar** which has two costumes :

— the **play** button
— the **stop** button

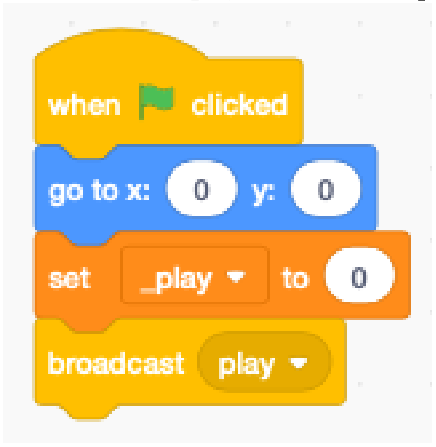
Both are centered and have the same size (38x51 pixels).



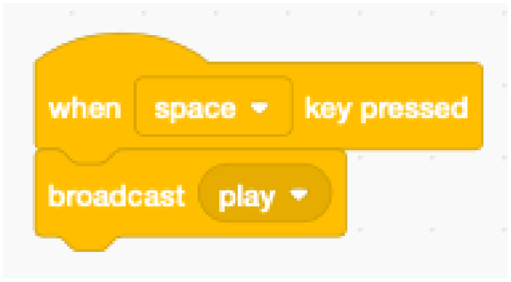
At start we

— place the sprite in the center
— set **_play** to 0

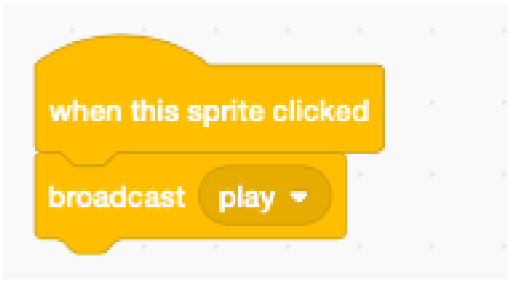
— broadcast **play** which will start playing



Pressing the space key toggles between play and stop.

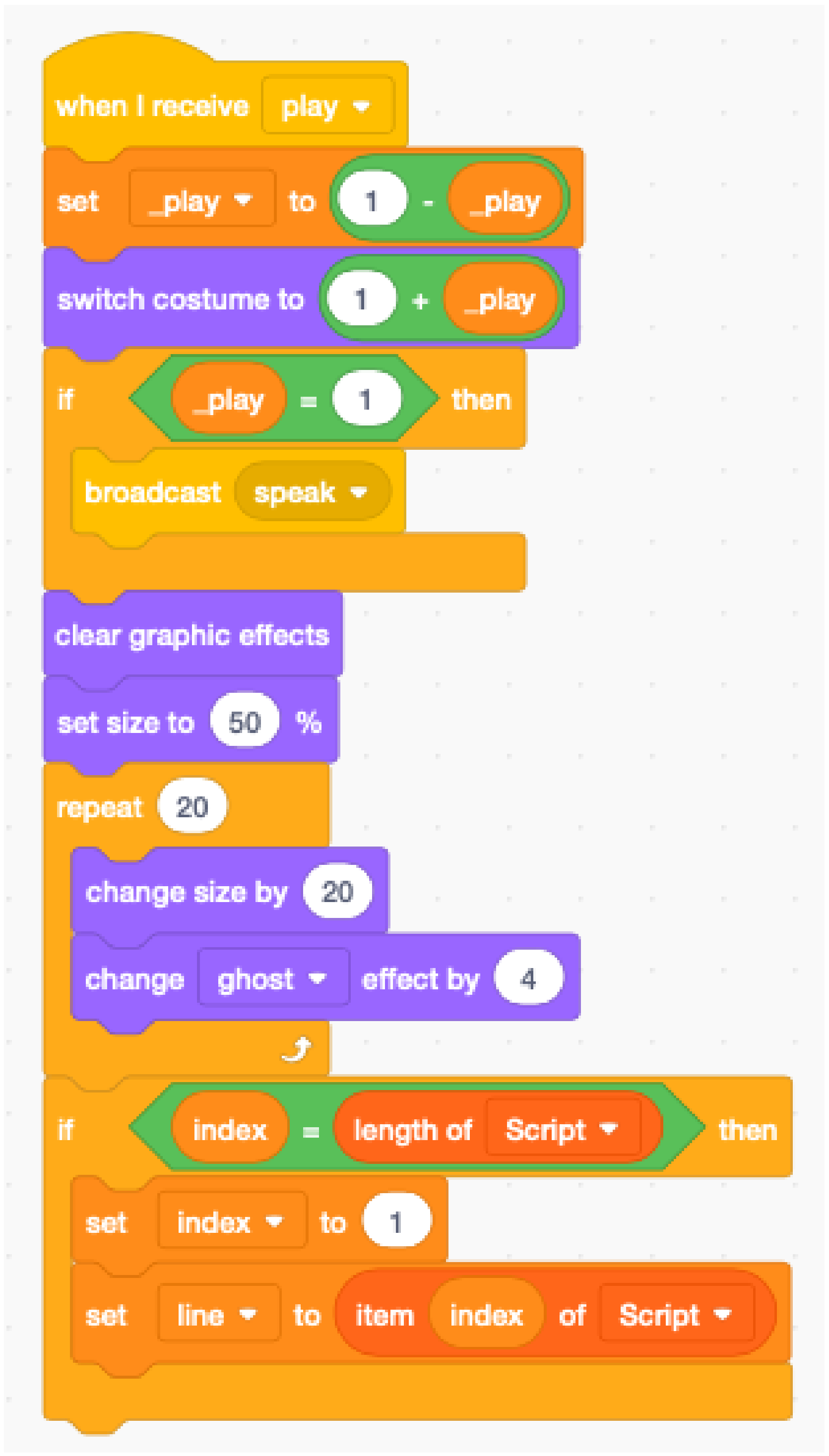


Clicking the large, half-transparent play/stop button does the same.



The **_play** variable is a boolean variable which toggles between 0 and 1. Adding 1 to it let's us set the costume.
— **_play=0**, stopped, show the play button (costume 1)
— **_play=1**, playing, show the stop button (costume 2)
If **_play=1** broadcast the message **speak**.

Each toggling of the play/stop button executes an animation :
— start small (size=50%) at the center
— grow larger and become transparent
If the index is on the last script position, we set the index to the first position. This allows to restart from the beginning and not with the last script position.

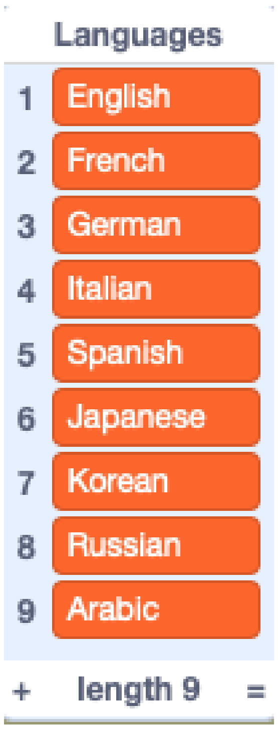


```
when I receive [play v]
set [_play v] to ((1) - (_play))
switch costume to ((1) + (_play))
if <(_play) = (1)> then
    broadcast [speak v]
end
clear graphic effects
set size to (50) %
repeat (20)
    change size by (20)
    change [ghost v] effect by (4)
end
if <(index) = (length of [Script v])> then
    set [index v] to (1)
    set [line v] to (item (index) of [Script v])
end
```

11.5 Speak different languages

We are going to use Google translate and Text-to-Speech to display and speak the dialog in several languages.

The **Languages** list contains the 9 preconfigured lanugages.



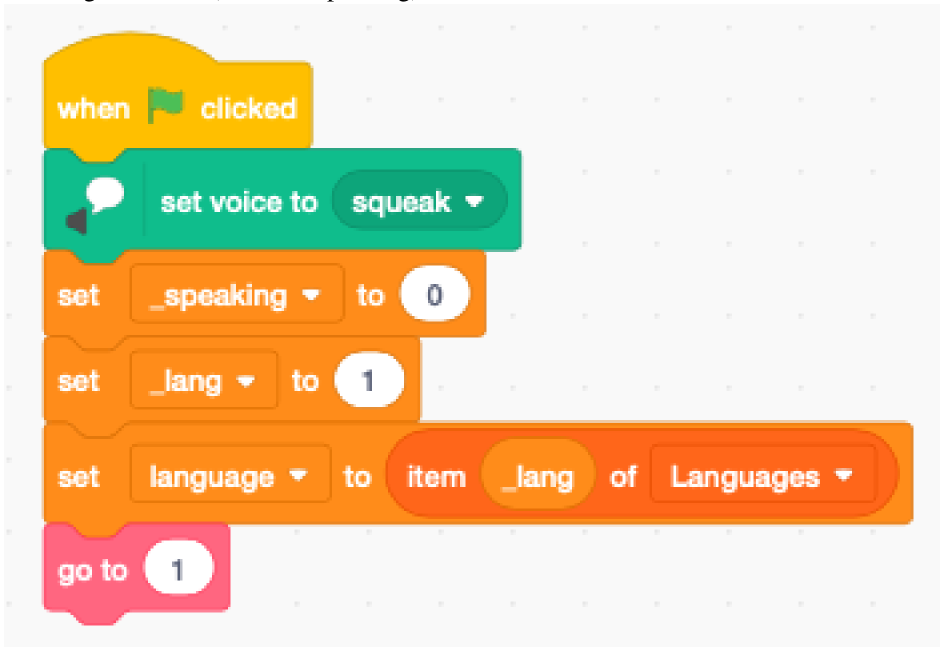
Languages

1	English
2	French
3	German
4	Italian
5	Spanish
6	Japanese
7	Korean
8	Russian
9	Arabic

+ length 9 =

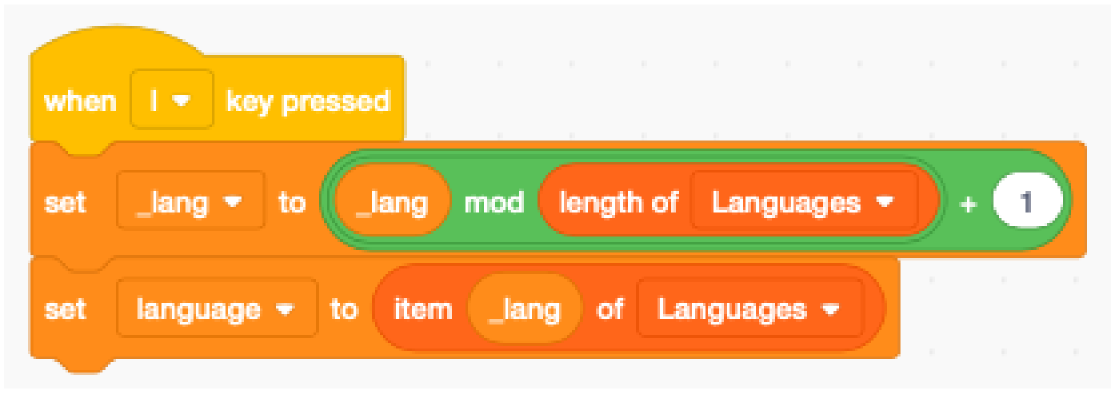
At start we :

- — set the voice to **squeak** (seems appropriate for Scratchy)
- — set the language index **_lang=1** (English)
- — go to line 1 (and start speaking)



```
when flag clicked
set voice to squeak
set _speaking to 0
set _lang to 1
set language to item _lang of Languages
go to 1
```

Pressing the **L** key cycles through the 9 preset languages.

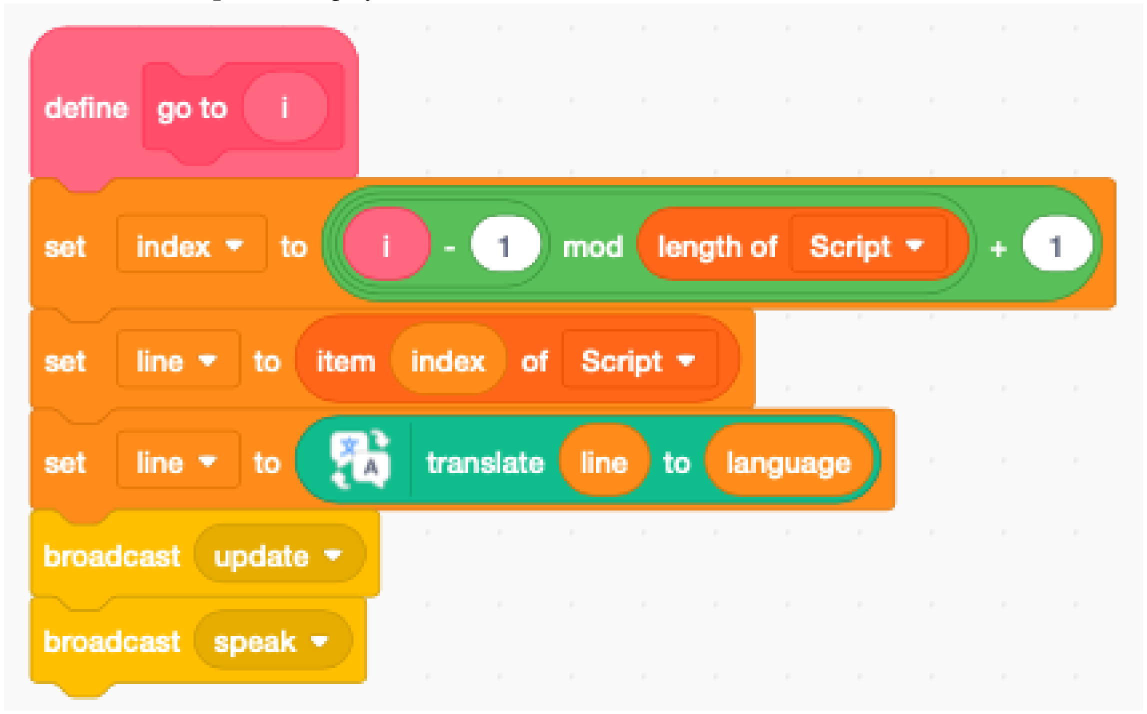


```
when l key pressed
set _lang to ((_lang mod (length of Languages)) + 1)
set language to (item _lang of Languages)
```

11.6 Set the line

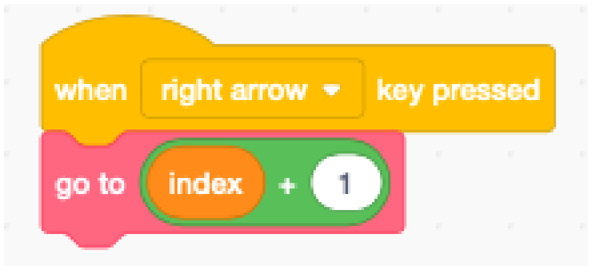
To set the current line number we create the function **go to (i)** It sets the current line to a number from 1 to lenght(Script). The function

- — limits the **index** variable to the correct range using a **mod** expression
- — sets the **line** variable to the **Script** item pointed **index**
- — translates the line to the current language
- — broadcasts **update** to set the progress bar
- — broadcasts **speak** to display and read out the line

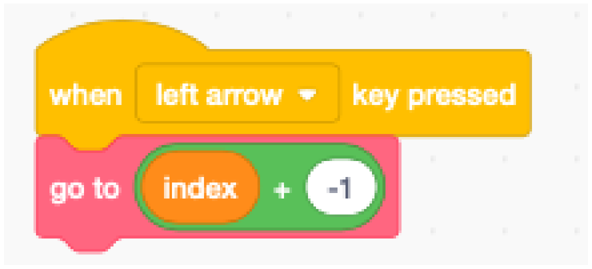


```
define go to (i)
set index to (((i - 1) mod (length of Script)) + 1)
set line to (item index of Script)
set line to (translate line to language)
broadcast update
broadcast speak
```

Now we can use the arrow keys to increment the line (right)



or decrement the line (left)

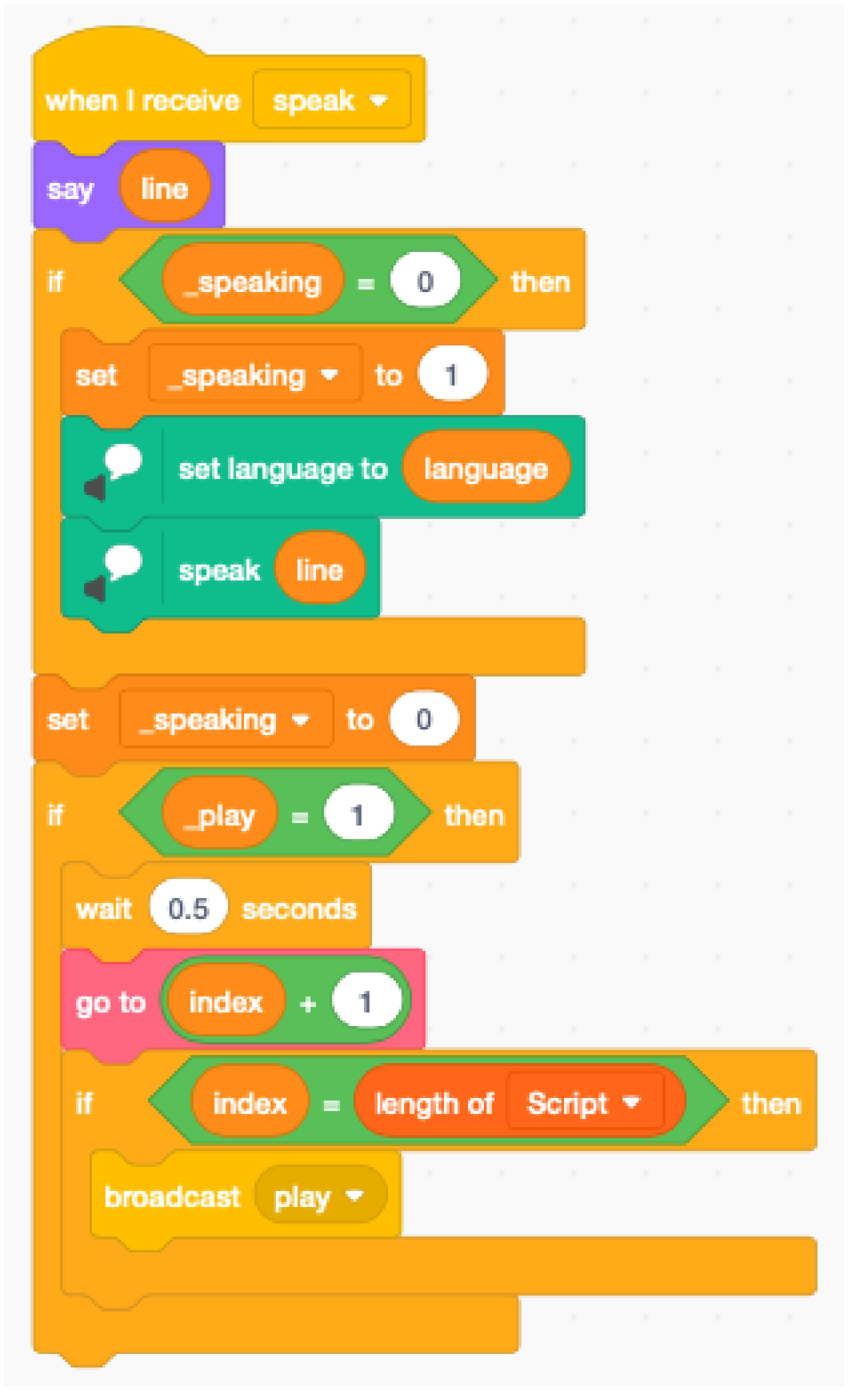


11.7 Speak a line

First we display the speech bubble. Then we speak the line, unless we are currently already speaking.

If **_play=1** then we :

- wait 0.5 seconds
- go to the next line (and read it)



```
when I receive speak
say line
if _speaking = 0 then
    set _speaking to 1
    set language to language
    speak line
set _speaking to 0
if _play = 1 then
    wait 0.5 seconds
    go to index + 1
    if index = length of Script then
        broadcast play
```

11.8 Cut, copy, paste

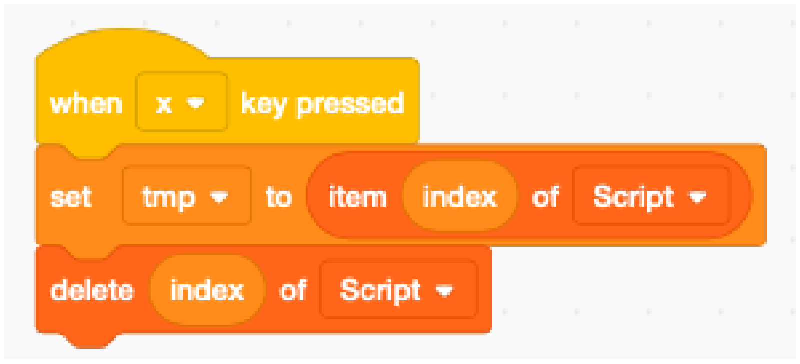
To edit complete lines of the **Script** list we add the three standard functions :

— cut (x)

— copy (c)

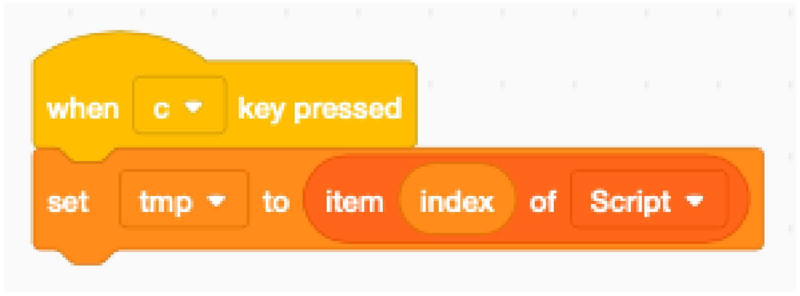
— paste (v)

Cutting a line copies the current line first to the **tmp** variable and deletes the item.



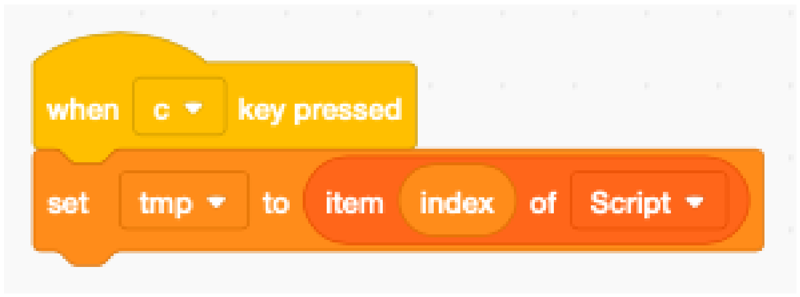
```
when x key pressed
set tmp to item index of Script
delete index of Script
```

Copying a line just copies the current line to the **tmp** variable.



```
when c key pressed
set tmp to item index of Script
```

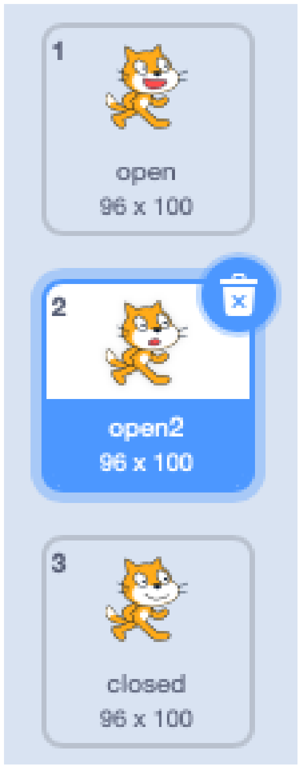
Pasting a line inserts the **tmp** variable at the current **index** position.



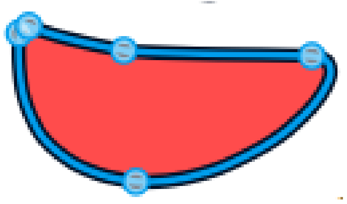
```
when c key pressed
set tmp to item index of Script
```

11.9 Draw the mouth

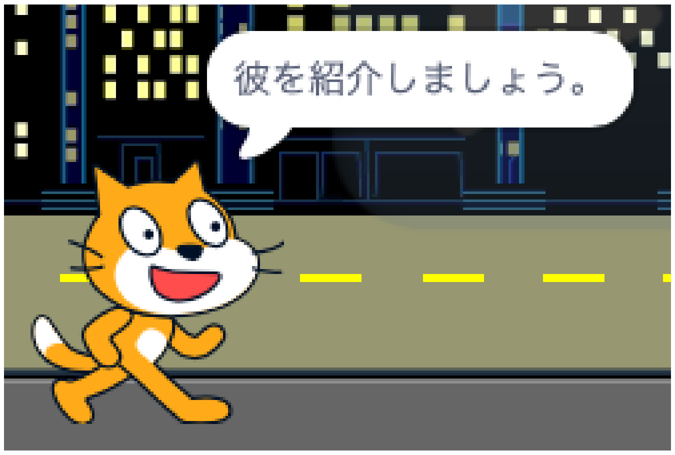
To give the indication of speaking, we should animathe the mouth. Duplicate three times the first cat sprite.



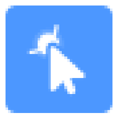
Now color the mouth of the first sprite with 70% red.



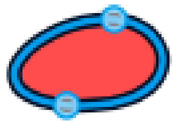
The cat sprite will look like this :



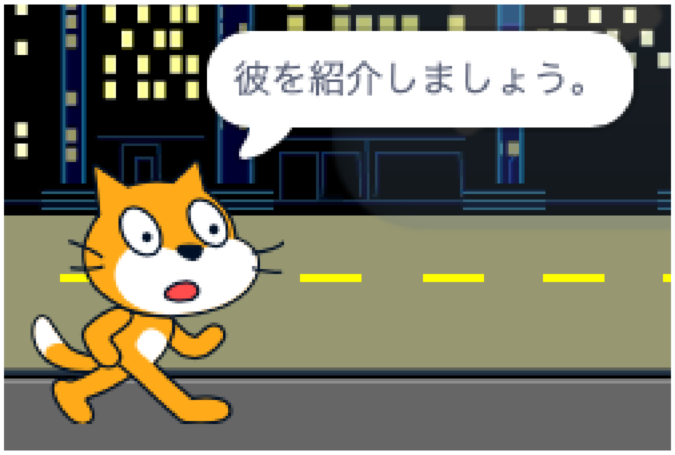
Now use the reshape tool to remove picture points.



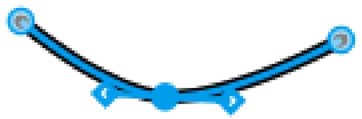
Remove the edges of the mouth to get this.



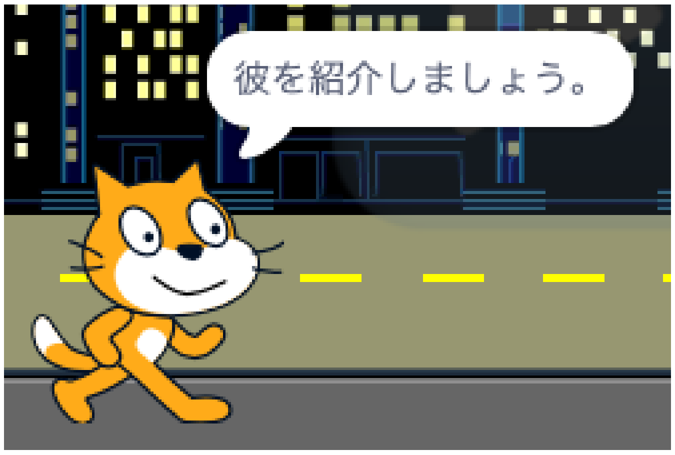
The cat sprite will look like this.



Remove the mouth completely. Draw a new black line with a with of 2.4 points. With the reshape tool click in the center and pull the line downwards until you get this :



The cat sprite will lok like this :



11.10 Animate the mouth

To create the animation, we select randomly one of the 3 mouth sprites. In between we wait 0.1 seconds. The loop repeats forever, and is stopped from the other script.

4_scenario/Animate.png

CHAPITRE 12

Scripts

12.1 Stopping scripts

https://scratch.mit.edu/projects/391832106

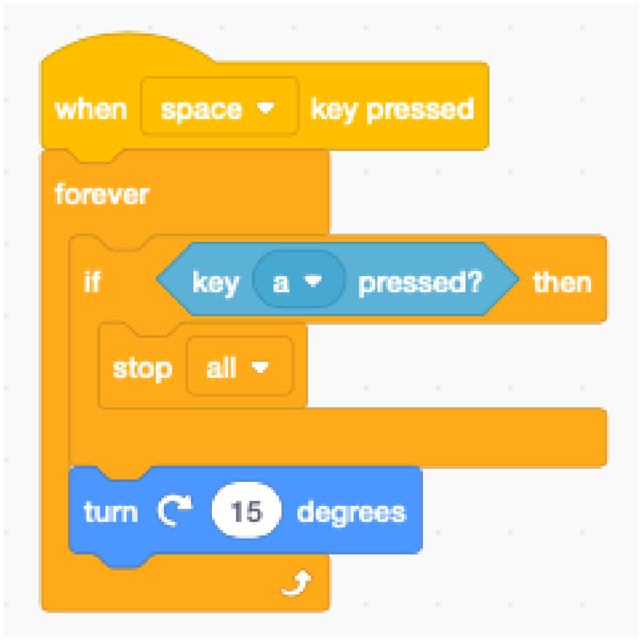
This project has three sprites A, B and C.

When pressing the SPACE key, three forever loops are started which all rotate the letter. All three loops are defined in the A sprite.

Pressing :

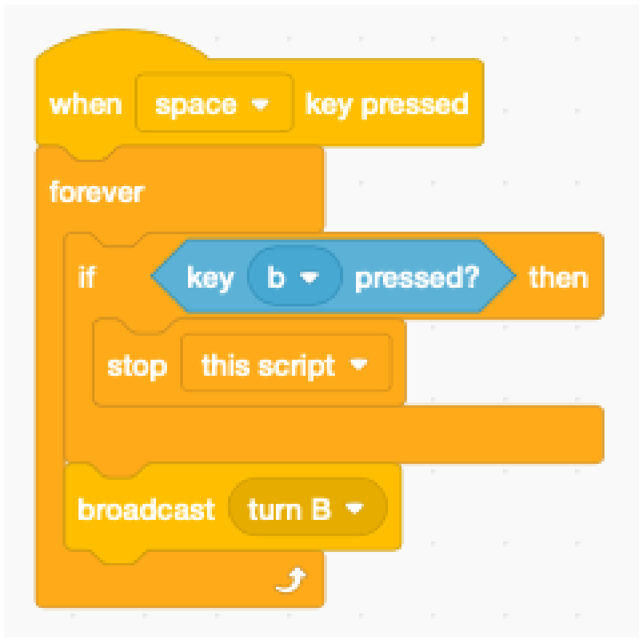
- A - stops all scripts (A, B, C)
- B - stops the current script (B)
- C - stops the other scripts (A, B)

Scripts defined under the A sprite only can move or modify the A sprite. In order to modify the B or C sprite, they need to broadcast a message.



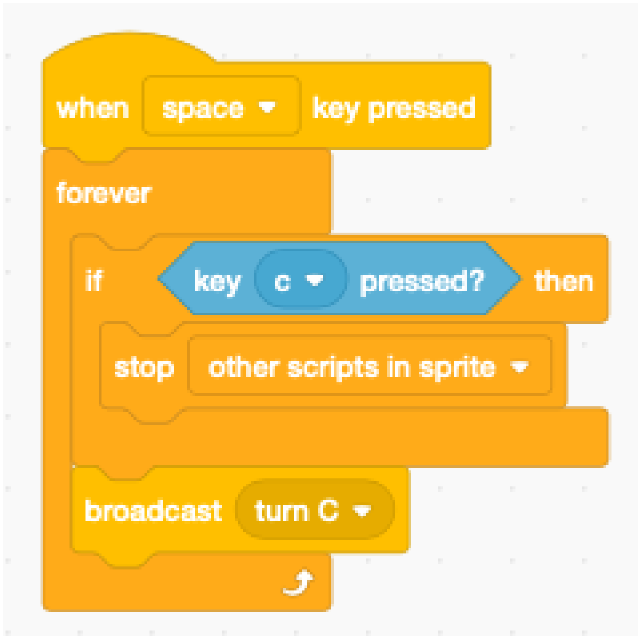
```
when space key pressed
forever
    if key a pressed? then
        stop all
    turn ↻ 15 degrees
```

For a script defined under the A sprite, a message **turn B** needs to be broadcast to a script under the sprite B.



```
when space key pressed
forever
    if key b pressed? then
        stop this script
    broadcast turn B
```

For a script defined under the A sprite, a message **turn C** needs to be broadcast to a script under the sprite C.

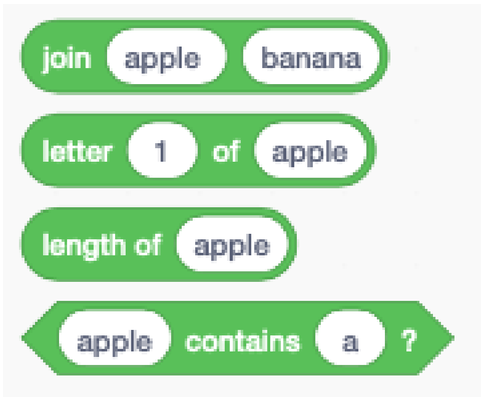


```
when [space ▾] key pressed
forever
    if <key (c ▾) pressed?> then
        stop [other scripts in sprite ▾]
    broadcast (turn C ▾)
```

CHAPITRE 13

String

This section shows a couple of tricks to work with strings. There are 4 operations with strings

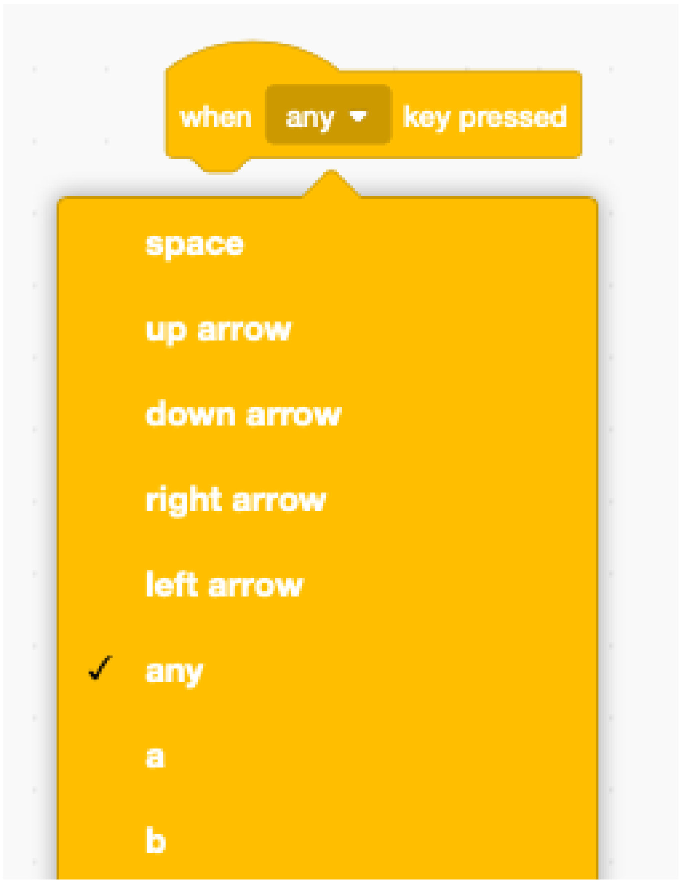


— **join(apple) (banana)** returns `apple banana` (there is a space after apple)
— **letter (1) of (apple)** returns `a`
— **length of (apple)** returns `5`
— **(apple) contains (a)** returns `True`

13.1 Detect a key press

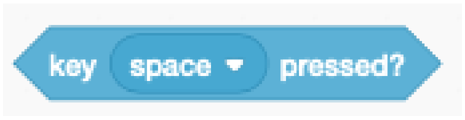
https://scratch.mit.edu/projects/381350556

Scratch can react to key presses. It can react to a specific key such as :
— space key
— arrow keys
— letter keys
— number keys

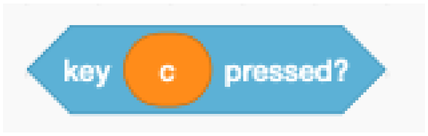


It also can react to *any* key, which includes :
- symbols
- punctuation
- parenthesis

Unfortunately Scratch has no reporter block to tell you which key was pressed with the *any* option. However it has the boolean function **key X pressed**.

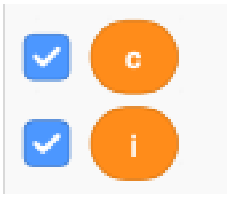


To find out which key was pressed, we iterate inside a loop through the characters of interest.



13.2 Iterate through a string

Our string functions will use two special variables to which we give short (one-letter) names :
- the index **i** pointing to a character inside a string
- the caracter **c** inside the string

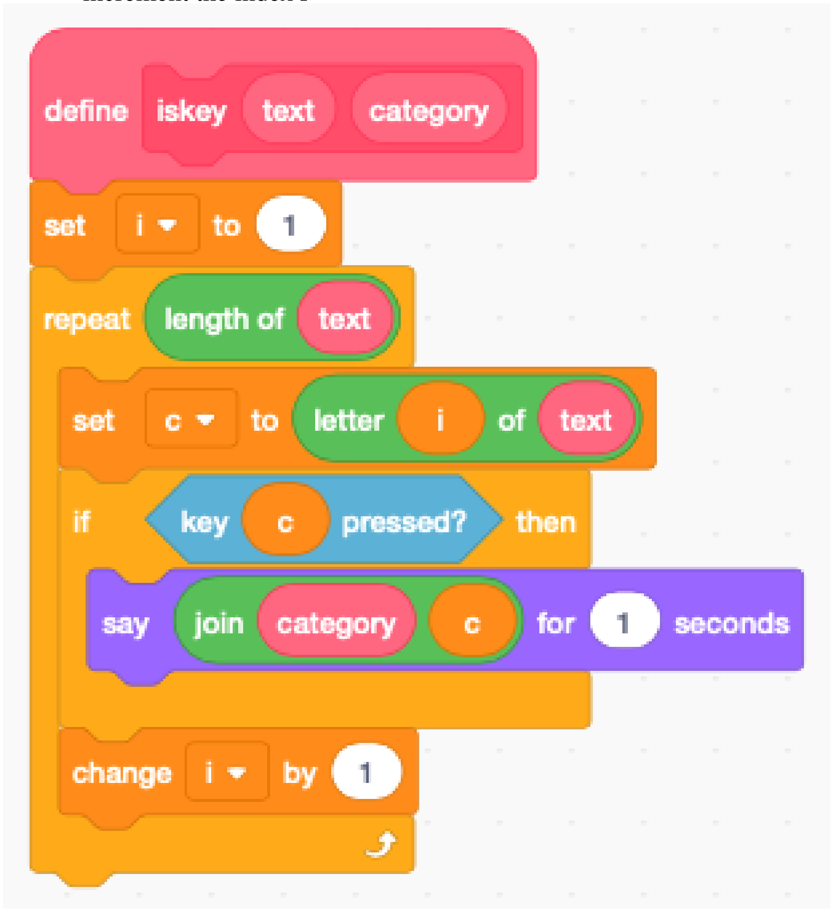


We create the following function **iskey** which has two parameters

- the **characters** of interest
- the name of that **category**

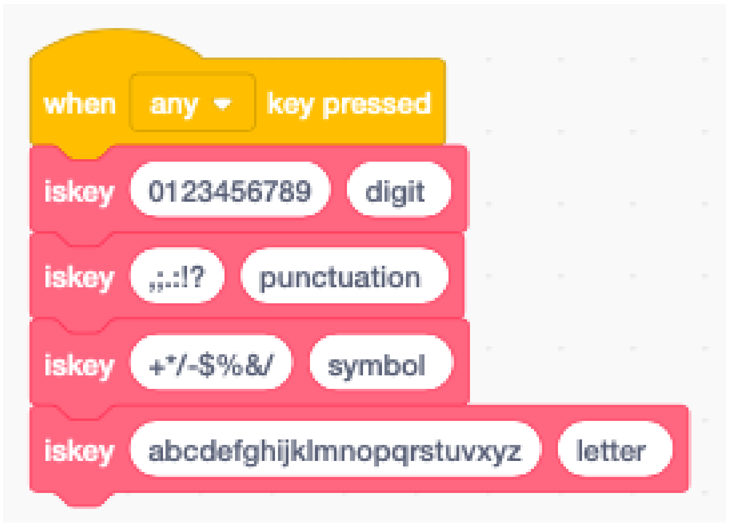
The function shows the general alogrithm :

- set the index **i** to 1 (pointing to the first caractor of text)
- repeat for the length of the text
- set **c** to each consecutive letter of the text
- do something with the caracter **c**
- increment the index **i**



```
define iskey (text) (category)
set [i v] to (1)
repeat (length of (text))
    set [c v] to (letter (i) of (text))
    if <key (c) pressed?> then
        say (join (category) (c)) for (1) seconds
    end
    change [i v] by (1)
end
```

In the example below we detect : digits, punctuation, symbols and letters.

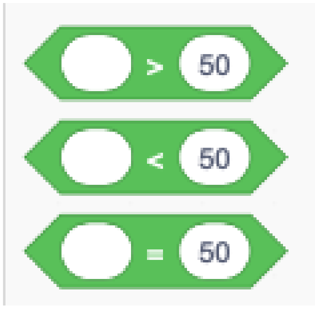


```
when any key pressed
iskey 0123456789 digit
iskey ,;.:!? punctuation
iskey +*/-$%&/ symbol
iskey abcdefghijklmnopqrstuvxyz letter
```

Our cat Scratchy announces the category and the key it recognizes. This method can be used to start a specific action when certain keys are pressed.

13.3 String comparison

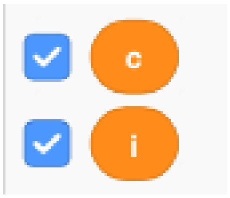
Scratch uses the ASCI code to compare characters. Upper case letters (A..Z) are first transformed into lower-case letters (a..z). There is no easy way to distinguish between upper-case and lower-case letters.



```
( ) > 50
( ) < 50
( ) = 50
```

13.4 Repeat a string

The **repeat** function repeats a text n times. Scratch functions do not allow a return value. In order to return a result we define a variable which we call **result**. The **split** function will requires two return strings, so we define these two variables.

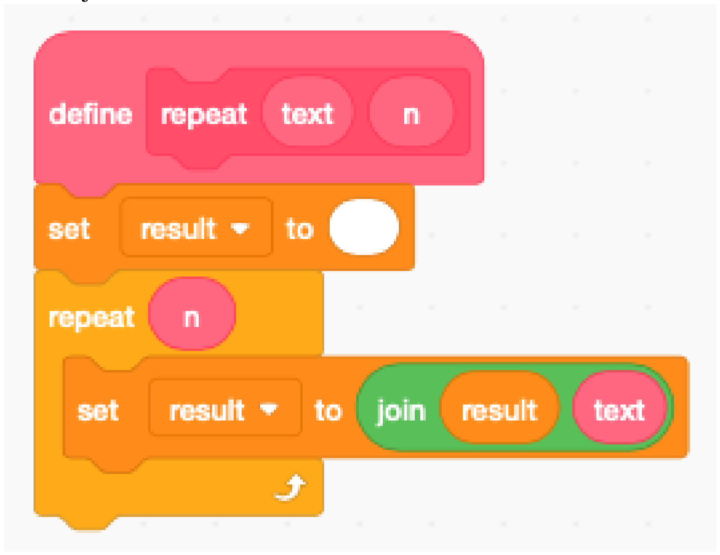


```
☑ c
☑ i
```

The algorithm is quite simple :

- erase the **result** variable
- repeat and iterate **n** times

— join the text at the end of the result

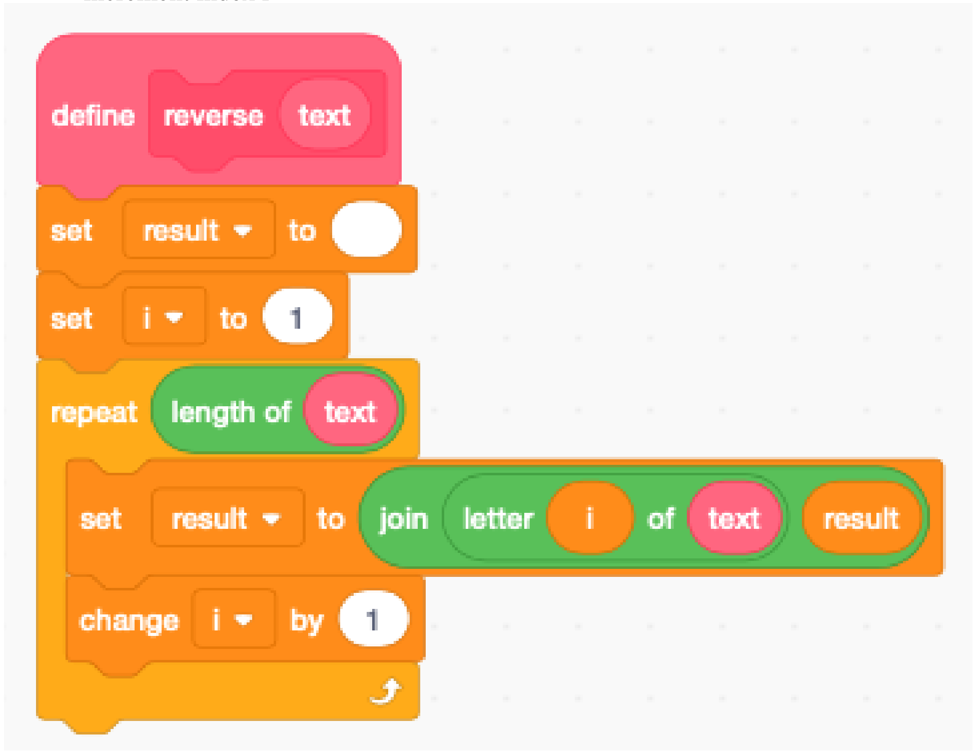


```
define repeat (text) (n)
set [result] to ( )
repeat (n)
    set [result] to (join (result) (text))
```

13.5 Reverse a string

To reverse a string we extract letter by letter and assemble them backwards.

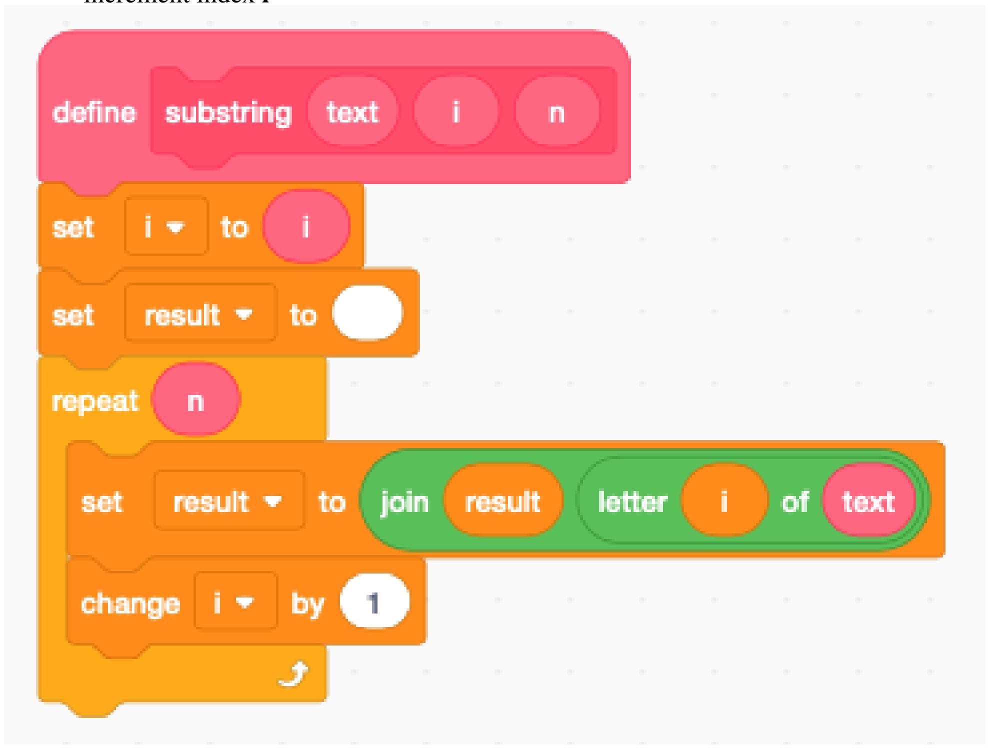
— erase the **result** variable
— set the index **i** to the first character
— repeat for the lenght of text
— join the i-th letter in front of the result
— increment index **i**



```
define reverse (text)
set [result] to ( )
set [i] to (1)
repeat (length of (text))
    set [result] to (join (letter (i) of (text)) (result))
    change [i] by (1)
```

13.6 Extract a sub-string

To extract a substring from position **i** for a length **n** we :
- set the index **i** to the start character
- set the result to empty
- repeat **n** times
- copy a character to the result string
- increment index **i**

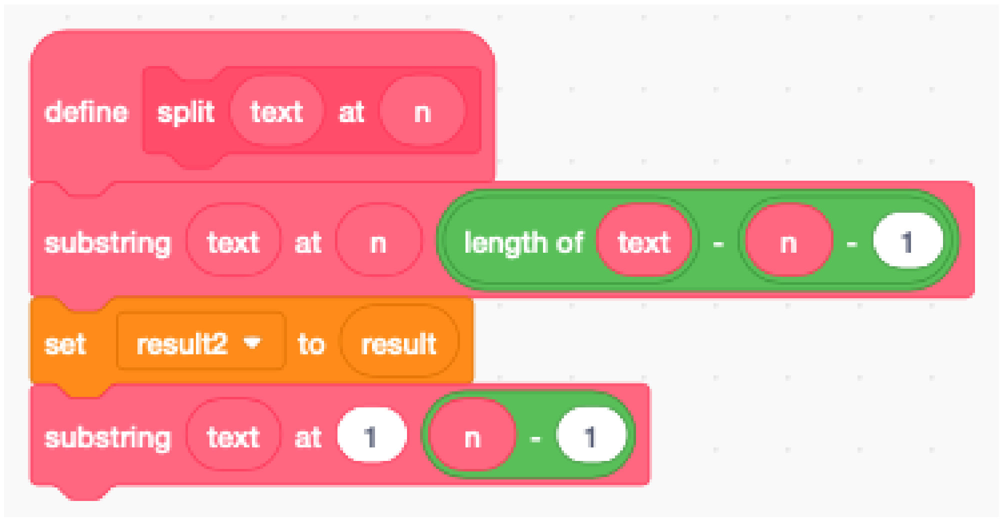


```
define substring (text) (i) (n)
set i to (i)
set result to ()
repeat (n)
    set result to (join (result) (letter (i) of (text)))
    change i by (1)
```

13.7 Split a string

The **split** function splits a text into 2 sub-strings at position **n** :
- The frist n-1 letters are in **result**
- The remaining letters from n onwards are in **result2**

We use the function **substring** twice. Be careful to extract the second part first, as the **substring** function uses **result**.

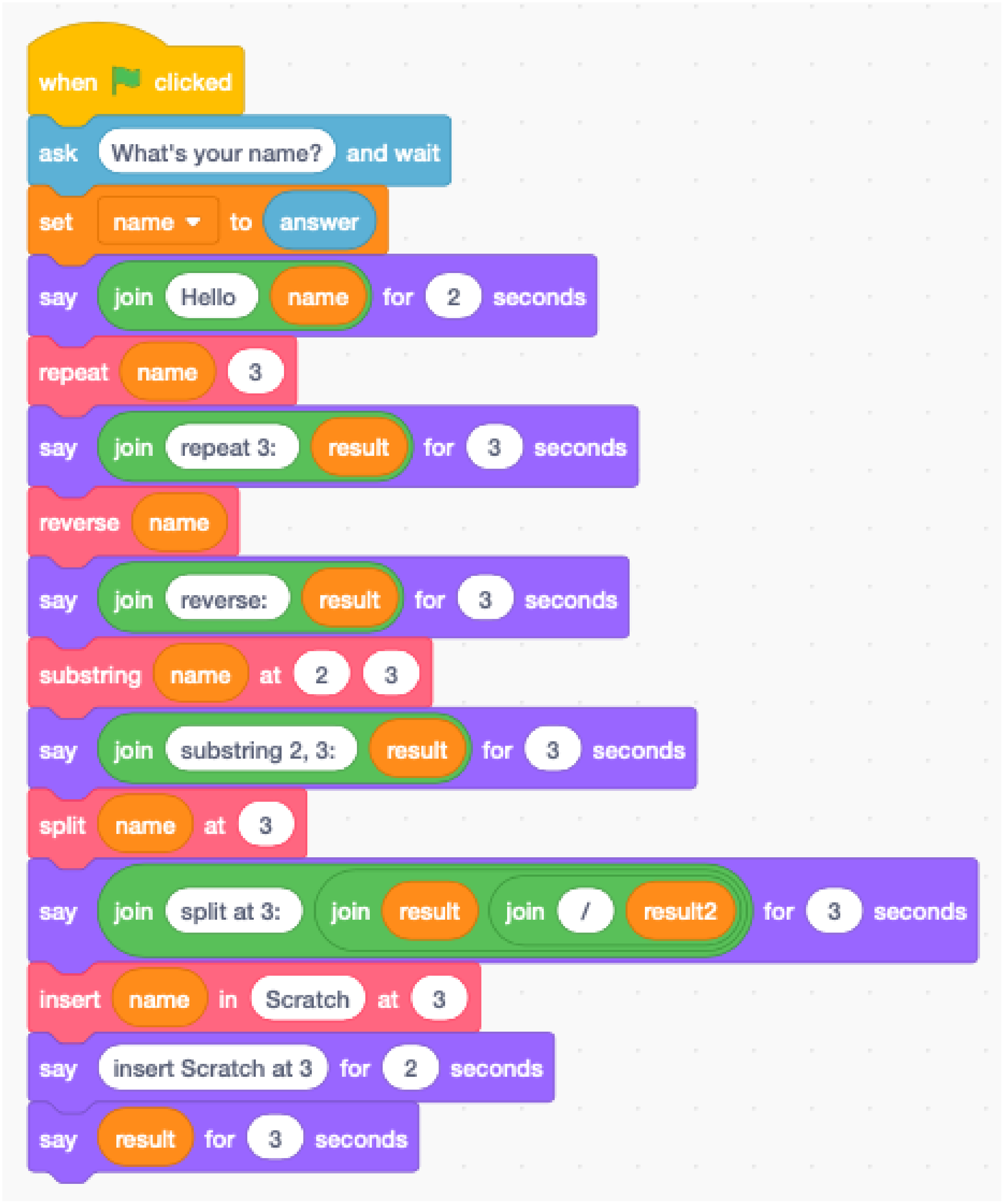


```
define split (text) at (n)
substring (text) at (n) ((length of (text)) - (n) - (1))
set [result2 v] to (result)
substring (text) at (1) ((n) - (1))
```

13.8 Demo

https://scratch.mit.edu/projects/390866776

The following demo program shows the effect of the 5 string manipulation functions.



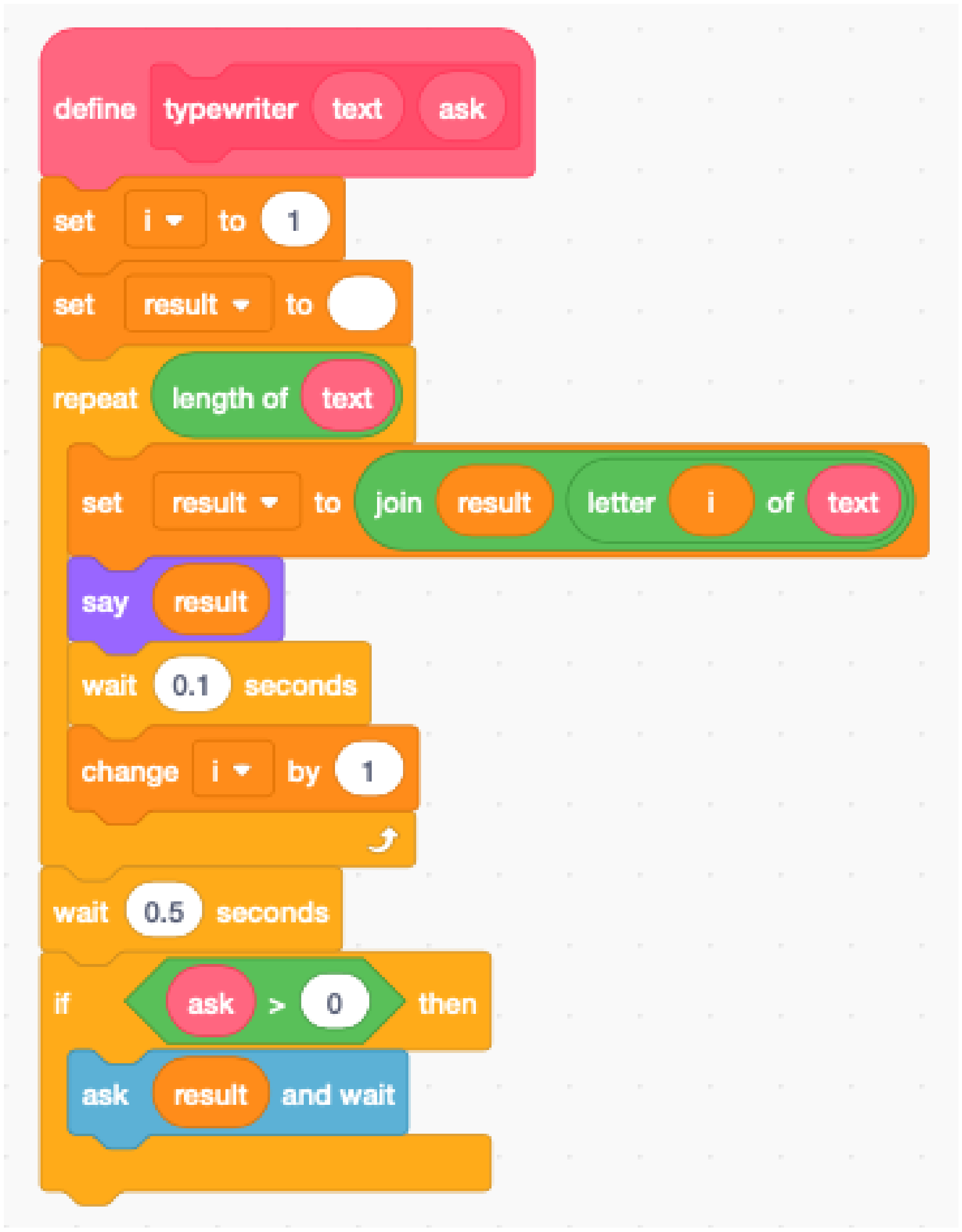
```
when flag clicked
ask (What's your name?) and wait
set [name] to (answer)
say (join (Hello) (name)) for (2) seconds
repeat (name) (3)
say (join (repeat 3:) (result)) for (3) seconds
reverse (name)
say (join (reverse:) (result)) for (3) seconds
substring (name) at (2) (3)
say (join (substring 2, 3:) (result)) for (3) seconds
split (name) at (3)
say (join (split at 3:) (join (result) (join (/) (result2)))) for (3) seconds
insert (name) in (Scratch) at (3)
say (insert Scratch at 3) for (2) seconds
say (result) for (3) seconds
```

13.9 Imitate a typewriter

Sometimes written conversation looks more natural if it is paced like the text appearing on a typewriter.

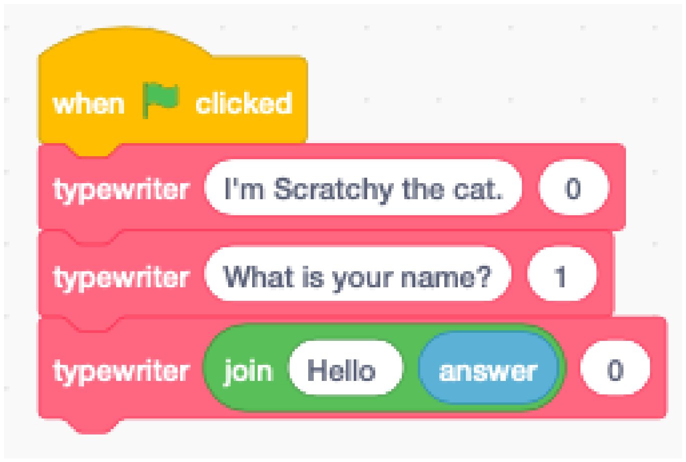
https://scratch.mit.edu/projects/390908846

The fonction **typewriter** has a text argument, and a second **ask** argument. If it is 1 the text is asked as a question, and the user is invited to give an answer.



```
define typewriter (text) (ask)
set i to 1
set result to
repeat (length of (text))
    set result to (join (result) (letter (i) of (text)))
    say (result)
    wait 0.1 seconds
    change i by 1
end
wait 0.5 seconds
if <(ask) > 0> then
    ask (result) and wait
end
```

The following text displays 3 phrases and asks a question. The answer is integrated into the 3rd phrase.



```
when green flag clicked
typewriter (I'm Scratchy the cat.) (0)
typewriter (What is your name?) (1)
typewriter (join (Hello) (answer)) (0)
```

CHAPITRE 14

Tablet

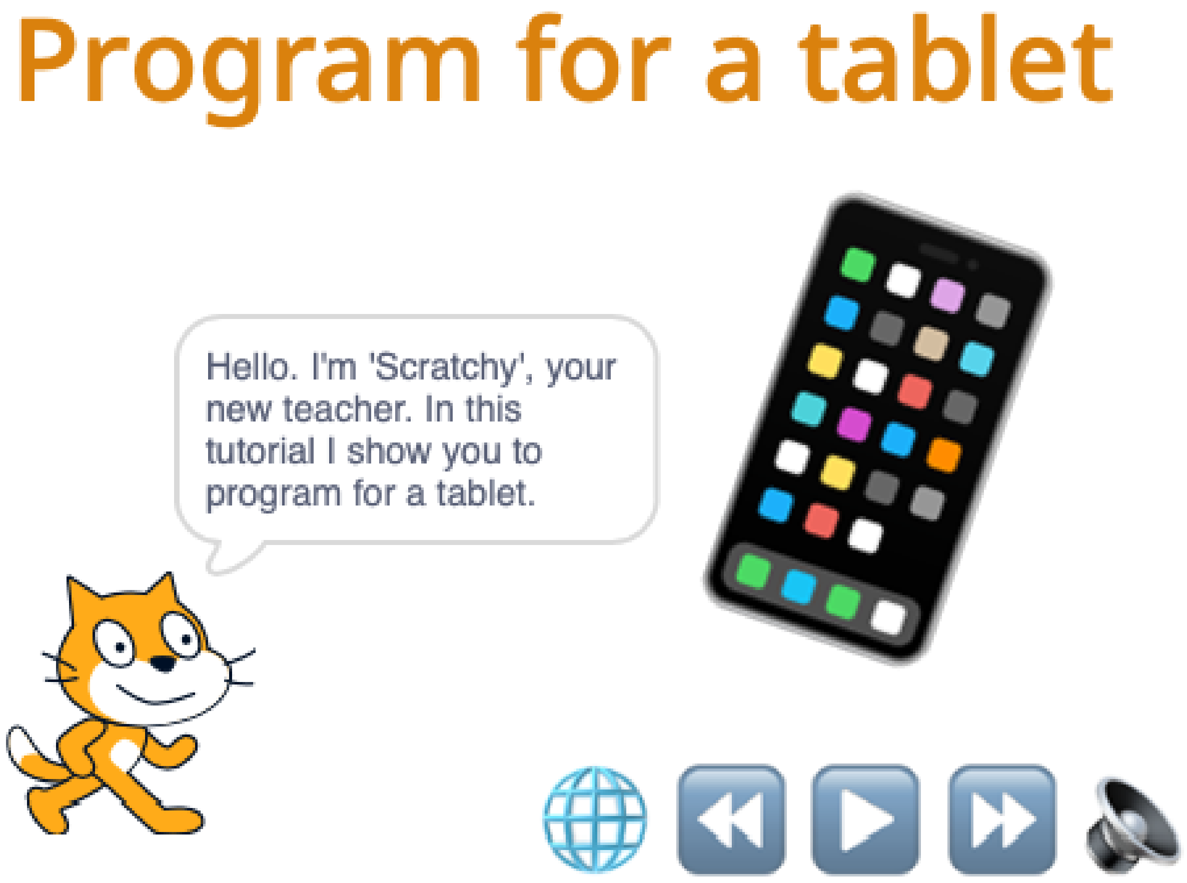
Many users will use Scratch applications on a tablet. Therefore we should take care in our program to make it work with a tablet.

Tablets do not have a keyboard so the only input is the pointer.

https://scratch.mit.edu/projects/396114914

14.1 Introduction

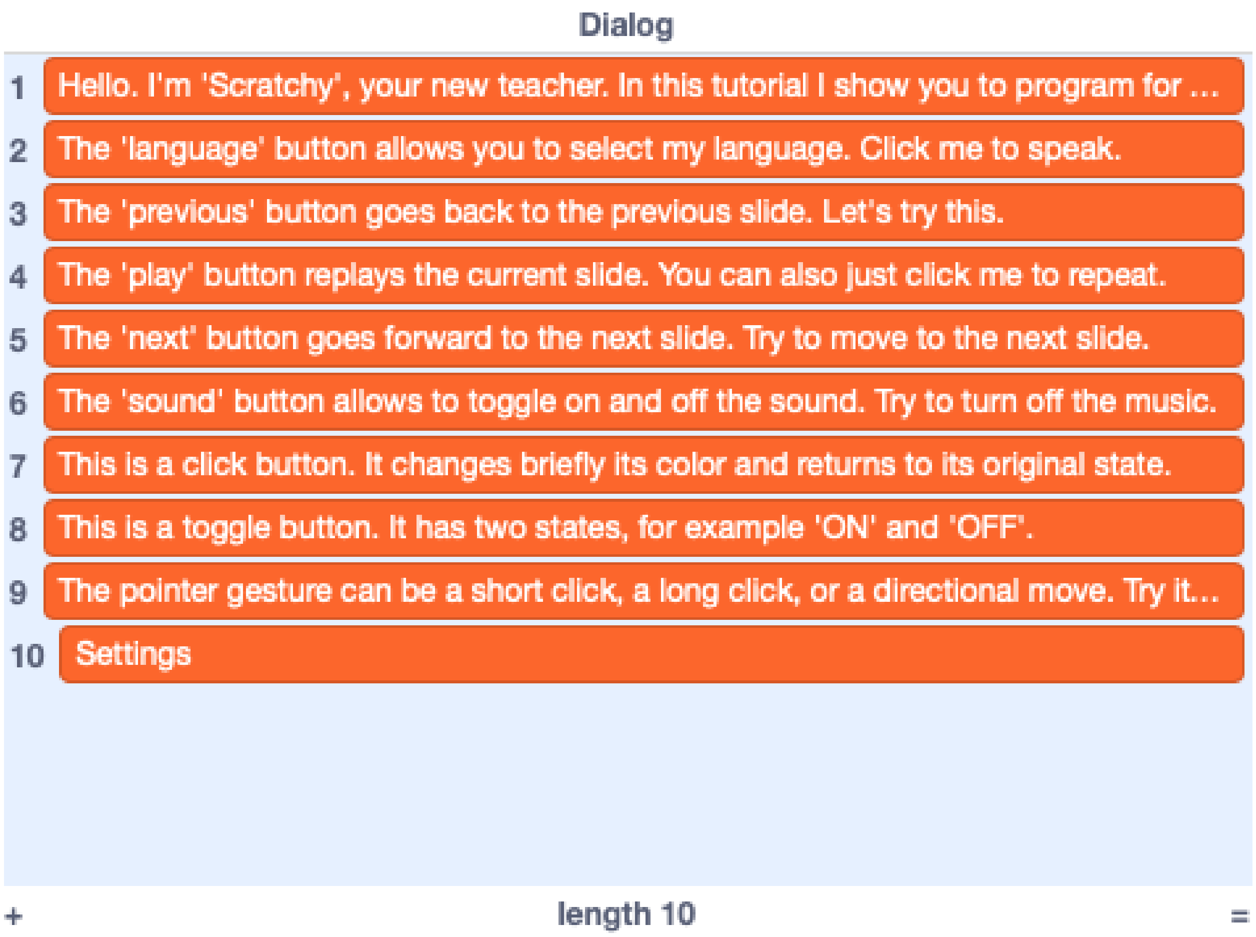
We are going to develop a short, tutorial-type program. This program has 10 slides. Each slide has a title and most of the slides have icons. Scratchy is the narrator and displays the text. At the bottom there is 5-button navigation menu.


Program for a tablet
Hello. I'm 'Scratchy', your new teacher. In this tutorial I show you to program for a tablet.

14.2 Dialog

There is one line of dialog per slide and it is kept in the list **Dialog**.

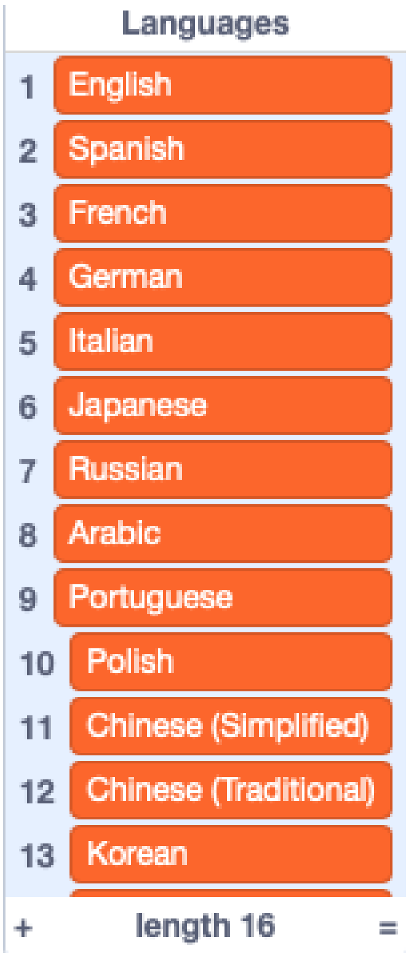
These are the 10 lines of text (or dialog).


Dialog

1 Hello. I'm 'Scratchy', your new teacher. In this tutorial I show you to program for ...
2 The 'language' button allows you to select my language. Click me to speak.
3 The 'previous' button goes back to the previous slide. Let's try this.
4 The 'play' button replays the current slide. You can also just click me to repeat.
5 The 'next' button goes forward to the next slide. Try to move to the next slide.
6 The 'sound' button allows to toggle on and off the sound. Try to turn off the music.
7 This is a click button. It changes briefly its color and returns to its original state.
8 This is a toggle button. It has two states, for example 'ON' and 'OFF'.
9 The pointer gesture can be a short click, a long click, or a directional move. Try it...
10 Settings

14.3 Languages

In our program you can choose among 16 different languages. These languages are kept in the list **Languages**.

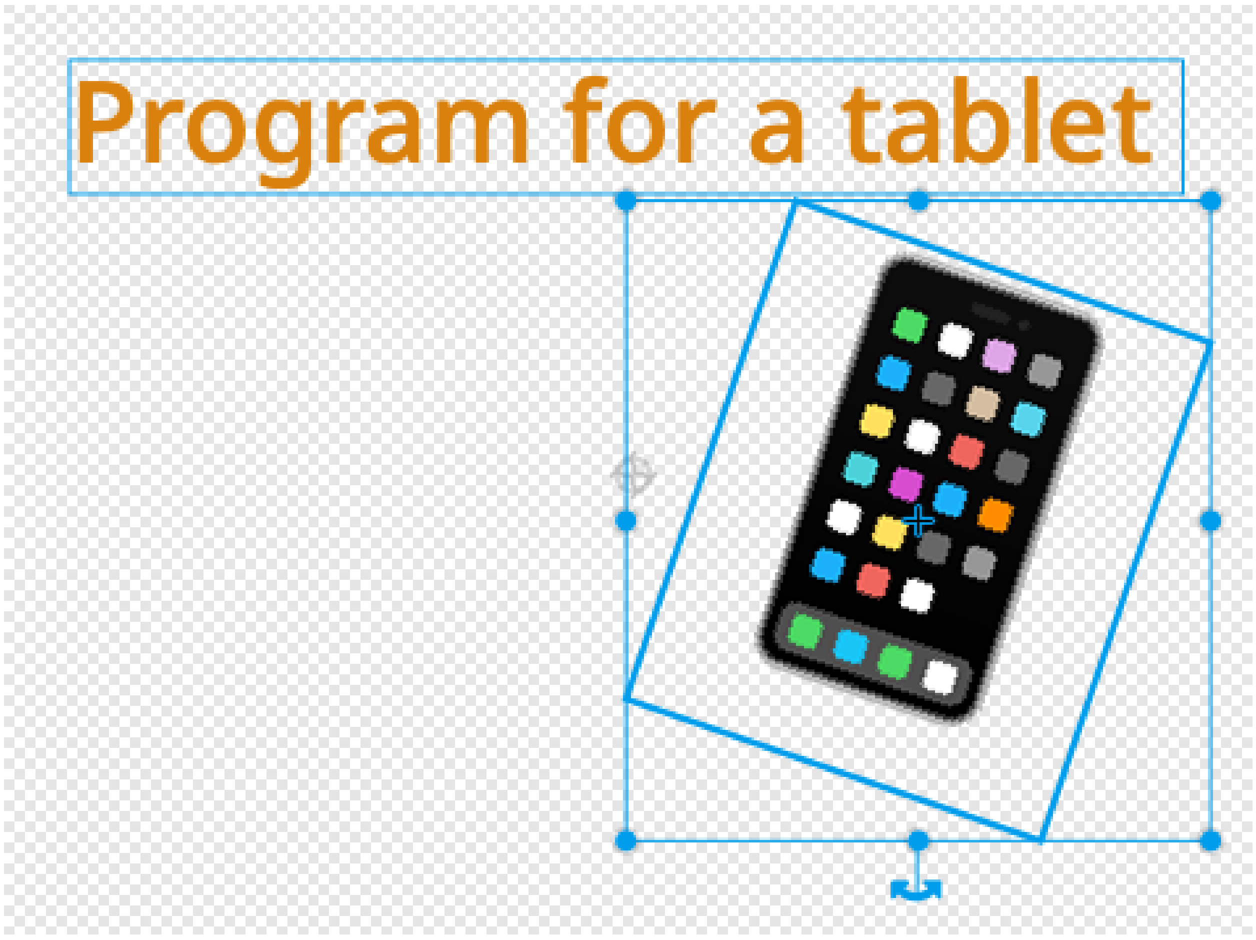


Languages

1	English
2	Spanish
3	French
4	German
5	Italian
6	Japanese
7	Russian
8	Arabic
9	Portuguese
10	Polish
11	Chinese (Simplified)
12	Chinese (Traditional)
13	Korean

14.4 Backdrops

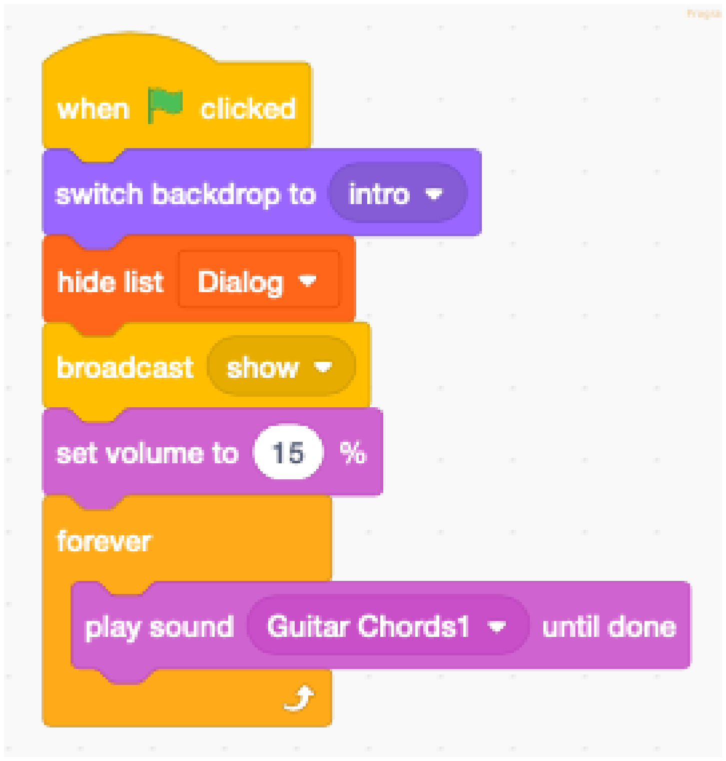
The backdrops are all vector images. This allows to easily change the titles. It also gives perfect resolution in full-screen mode. The smartphone image below is an emoji. That is a very useflul method for getting high-resolution icons.



14.5 The stage

At the start we :

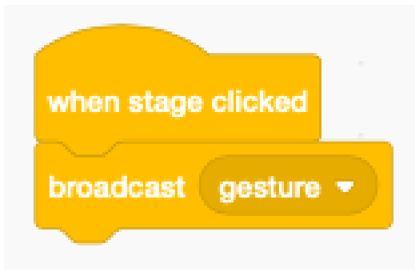
- switch to the title slide (intro)
- hide the Dialog list
- broadcast **show** (this hides all sprites which are not visible on this slide)
- set the volume for the background music to a low value (15%)
- enter the forever loop to play the background music



```
when [flag] clicked
switch backdrop to (intro ▼)
hide list [Dialog ▼]
broadcast (show ▼)
set volume to (15) %
forever
    play sound (Guitar Chords1 ▼) until done
```

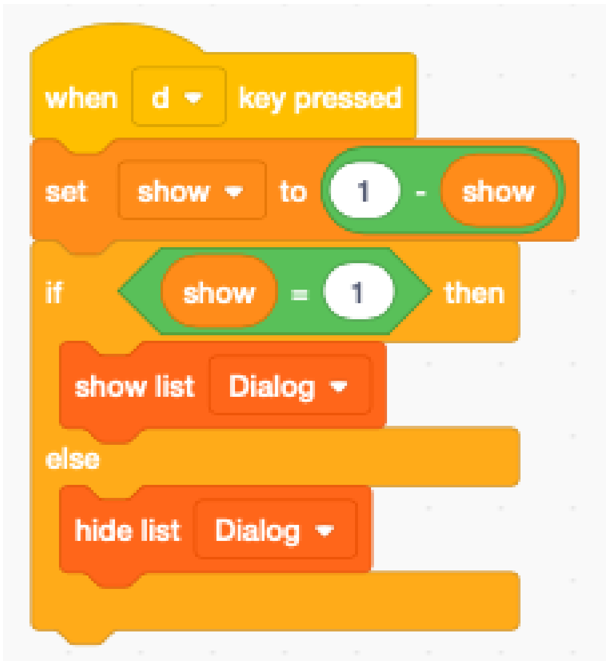
Only the stage receives the **when stage clicked** event. The sprites only have access to their own **when sprite clicked** event.

When the stage is clicked (outside of a sprite), the message **gesture** is broadcast. It will be dealt with in the code of the **Gesture** sprite.



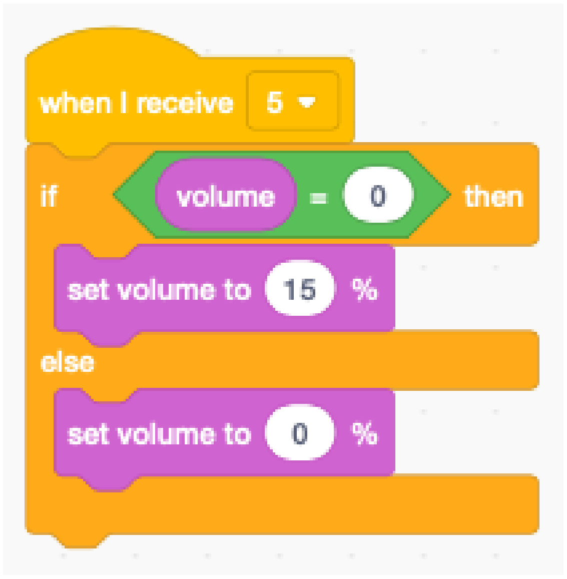
```
when stage clicked
broadcast (gesture ▼)
```

The next function is a helper function to show the dialog. It is only used during development. Scratch can display at the maximum 13 lines of a list. Typically the backdrop editor is open, and you add a line of text for each backdrop, taking care to keep the same numbering.



```
when [d] key pressed
set [show] to ((1) - (show))
if <(show) = (1)> then
    show list [Dialog]
else
    hide list [Dialog]
```

Only the stage can change the volume of its background music. The broadcast message number **5** is used to toggle the music. This corresponds to the button number 5.



```
when I receive [5]
if <(volume) = (0)> then
    set volume to (15) %
else
    set volume to (0) %
```

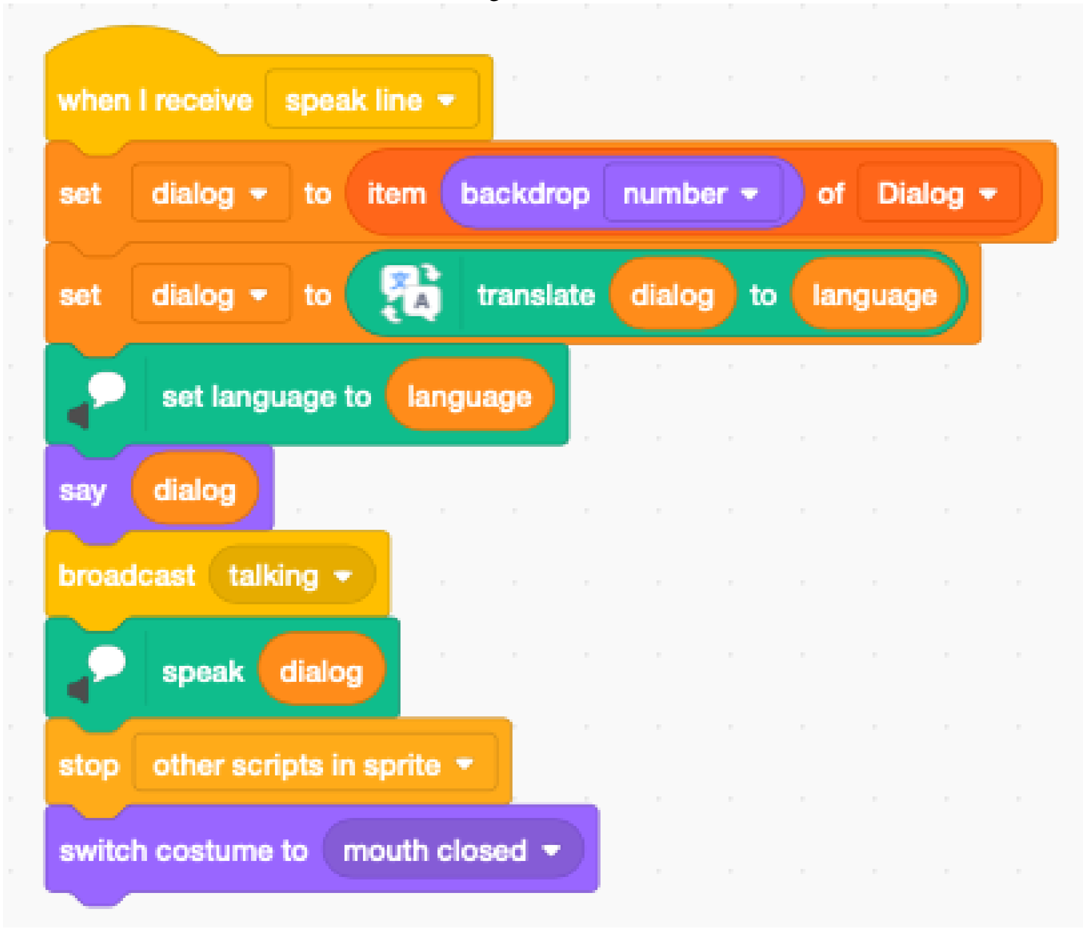
14.6 Scratchy the narrator

Scratchy is our narrator. The speech bubbles are the only way to display text under program control.

When the sprite Scratchy receives a **speak line** broadcast it :

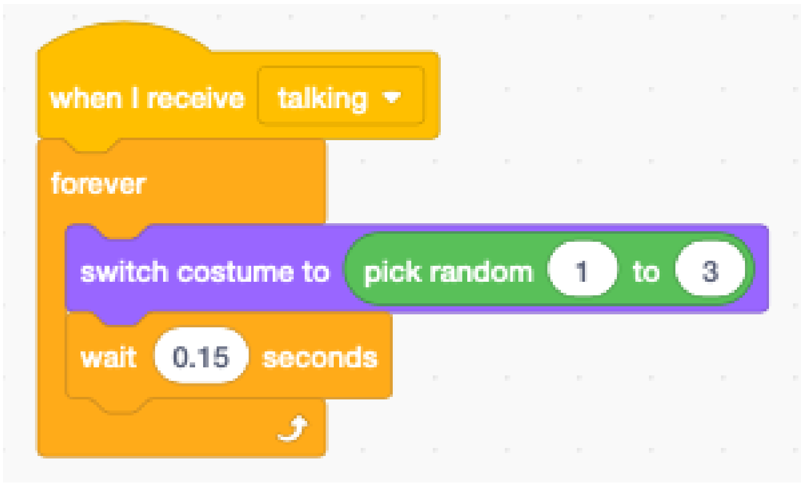
- — takes the line of dialog which correspond to the current backdrop number
- — translates this to the chosen target language
- — displays (say) the dialog in a speech bubble
- — broadcast **talking** (to start moving the mouth)
- — speak the dialog in the selected language

— stop the **talking** script
— close the mouth at the end of the dialog



```
when I receive [speak line ▾]
set [dialog ▾] to (item (backdrop [number ▾]) of [Dialog ▾])
set [dialog ▾] to (translate (dialog) to (language))
set language to (language)
say (dialog)
broadcast [talking ▾]
speak (dialog)
stop [other scripts in sprite ▾]
switch costume to [mouth closed ▾]
```

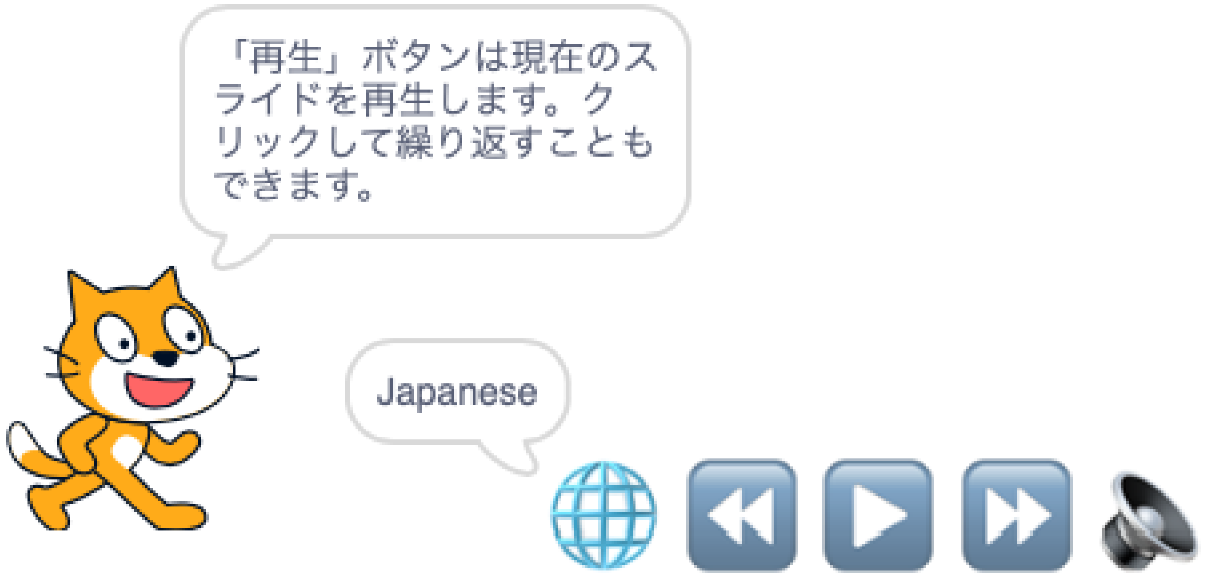
The mouth movement needs to be done in a separate loop.



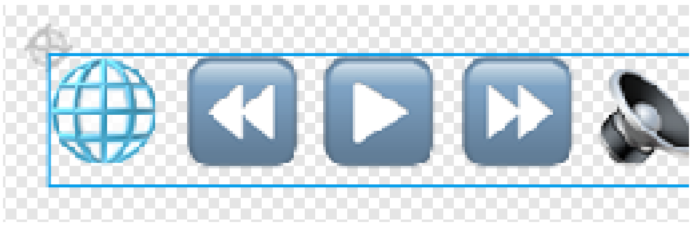
```
when I receive [talking ▾]
forever
    switch costume to (pick random (1) to (3))
    wait (0.15) seconds
```

14.7 The icon menu

Sprites are clickable. It would be possible to have a separate sprite for each menu button. Here we use a different approach. We have one single sprite which is made up of a string with 5 emoji characters.


「再生」ボタンは現在のスライドを再生します。クリックして繰り返すこともできます。
Japanese

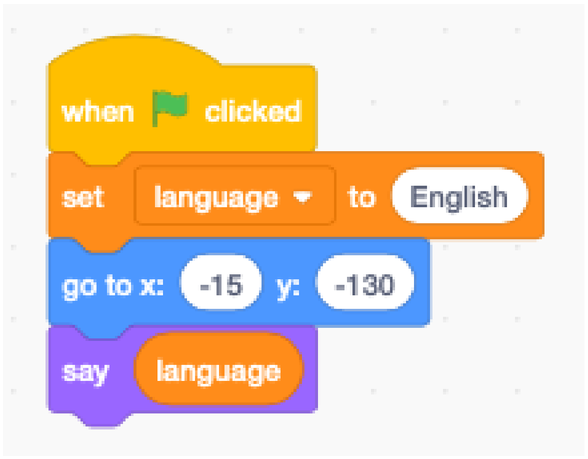
We ajust the total size of the string to 250 points, so that each emoji has a width of 50 points. The origin is also important. The origin marker is placed in the upper left corner.



At the start of the progam we

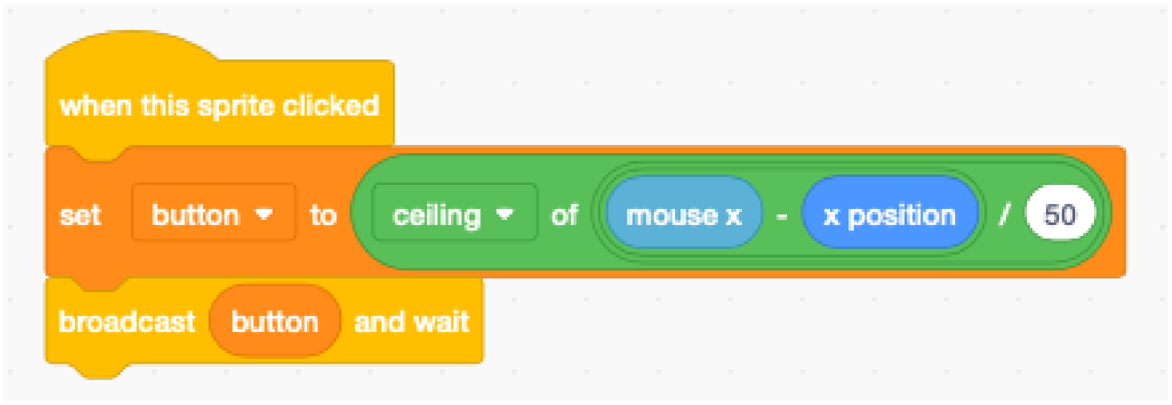
- set the language to English
- place the sprites
- display *English* in the speech bubble

Displaying the language next to the globe, makes its meaning clearer to the user.



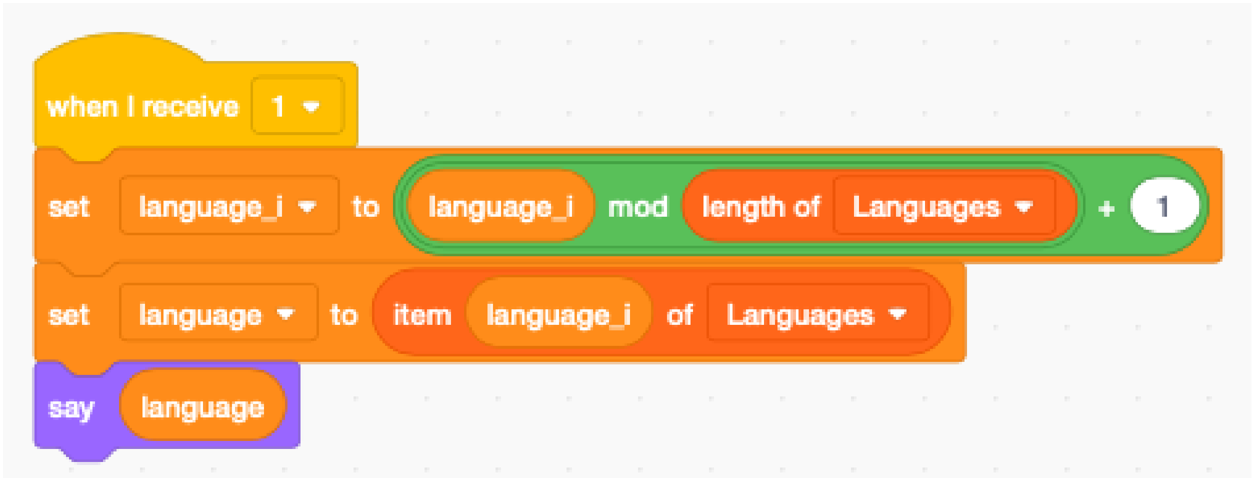
```
when flag clicked
set language to English
go to x: -15 y: -130
say language
```

When this sprite is clicked, we
— calculate the mouse position from the x-origin of the sprite
— divide by the button width (50)
— round up (ceiling)
This will give us a number from 1 to 5 which corresponds to the button clicked. We broadcast directly this number.



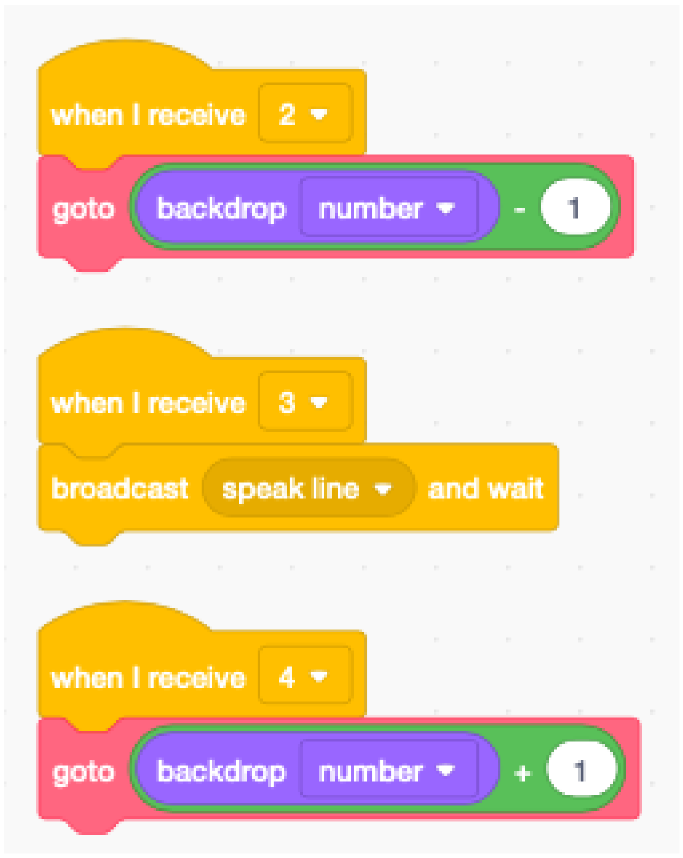
```
when this sprite clicked
set button to ceiling of mouse x - x position / 50
broadcast button and wait
```

Button 1 (globe) increments the language index. We have to keep the index (language_i) in a variable.



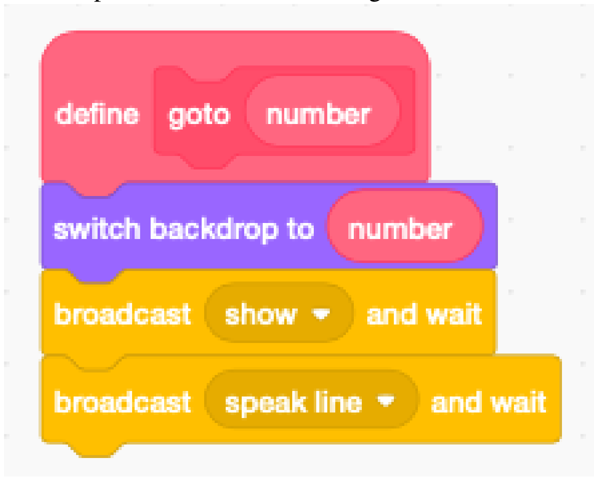
```
when I receive 1
set language_i to language_i mod length of Languages + 1
set language to item language_i of Languages
say language
```

Button 2, 3 and 4 are navigation buttons.



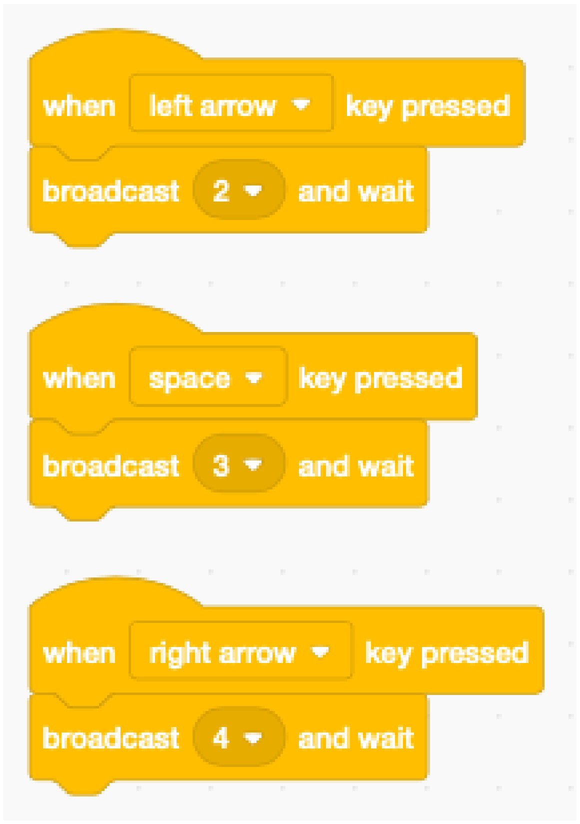
```
when I receive 2
goto (backdrop number) - 1

when I receive 3
broadcast speak line and wait

when I receive 4
goto (backdrop number) + 1
```

This is the function to go to a new slide
- — switch the backdrop
- — show/hide the sprites
- — speak the new line of dialog



```
define goto number
switch backdrop to number
broadcast show and wait
broadcast speak line and wait
```

The last 3 scripts are a convenience when using a keyboard :
- — left arrow : previous slide
- — space key : replay current
- — right arrow : next slide

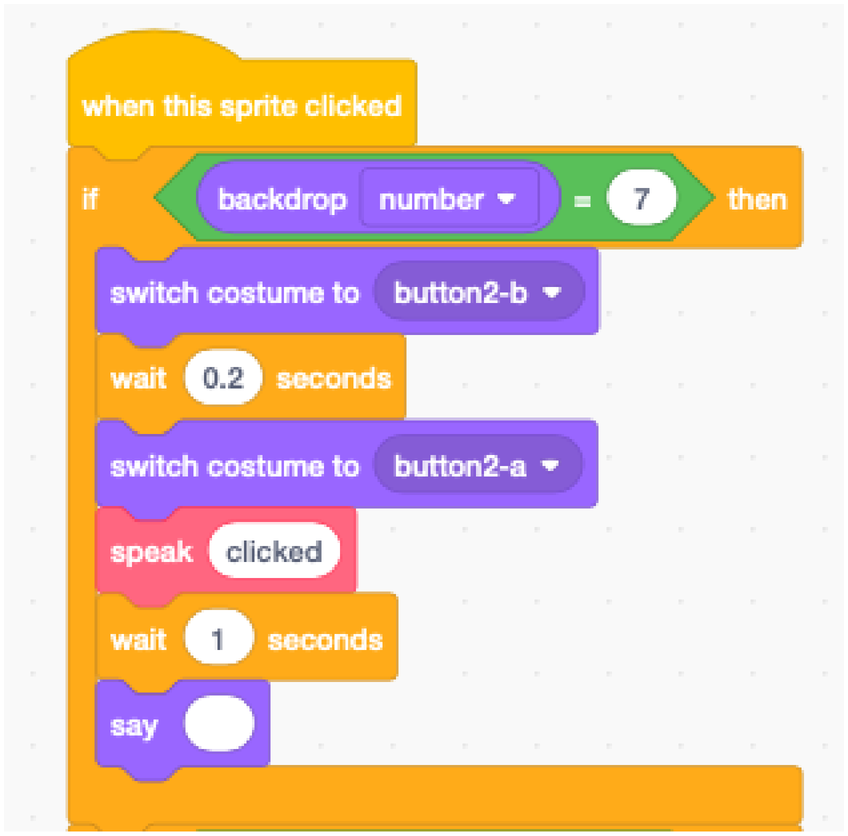


```
when left arrow ▾ key pressed
broadcast 2 ▾ and wait

when space ▾ key pressed
broadcast 3 ▾ and wait

when right arrow ▾ key pressed
broadcast 4 ▾ and wait
```

14.8 A click button

This example button is only shown in slide 7 and 8. In slide 7 it works as a click button. It :

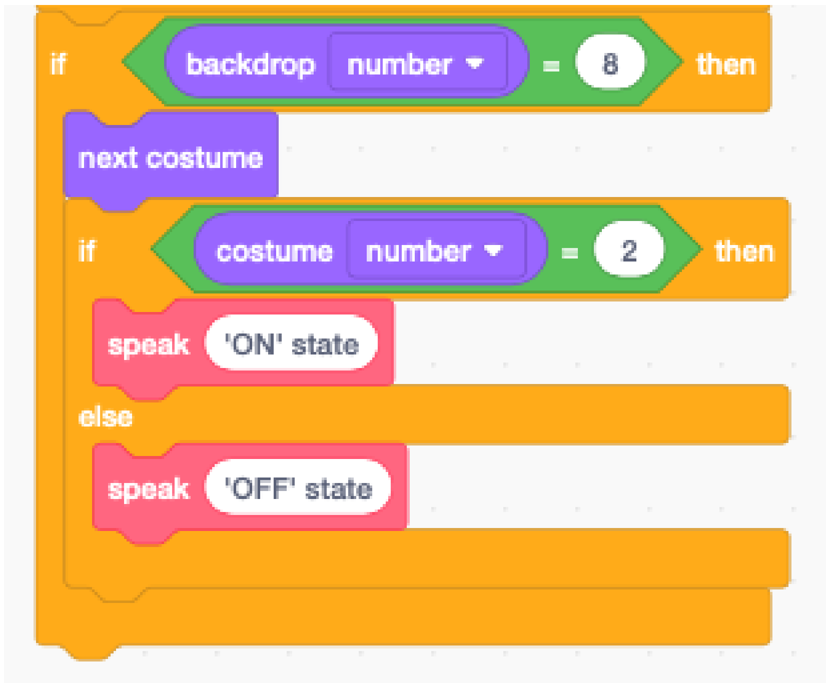
— switches to the second costume (orange)
— waits 0.2 seconds (a brief flash)
— switches back to the first costume (blue)
— it prononces *clicked*



```
when this sprite clicked
if <(backdrop number) = 7> then
    switch costume to button2-b
    wait 0.2 seconds
    switch costume to button2-a
    speak clicked
    wait 1 seconds
    say
```

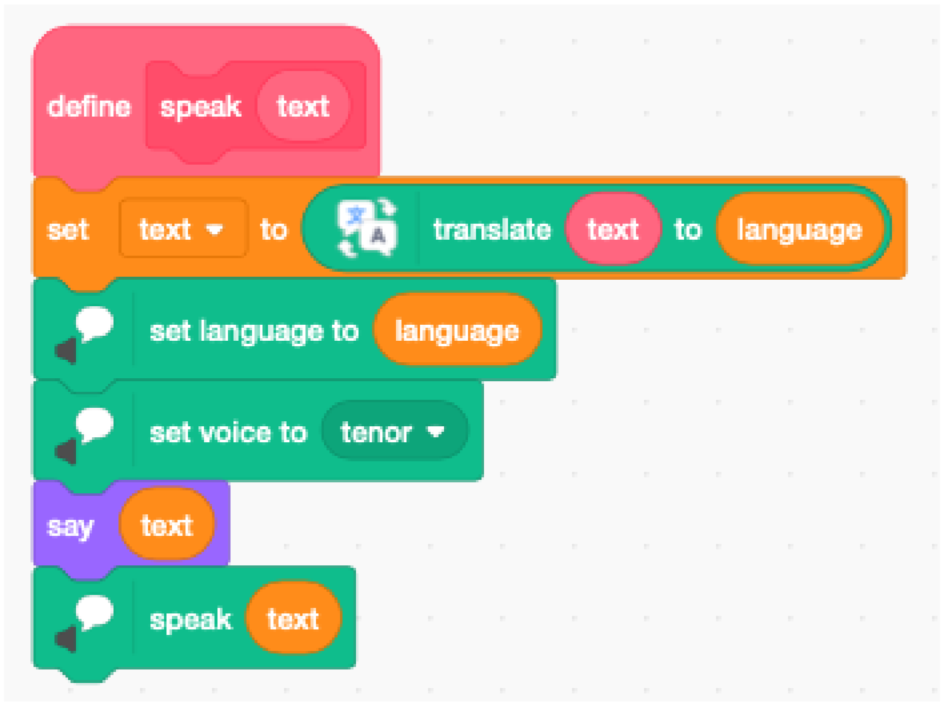
14.9 A toggle button

In slide 8 the same button behaves as a toggle button with two states. It toggles between costume 1 and 2. The costume number is used as the state variable.



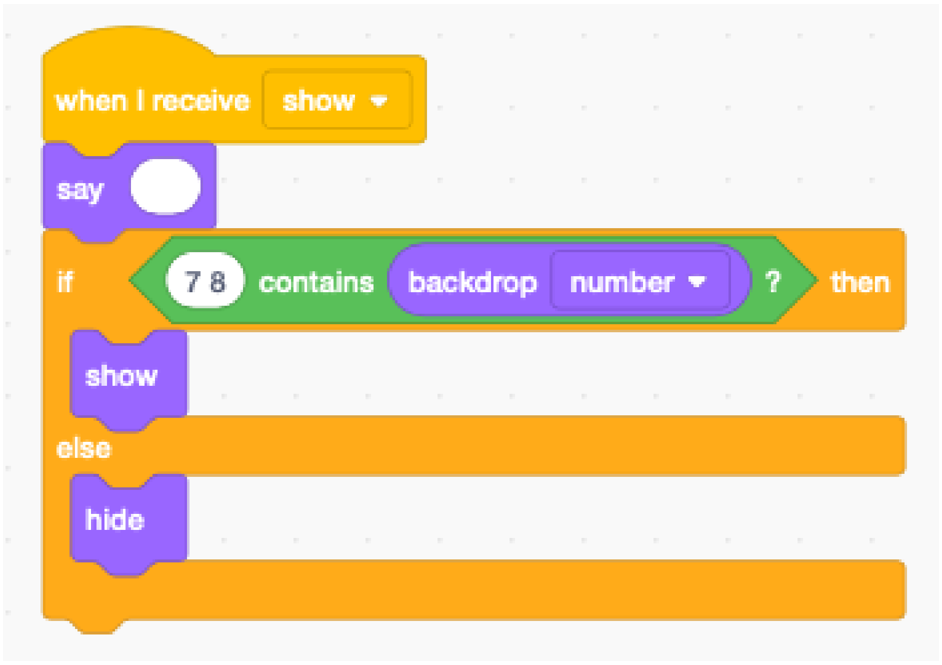
```
if <(backdrop number) = 8> then
    next costume
    if <(costume number) = 2> then
        speak 'ON' state
    else
        speak 'OFF' state
```

The function **speak** pronounces the *text* argument in the selected language. To make it different from Scratchy we set the voice to *tenor*.



```
define speak text
set text to (translate (text) to (language))
set language to (language)
set voice to tenor
say (text)
speak (text)
```

Finally the **show** broadcast, hides the speach bubble, and then shows the button sprite only if we are in background number 7 or 8.

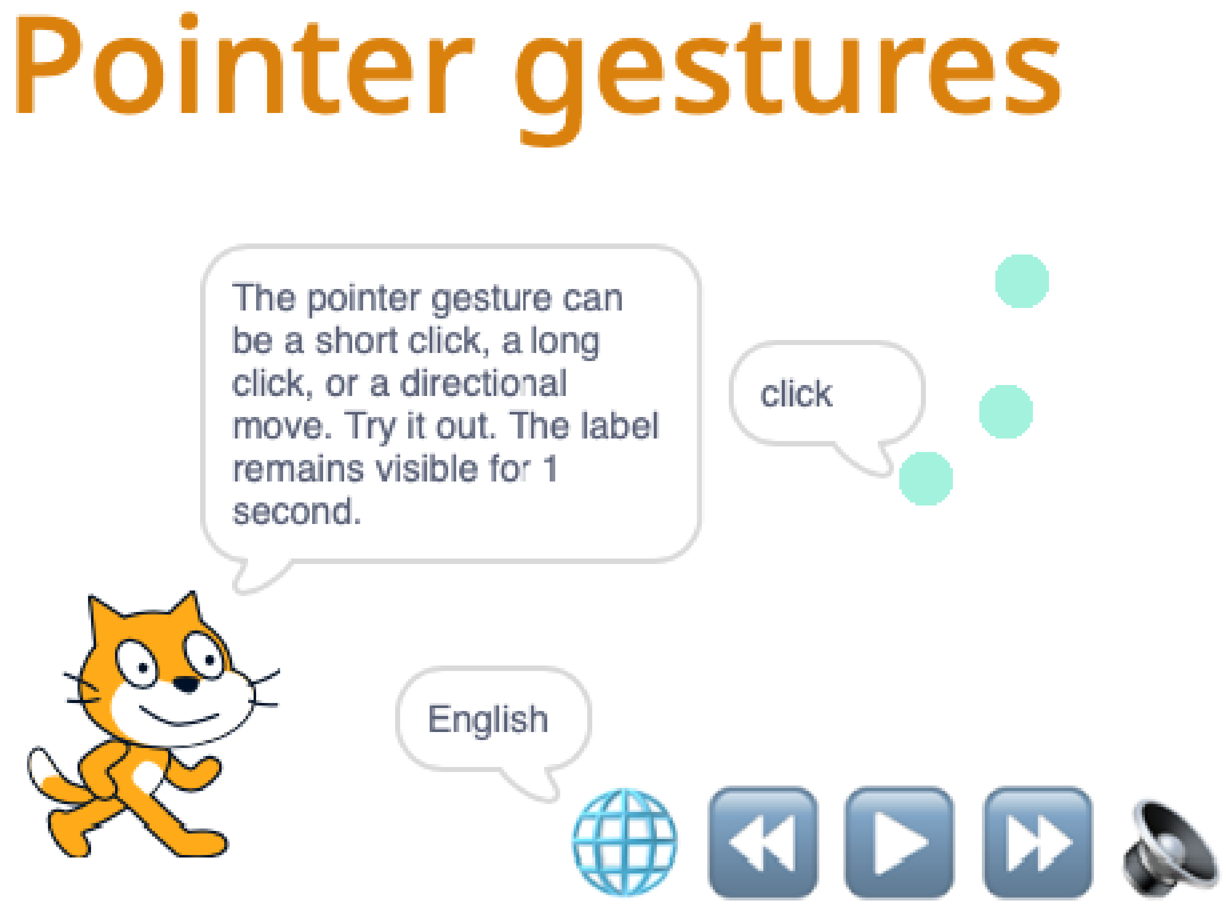


```
when I receive show
say
if 7 8 contains backdrop number ? then
    show
else
    hide
```

14.10 Pointer gestures

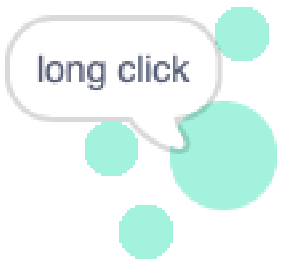
On tablets, Scratch does not provide a soft-keyboard. To use one, we would need to program it in Scratch. However mouse clicks and gestures are the natural way of interacting with a tablet.

To give the user some feedback, we visulize the result of the gesture. On the following image we see three simple clicks. An explanation of the gesture is visible for 1 second.

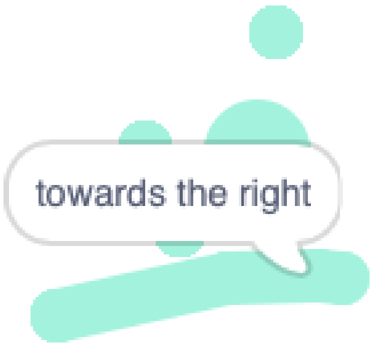

Pointer gestures

The pointer gesture can be a short click, a long click, or a directional move. Try it out. The label remains visible for 1 second.

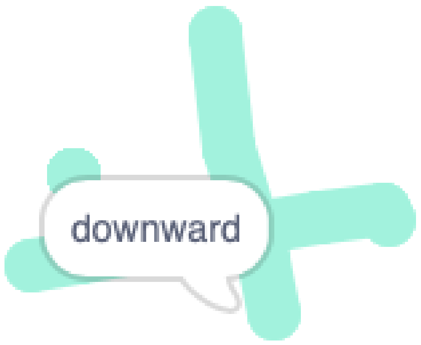
If we press longer than 0.3 seconds, the click is detected as a **long click**. In the image below we have 3 normal clicks and one long click.



This is a gesture towards the right. It can be used to go to the next page.

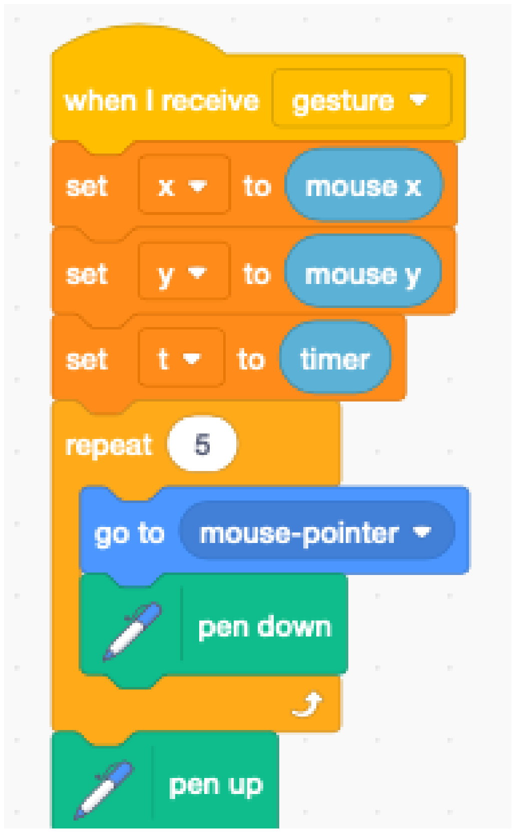


This is a gesture towards the bottom.



14.11 The gesture algorithm

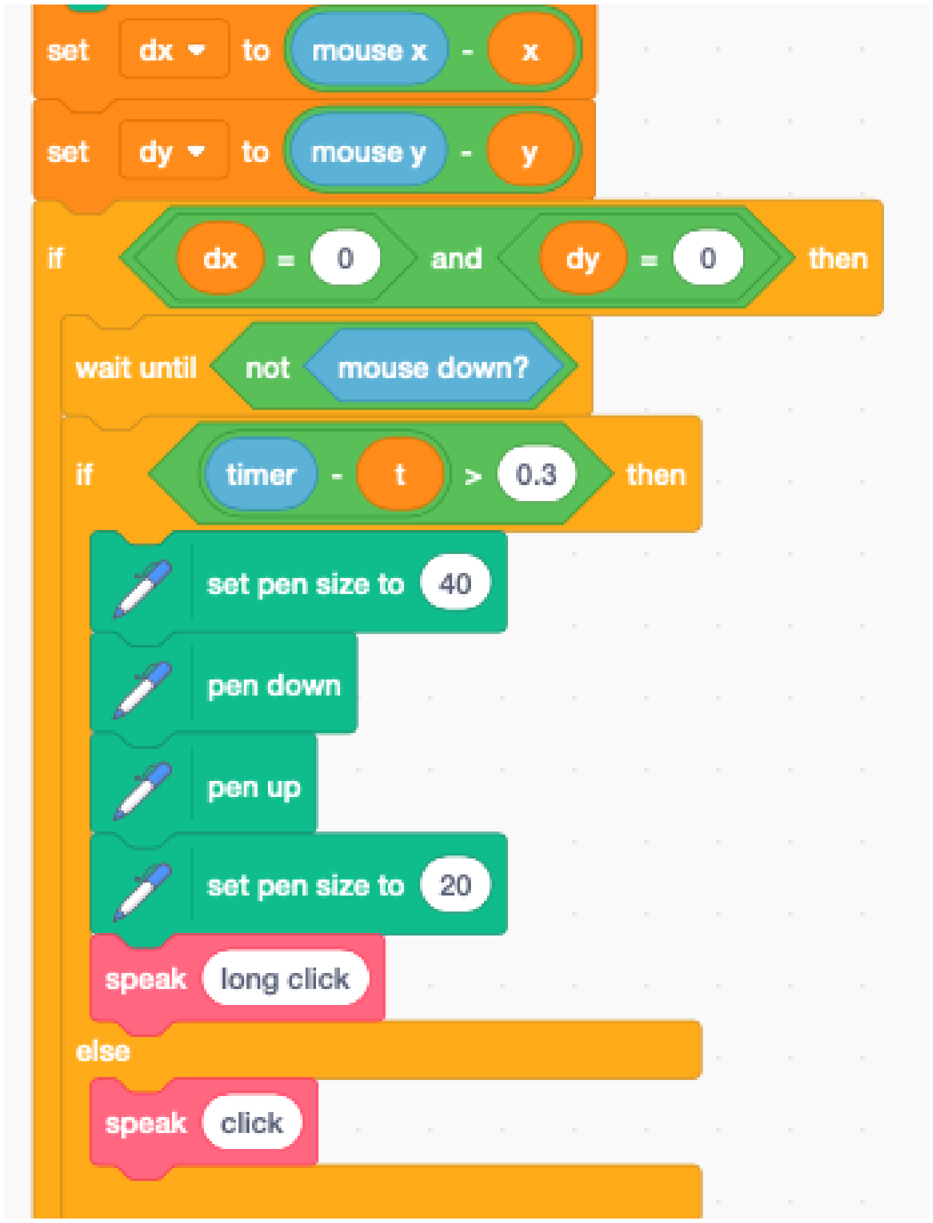
At the start we record the initial mouse position (x, y) and the time t. The the **Gesture** sprite goes for 5 frames to the mouse position with the pen down.



```
when I receive gesture
set x to mouse x
set y to mouse y
set t to timer
repeat 5
    go to mouse-pointer
    pen down
pen up
```

At the end of the gesture, we calculate the displacement vector (dx, dy). If the displacement vector is zero, we have a click. We then wait until the mouse is up and look at the elapsed time.

If it's more than 0.3 secondes we have a long click and we draw a bigger circle, to make it visually clear to the user.



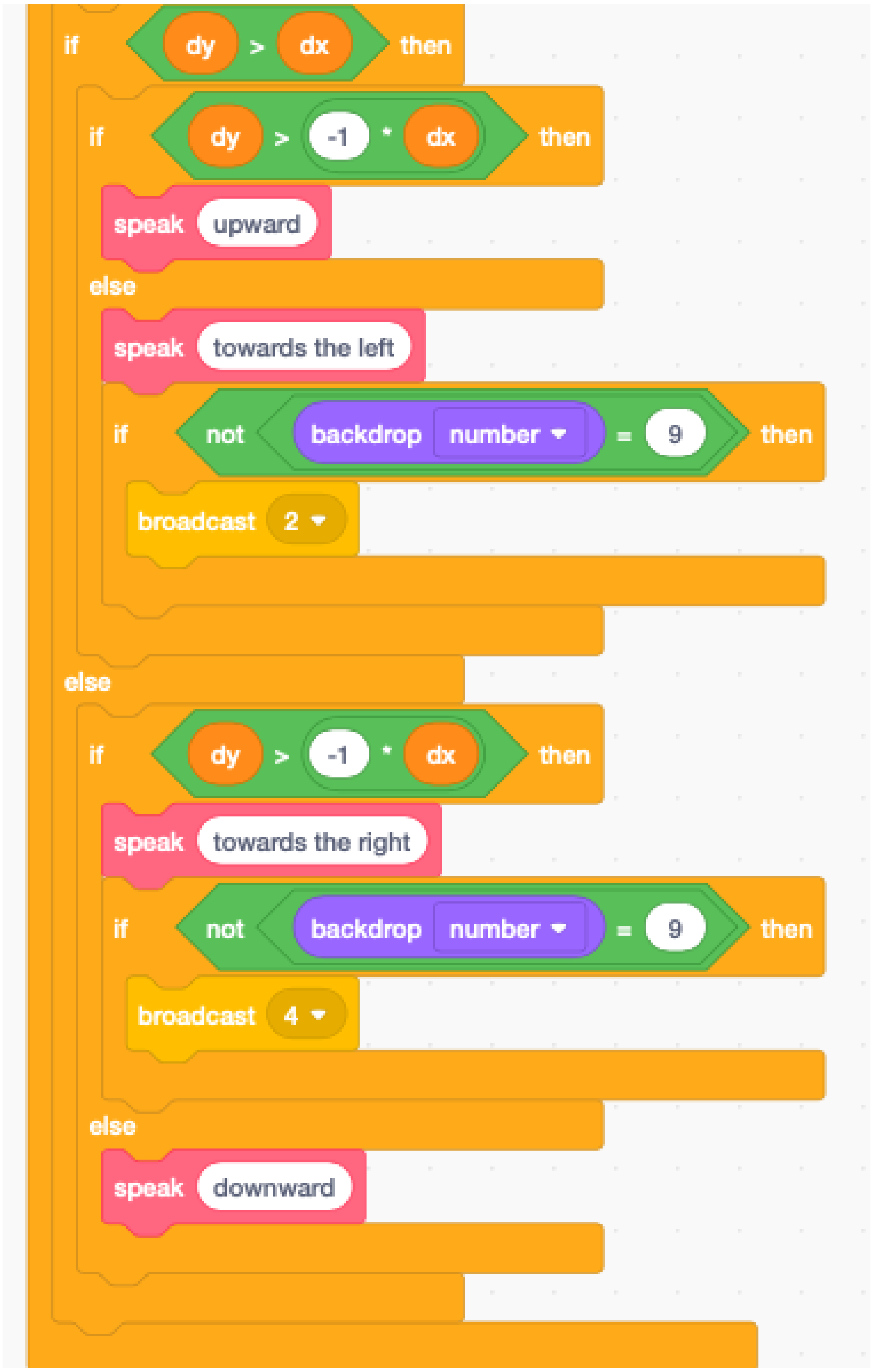
```
set dx to (mouse x - x)
set dy to (mouse y - y)
if <(dx = 0) and (dy = 0)> then
    wait until <not <mouse down?>>
    if <(timer - t) > 0.3> then
        set pen size to 40
        pen down
        pen up
        set pen size to 20
        speak long click
    else
        speak click
```

If the displacement value is not zero, we determine in which direction it is. The line dx=dy is the diagonal which goes up from left to right. The line dx=-dy is the diagonal which goes down from left to right.

Using these conditions, we obtain the 4 cases

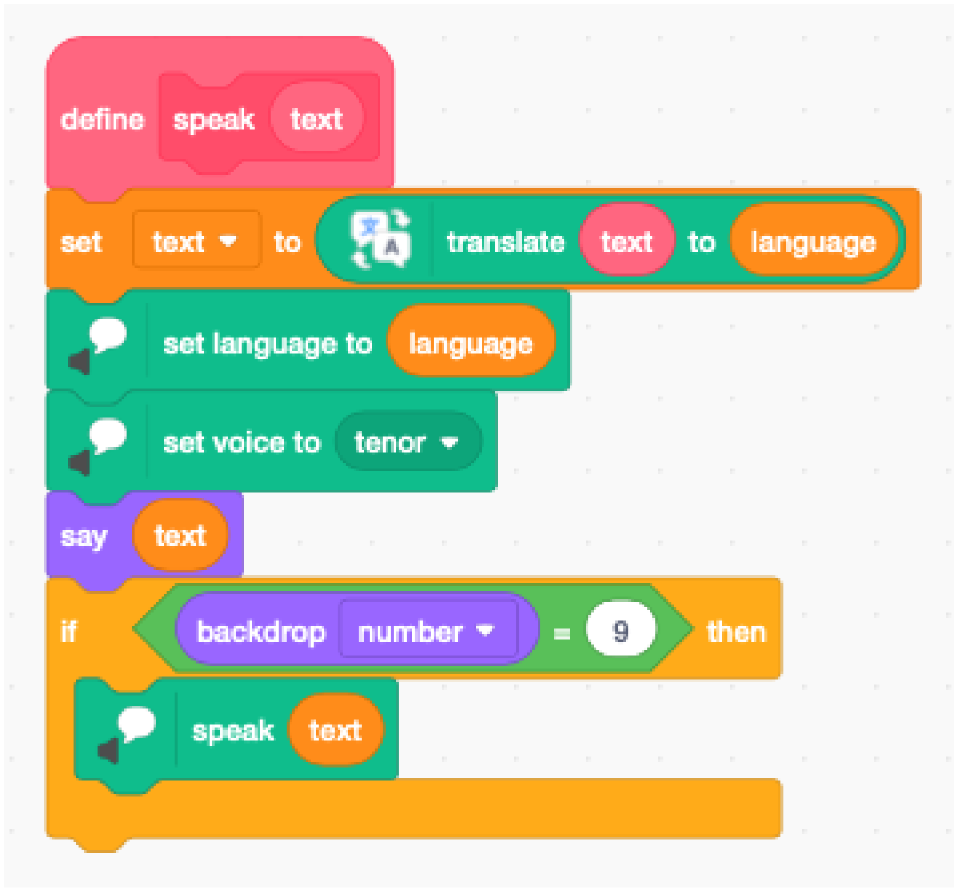
- towards the left
- towards the right
- upwards
- downwards

Because of the automatic translation we make the words in English as specific as possible. That is the reason of not just using left/right or up/down.



```
if dy > dx then
    if dy > -1 * dx then
        speak upward
    else
        speak towards the left
        if not backdrop number = 9 then
            broadcast 2
else
    if dy > -1 * dx then
        speak towards the right
        if not backdrop number = 9 then
            broadcast 4
    else
        speak downward
```

We want to use text as a means of feedback for gestures on all 10 slides. However the speaking voice, we only want it in slide number 9, where we present gestures.

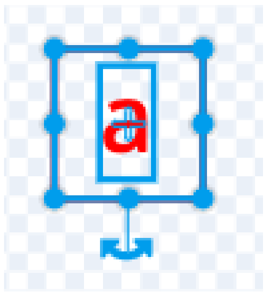


```
define speak (text)
set [text ▾] to (translate (text) to (language))
set language to (language)
set voice to [tenor ▾]
say (text)
if <(backdrop [number ▾]) = (9)> then
    speak (text)
```

14.12 Virtual keyboard

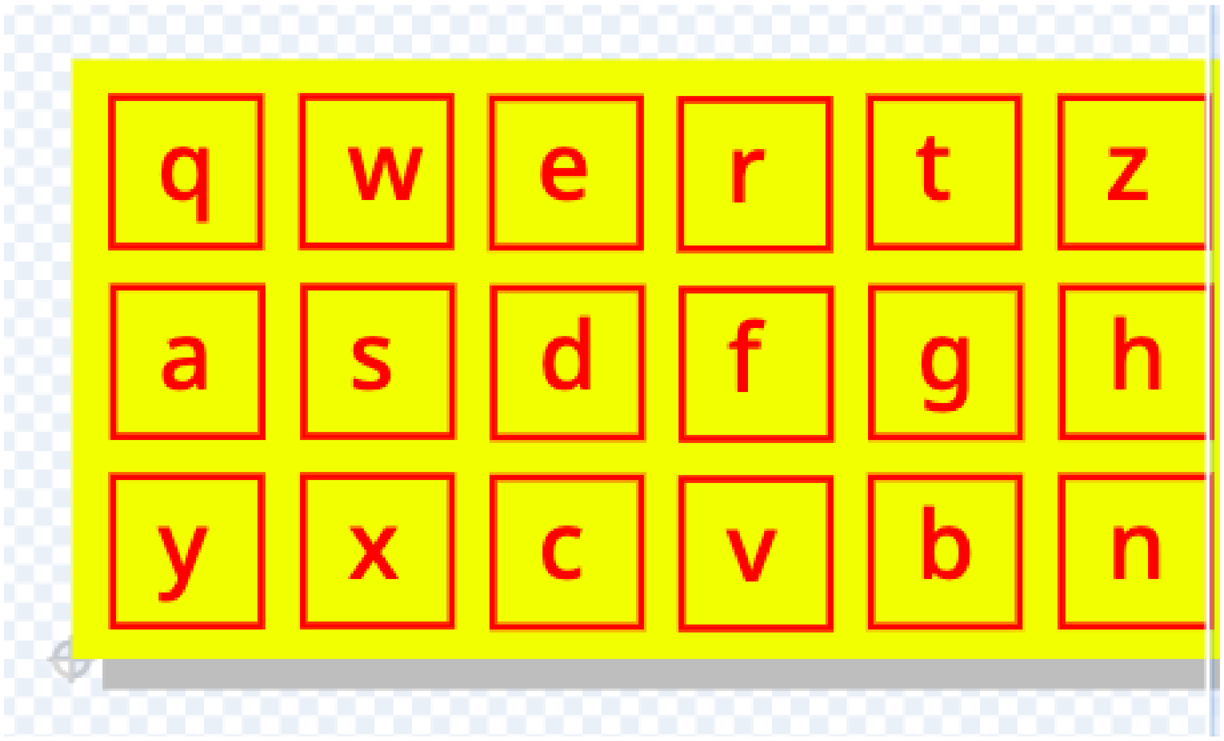
Tablets use a virtual keyboard. With the **ask () and wait** block the virtual keyboard of the tablet OS can be used. To have more control, you can also program a virtual keyboard within Scratch by yourself.

We create the image of a single key by drawing a rectangle with a single letter inside. Make the size about 32 points wide. Select both items and group them.



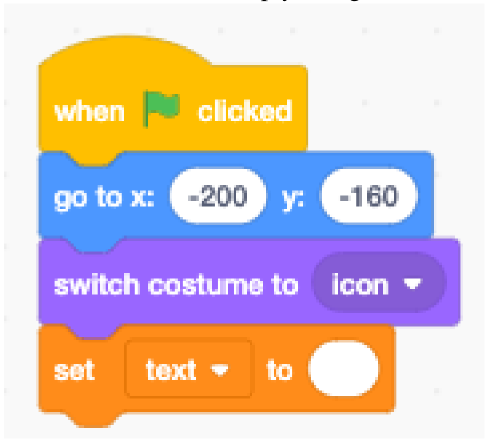
Then copy-paste that key and align the second key with the first one. Use a horizontal (and vertical) spacing of 40 points. Then select the two keys and copy-paste them again and align. Repeat with 4 keys to get 8 keys. Continue this to obtain a keyboard with 3 lines of 10 keys. Then you relabel the keys according the keyboard layout you want to show (qwertz…).

Then place a colored filled rectangle behind the keys. You can add a second grey rectangle to add a shadow. Place the origin of the sprite to the lower left corner.



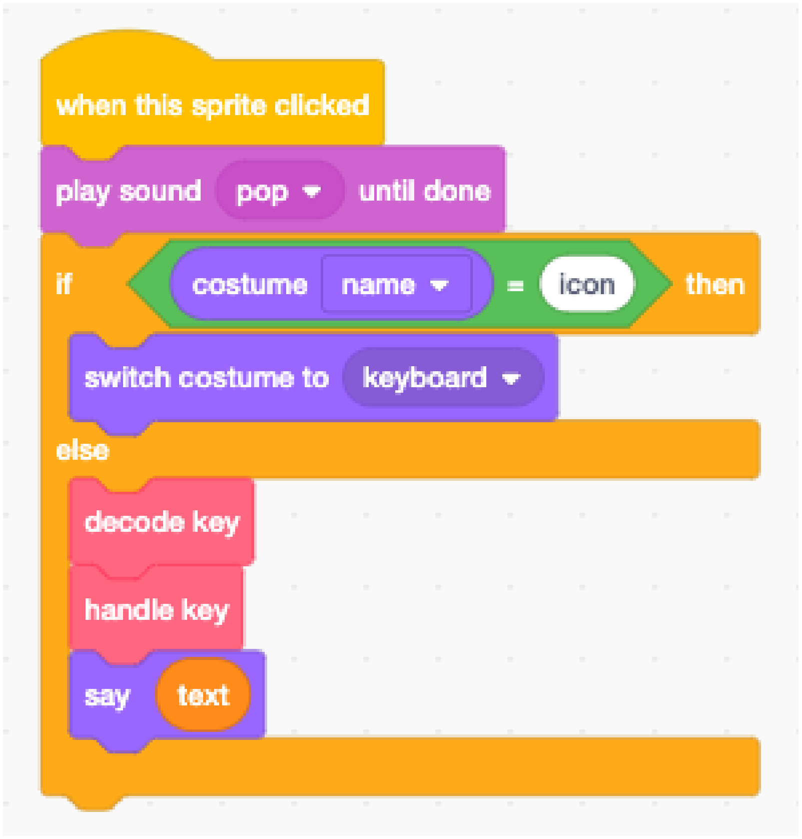
14.13 Decoding the keyboard

At the start we :
- position the sprite to the lower left corner
- switch the costume to **icon** (small keyboard icon abc)
- set **text** to an empty string



```
when flag clicked
go to x: -200 y: -160
switch costume to icon
set text to
```

When the keyboard of the icon is clicked we :
- play a sound
- switch to the costume to **keyboard** if the costume was the icon
- decode the key
- handle the key
- display the new text

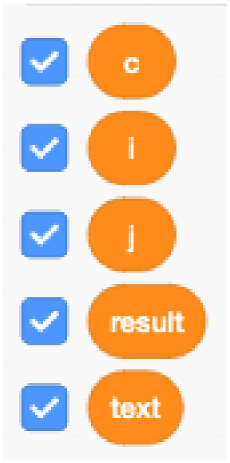


```
when this sprite clicked
play sound (pop ▾) until done
if <(costume (name ▾)) = (icon)> then
    switch costume to (keyboard ▾)
else
    decode key
    handle key
    say (text)
```

14.14 Variables

The code for this sprite needs 5 variables :

— **c** - the current character pressed
— **i** - the horzontal index
— **j** - the vertical index
— **text** - the current text
— **result** - the return value for the *substring** function



14.15 Decode the key

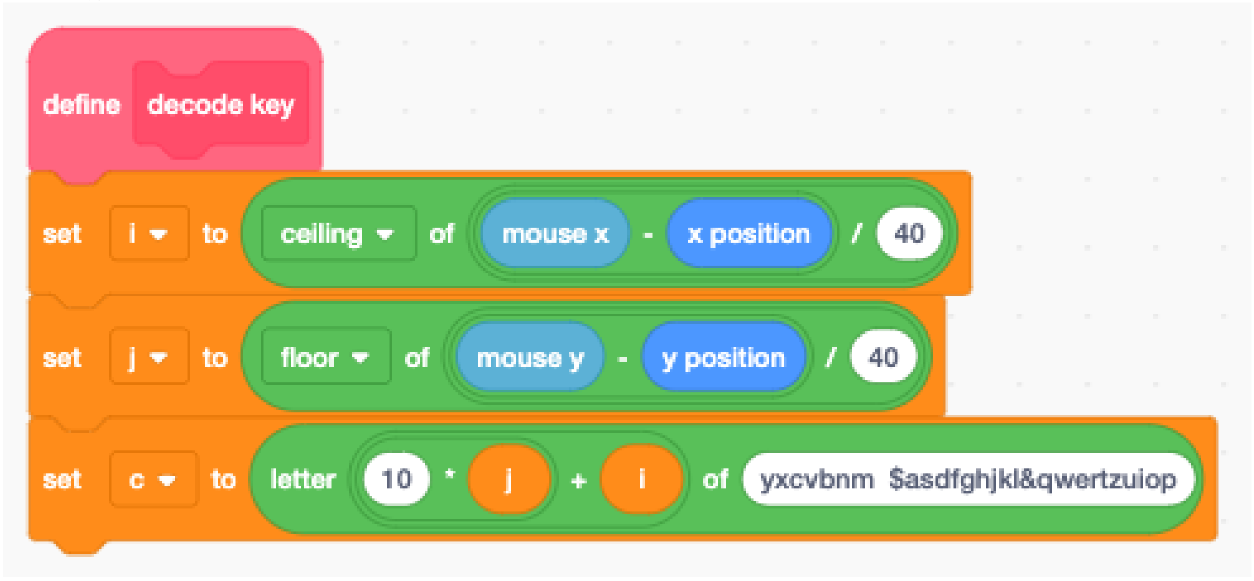
Based on the mouse position, we can obtain the indexes **(i, y)** of the pressed key.

- using **ceiling** the index **i** goes from 1 to 10
- using **floor** the index **j** goes from 0 to 2

Then we combine the two indexes with the expression **(10*j) + i** to get an index from 1 to 30 which we use to look up the character pressed.

We use 2 special characters

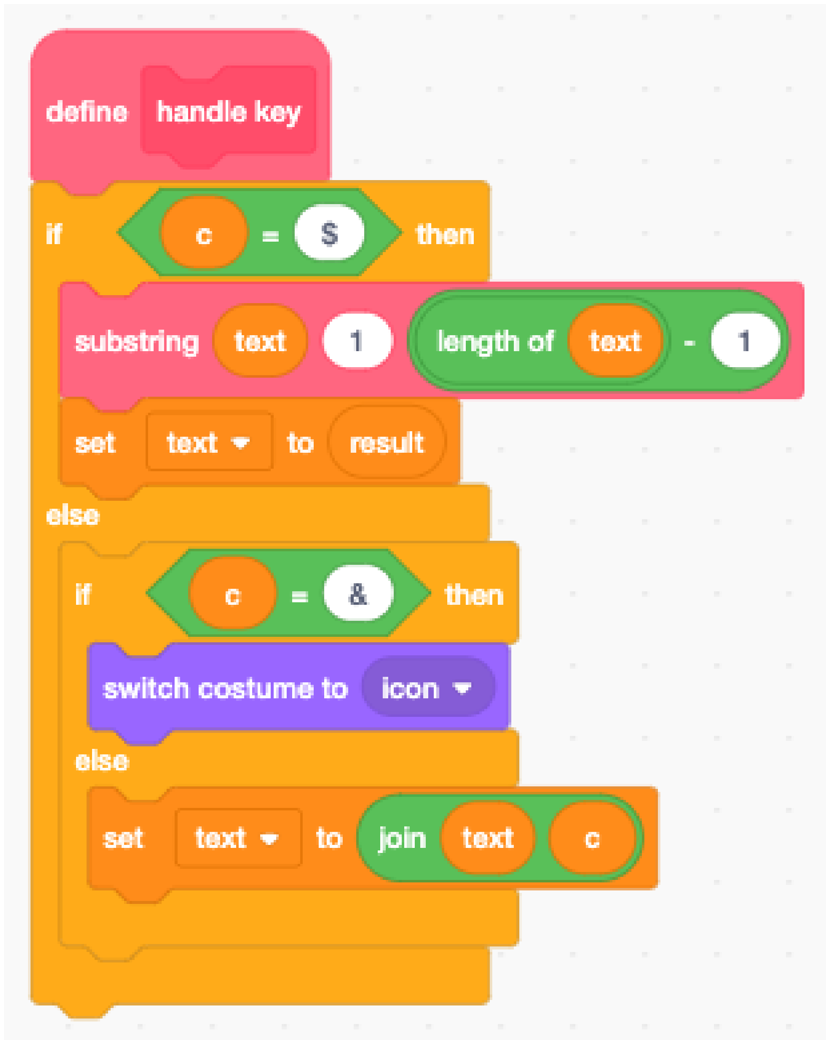
- **$** to indicate *delete*
- **&** to indicate *return*



```
define decode key
set i to ceiling of ((mouse x - x position) / 40)
set j to floor of ((mouse y - y position) / 40)
set c to letter ((10 * j) + i) of yxcvbnm  $asdfghjkl&qwertzuiop
```

When handling the key we look at the current character **c** and

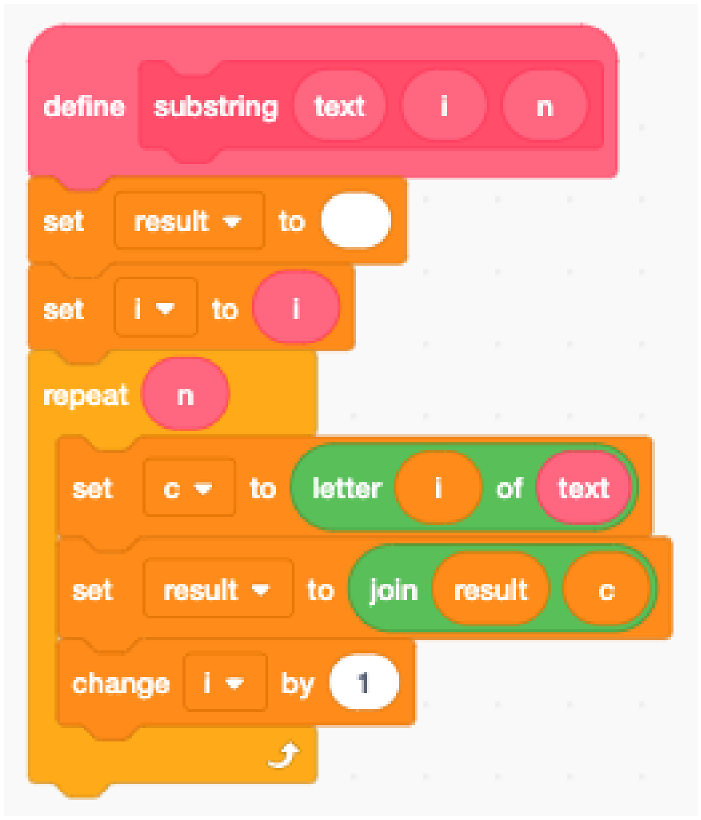
- delete the last character when it is **$**
- return to the icon costume when it is **&**
- append it to **text** otherwise



```
define handle key
if <(c) = $> then
    substring (text) (1) ((length of (text)) - (1))
    set [text ▾] to (result)
else
    if <(c) = &> then
        switch costume to [icon ▾]
    else
        set [text ▾] to (join (text) (c))
```

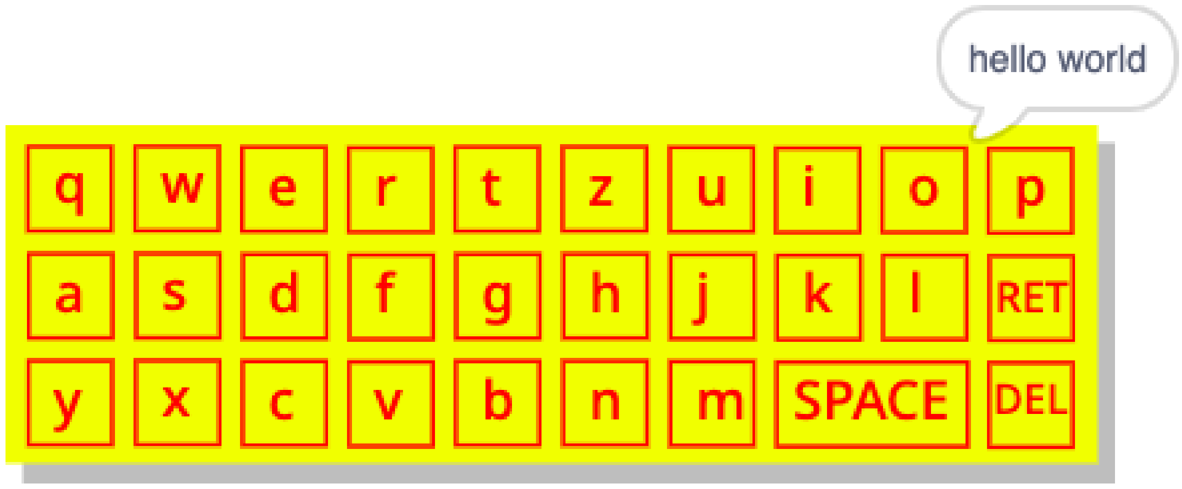
To delete the last character of **texte** we need the **substring** function.

The **substring** function returns a substring of *text* starting at position *i* with a length of *n* and returns it in the variable **result**

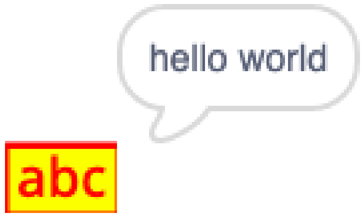


```
define substring (text) (i) (n)
set [result v] to ()
set [i v] to (i)
repeat (n)
    set [c v] to (letter (i) of (text))
    set [result v] to (join (result) (c))
    change [i v] by (1)
end
```

This is the keyboard after entering the text *hello world*.



This is the icon after pressing the RETURN key.



This is the online version :

https://scratch.mit.edu/projects/402881699

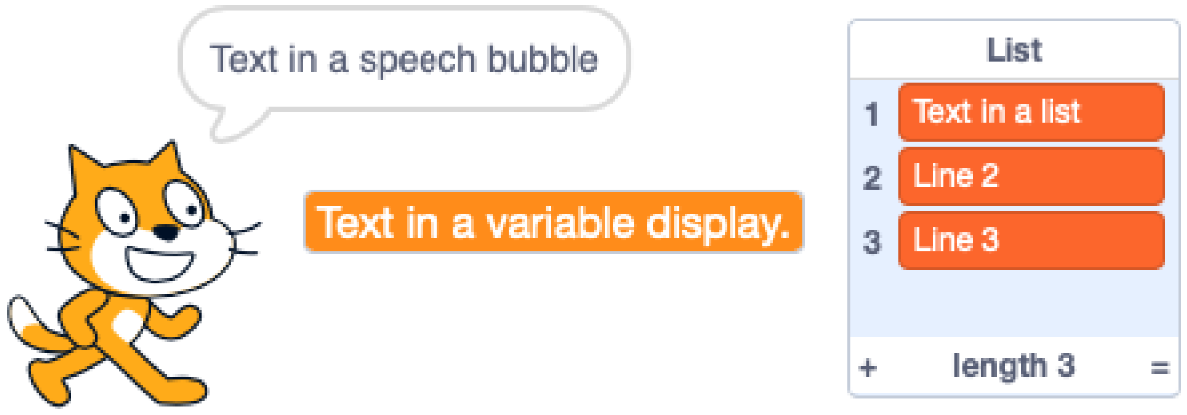
CHAPITRE 15

Text

Scratch cannot place arbitrary text onto the stage. Normally it can only
— handle predefined sprites
— produce arbitrary pen drawings

15.1 Display text

For displaying text in a Scratch app you can use :
— the speech bubbles
— the variable display
— the list display

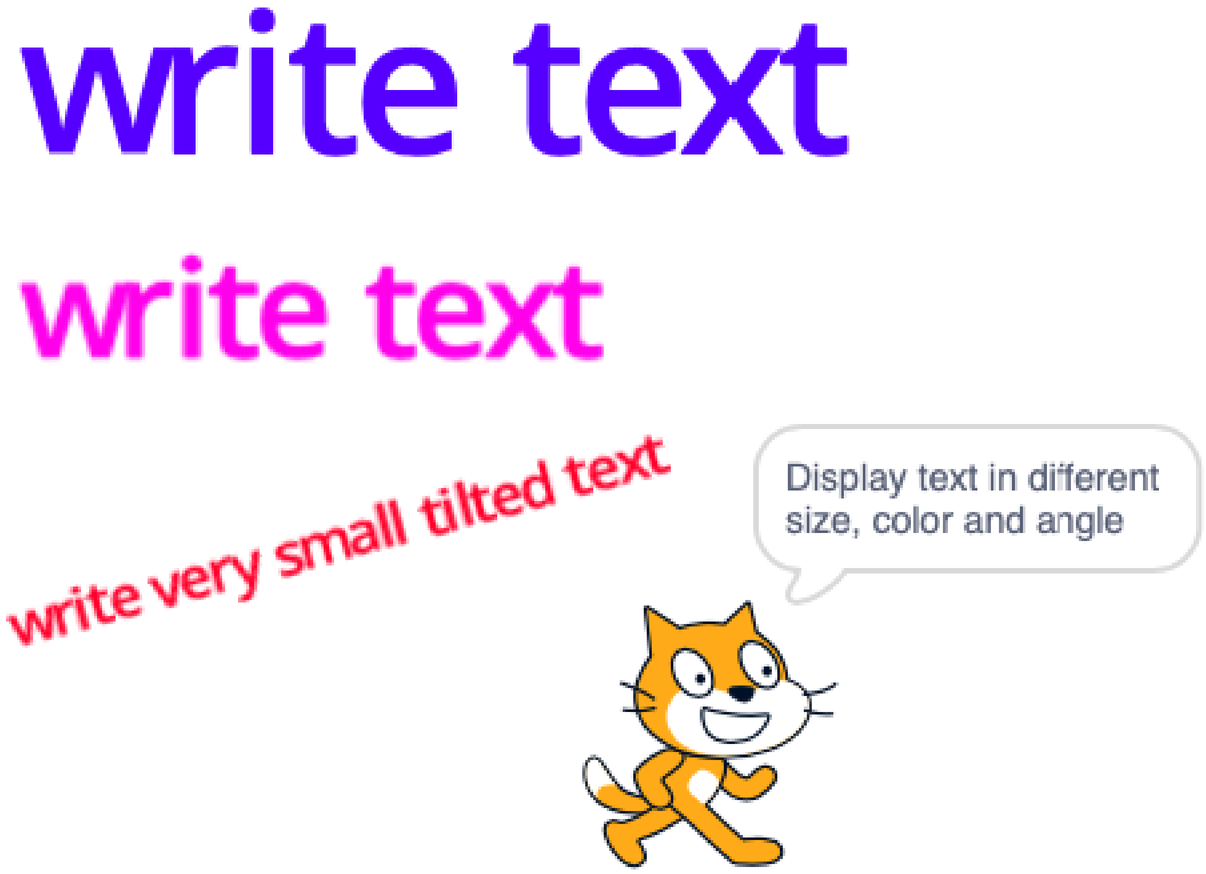


What we are developping in this section, is to define a special sprite which has as costumes the symbols of an alphabet (letters, digits, punctuation). Then we use these **Letter** sprite to create text.

By using sprites as letters, our text can have any kind of
— position
— size

— color
— angle

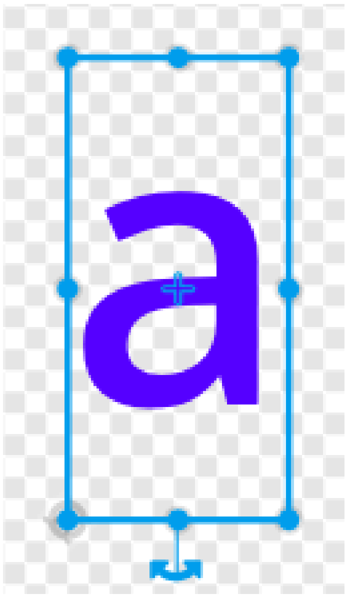
If the text is static we can use **stamps**. If the text needs to be dynamic we use **clones**.


write text
write text
write very small tilted text
Display text in different size, color and angle

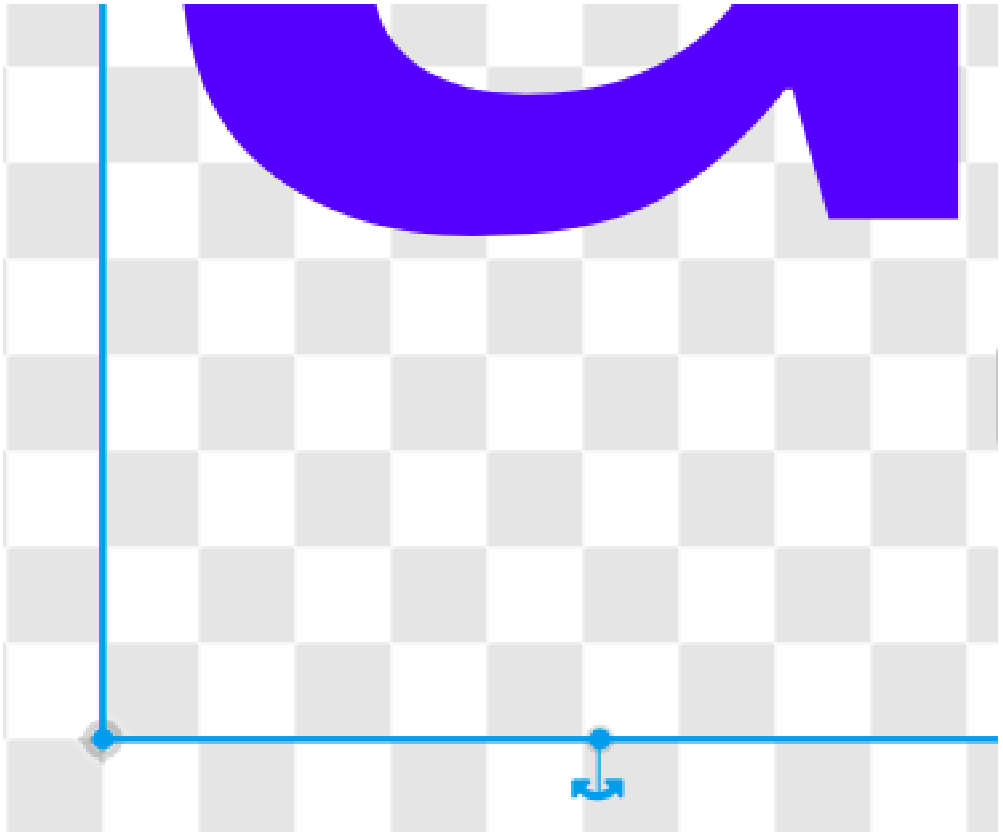
15.2 Define the letters

The first thing is to define a **Letter** sprite. We do it only for the 26 lower-case letters, but we could include upper-case letters, digits, punctuation and other special characters.

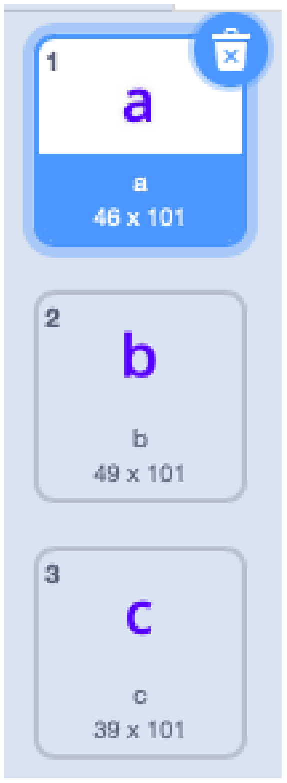
This is the letter **a**. It is a vector drawing with the letter a. Care has been taken to create a letter with a hight of exactly 101 points.



The origin of the sprite needs to be placed in the lower left corner of the letter box. This is best done at the highest zoom level.

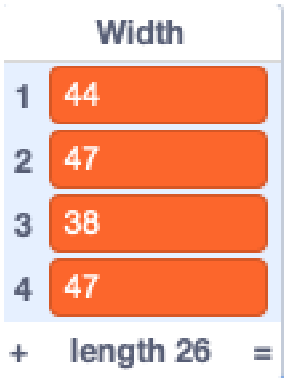


Now we can duplicate this costume 25 times. Then we replace all the letters with the remaining letters of the alphabet. At the end we rename all the sprites so that their names match with their letters. Here are the first three letters.



We notice that all three boxes have a hight of 101 points, but the width varies from 46, to 49, to 36 points.

This is very close to the values which will be calculated in the next section and stored in a list called **Width**.

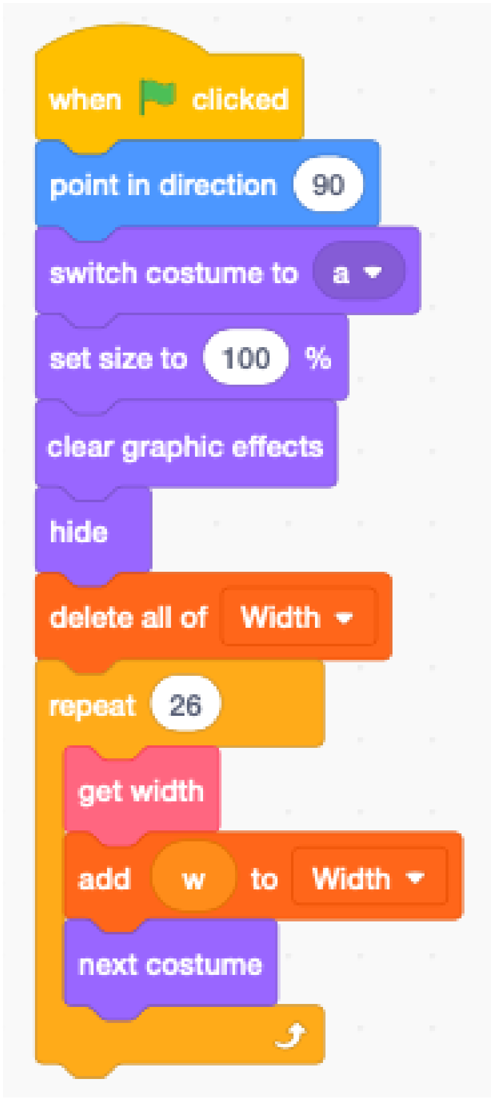


15.3 Calculate the letter width

Scratch has no function to return the width of a sprite. The only way of measuring the size of a sprite is to move it pixel by pixel until it touches one of the four edges.

We create a **Width** list for the 26 values of the letter width. At the start we

— reset the letter to horizontal (90°)
— switch to the first costume (letter a)
— set its size to 100%
— clear any graphic effets
— delete all items of the list
— enter a loop to get the width for each letter

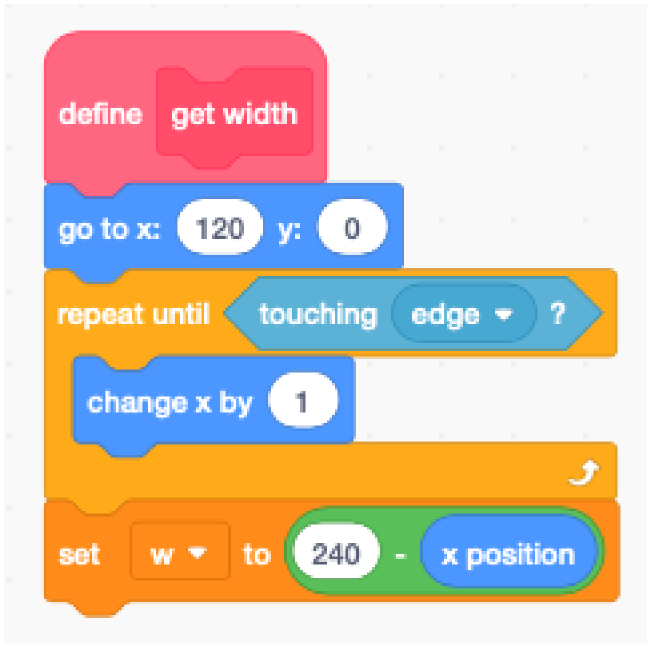


```
when [flag] clicked
point in direction (90)
switch costume to (a ▾)
set size to (100) %
clear graphic effects
hide
delete all of [Width ▾]
repeat (26)
    get width
    add (w) to [Width ▾]
    next costume
```

Since all costumes have the origin at the lower left corner, detecting the intersection with the right edge is all we need.

The function works as follows :

- place the letter at x=120.
- advance pixel by pixel until the rigth side touches the edge
- width **w = 240 - x**



```
define get width
go to x: 120 y: 0
repeat until <touching edge ?>
    change x by 1
set w to 240 - x position
```

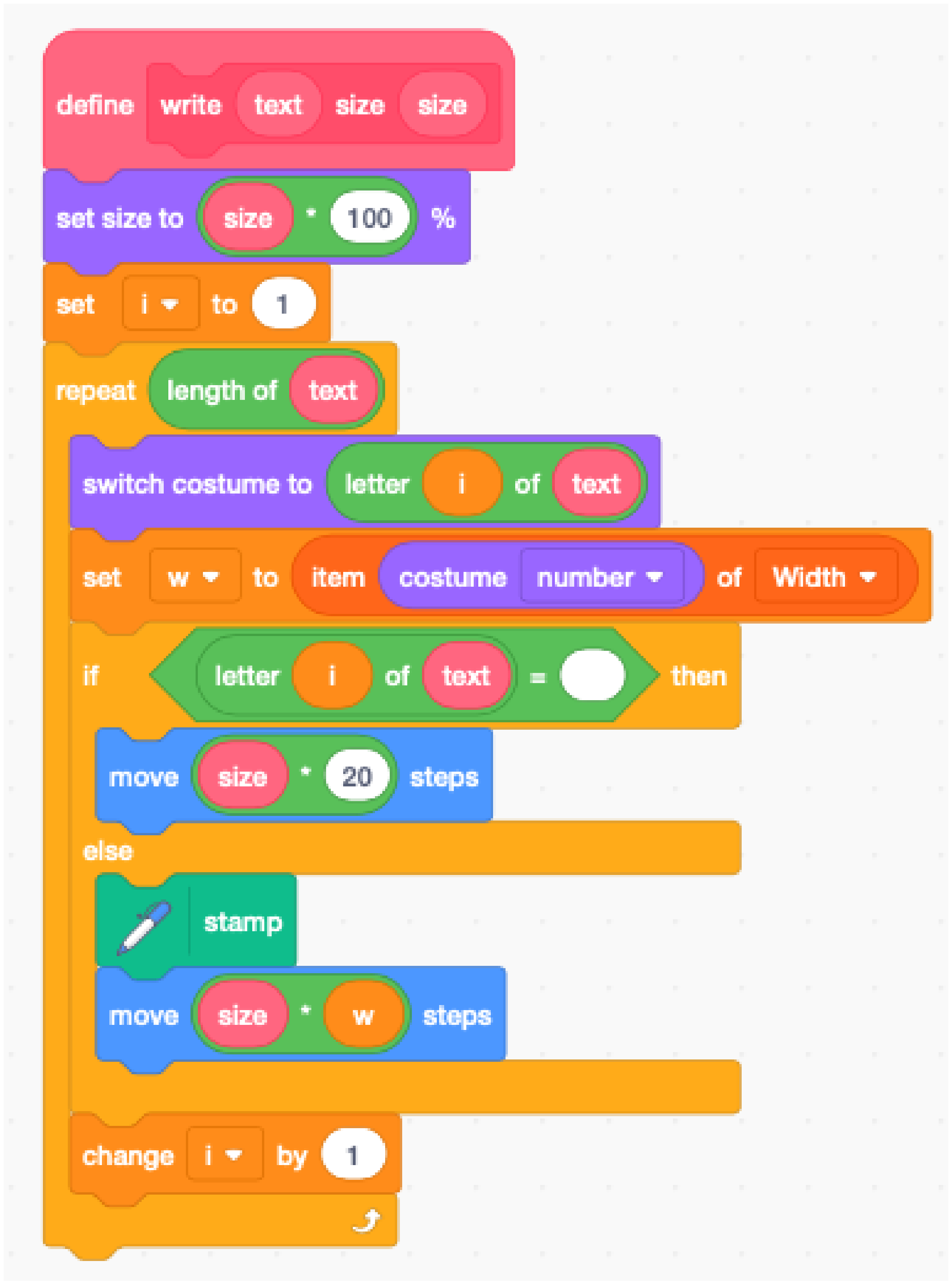
15.4 Write a text

We can now define a function to assemble the letters of a text. The function

- sets the sprite size
- resets the index **i**
- enters a loop for each letter of the text

Inside the loop we

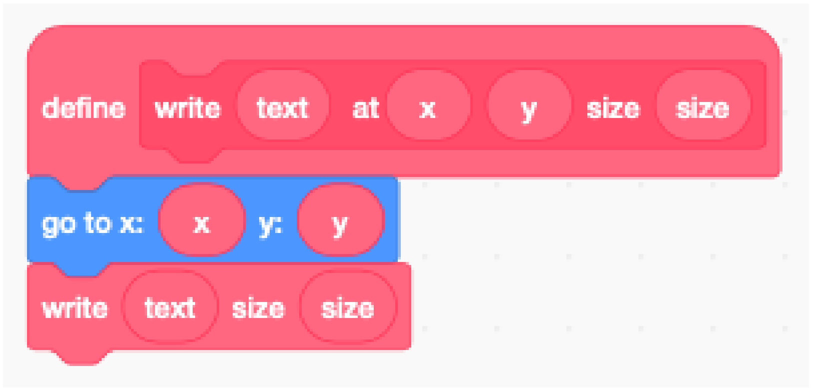
- switch the costume to current character
- get its width from the **Width** list
- if the character is a space, we advance 20 steps
- otherwise we advance for the width of the letter



```
define write (text) size (size)
set size to ((size) * (100)) %
set [i v] to (1)
repeat (length of (text))
    switch costume to (letter (i) of (text))
    set [w v] to (item (costume [number v]) of [Width v])
    if <(letter (i) of (text)) = ( )> then
        move ((size) * (20)) steps
    else
        stamp
        move ((size) * (w)) steps
    end
    change [i v] by (1)
end
```

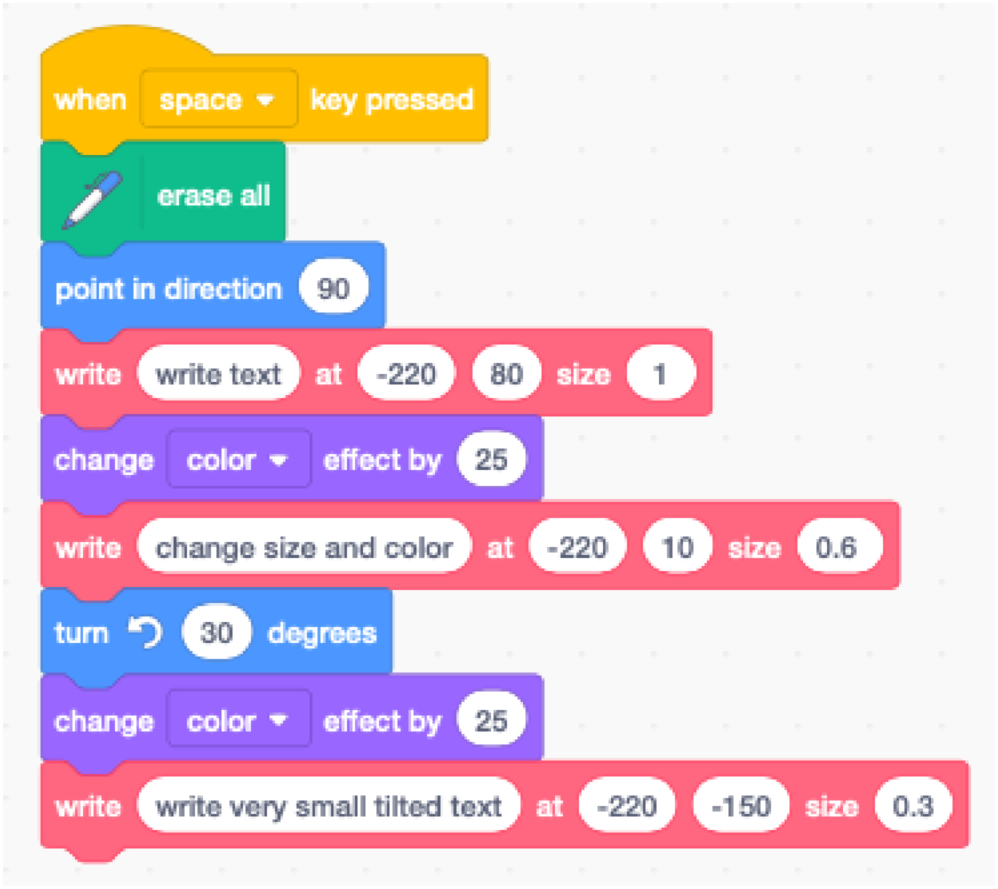
The previous function draws text based current sprite's
- position
- direction
- color
- transparency

For convenience we define a second **write** function which includes position.



```
define write (text) at (x) (y) size (size)
go to x: (x) y: (y)
write (text) size (size)
```

Now we can define our demo, started by pressing the space key.



```
when [space ▾] key pressed
erase all
point in direction (90)
write (write text) at (-220) (80) size (1)
change [color ▾] effect by (25)
write (change size and color) at (-220) (10) size (0.6)
turn ⟲ (30) degrees
change [color ▾] effect by (25)
write (write very small tilted text) at (-220) (-150) size (0.3)
```

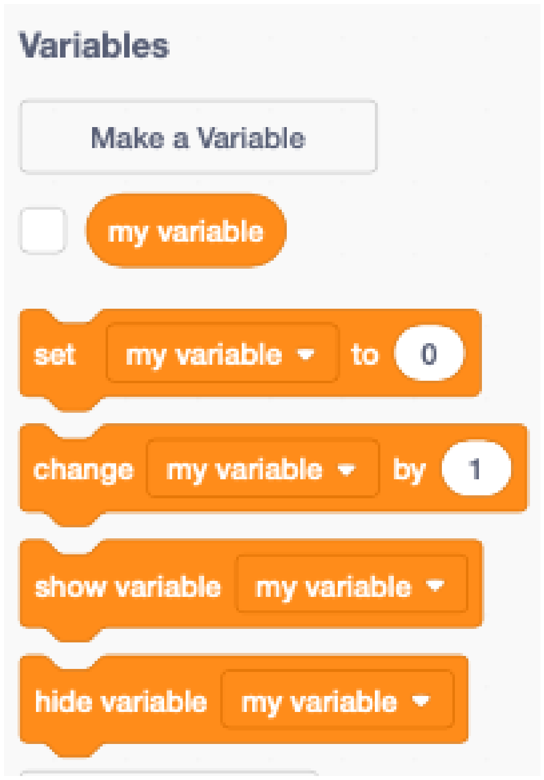
CHAPITRE 16

Variables

Variables are places in the memory of the computer to store values. You can store numbers and strings.

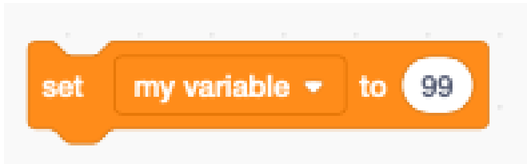
16.1 Blocks

In a new project there exists already one global variable called **my variable**. You can use it, rename it or delete it. When you activate the checkmark, the variable is displayed on the stage.

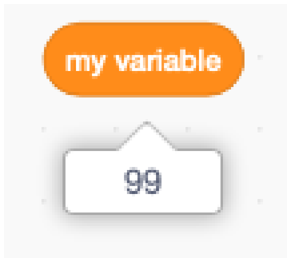


16.2 Set variable to

You can set a variable to a number value or to a string value. The following command will set the variable to the number value of 99.

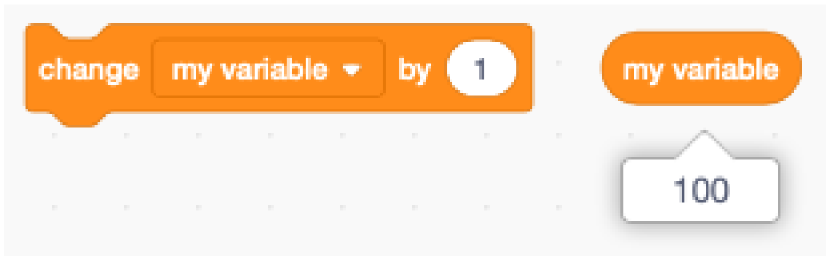


To check the value of the variable you can drag a **reporter** block onto the programming canvas and click on it. The current value of the variable is displayed.

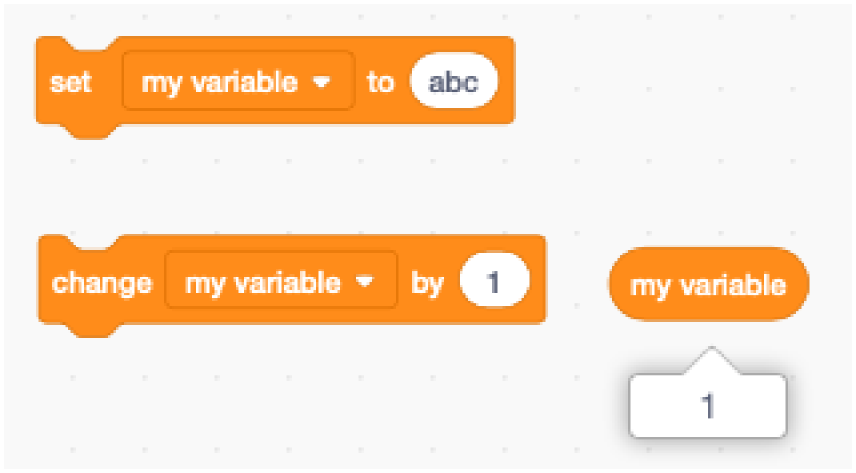


16.3 Change variable by

The **change by** block adds a number to the existing value. For example if the variable has been set to 99 and the **change by 1** block is executed, then the new value of the variable will be **99 + 1 = 100**.

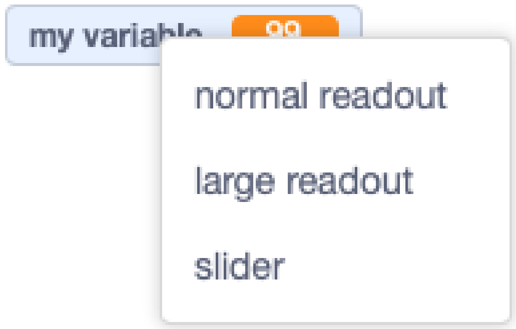


If the variable contains a string value (such as *abc*), and you try to increment it, it's value will be considered to be zero. After the change the variable has the value of the increment such as **0 + 1 = 1**.

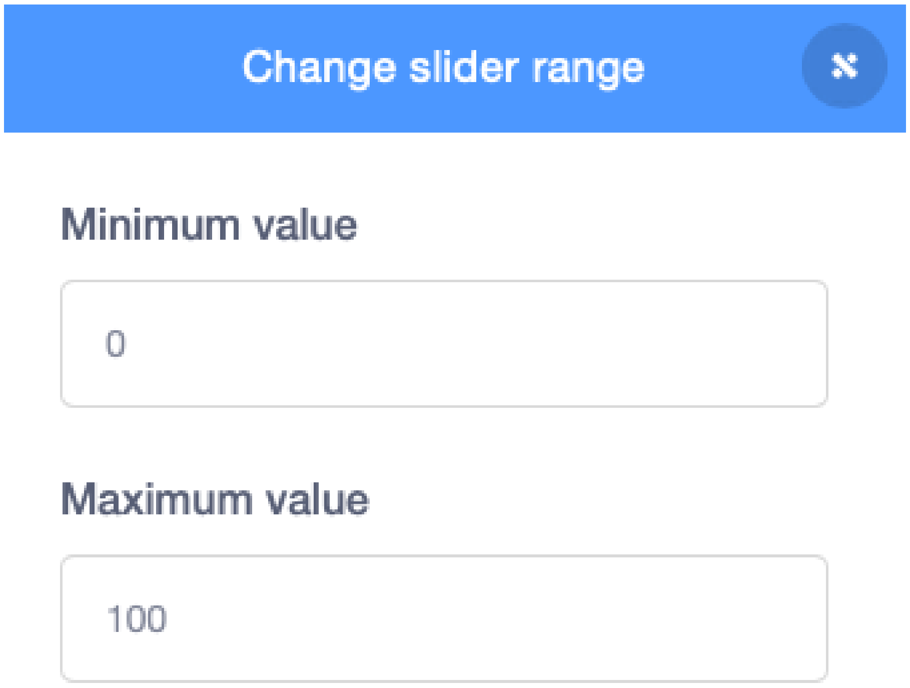


16.4 Display a variable

There are three ways to display a variable :
- normal readout, which has a label next to it
- large readout, without a label
- slider, the only way to modify the variable (works also on tablets)



When you select **slider** there is an option to select the minimum and maximum value.


Change slider range

Minimum value

0

Maximum value

100

When a variable is displayed on the stage, you drag it with the mouse to the desired position.

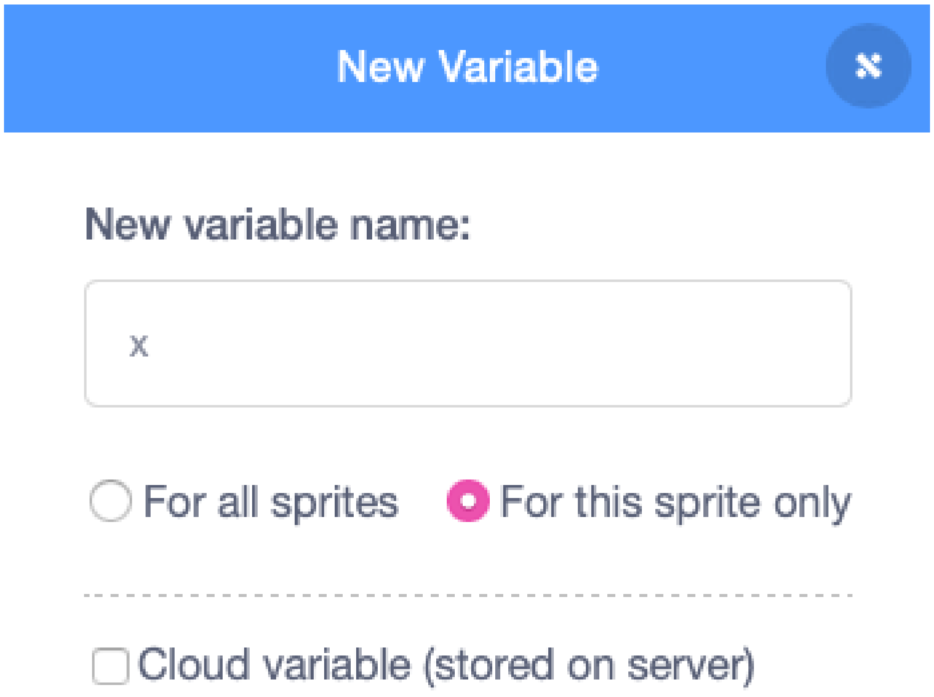
However there is no way to do this with code.

16.5 Create a new variable

When clicking on the **Make a Variable** button you can create a new variable. In the example below we give it the name **x**.

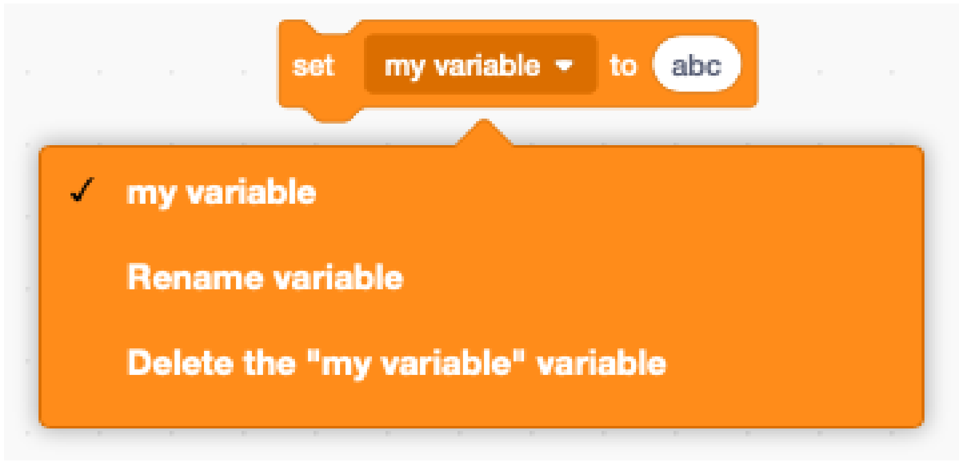
You must also choose if this variable
- is available for all sprites,
- is available only this sprite.

You also can make it a **Cloud variable** which will be stored on the server. This feature requires to be logged-in.. It allows to create multi-player games.



16.6 Rename or delete

All variable drop-down menus allow you to rename or delete them. You could delete or rename the **my variable**.

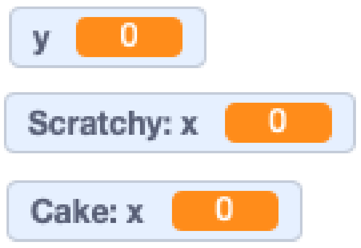


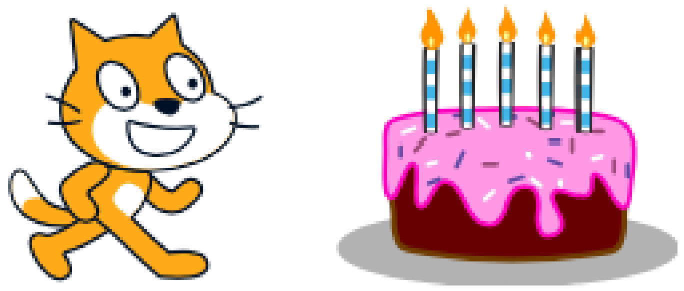
16.7 Global and local variables

When a variable is created for a single sprite only, it is called a **local** variable. Two local variables can have the same name. Variables which are available to all sprites are called **global**.

In the example below we have 3 variables :
- a global variable **y**
- a local variable **x** for the sprite **Scratchy** and

— a local variable **x** for the sprite **Cake**.

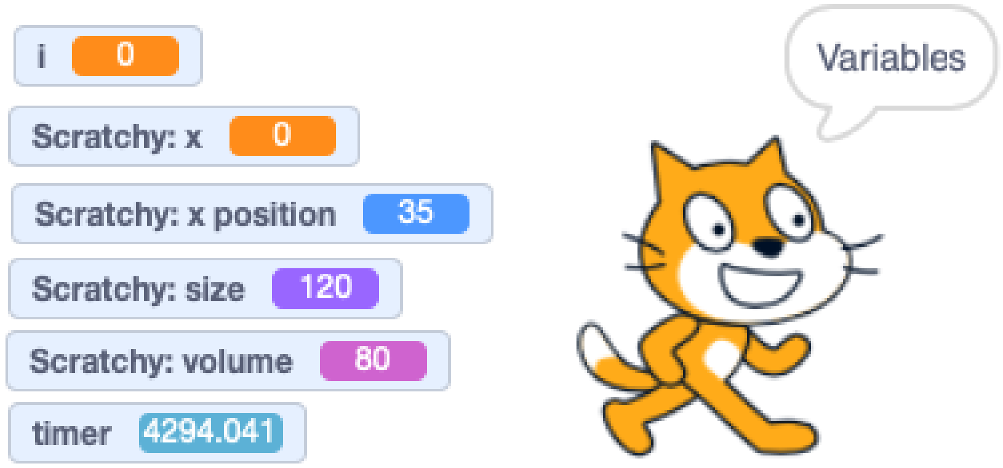




Stage variables are global and are available to all sprites.

Several **reporter** values can be displayed on the stage. Colors are used to distinguish them.

— orange for user-created global and local variables
— blue for sprite position and direction
— violet for sprite size
— purple for sound volume
— turquoise for global information such as the timer



Your program can only show and hide the user-created variables (orange). The visibility of the other reporter variables can not be programmed.

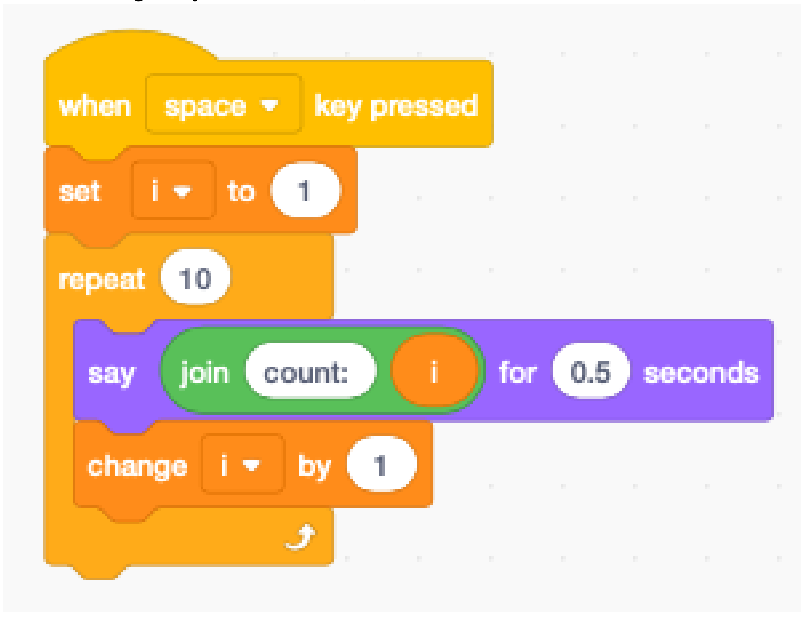
16.8 Variable as a counter

A typical use of a variable is to be used in a loop as a counter. In that case we use the variable **i**. This stands for

— integer, or
— index

A counter loop consists of these blocks :

— set **i** to the initial value (often 1)
— enter a repeat loop
— do something with variable **i**
— change **i** by an increment (often 1)



```
when space key pressed
set i to 1
repeat 10
    say join count: i for 0.5 seconds
    change i by 1
```

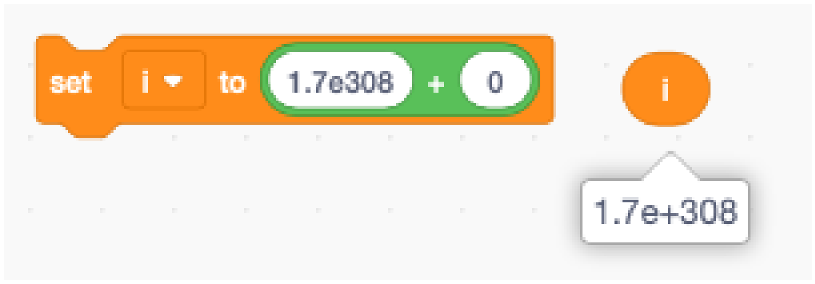
The above code makes Scratchy count from 1 to 10.

16.9 Largest number

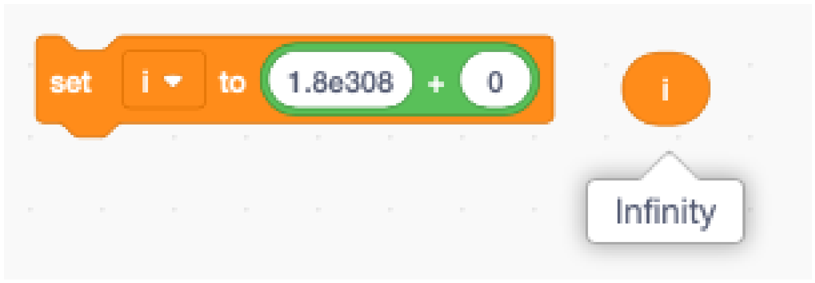
The numbers which can be stored in variables are floating point numbers. They are called **double-precision foating point numbers** and are encoded internally with 64 bits. It has about 15 significant digits.

The largest number which can be represented is close to **1.8e308**.

This value can still be used :

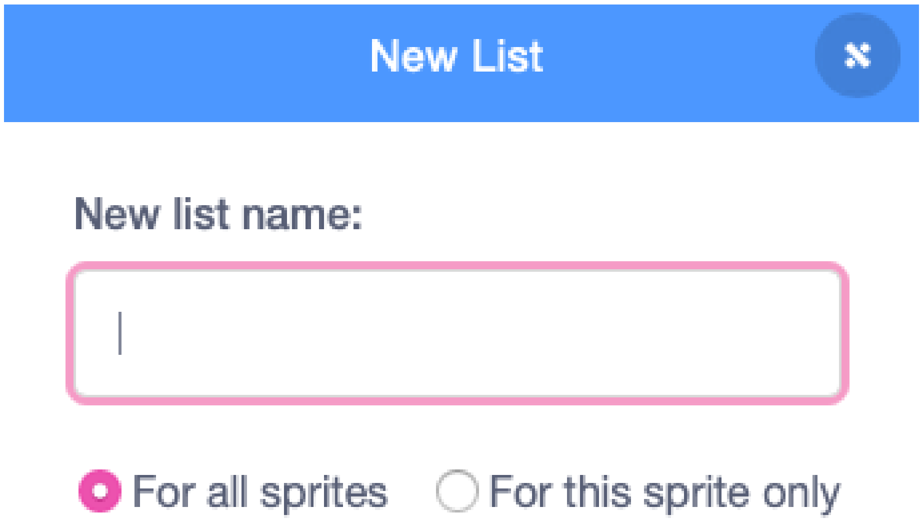


This slighty larger value results in **Infinity**.

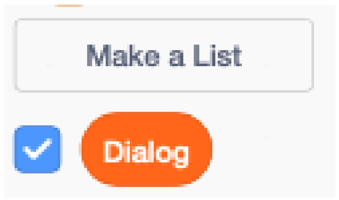


16.10 Make a list

Besides variables Scratch has a second way of storing information : lists. Lists are groups of variables.



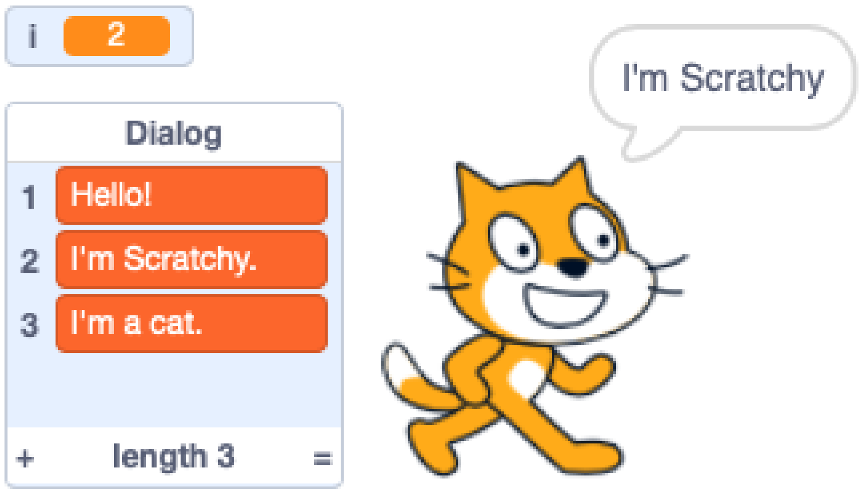
After making a new list you can display it on the stage.



This is the easiest way to enter data

— click (+) to create a new item

— click and drag (-) to resize the list
— when editing press ENTER to create a new item
The following image shows a list called **Dialog** with 3 items.

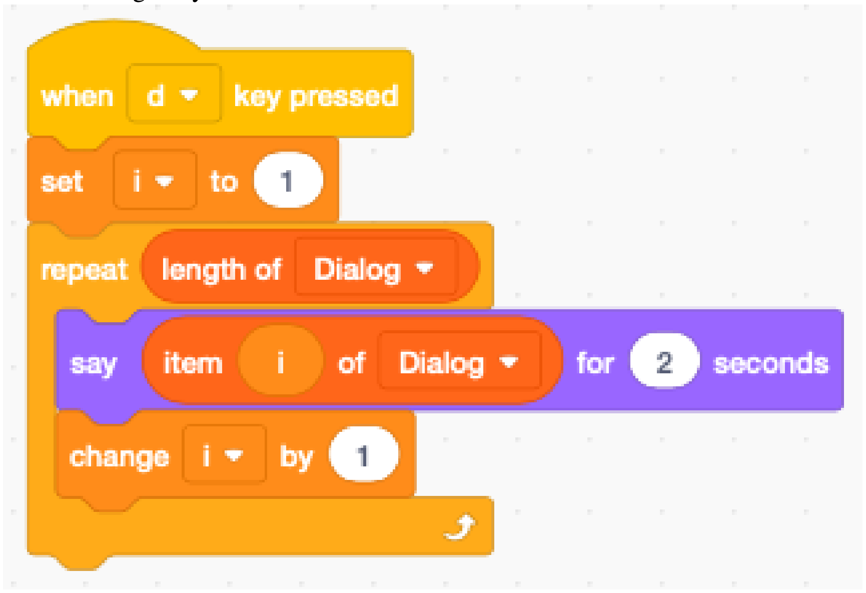


The image above shows index **i = 2** accessing the item *I'm Scratchy*.

16.11 Cycle through a list

A frequent task is to cycle through all the elements of a list. Like in the counter example from before, we are using an index variable **i**.

The steps are
— initialize the index **i** to 1 (first item)
— repeat for the lenght of the list
— do something with the item number **i**
— change **i** by 1



```
when d key pressed
set i to 1
repeat (length of Dialog)
    say (item i of Dialog) for 2 seconds
    change i by 1
```

CHAPITRE 17

Clones

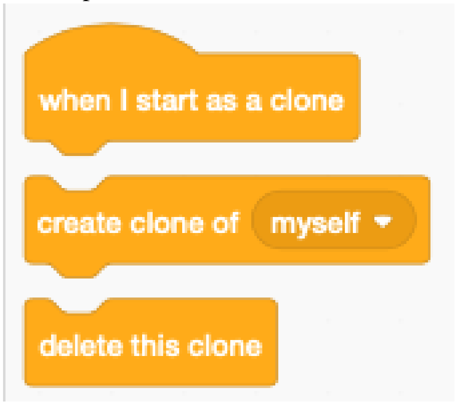
A **clone** is an exact copy of a sprite which inherits also all the scripts. This is useful if you want to create multiple spirits which have similar behavior.

There are 3 clone-related blocks :

— a hat block to start as a clone
— a stack block to clone the current sprite
— an end block to delete the clone (and stops its scripts)

If a variable is marked as *for this sprite only*, each clone will have its own private variable, just as each clone has its :

— position
— direction
— pen



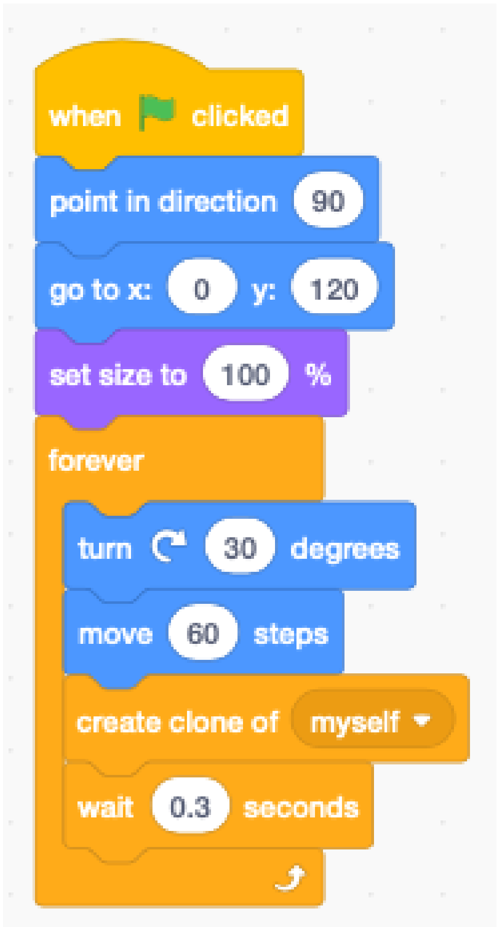
17.1 Decaying sprite trail

https://scratch.mit.edu/projects/390996965

You can use clones to create a mouse trail. Usually these clones will fade out and disappear. For exemple they can decrease their size, or use the ghost effect to become invisible.

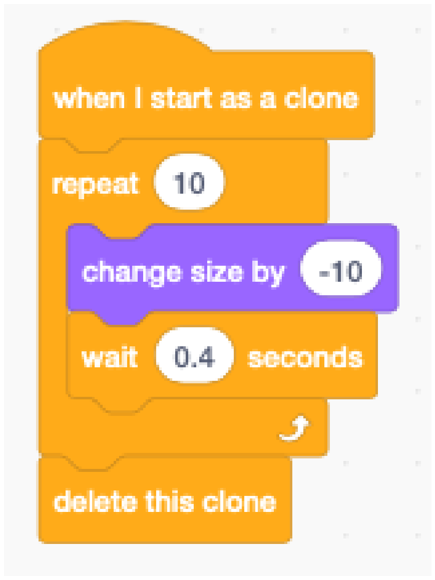
At start the sprite is initialized with an initial :
- position
- direction
- size

Inside the **forever** loop the sprite is rotated by 30 degrees and advanced. It executes a circle. At each of the 12 positions a clone is created every 0.3 seconds.



```
when 🏴 clicked
point in direction (90)
go to x: (0) y: (120)
set size to (100) %
forever
    turn ↻ (30) degrees
    move (60) steps
    create clone of (myself ▾)
    wait (0.3) seconds
```

Every 0.4 seconds the clone size decreases by 10%. After 10 iterations the clone is deleted.

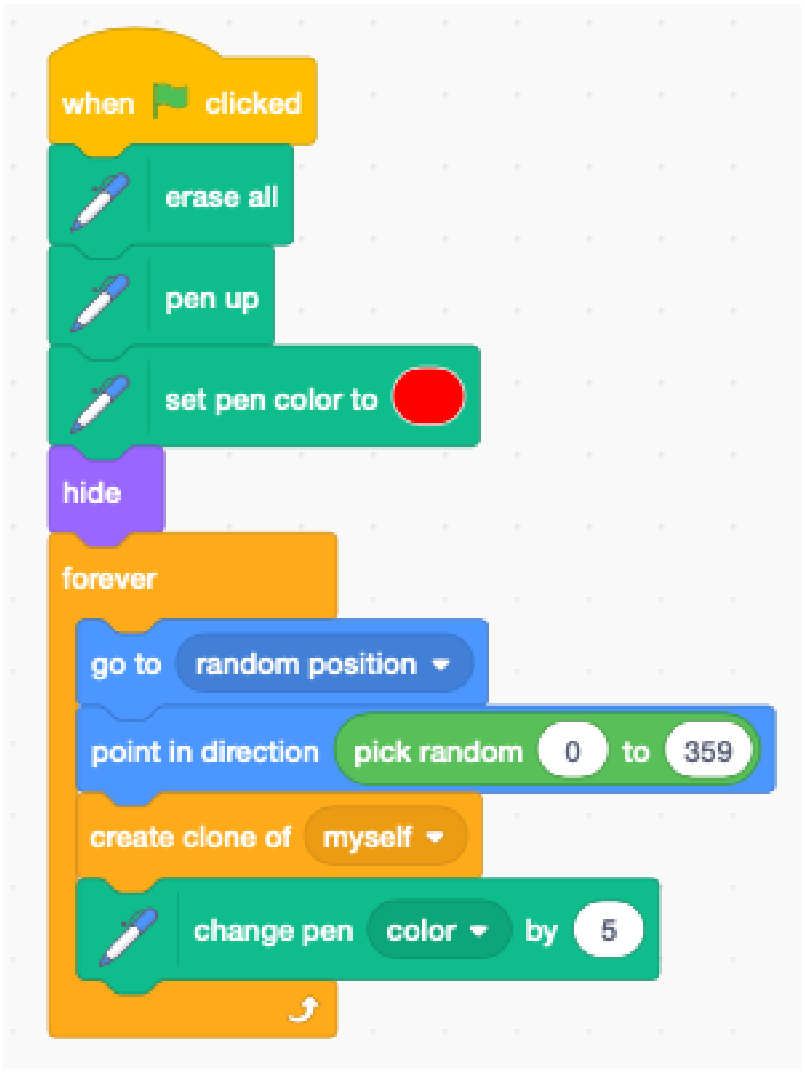


```
when I start as a clone
repeat (10)
    change size by (-10)
    wait (0.4) seconds
end
delete this clone
```

17.2 Pen trail towards the edge

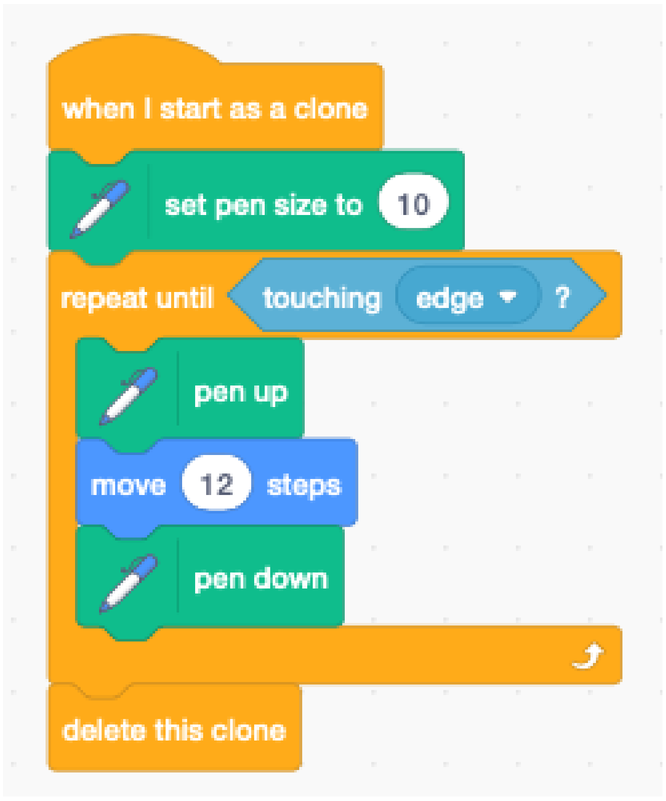
https://scratch.mit.edu/projects/391011669

This project creates clones at random positions and random directions.



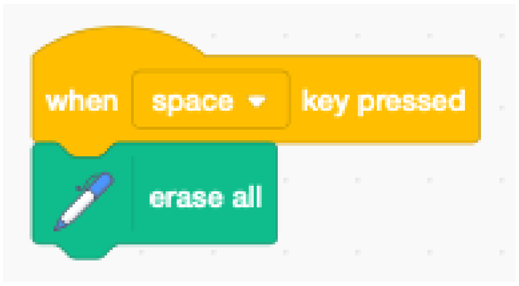
```
when clicked
erase all
pen up
set pen color to
hide
forever
    go to random position
    point in direction pick random 0 to 359
    create clone of myself
    change pen color by 5
```

Each clone continues to move forward in its chosen direction. When it reaches the edge, the clone is deleted.



```
when I start as a clone
set pen size to (10)
repeat until <touching (edge ▾) ?>
    pen up
    move (12) steps
    pen down
end
delete this clone
```

Pressing the space bar erases the stage.



```
when [space ▾] key pressed
erase all
```

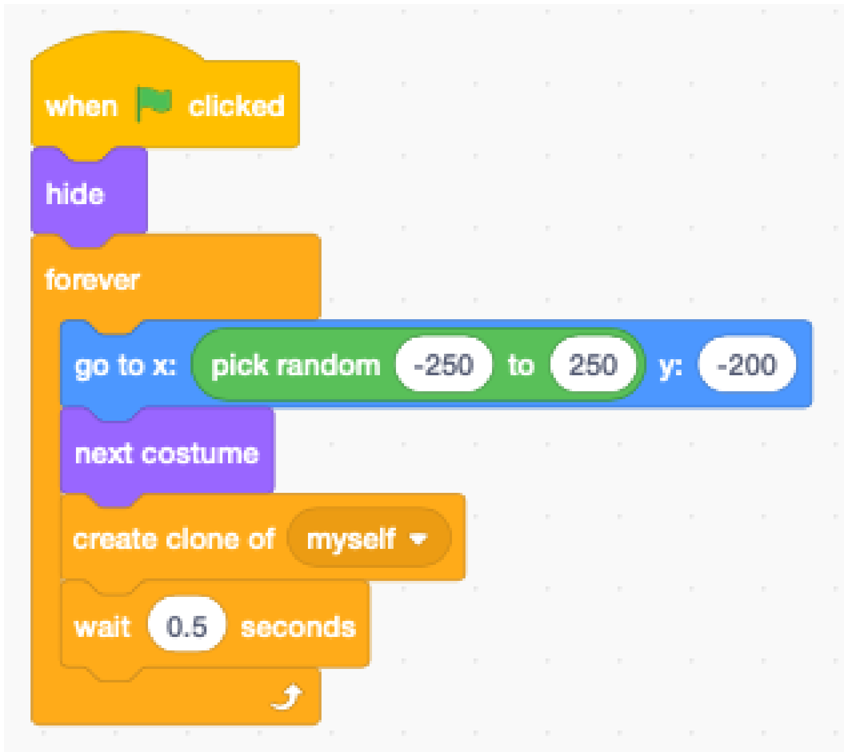
17.3 Disappearing balloons

https://scratch.mit.edu/projects/391035000

An image with a large portion of a sky had been chosen. The ballons appear at random positions at the bottom of the image.

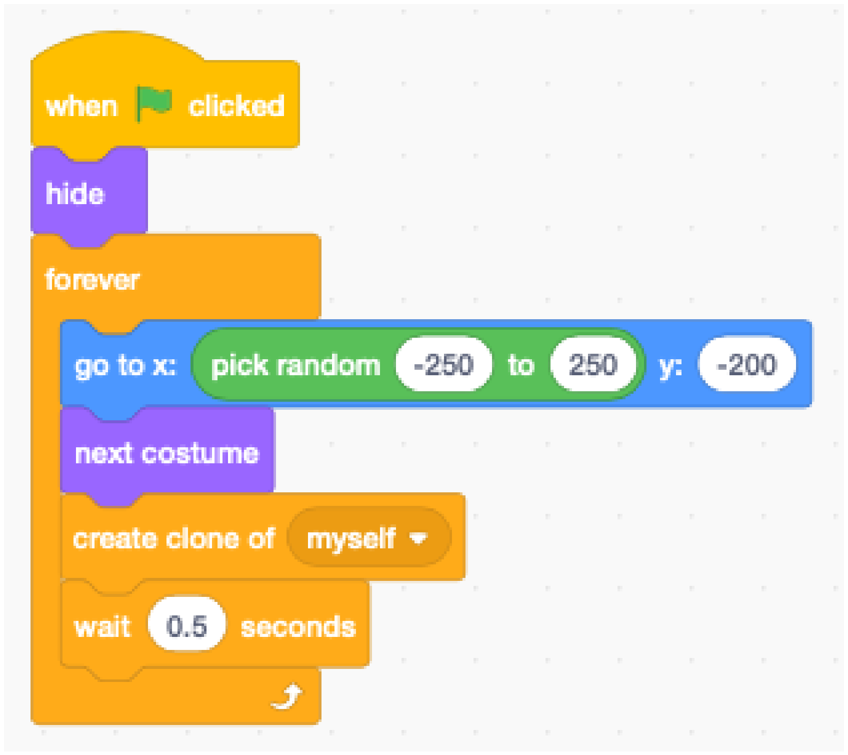
First we hide the orginal sprite, then we enter a **forever** loop and :

- go to a random location
- switch the costume to a different color
- create a clone
- wait 0.5 seconds



```
when flag clicked
hide
forever
    go to x: (pick random (-250) to (250)) y: (-200)
    next costume
    create clone of (myself)
    wait (0.5) seconds
```

Each cloned balloon moves upwards into the sky. At each iteration it gets smaller, until it disappears. To make the movement look more natural, a small random change is introduced into the direction.



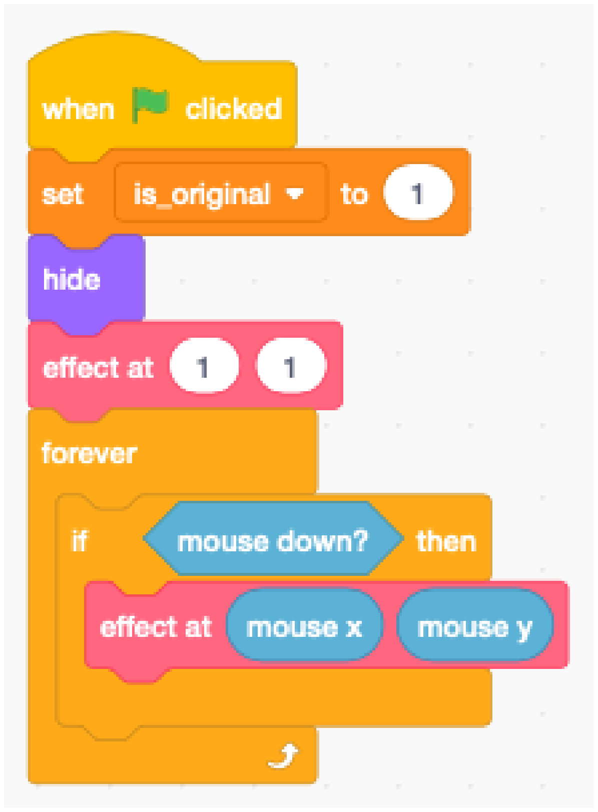
```
when flag clicked
hide
forever
    go to x: (pick random (-250) to (250)) y: (-200)
    next costume
    create clone of (myself)
    wait (0.5) seconds
```

17.4 Particle effects

Clones can be used for special effects such as :
— fire, sparks, explosions
— fog, clouds
— bubbles,
— birds, fish, leaves
https://scratch.mit.edu/projects/391003021

For each clone we define 3 individual variables :
— the x displacement **dx**
— the y displacement **dy**
— a boolean variable **is_original** to indicate if the sprite is the original or a clone

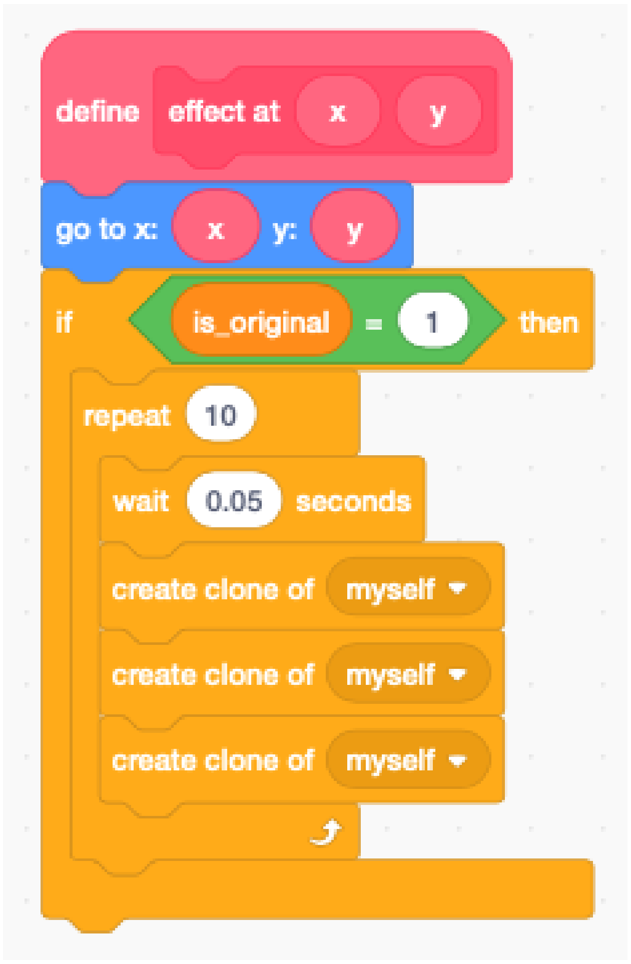
At the start of the program there is only the parent sprite.
— set **is_orignal** to 1 (it's not a clone)
— hide the parent sprite
— create a spark effect at position (1, 1)
— enter a **forever** loop
— create sparks at the mouse position when the mouse is down



```
when 🏴 clicked
set is_original ▾ to 1
hide
effect at 1 1
forever
    if mouse down? then
        effect at mouse x mouse y
```

The function **effect at (x, y)** does the following :

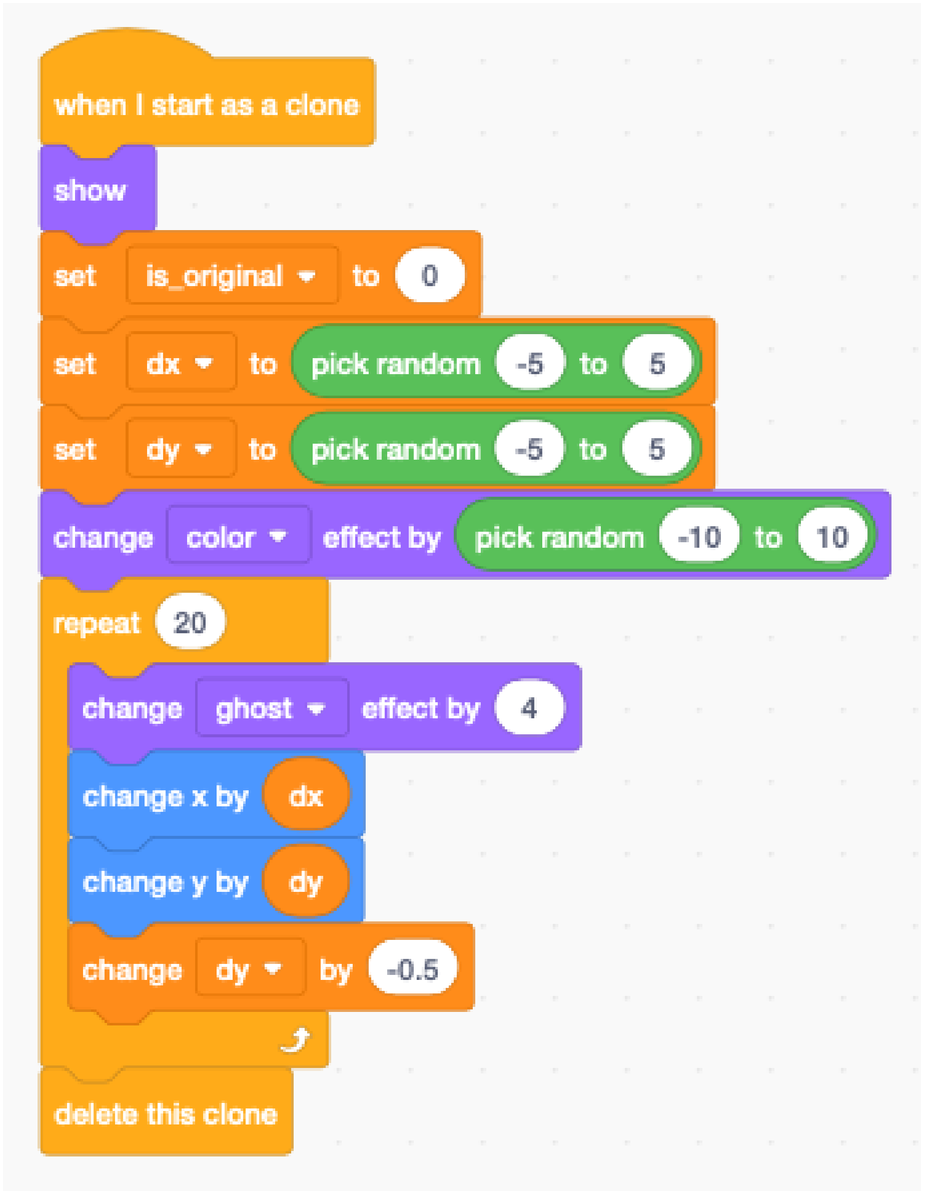
- go to position (x, y)
- check if the sprite is an original (is_original = 1)
- repeat 10 times
- wait 0.05 seconds and create 3 clones



```
define effect at (x) (y)
go to x: (x) y: (y)
if <(is_original) = (1)> then
    repeat (10)
        wait (0.05) seconds
        create clone of (myself ▾)
        create clone of (myself ▾)
        create clone of (myself ▾)
    end
end
```

When a clone starts it does

- show the sprite
- set **is_original** to 0 (it's a clone)
- set a random speed (dx, dy)
- add a random color variation
- repeat 20 times
- become more transparent (add 4% ghost effect)
- change position by (dx, dy)
- add gravity to **dy**
- delete the clone after 20 iterations



```
when I start as a clone
show
set is_original to 0
set dx to pick random -5 to 5
set dy to pick random -5 to 5
change color effect by pick random -10 to 10
repeat 20
    change ghost effect by 4
    change x by dx
    change y by dy
    change dy by -0.5
delete this clone
```

CHAPITRE 18

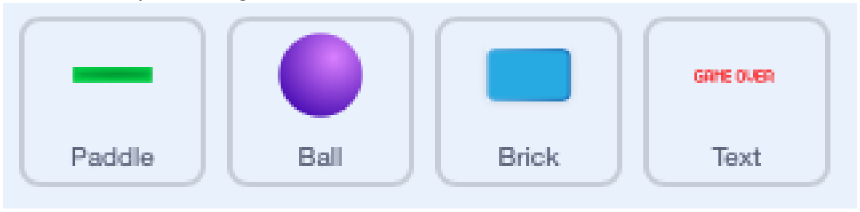
Bricks

The following game uses a paddle to bounce a moving ball. The ball must hit a wall of bricks. When the ball hits a brick, the brick is destroyed. When all the bricks are destroyed, you win the game.

https://scratch.mit.edu/projects/393952500

18.1 Sprites

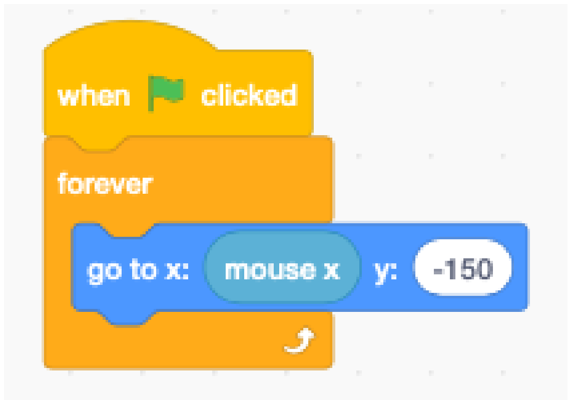
This game uses 4 sprites :
- ball
- paddle
- brick
- text (you win, game over)



18.2 Paddle

In this game we don't need Scratchie the cat. Delete it and load the **Paddle** sprite. We use a very simple script to move the paddle with the mouse.

We set the x coordinate to the **mousex** coordinate and the y coordinate to a fixed value at -150.

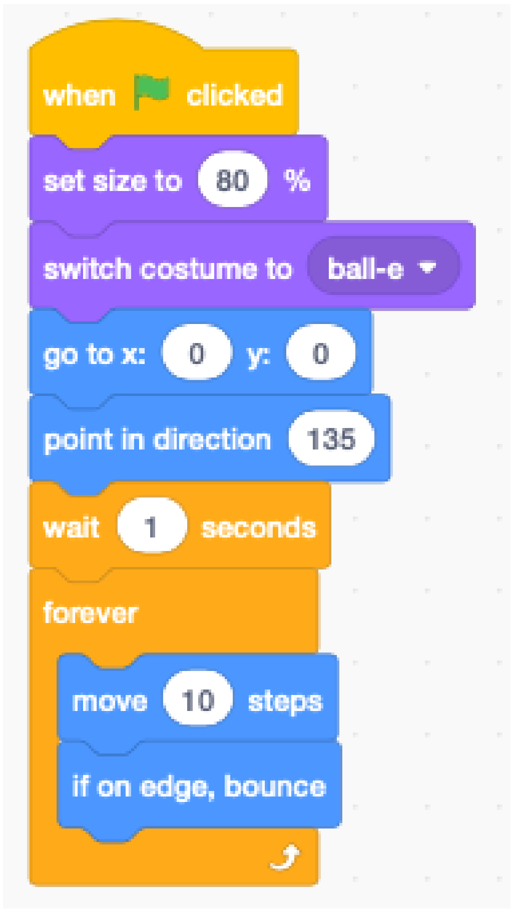


```
when [flag] clicked
forever
    go to x: (mouse x) y: (-150)
```

18.3 Ball

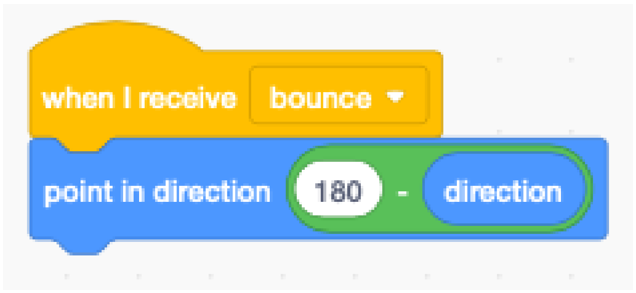
Add a new sprite **Ball** sprite.

When the program starts :
- set ball size to 80%
- switch to costume **ball-e** (violet)
- go to the center position (0, 0)
- point to left and down (135 degrees)
- wait 1 second until the bricks are placed
- enter the **forever** loop
- move 10 steps and bounce from edges



```
when 🏳 clicked
set size to (80) %
switch costume to (ball-e ▾)
go to x: (0) y: (0)
point in direction (135)
wait (1) seconds
forever
    move (10) steps
    if on edge, bounce
```

When the ball receveives the message **bounce** it changes direction.



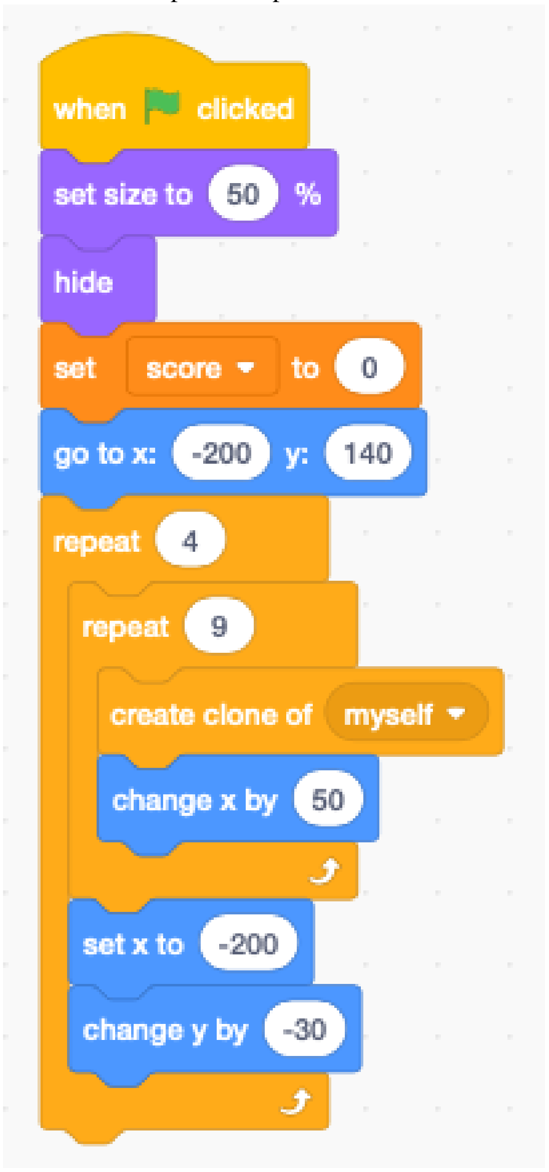
```
when I receive (bounce ▾)
point in direction ((180) - (direction))
```

There are two events which make the ball bounce ; when it hits
- the paddle, or
- a brick

18.4 Bricks

You can load the **Button3** sprite and delete the gray costume. When the program starts we :
- set the size to 50%
- hide the original brick
- set the score to 0
- place the original brick sprite to position (-200, 140)
- enter a 4x9 nexted loop

- — clone the brick 36 times
- — 9 columns spaced by 50 pixels
- — 4 rows spaced 30 pixels

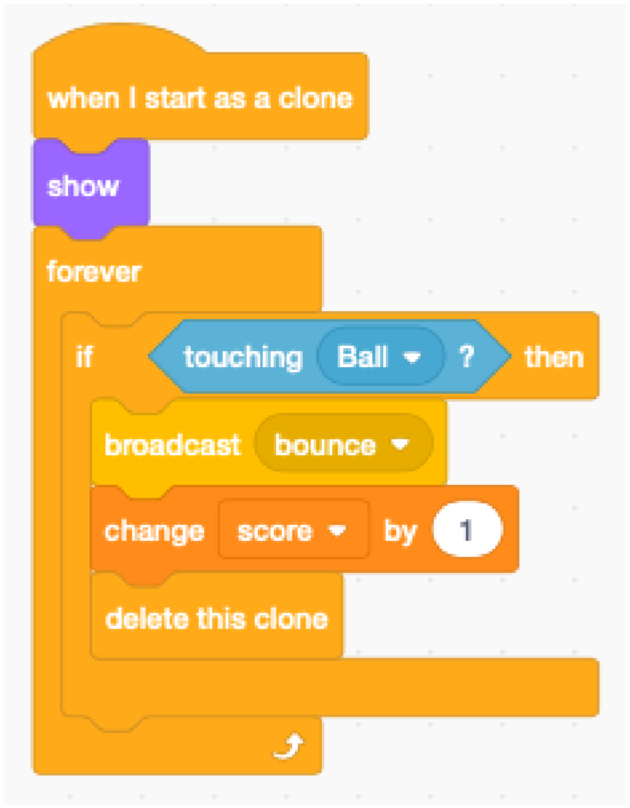


```
when 🏴 clicked
set size to (50) %
hide
set [score ▾] to (0)
go to x: (-200) y: (140)
repeat (4)
    repeat (9)
        create clone of (myself ▾)
        change x by (50)
    end
    set x to (-200)
    change y by (-30)
end
```

Whenever a clone starts we :

- — show the sprite
- — enter a **forever** loop

if a brick is touched by the ball - the message **bounce** is broadcast (to the ball) - the socre increases - the brick is removed



```
when I start as a clone
show
forever
  if <touching (Ball ▾) ?> then
    broadcast (bounce ▾)
    change [score ▾] by (1)
    delete this clone
```

18.5 Text

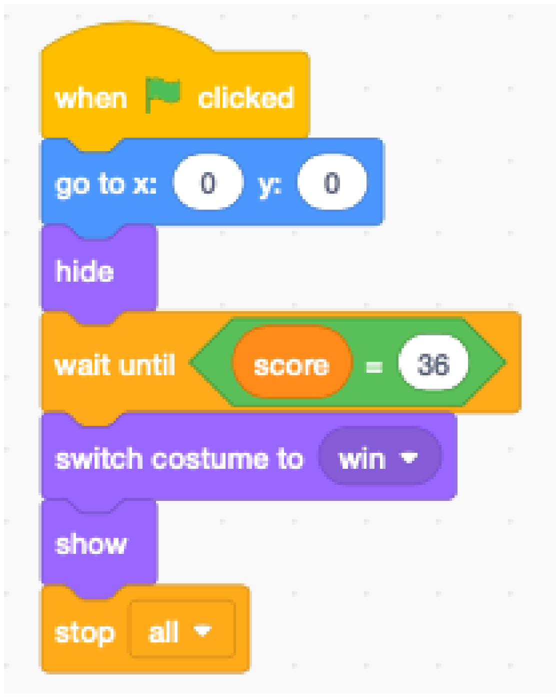
Create a **Text** sprite with two messages :

- you win (green)
- game over (red)

Be careful to center both texts.

When the program starts :

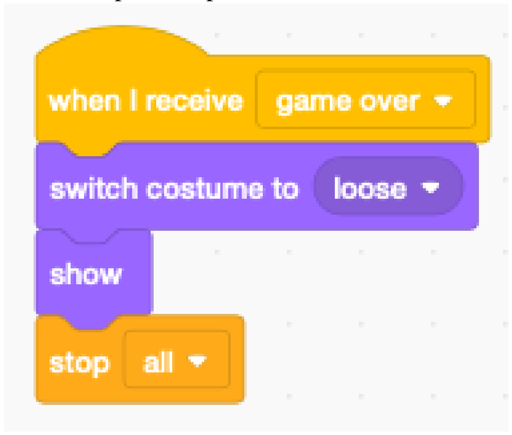
- go to the center (0, 0)
- hide the text
- wait until the sore is 36
- switch to **you win**
- show the sprite
- stop all scripts



```
when green flag clicked
go to x: 0 y: 0
hide
wait until <(score) = 36>
switch costume to win
show
stop all
```

When the message **gameover** is received, then

- show the costume **game over**
- show the sprite
- stop all scripts



```
when I receive game over
switch costume to loose
show
stop all
```

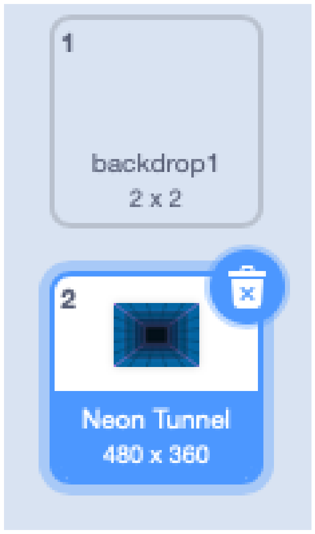
18.6 Polish the game

Now that we have a functional game, let's polish the game.

https://scratch.mit.edu/projects/393997612

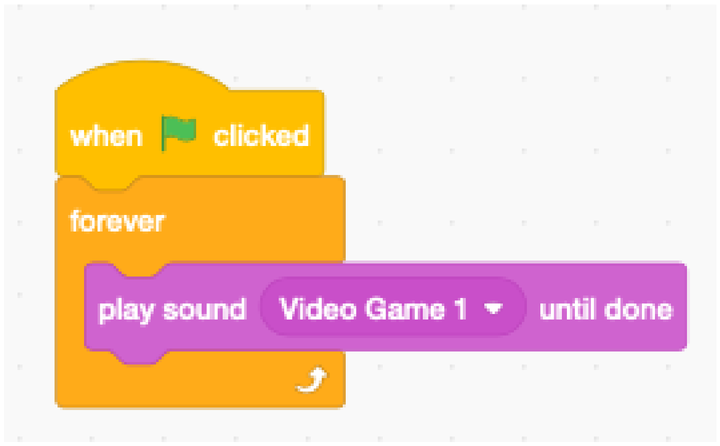
18.7 Add a background

Add a backdrop to your game. Here we choose the **Neon Tunnel** backdrop.



18.8 Add music

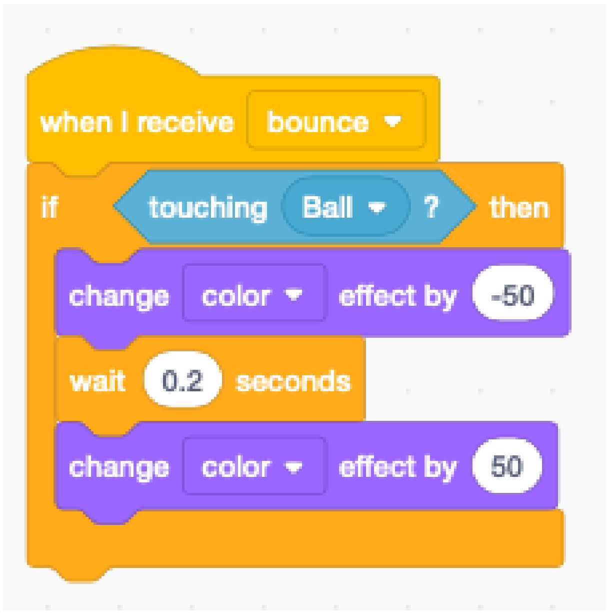
Inside the Stage, we add the **Video Game 1** sound. This little script plays it in a loop.



```
when green flag clicked
forever
    play sound (Video Game 1 ▾) until done
```

18.9 Make the paddle flash

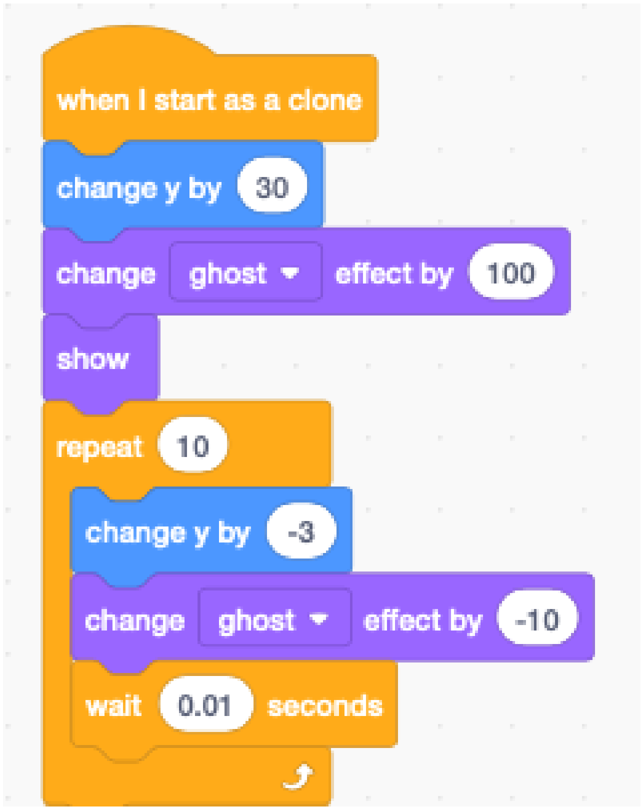
When the ball hits the paddle, we make it flash by changing the color effect for 0.2 seoconds and then go back to the original color.



```
when I receive bounce
if touching Ball ? then
    change color effect by -50
    wait 0.2 seconds
    change color effect by 50
```

18.10 Animate the brick entry

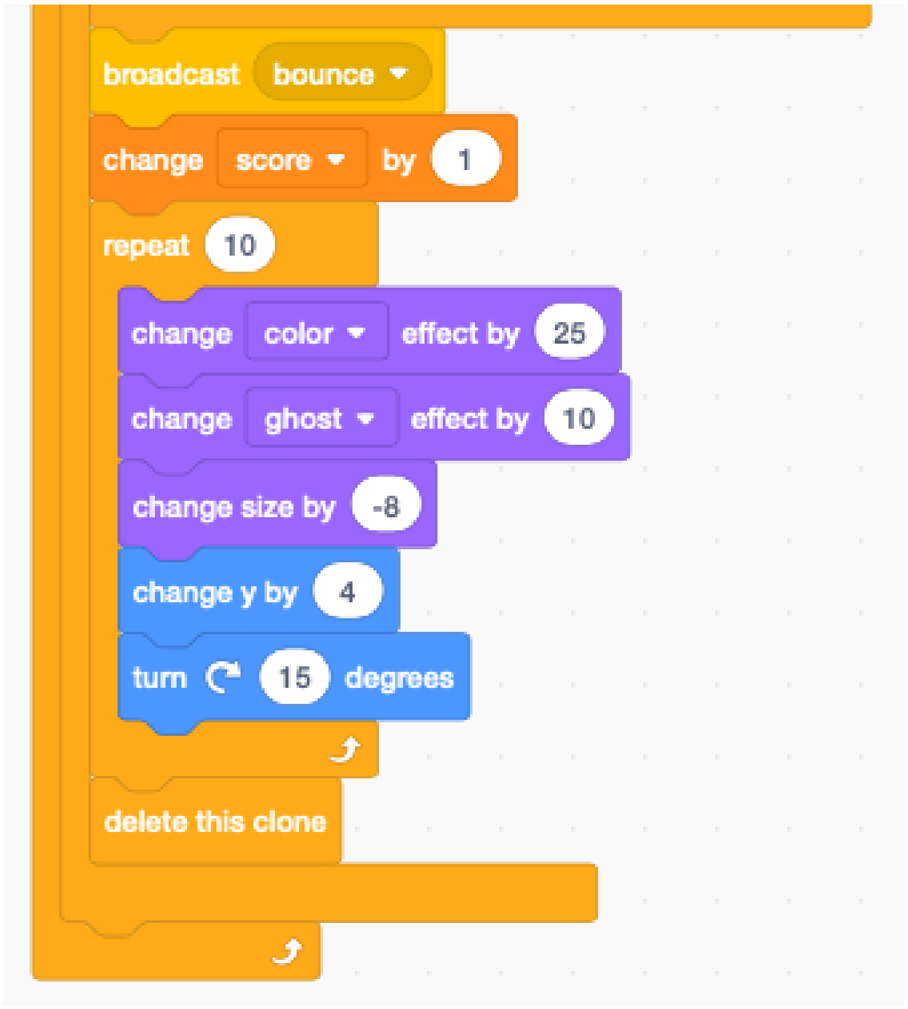
To make the brick entry more interesting, we can animate them.



```
when I start as a clone
change y by 30
change ghost effect by 100
show
repeat 10
    change y by -3
    change ghost effect by -10
    wait 0.01 seconds
```

18.11 Animate the brick exit

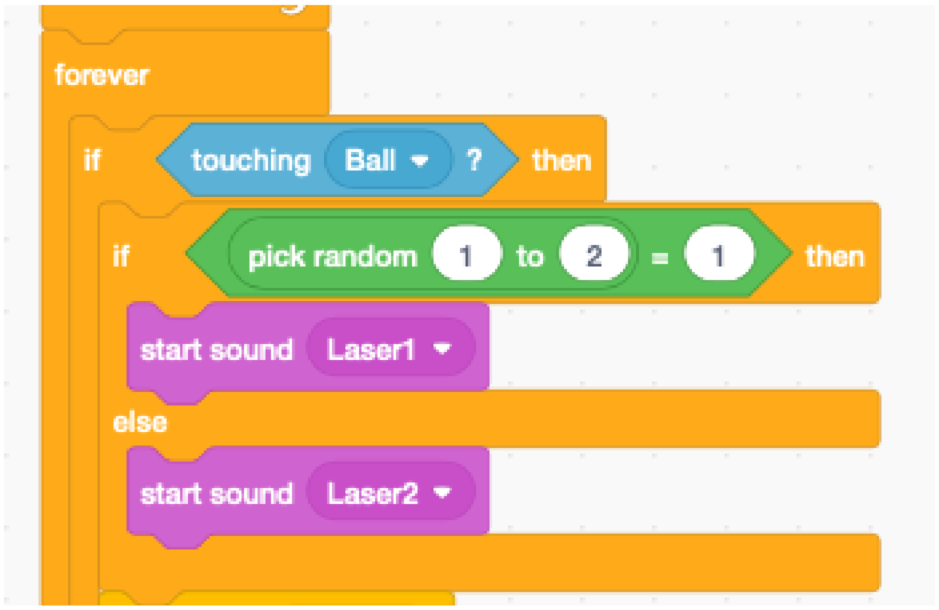
Inside the brick clone loop we add the following code :



```
broadcast [bounce ▾]
change [score ▾] by (1)
repeat (10)
    change [color ▾] effect by (25)
    change [ghost ▾] effect by (10)
    change size by (-8)
    change y by (4)
    turn ↻ (15) degrees
end
delete this clone
```

18.12 Add sound effect to brick

We add two sounds (Laser1 and Laser2) to the brick sprite. When the ball touches a brick, we chose one of the sounds randomly.

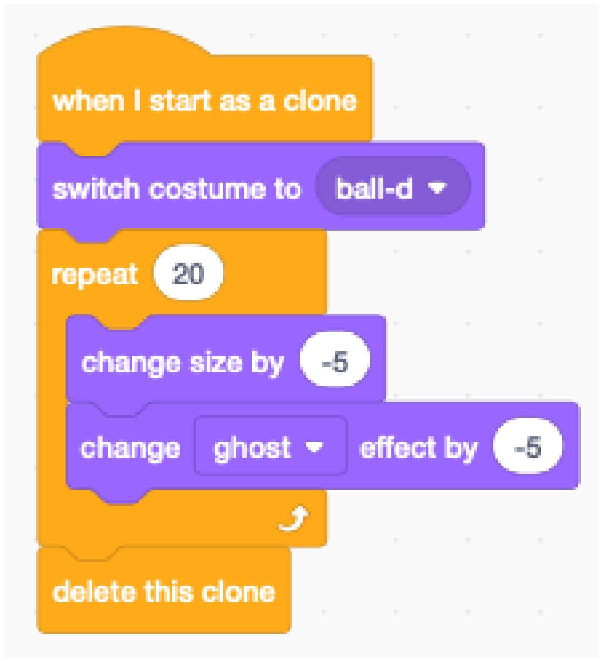


```
forever
  if <touching (Ball ▾) ?> then
    if <(pick random (1) to (2)) = (1)> then
      start sound (Laser1 ▾)
    else
      start sound (Laser2 ▾)
```

18.13 Add a trail to the ball

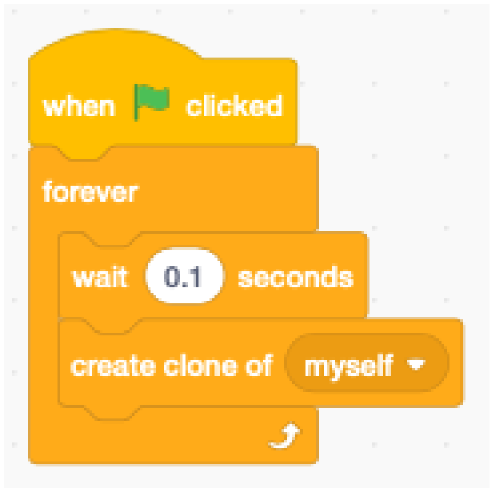
We use cloning to add a trail to the moving ball. First we change the color of the clone to the green costume (ball-d).

Each ball clone goes through 20 iterations and becomes smaller and more transparent (ghost effect). After 20 iterations the clone is deleted.



```
when I start as a clone
switch costume to (ball-d ▾)
repeat (20)
  change size by (-5)
  change (ghost ▾) effect by (-5)
delete this clone
```

To control the timing of the clone trail, we use a second **forever** loop in the ball sprite. It creates a new ball clone and every 0.1 seconds.

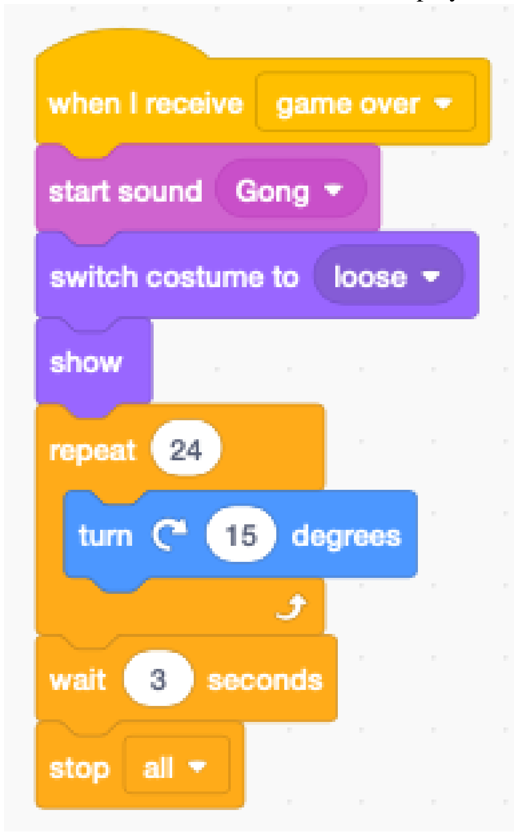


```
when [flag] clicked
forever
    wait (0.1) seconds
    create clone of (myself ▾)
```

18.14 Animate Game Over sprite

Let's animate the game-over sprite :

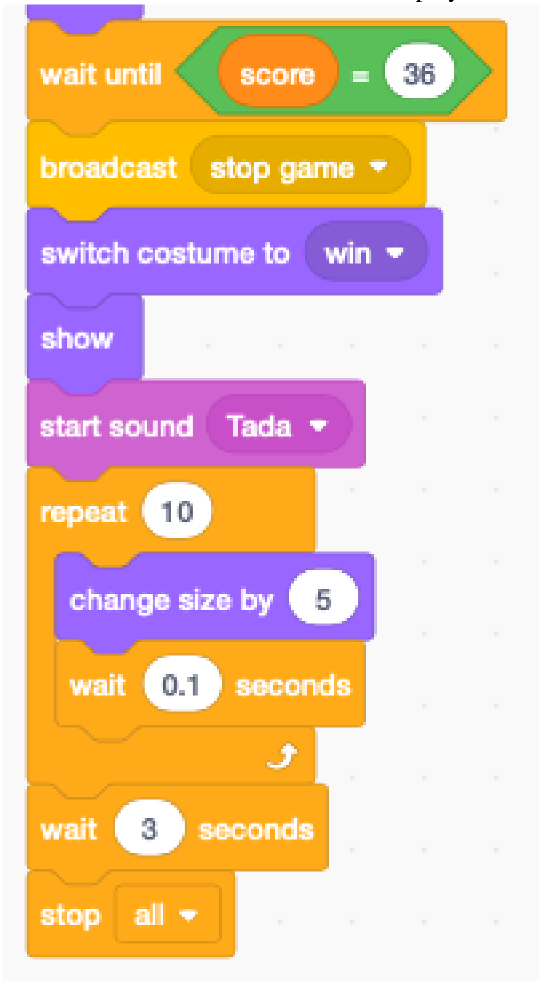
- add a **Gong** sound
- make the sprite rotate
- wait 3 seconds to let the music play



```
when I receive [game over ▾]
start sound (Gong ▾)
switch costume to (loose ▾)
show
repeat (24)
    turn ↻ (15) degrees
wait (3) seconds
stop [all ▾]
```

18.15 Animate You Win sprite

When the score reaches 36, we
- — add a **Tada** sound
- — repeat 10 times
- — increase the sprite size by 5 %
- — wait 3 seconds to let the music play



```
wait until score = 36
broadcast stop game
switch costume to win
show
start sound Tada
repeat 10
    change size by 5
    wait 0.1 seconds
wait 3 seconds
stop all
```

CHAPITRE 19

Platformer

CHAPITRE 20

Pong

Pong is a game based on ping-pong. A ball bounces off by three walls and the the player must hit the ball with a paddle.

CHAPITRE 21

Adventure game

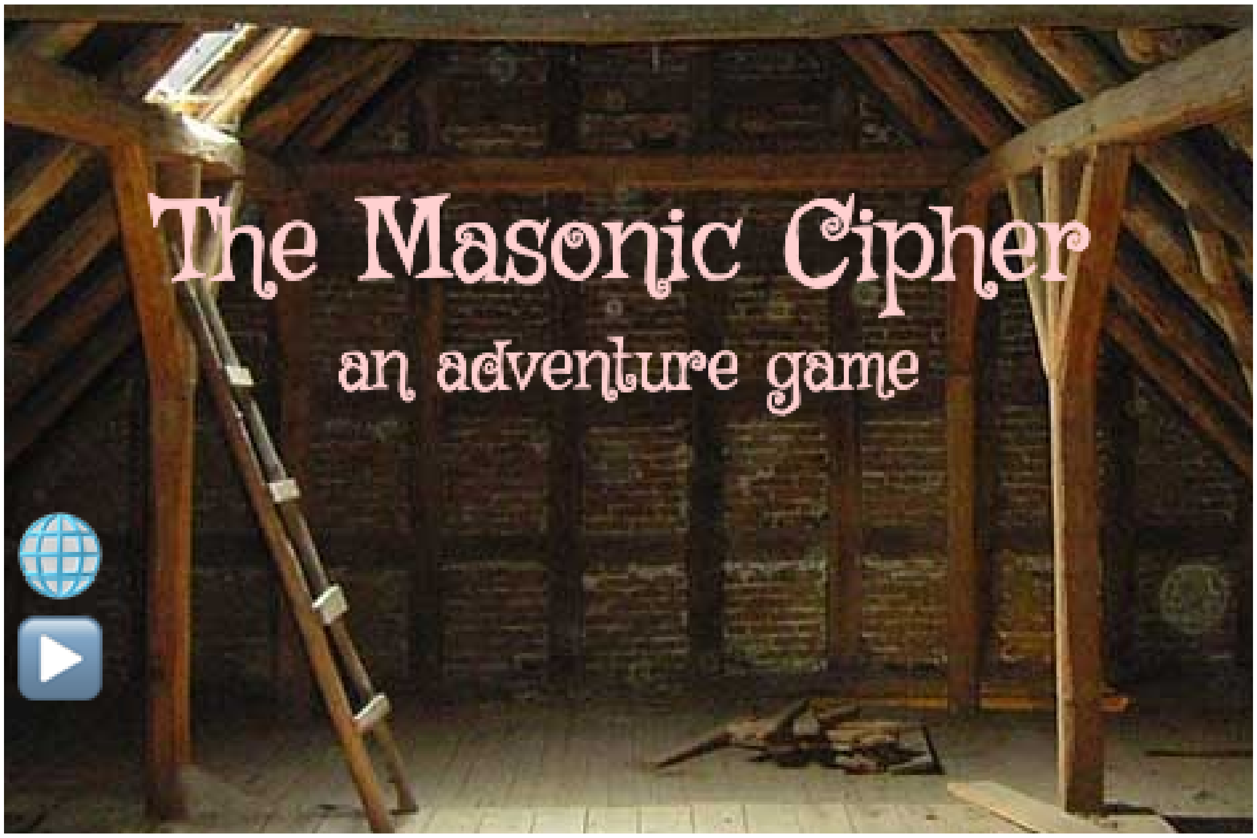
In this section we are going to program an adventure game. The game has a main character who narrates the story (**Boy**). At certain points the player must solve a puzzle to advance in the story.

There is a secondary character (**Girl**) in the story.

A Scratch program can easily get very complicated if the code is spread over several sprites. To make the program understandable, you can either

- place the story line into the stage (backdrops), or
- use a main character to narrate the story

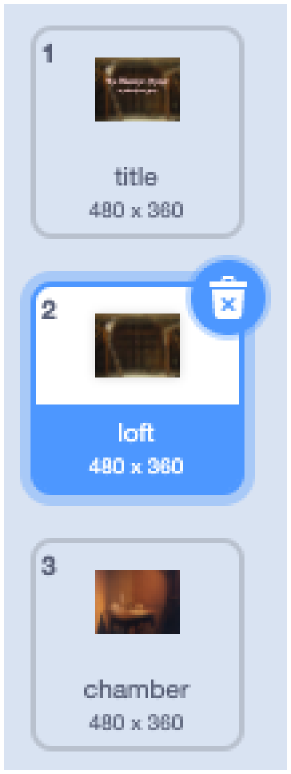
Here we choose a main charactor to narrate the story.



21.1 Backdrops

To organize the story, we place all the backdrops in the order of appearance :

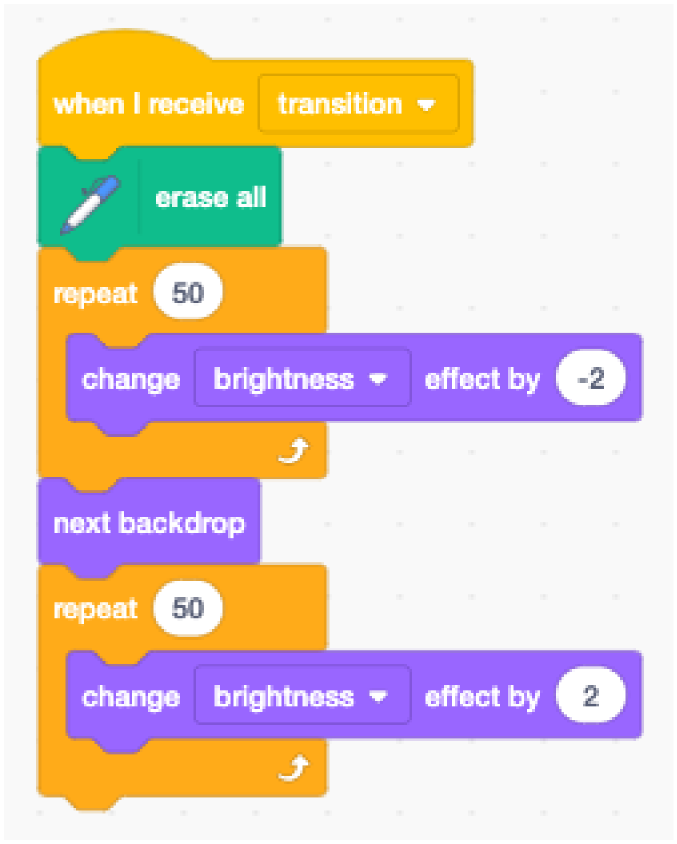
- — title
- — loft
- — chamber
- — museum



Since the story is driven by the main sprite (**Boy**), there is almost no code here. The only code is a slow transition from the current backdrop to the next one. For this to work, the backdrops need to be in the correct order.

The intro screen shows the title and two buttons :

- select language
- start the game

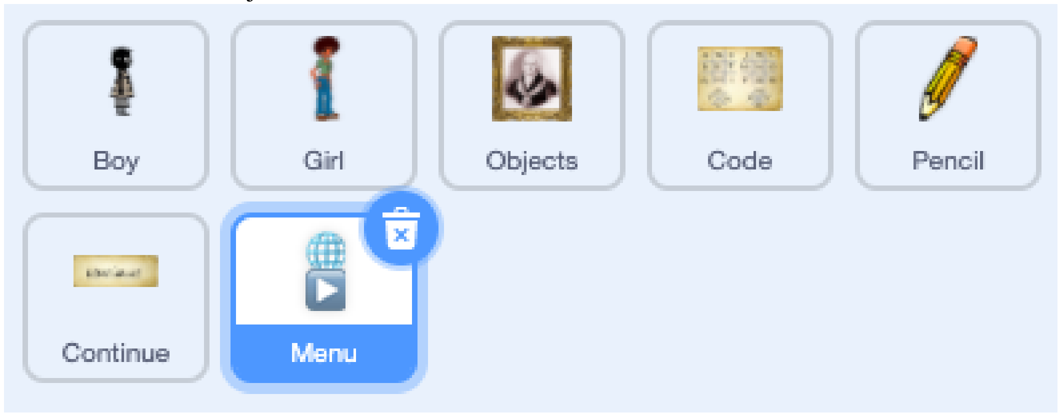


```
when I receive [transition ▾]
erase all
repeat (50)
    change [brightness ▾] effect by (-2)
next backdrop
repeat (50)
    change [brightness ▾] effect by (2)
```

21.2 Sprites

In this game we have 7 sprites :

— the actors : **Boy** and **Girl**
— the objects : **Objects**, **Code**, **Pencil**
— the interface objects : **Continue**, **Menu**

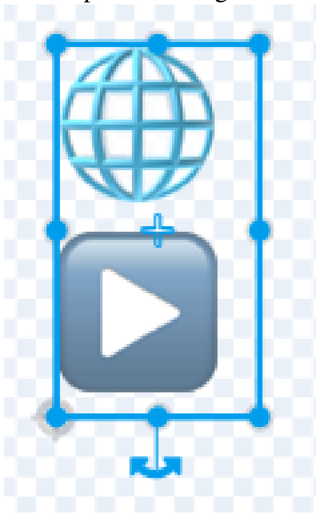


21.3 The menu

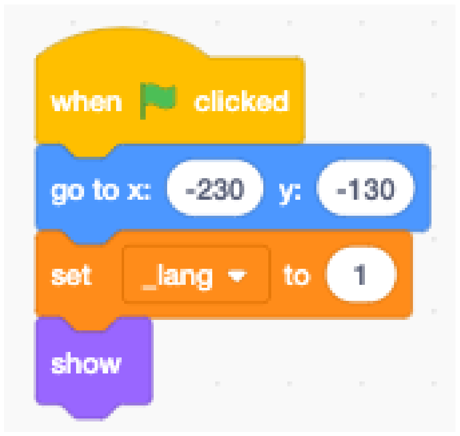
In this game we have a simple menu with two buttons. We have used emoticons to create them. This is an easy way to create them. They are universally understood, and it's easy to add more buttons.

Pay attention to these things :

— make the height of each icon 40 points (in this case the total height is 80 points)
— add an extra space after the first icon (to avoid cropping icons in Android, which has larger icons than iOS)
— place the origin at the lower left cornder

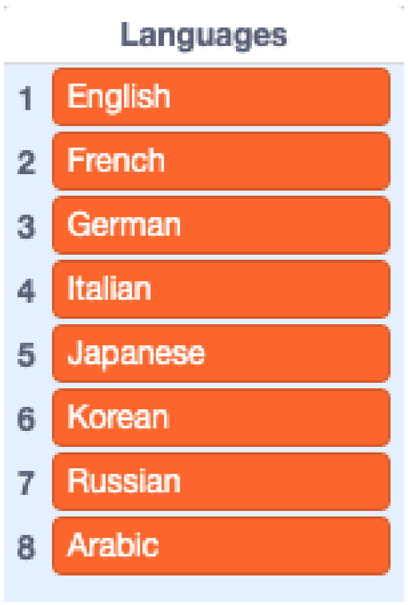


At start we place the icons and set the language to the first language (English).



```
when flag clicked
go to x: -230 y: -130
set _lang to 1
show
```

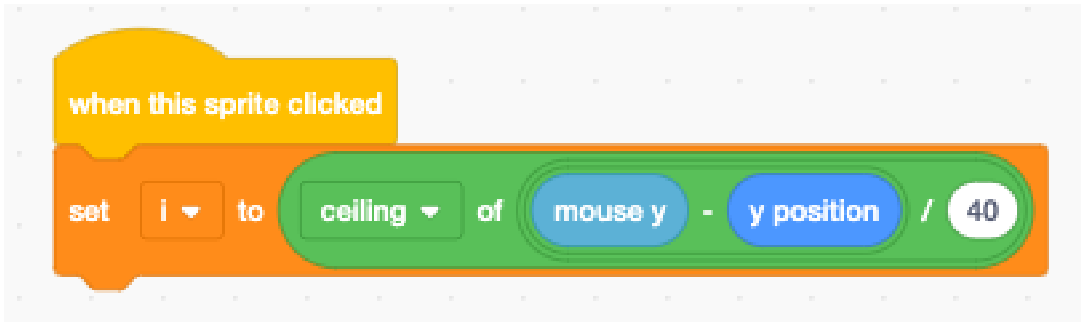
The languages of the game are kept in a list. We use the variable **_lang** as an index.



Languages

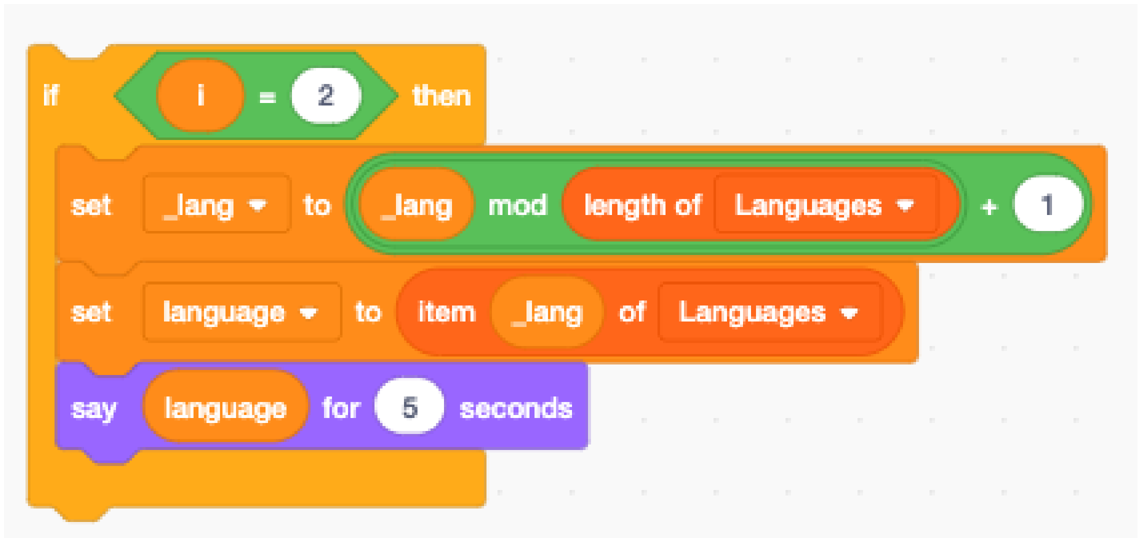
1	English
2	French
3	German
4	Italian
5	Japanese
6	Korean
7	Russian
8	Arabic

When the sprite is clicked, we use the **mouse y** position to calculate the index of the button.



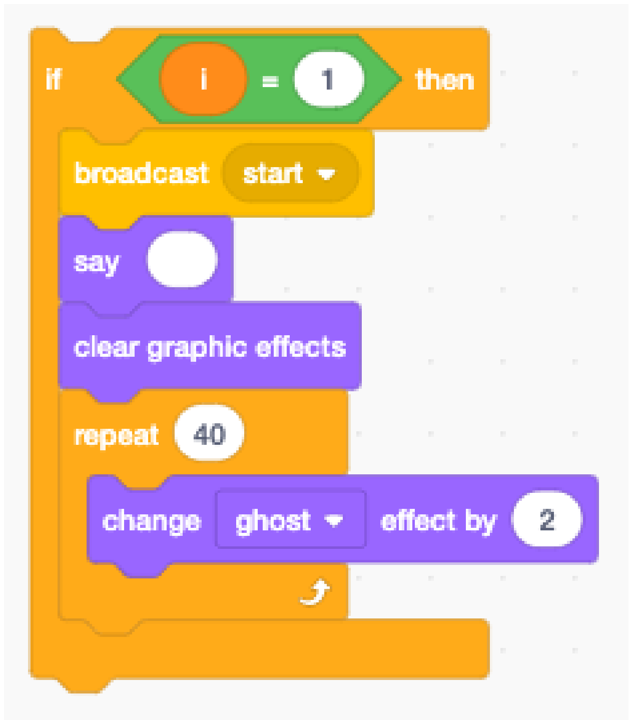
```
when this sprite clicked
set i to ceiling of ((mouse y - y position) / 40)
```

If the **language** icon is clicked, we iterate to the next language in the language list. The selected language is displayed for 5 seconds.



```
if i = 2 then
    set _lang to ((_lang mod length of Languages) + 1)
    set language to item _lang of Languages
    say language for 5 seconds
```

If the **start** icon is clicked, we make the make the buttons almost transparent and start the game. We keep the buttons in a very transpartent state on purpose, in order to change the languge or restart during the game.

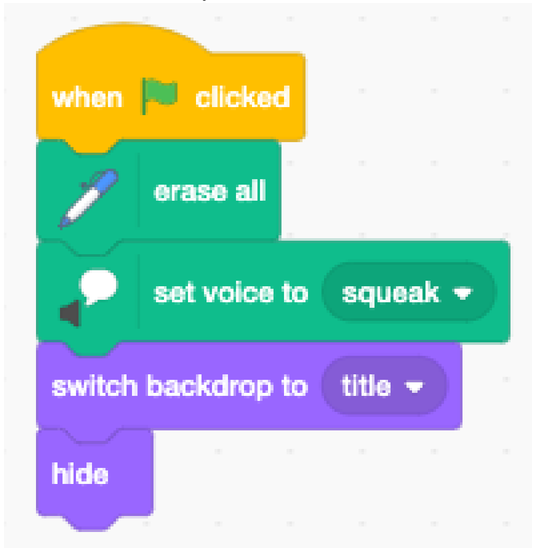


```
if (i) = (1) then
    broadcast start
    say ()
    clear graphic effects
    repeat 40
        change ghost effect by 2
```

21.4 The narrator

At the start of the game, we :

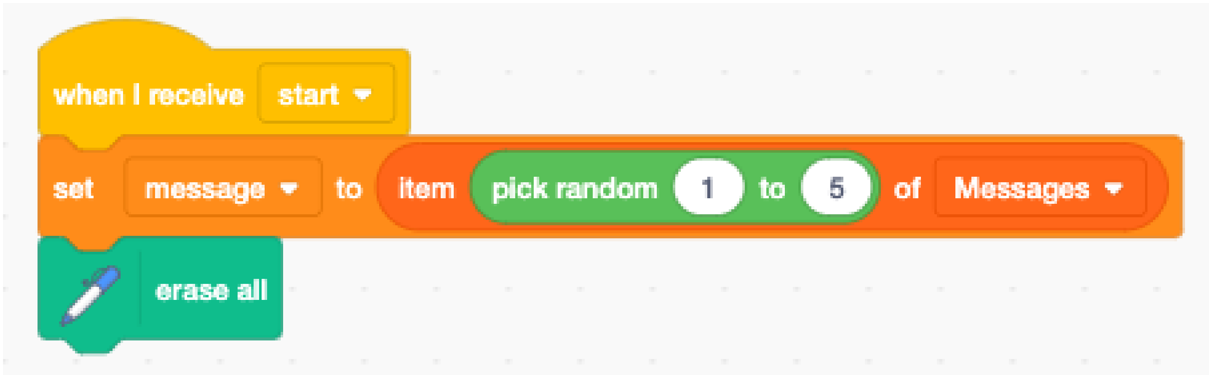
- — display the title screen
- — set the boy's voice to squeak
- — hide the boy



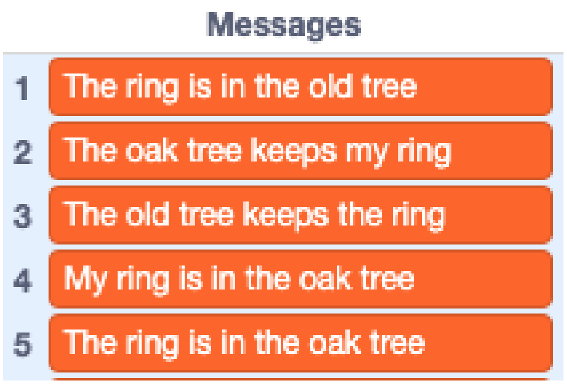
```
when ⚑ clicked
erase all
set voice to squeak
switch backdrop to title
hide
```

21.5 Choose a secret message

When receiveing the **start** message the game starts. We start by choosing a random secret message.



The **Message** list contains 5 possible messages.

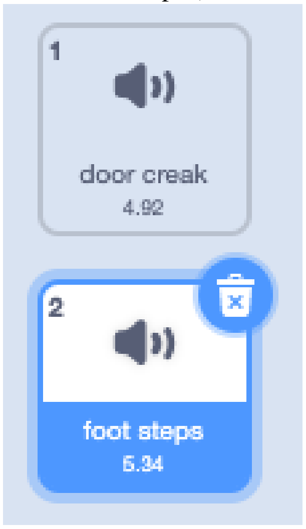

Messages

1	The ring is in the old tree
2	The oak tree keeps my ring
3	The old tree keeps the ring
4	My ring is in the oak tree
5	The ring is in the oak tree

21.6 Overlapping sounds

The story starts with an empty loft. Sounds prepare the entry of the main character.

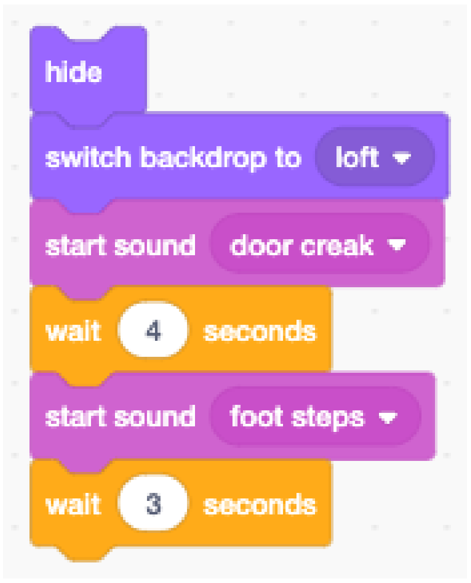
These two sounds are used :
- door creak (4.9 seconds)
- foot steps (5.3 seconds)



To create an atmosphere sounds often overlap.

We start the second sound after 4 secondes. That gives us 0.9 seconds where the door and the foot-steps overlap.

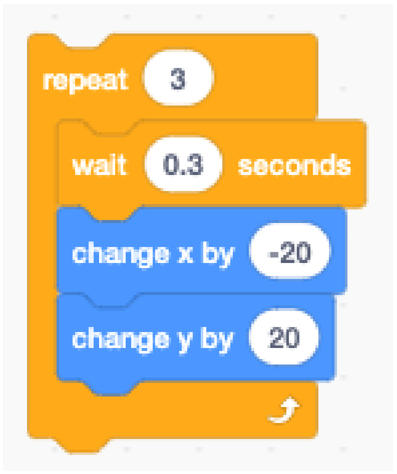
After the second sound we wait 3 secondes. Then the main character appears. During 2.3 seconds, the footstep sound overlaps with the character movement.



```
hide
switch backdrop to loft
start sound door creak
wait 4 seconds
start sound foot steps
wait 3 seconds
```

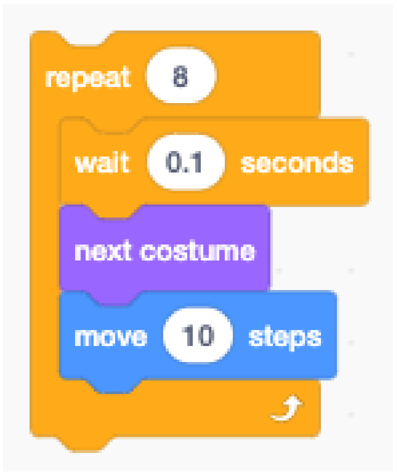
21.7 Walking up the stairs

To create the illusion that the boy is walking up the stairs, we make him move by (-20, -20) points and stop for 0.3 seconds.



```
repeat 3
    wait 0.3 seconds
    change x by -20
    change y by 20
```

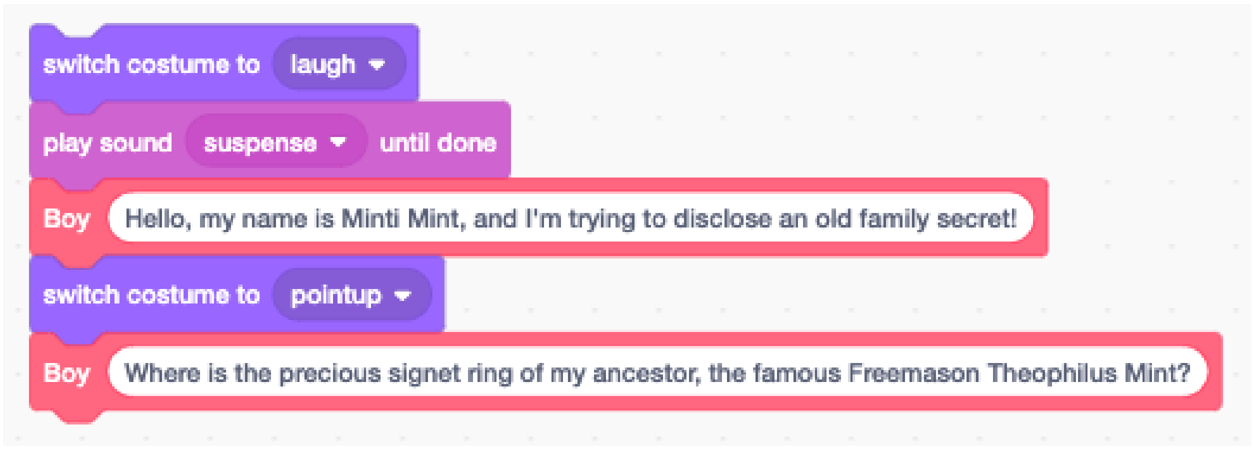
The first 9 costumes are a walking sequence. We repeat these costumes and move the charactor to the right.



```
repeat (8)
    wait (0.1) seconds
    next costume
    move (10) steps
end
```

21.8 Look, sound and speech

We accompany phrases with gesture (laugh, point up). Sounds are interspersed in between phrases. The **suspense** is played to the end, before the conversation continues.

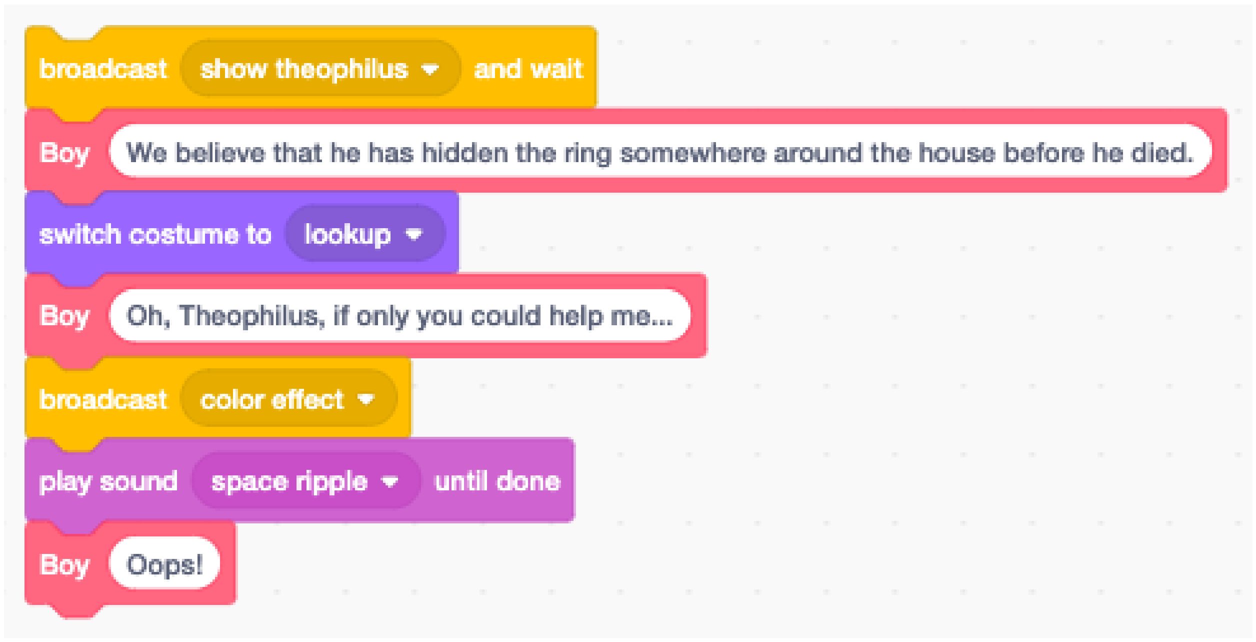


```
switch costume to (laugh ▾)
play sound (suspense ▾) until done
Boy (Hello, my name is Minti Mint, and I'm trying to disclose an old family secret!)
switch costume to (pointup ▾)
Boy (Where is the precious signet ring of my ancestor, the famous Freemason Theophilus Mint?)
```

21.9 Interact with objects

The objects (Theophilus" portrait, knob, ring) are in a different sprite. The only way to interact with these objects is to send messages. We send the message

— **show theophile** to make the portrait appear,
— **color effect** to add a color animation.

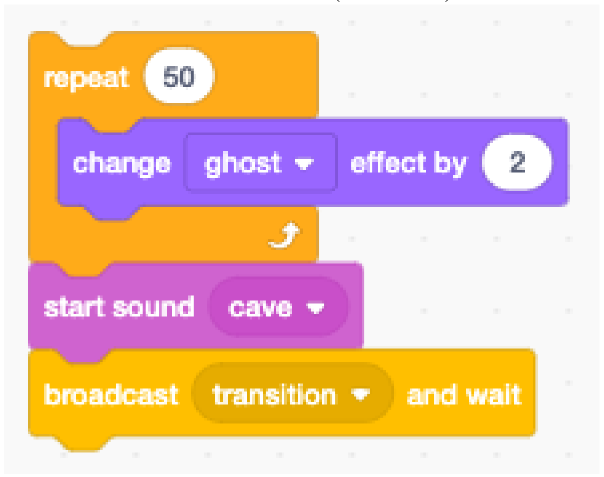


```
broadcast [show theophilus v] and wait
Boy (We believe that he has hidden the ring somewhere around the house before he died.)
switch costume to [lookup v]
Boy (Oh, Theophilus, if only you could help me...)
broadcast [color effect v]
play sound [space ripple v] until done
Boy (Oops!)
```

21.10 Transition

The next seend is often anticipated with sound. Here we start the scene transition with the **cave** sound have a

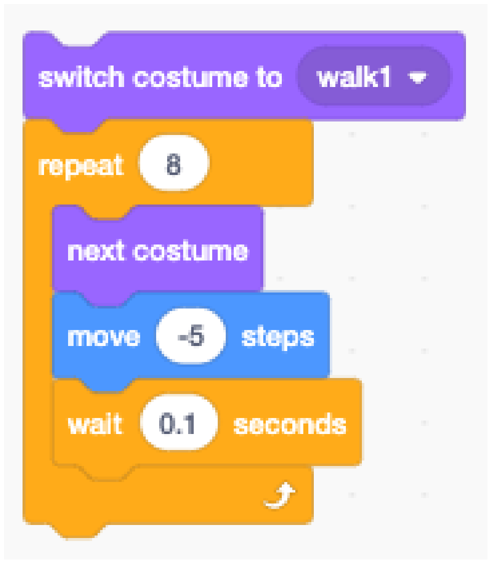
- — fade-out of the loft (2 seconds)
- — fade-in of the chamber (2 seconds)



```
repeat (50)
    change [ghost v] effect by (2)
end
start sound [cave v]
broadcast [transition v] and wait
```

21.11 Walk backwards

We use the 9 costumes of the walking sequence again, with a -5 point displacement, to simulate backward walking.

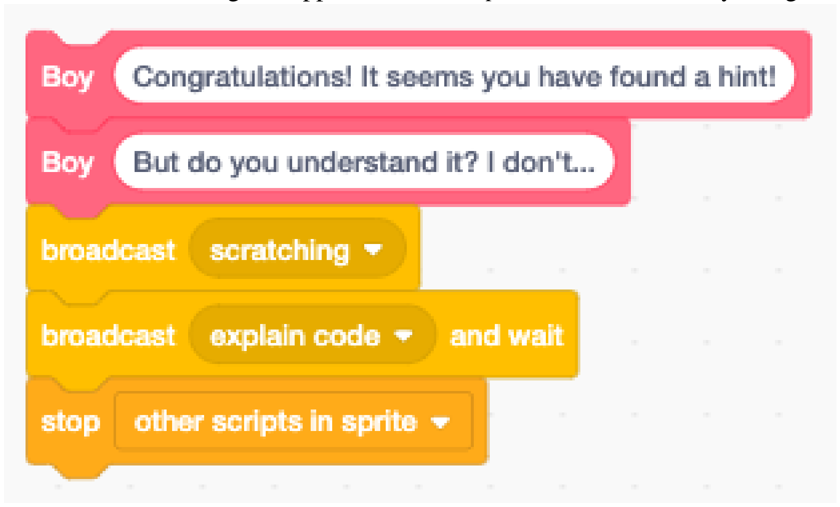


```
switch costume to walk1
repeat 8
    next costume
    move -5 steps
    wait 0.1 seconds
```

21.12 Parallel actions

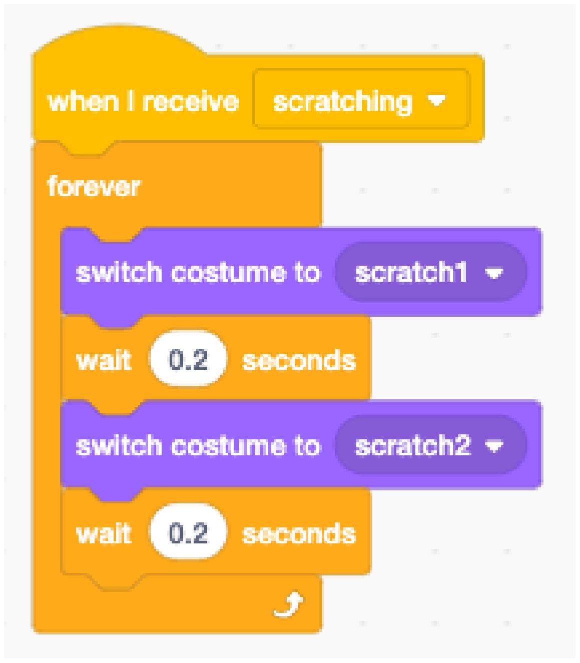
Parallel actions need to run in a separate stack. Here we want the boy to scratch his head, while he is listening to the explanation of the code chart.

— The broadcast **scratching** starts the parallel action.
— In parallel, the new code sprite appears and a variable-length explanation follows.
— The scratching is stopped when the *explain* action returns, by using the **stop other scripts in sprite** block.



```
Boy Congratulations! It seems you have found a hint!
Boy But do you understand it? I don't...
broadcast scratching
broadcast explain code and wait
stop other scripts in sprite
```

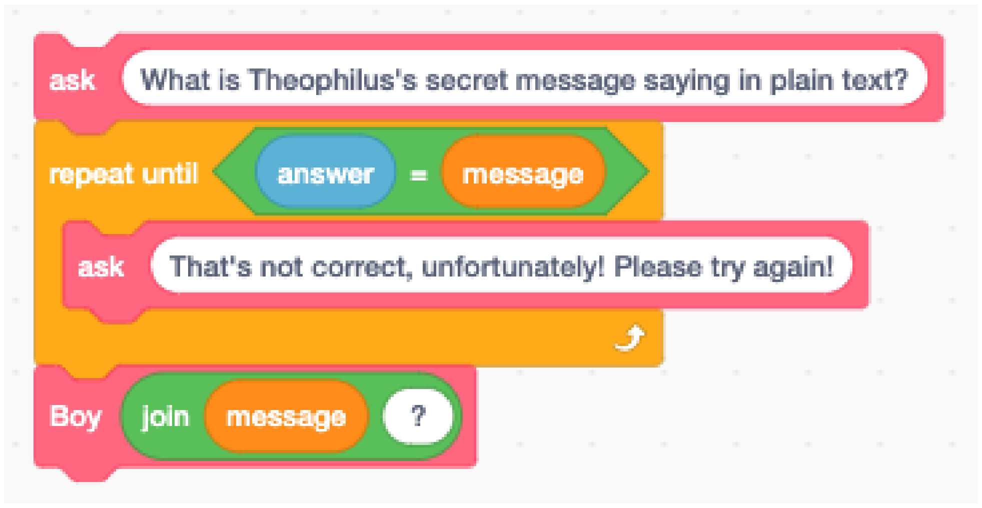
The scratch animation is done with only 2 costumes. Since the duration is variable, a **forever** loop is used. The loop is forcefully stopped with a **stop other scripts** block.



```
when I receive [scratching ▾]
forever
    switch costume to [scratch1 ▾]
    wait (0.2) seconds
    switch costume to [scratch2 ▾]
    wait (0.2) seconds
```

21.13 Ask for secret message

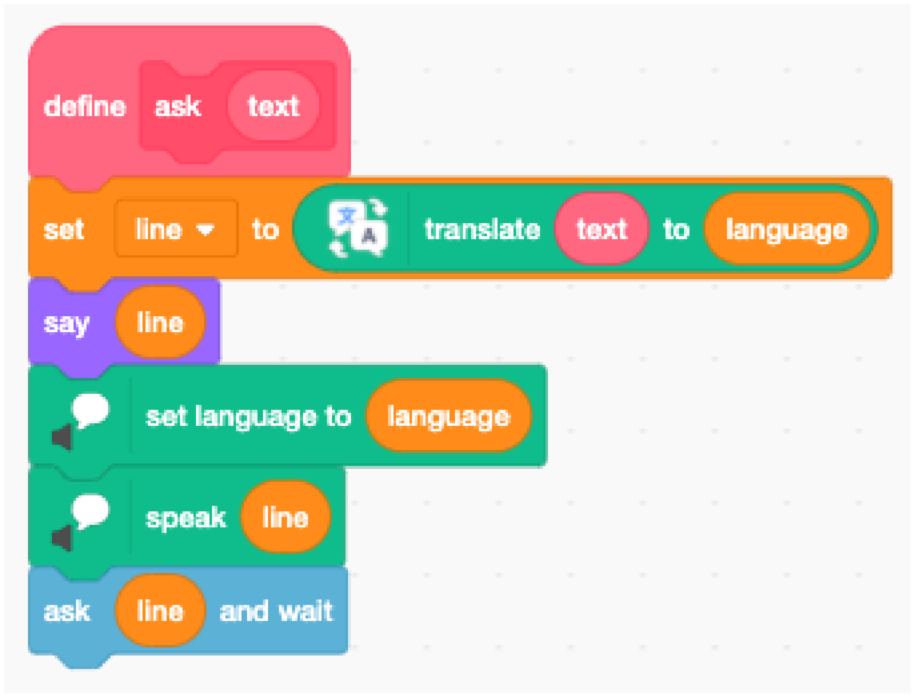
This code asks the user to decipher the secret message. The program only can advance if the message is correctly found.



```
ask (What is Theophilus's secret message saying in plain text?)
repeat until <(answer) = (message)>
    ask (That's not correct, unfortunately! Please try again!)
Boy (join (message) (?))
```

The **ask** function asks the question. It :

- — translates the text
- — writes it to the speech bubble
- — prononces it
- — displays it again and opens the **ask** dialog.



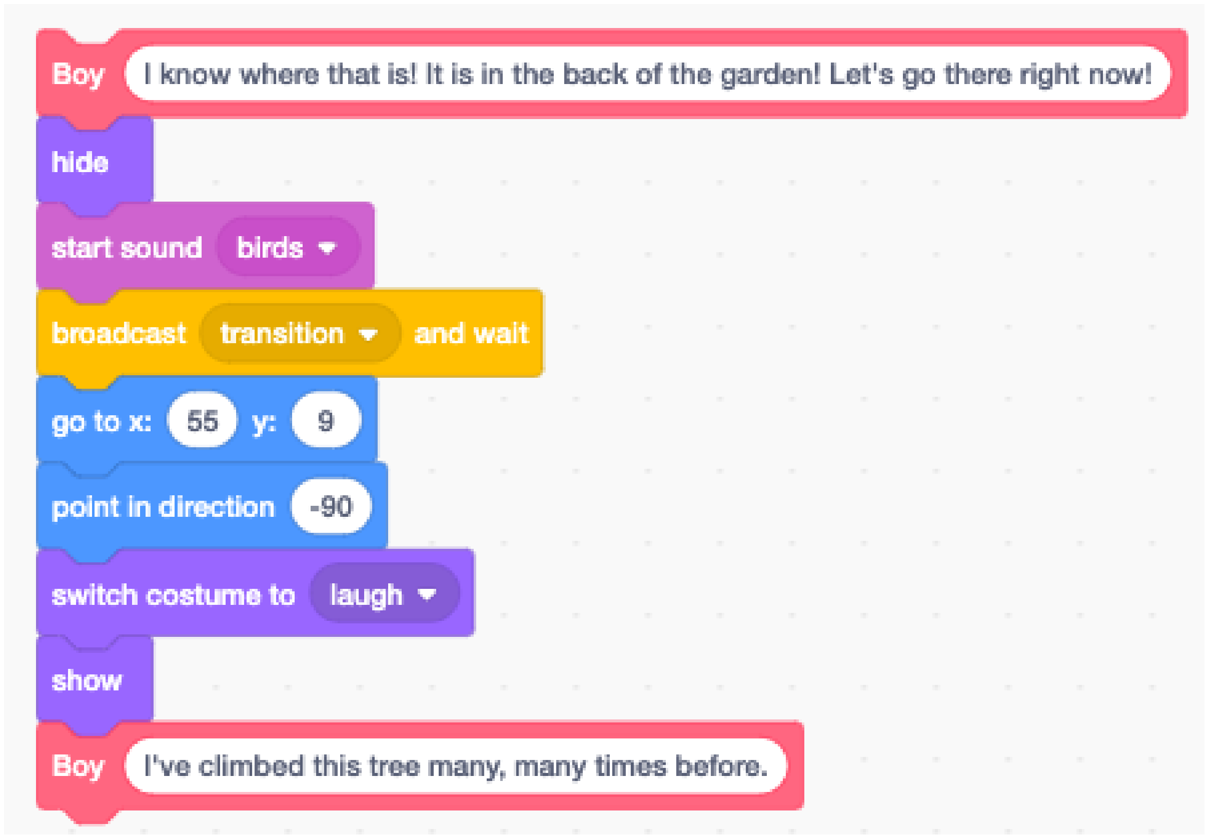
```
define ask (text)
set [line ▾] to (translate (text) to (language))
say (line)
set language to (language)
speak (line)
ask (line) and wait
```

21.14 Scene transition

Let’s look at the scene transition technique used :

Being at the end of the *chamber scene* :

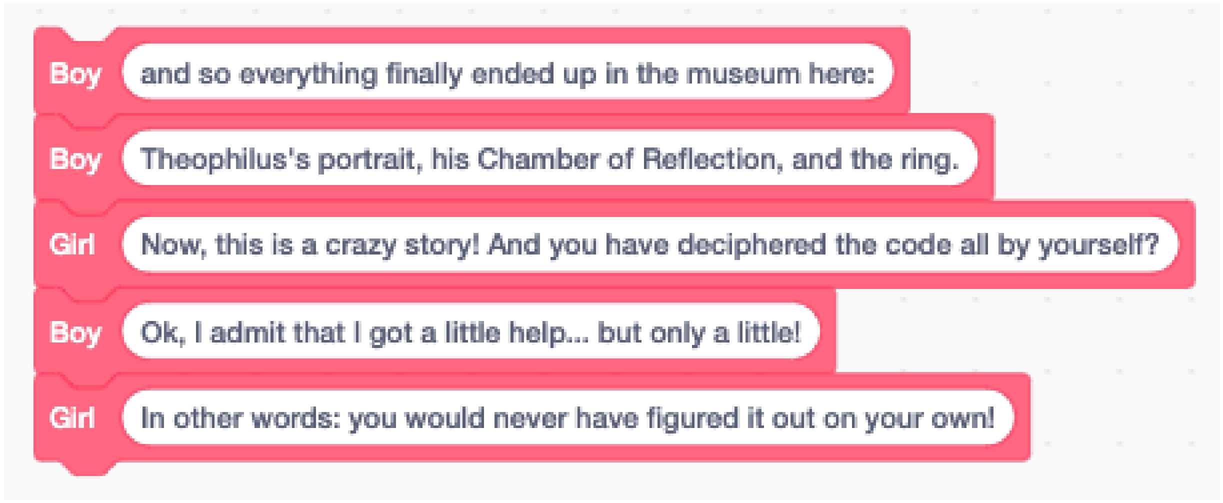
- the character talks about the next scene (I know the old tree)
- the character hides
- bird sounds start
- the chamber fades out
- the tree fades in
- the character is positioned
- the character appears in the new place
- The character talks about the new scene (I was here before)



```
Boy I know where that is! It is in the back of the garden! Let's go there right now!
hide
start sound birds ▾
broadcast transition ▾ and wait
go to x: 55 y: 9
point in direction -90
switch costume to laugh ▾
show
Boy I've climbed this tree many, many times before.
```

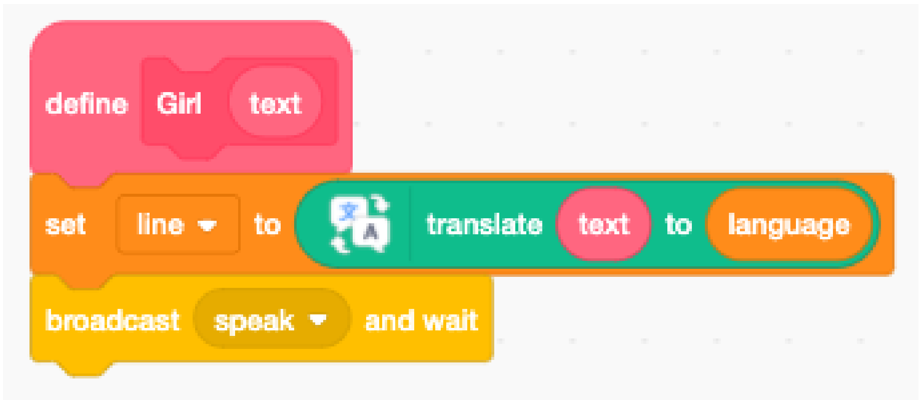
21.15 Talk between characters

In the *museum scene* we have a dialog between the boy and the girl. Speech bubbles only can be called in the code that belongs to that sprite.



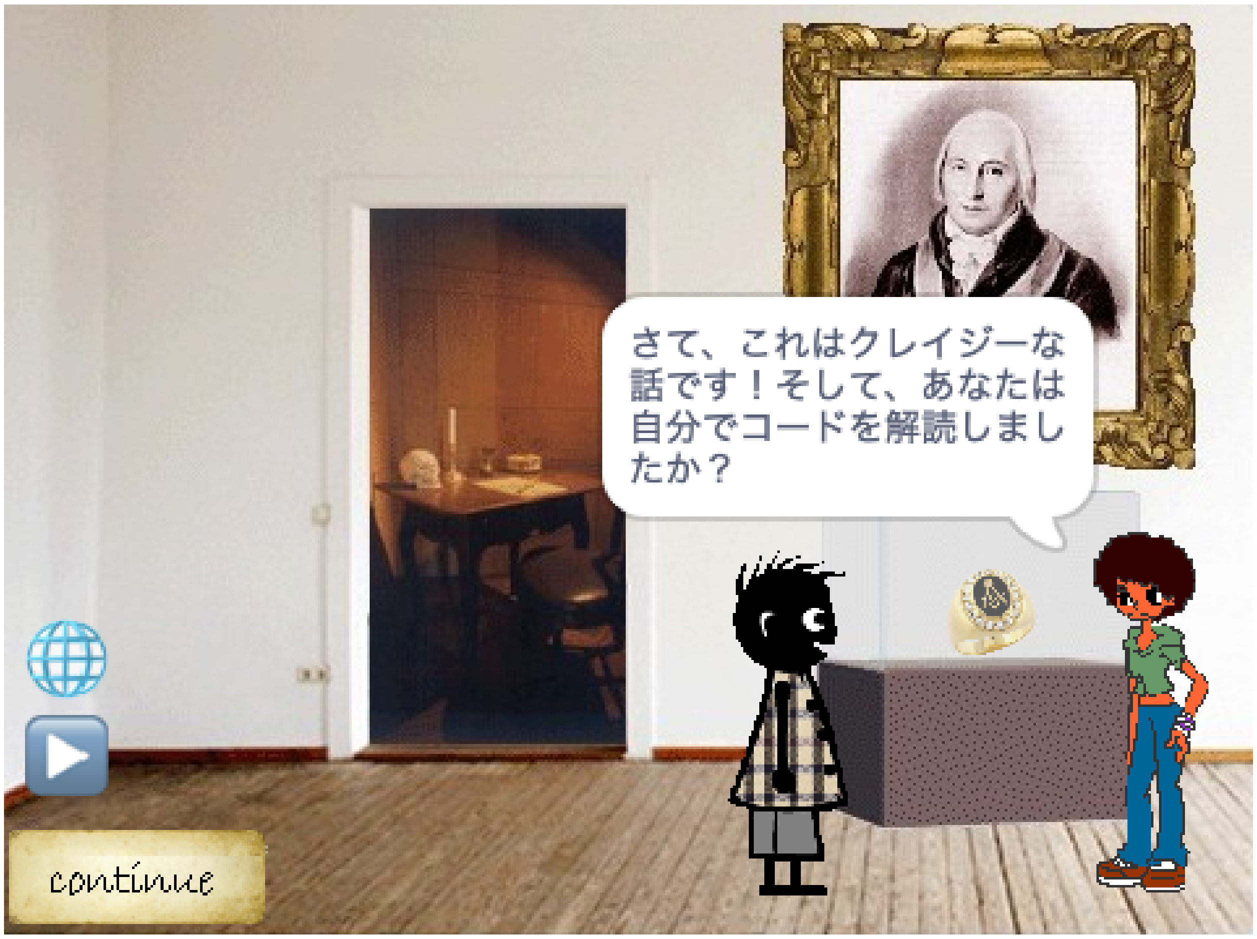
```
Boy and so everything finally ended up in the museum here:
Boy Theophilus's portrait, his Chamber of Reflection, and the ring.
Girl Now, this is a crazy story! And you have deciphered the code all by yourself?
Boy Ok, I admit that I got a little help... but only a little!
Girl In other words: you would never have figured it out on your own!
```

We use the message **speak** to call code belonging to the **Girl** sprite. The variable **line** contains the text to be spoken.



```
define Girl (text)
set [line ▾] to (translate (text) to (language))
broadcast [speak ▾] and wait
```

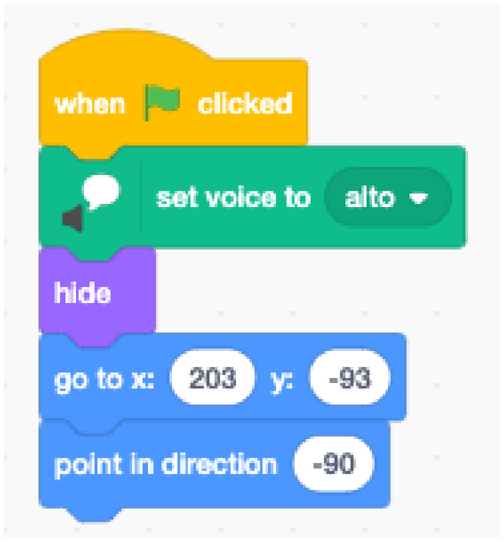
This is a talking scene in the museum, in Japanese.


さて、これはクレイジーな話です！そして、あなたは自分でコードを解読しましたか？
continue

21.16 The girl sprite

The code of **Girl** sprite is very short. The girl only appears in the *museum scene* and does not move around.

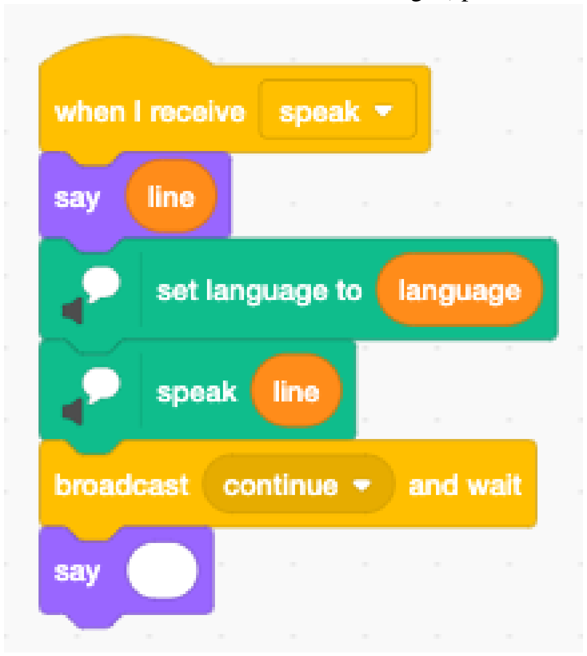
At the start the girl is placed, facing to the left, and hidden.



```
when 🏴 clicked
set voice to alto ▾
hide
go to x: 203 y: -93
point in direction -90
```

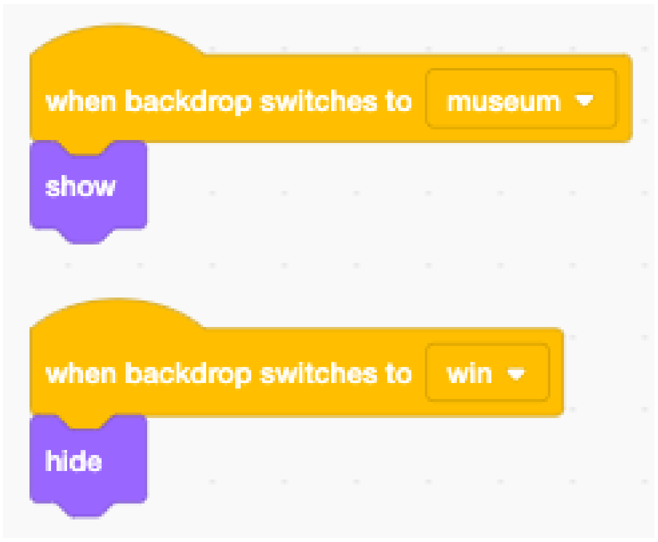
When it receives the **speak** broadcast, it

— displays the **line** in a speech bubble
— pronounces it in the chosen language
— broadcasts the **continue** message (space or click)



```
when I receive speak ▾
say line
set language to language
speak line
broadcast continue ▾ and wait
say
```

The sprite only appears in the museum scene. In the last scene the sprite is hidden.

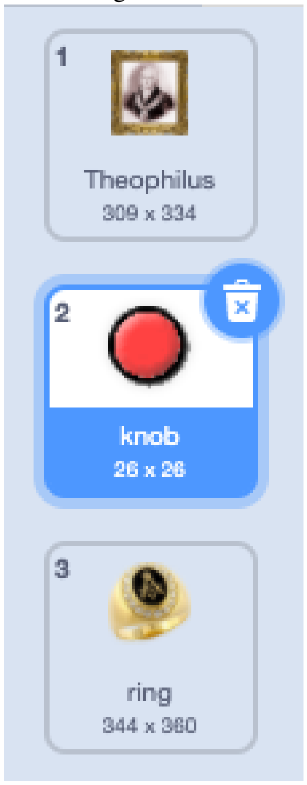


```
when backdrop switches to [museum ▾]
show

when backdrop switches to [win ▾]
hide
```

21.17 Interact with objects

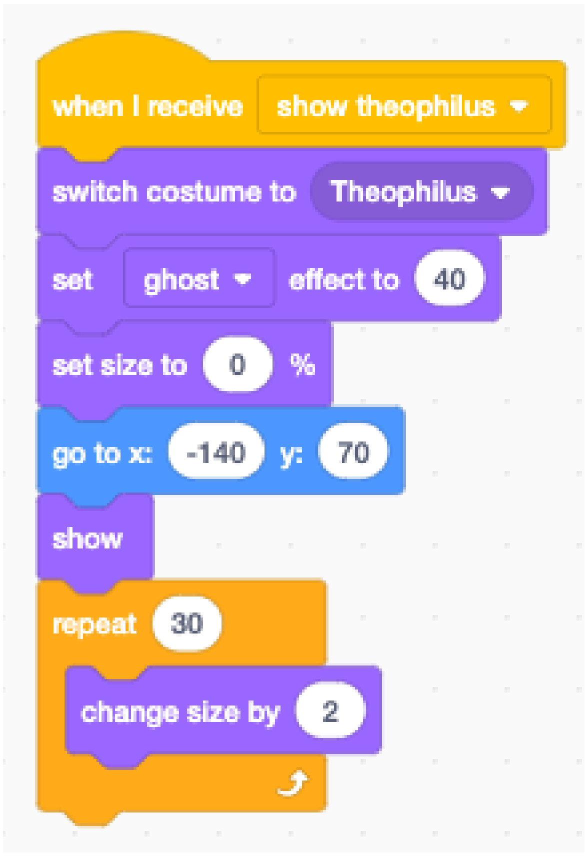
The **Obejects** sprite contains costumes of different objects

- Theophilus
- knob
- ring



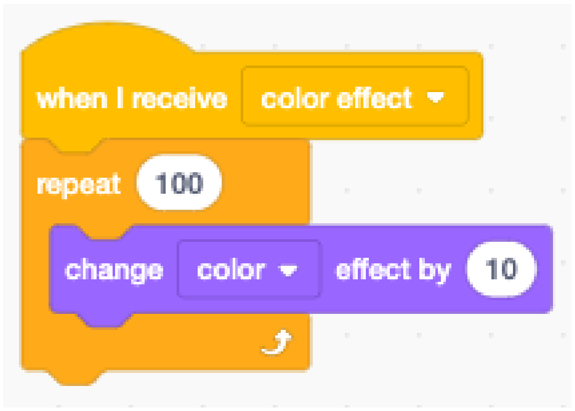
These objects are not animated and appear not simultaneously.

At the start all objects are hidded. Upon reception of ** show Theodophilus** the sprite blows up from 0% in size to 60%.



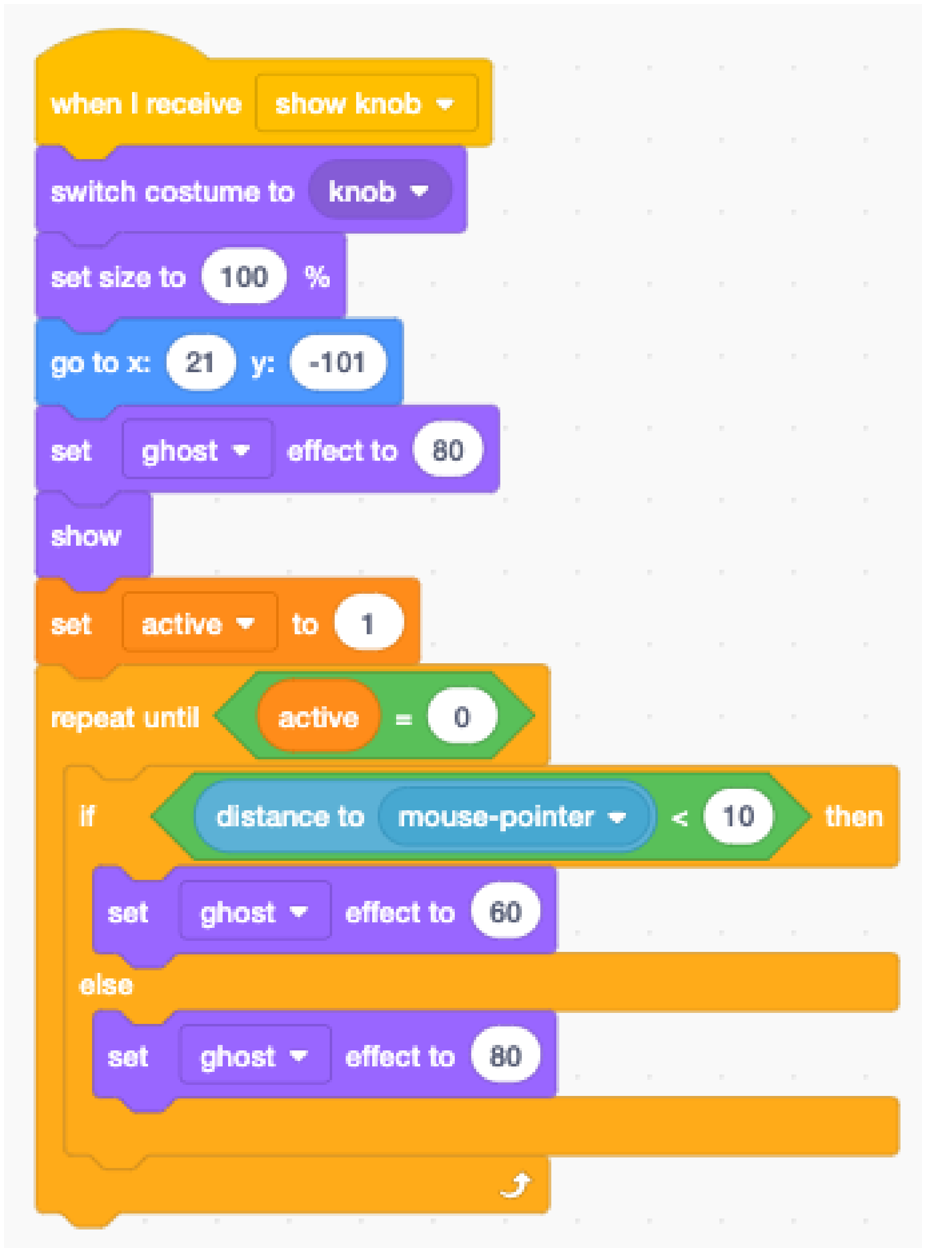
```
when I receive [show theophilus ▾]
switch costume to [Theophilus ▾]
set [ghost ▾] effect to (40)
set size to (0) %
go to x: (-140) y: (70)
show
repeat (30)
    change size by (2)
```

The message **color effect** triggers a color effect



```
when I receive [color effect ▾]
repeat (100)
    change [color ▾] effect by (10)
```

The knob of the drawer appears with a ghost effect of 80%. When the mouse comes near, the ghost effect changes and the knob becomes more solid.

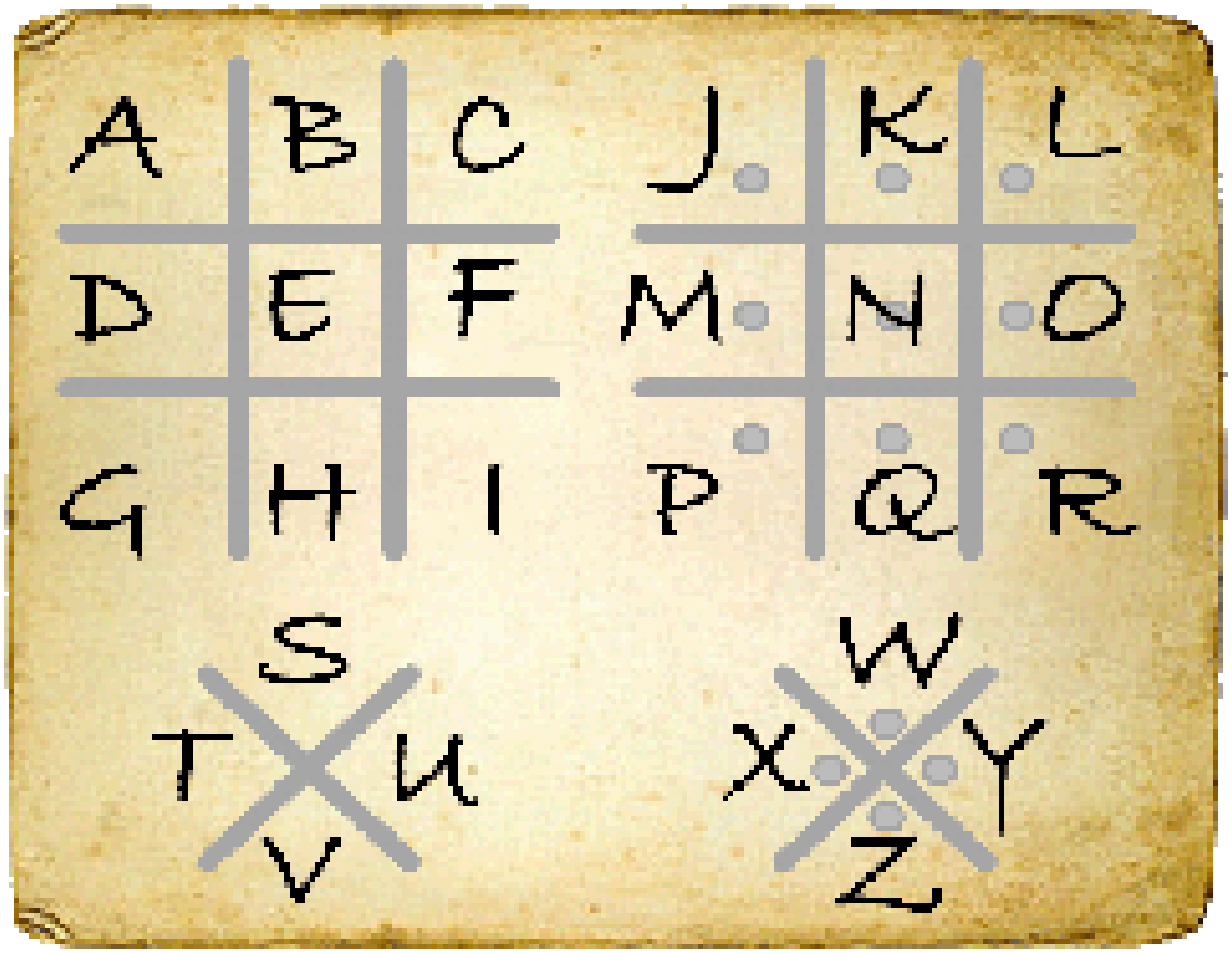


```
when I receive [show knob ▾]
switch costume to (knob ▾)
set size to (100) %
go to x: (21) y: (-101)
set [ghost ▾] effect to (80)
show
set [active ▾] to (1)
repeat until <(active) = (0)>
    if <(distance to (mouse-pointer ▾)) < (10)> then
        set [ghost ▾] effect to (60)
    else
        set [ghost ▾] effect to (80)
```

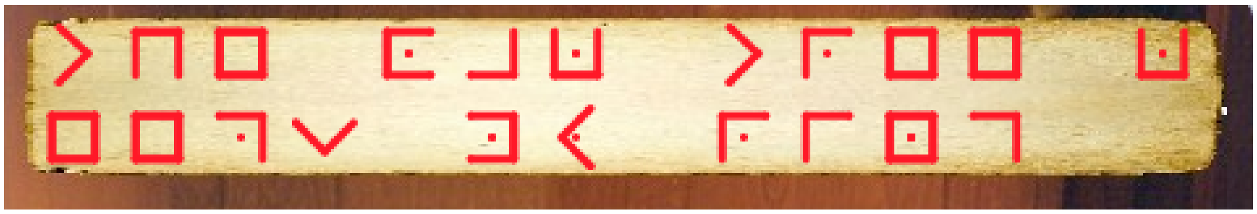
21.18 The cipher

The **Pencil** sprite translates the text in the **message** variable to a written code.

The pigpen cipher or masonic cipher exchanges letters for geometric symbols. The first 2x9 letters are arranged in a tic-tac-toe shape. The following 2x4 letters of the alphabet are arranged in an x shape. The surround pattern is forms the secret symbol. The second group is designated with an added dot.



For example the message *the oak tree keeps my ring* is expressed as this :



21.19 Play the game

This is the game :

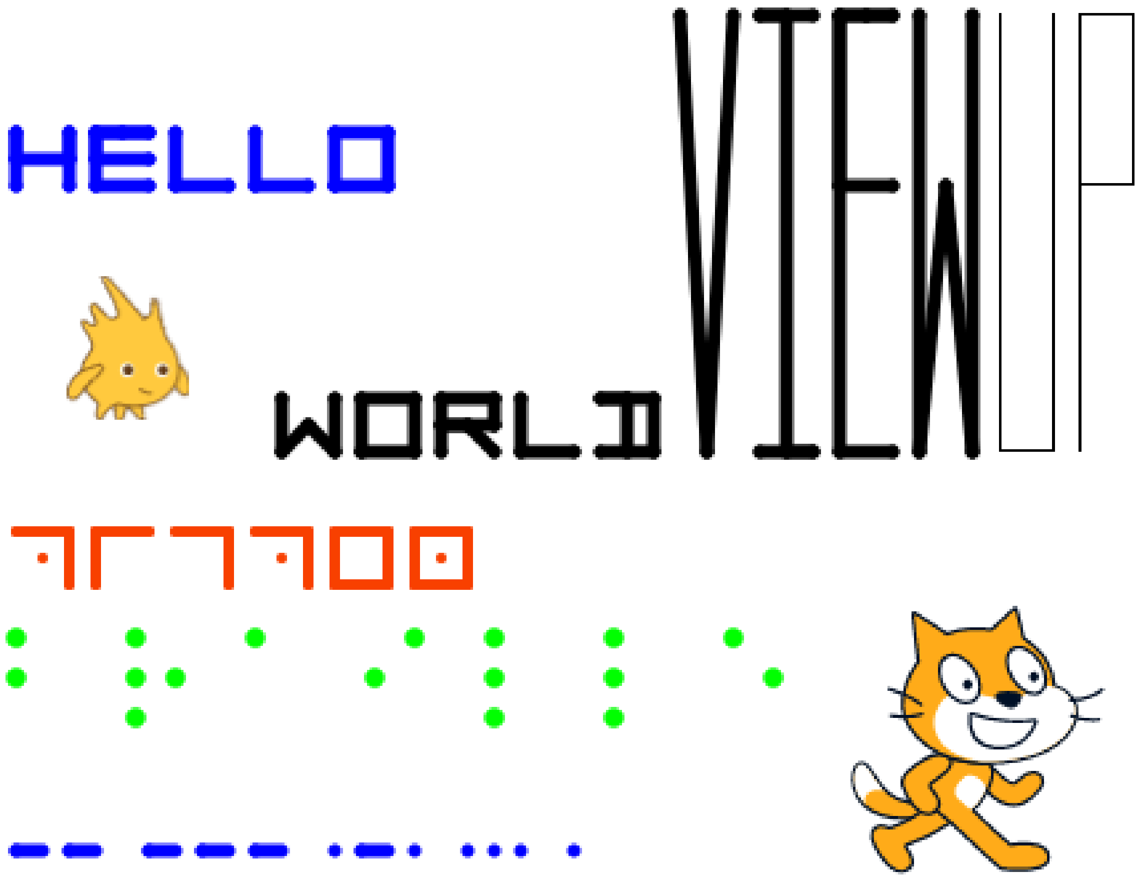
https://scratch.mit.edu/projects/402279634

CHAPITRE 22

Codes

In this chapter we look at how to display different codes. We are looking at

- 16 segment displays
- Pigpen code
- Braille code
- Morse code



https://scratch.mit.edu/projects/404785655

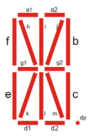
22.1 16-segment code

A sixteen-segment dispaly is a type of diasplay based on 16 segments which can individually be turned on and off. A 16-bit binary code encodes and represents all possbile characters.

For example the letter O could be represented with this 16-bit code :

```
1110001001000111
```

The segments are in the order from left to right, and top to down.



The following image shows how letters, numbers and special symbols can be represented.

Scratch cannot draw characters to the screen. There are two ways to print text in scratch :
- use a sprite for each letter
- draw the letter with the pen

22.2 Encoding the symbols

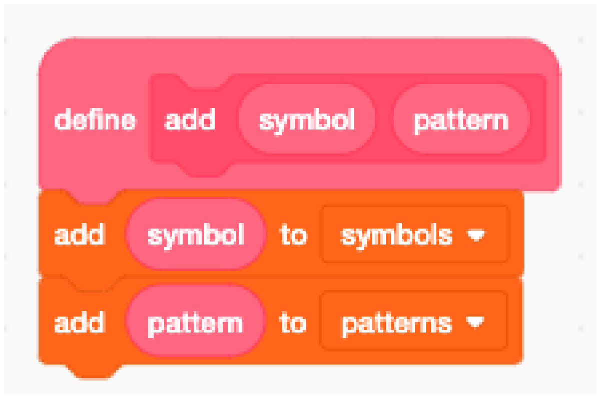
When drawing each letter with the pen, we need to encode somehow the geometric shape of the letter. We are going to use a system inspired by the 16 segment display. We use the 3x3 points of intersection and encode the number with a path going through these points.

The points are numbered like this :

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

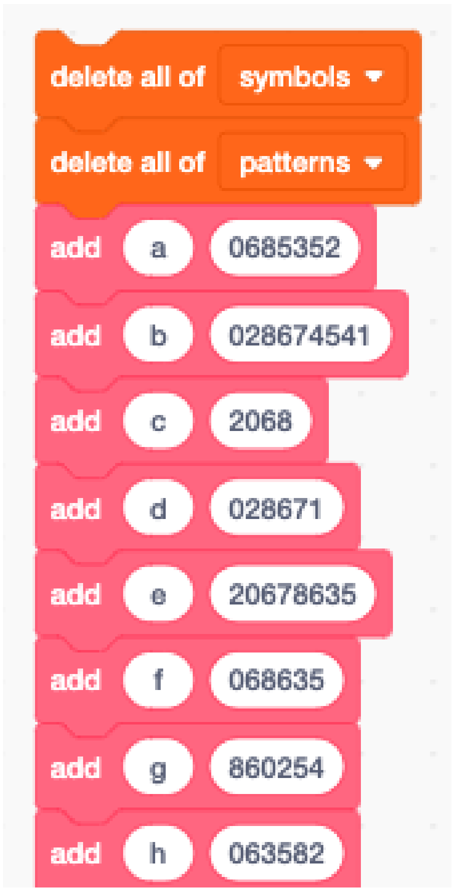
To store the information about how to draw the symbbols or caracters, we will use 2 lists :
- symbols
- patterns

The function **add** associations a symbol with a pattern and adds them to the two lists.



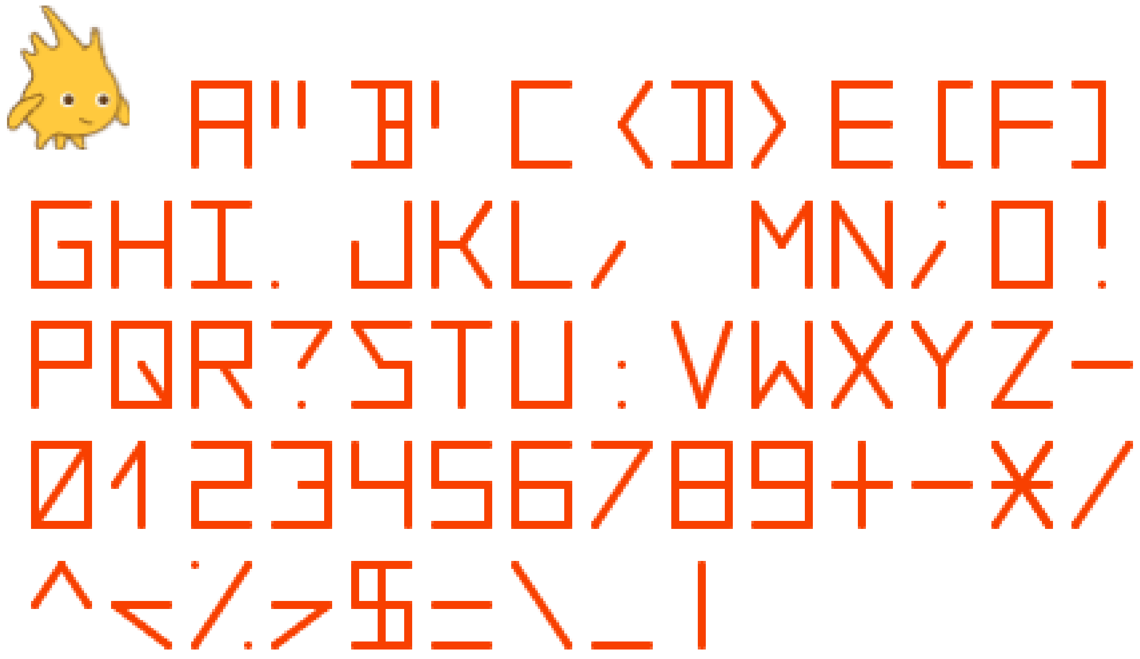
```
define add (symbol) (pattern)
add (symbol) to [symbols ▾]
add (pattern) to [patterns ▾]
```

The first letters from *a* to *h* are encoded this way.



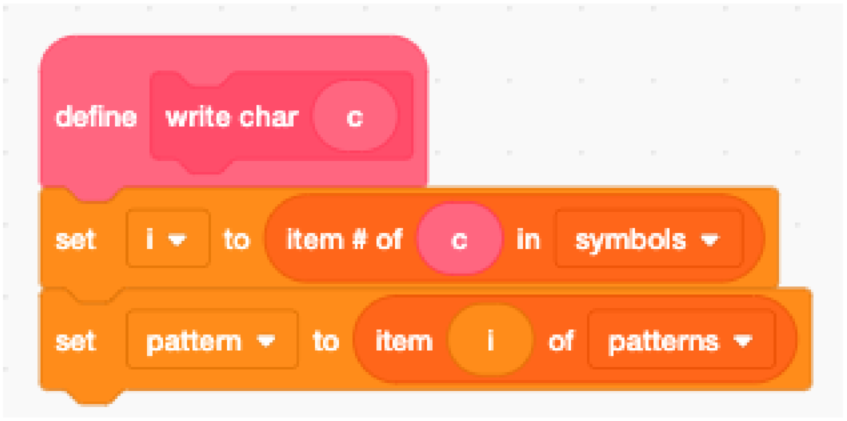
```
delete all of symbols
delete all of patterns
add a 0685352
add b 028674541
add c 2068
add d 028671
add e 20678635
add f 068635
add g 860254
add h 063582
```

This shows all the characters which have been encoded. More characters could easily be added if needed.



22.3 Draw a character

When drawing a specific symbol, we must first retrieve its code. This is done with the **find in list** block, which returns the index **i** of the pattern. In a second step we can look up the pattern associated with the symbol.



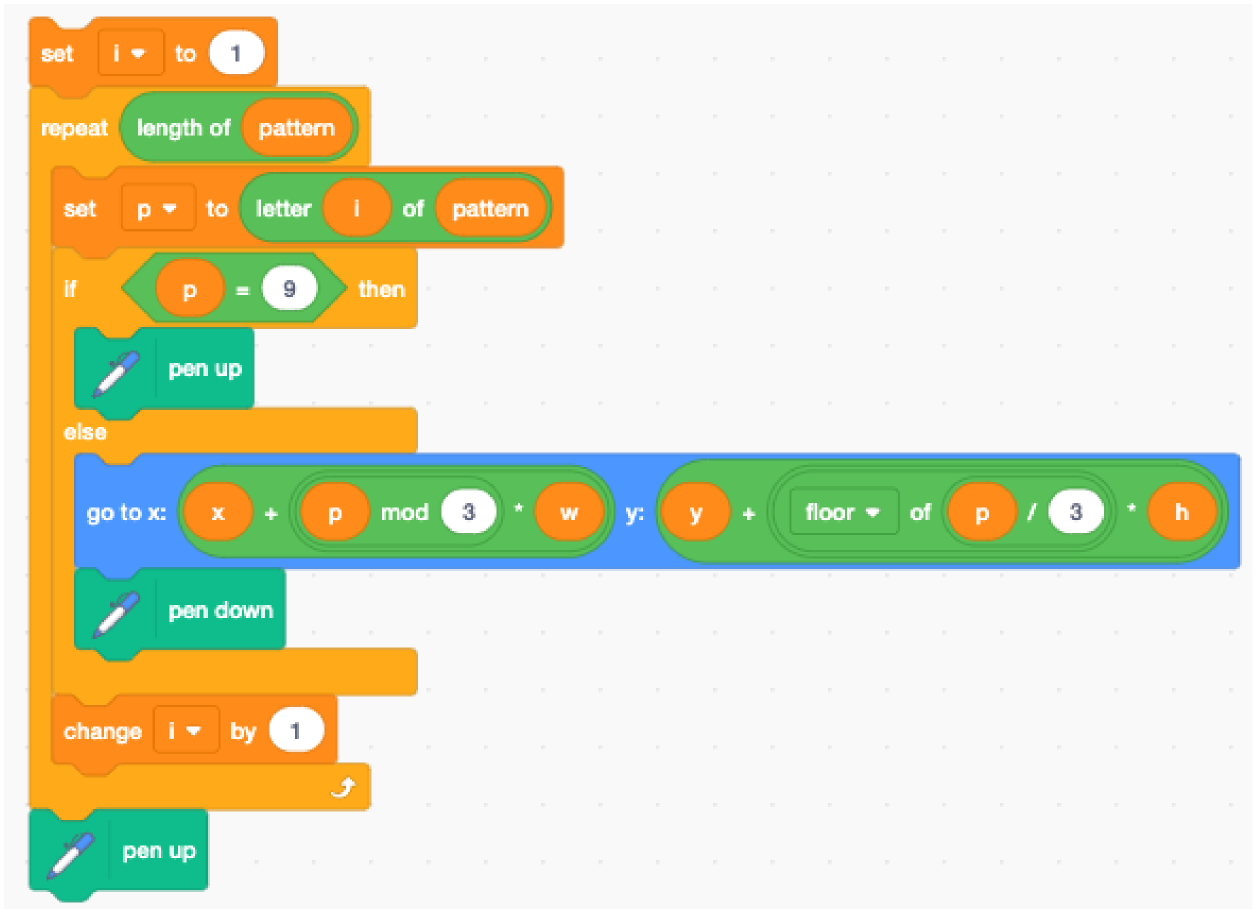
```
define write char (c)
set [i] to (item # of (c) in [symbols])
set [pattern] to (item (i) of [patterns])
```

We iterate through the sequence of points. The point index **p** starts from zero at the bottom left and goes from left to right.

We can use the two formulas to get the x and y offsets from the point index **p** (0 to 8) :

- x + mod(p, 3) * w
- y * floor(p/3) * h

The index 9 is used to lift up the pen, if we want to draw two disconnected lines.

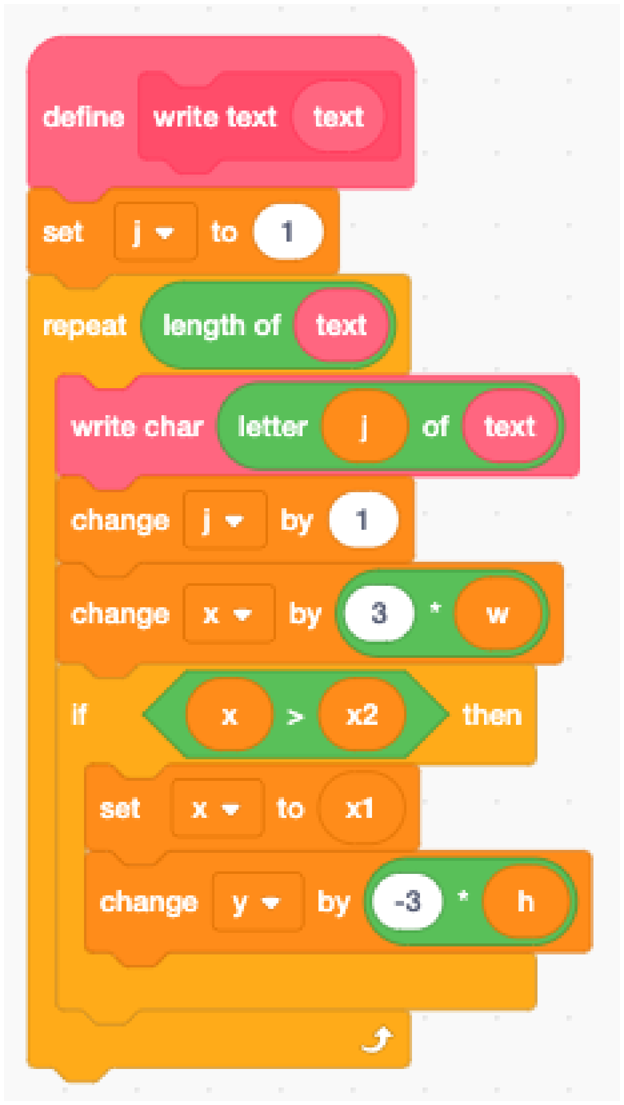


```
set i to 1
repeat length of pattern
    set p to letter i of pattern
    if p = 9 then
        pen up
    else
        go to x: x + p mod 3 * w y: y + floor of p / 3 * h
        pen down
    change i by 1
pen up
```

22.4 Draw a text

We used the index **i** to iterate inside the character function. To avoid interference we must use a different variable **i** in the **write text** function.

We write each new character next to the previous one. If the right margin (x2) is reached, the current character position is reset to the left margin (x1), and the y position is advancen to the next line.



```
define write text (text)
set [j] to (1)
repeat (length of (text))
    write char (letter (j) of (text))
    change [j] by (1)
    change [x] by ((3) * (w))
    if <(x) > (x2)> then
        set [x] to (x1)
        change [y] by ((-3) * (h))
    end
end
```

22.5 Settings (local variables)

A number of settings are encoded as local variables.

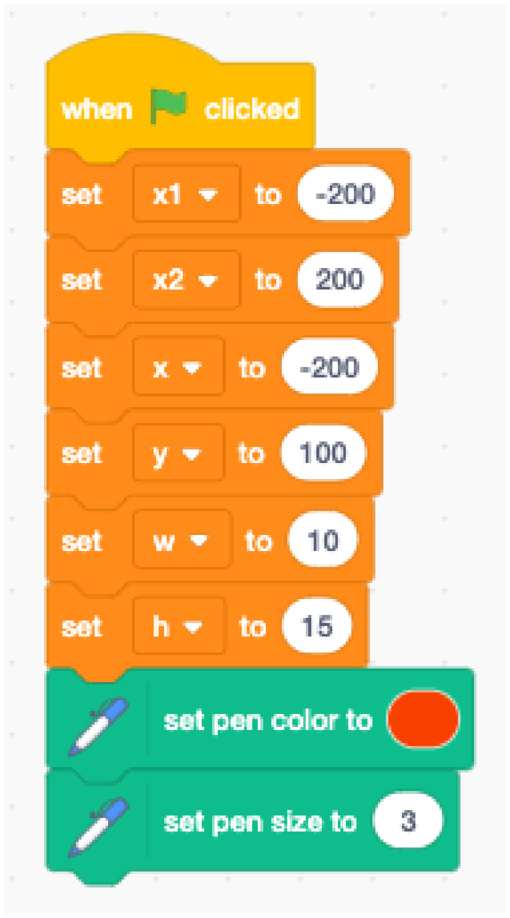
— the left and right margin (x1, x2)

— the current character position (x, y)

— the half-width and half-height (w, h)

— the pen thickness

— the pen color



```
when flag clicked
set x1 to -200
set x2 to 200
set x to -200
set y to 100
set w to 10
set h to 15
set pen color to
set pen size to 3
```

22.6 Parsing a command string

In order to change these parameters from another sprite, we will use a command string. The format of the command string is very simple : key=value. For example to set the left and write margins we would write :

```
x1=-100 x2=150
```

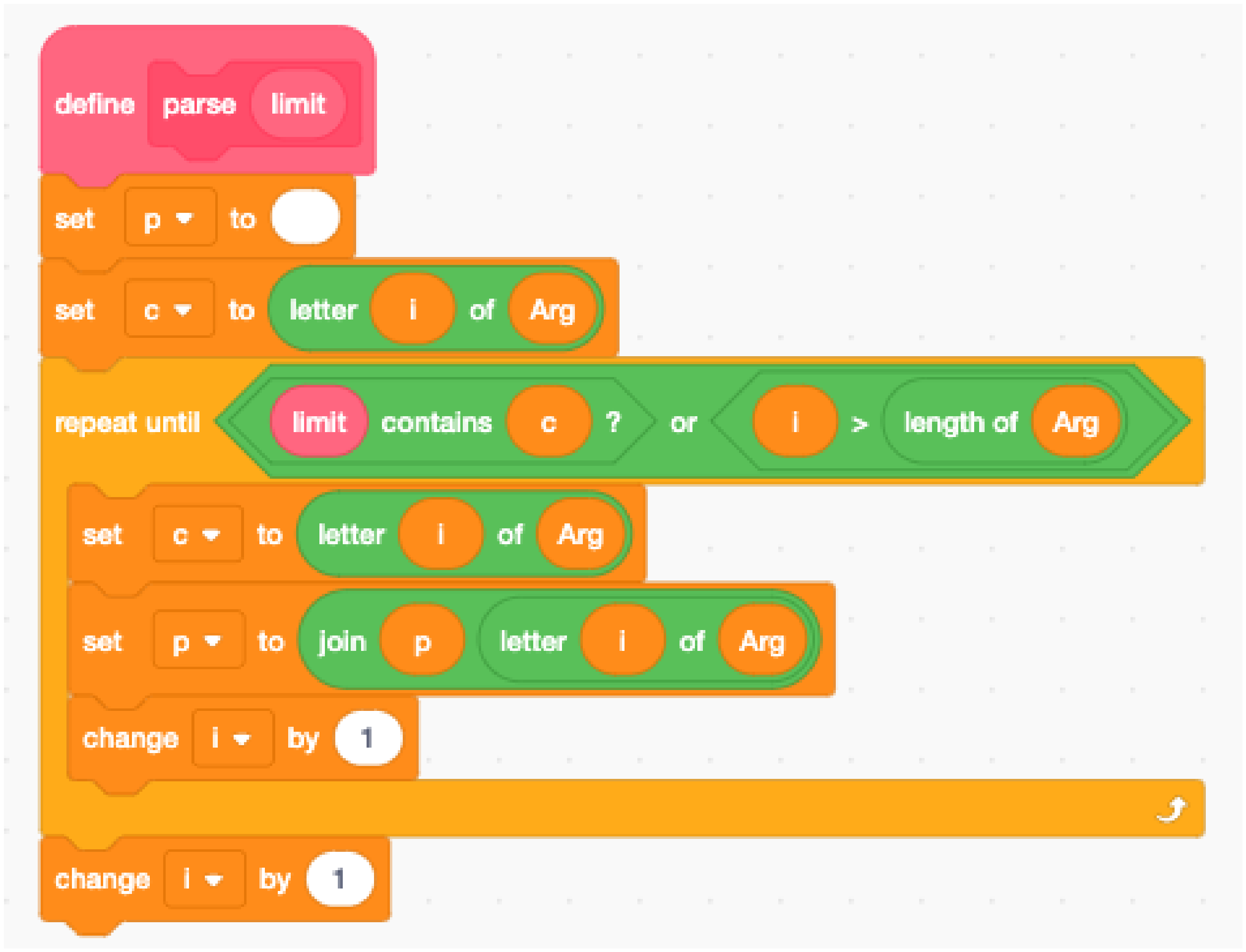
To set size :

```
s=5
```

The **parse** function takes the *limit** character set as an argument. The index **i** must point to the first character to analyze.

The **parse** function extracts the next token, which is separated by **=** or space ( ).

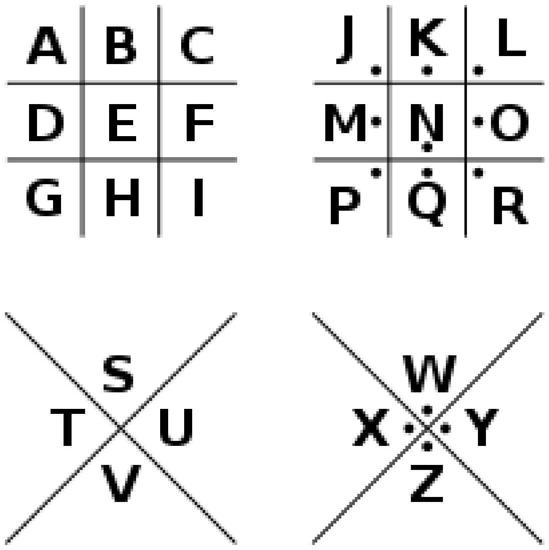
— **c** returns the limit character
— **p** returns the found token



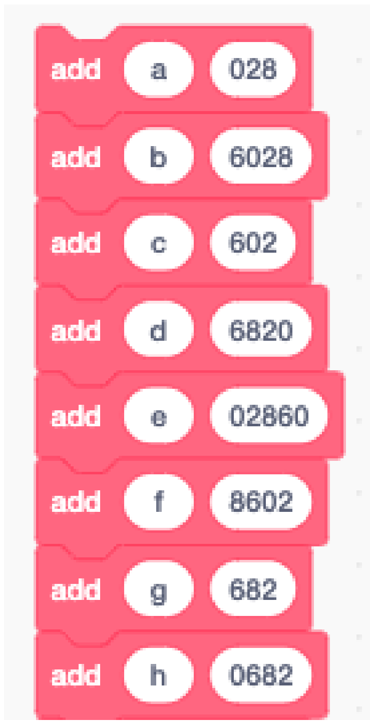
```
define parse (limit)
set [p v] to ()
set [c v] to (letter (i) of (Arg))
repeat until <<(limit) contains (c) ?> or <(i) > (length of (Arg))>>
    set [c v] to (letter (i) of (Arg))
    set [p v] to (join (p) (letter (i) of (Arg)))
    change [i v] by (1)
end
change [i v] by (1)
```

22.7 Pigpen cipher

The pigpen cipher or masonic cipher is a geometric, simple substitution code. We can easily program it using the 3x3 matrix and a dot.

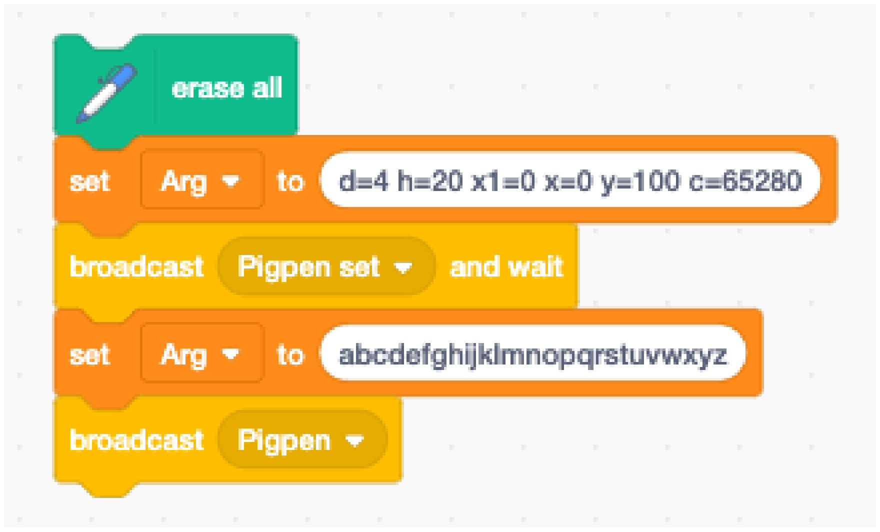


The encoding is very similar to the 16 segment encoding. We use the 9 to lift the pen and draw the center dot.



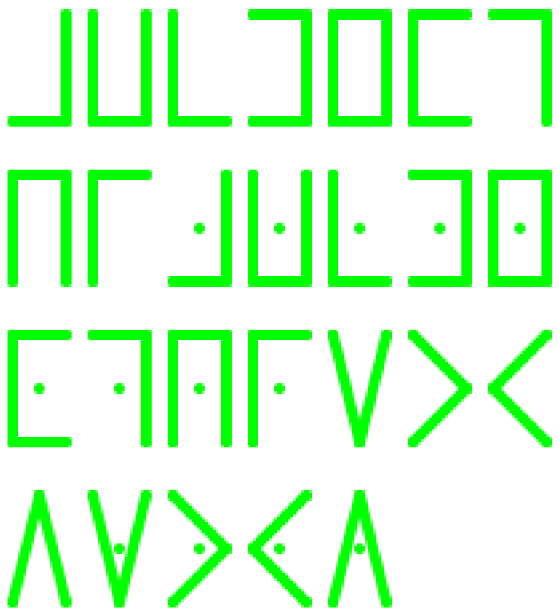
```
add a 028
add b 6028
add c 602
add d 6820
add e 02860
add f 8602
add g 682
add h 0682
```

We set the color to green.



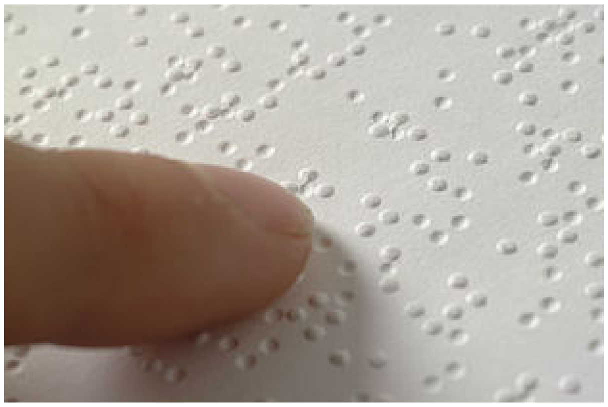
```
erase all
set Arg to d=4 h=20 x1=0 x=0 y=100 c=65280
broadcast Pigpen set and wait
set Arg to abcdefghijklmnopqrstuvwxyz
broadcast Pigpen
```

This is the alphabet printed in pipen cipher.



22.8 Braille code

The Braille code is a code used by blind people by using the sense of touch. It consist of a pattern of 6 raised dots. It is one of the first binary codes. Braille code pattern can be represented with a 6-bit binary code.



22.9 Morse code

6_codes/morse_code.png

CHAPITRE 23

Indices and tables

- genindex
- modindex
- search