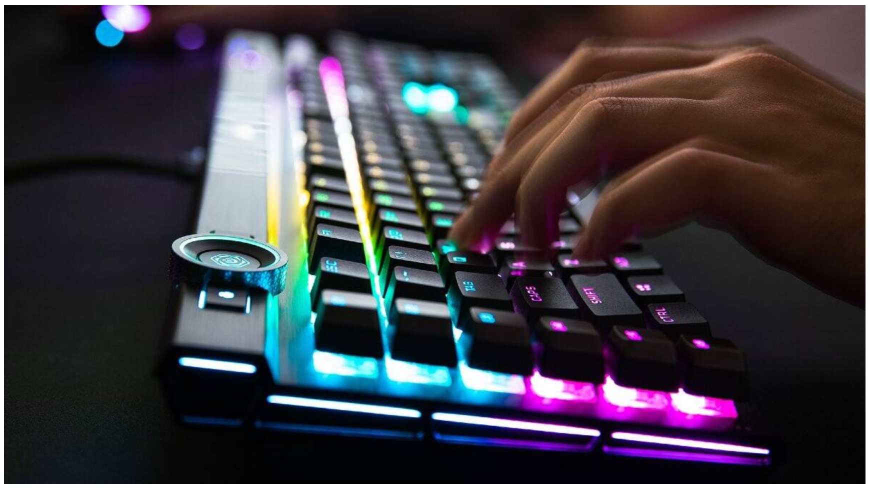
Snap! Keyboard interaction, stopping scripts, helicopter

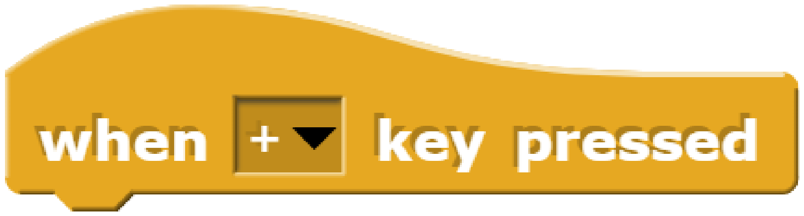
UBMS Snap! Programming Summer 2023

June 27, 2023

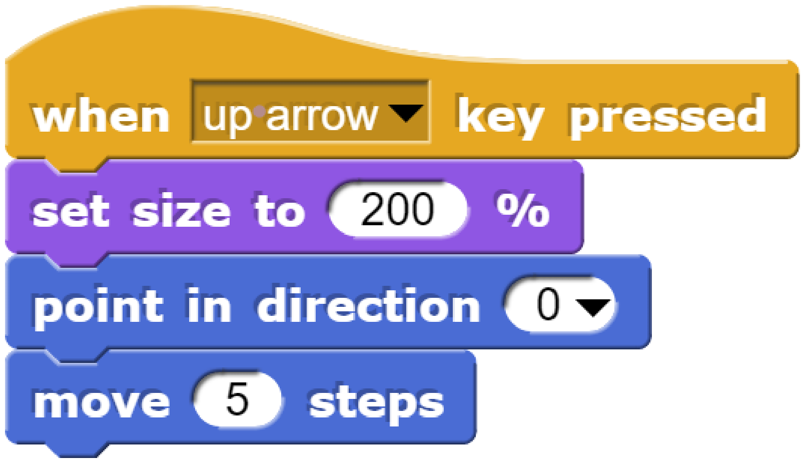


Event-controlled block

- Any time you run the selected key (0-9, a-z, arrows, enter, space, +/-), the following command is run

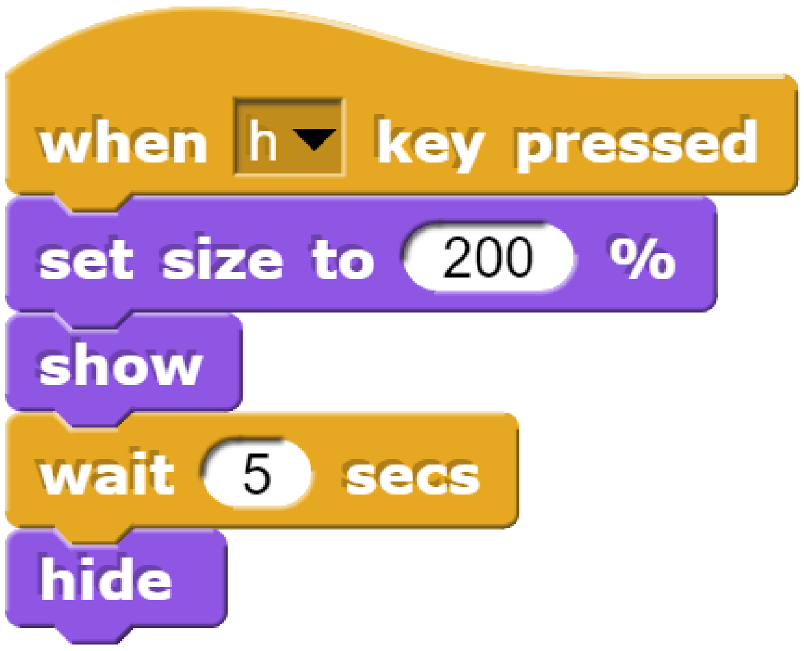


- Create a new project `Conditional_statements` to practice the material
- Save your reset script as a script in XML on your computer, import and run it while developing a script (link: tinyurl.com/3pnbn9wh)
- Build these two examples in Snap! yourself:
  1. before running them, consider what each of them might do
  2. what effect does it have if you keep pressing the keys?



```
when up arrow key pressed
set size to 200 %
point in direction 0
move 5 steps
```

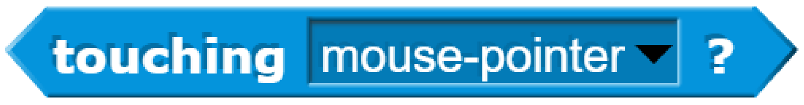
  - Doubles the size of the sprite
  - Sprite points upwards
  - Sprite moves 5 steps
  - Repeated keys only move the sprite further up

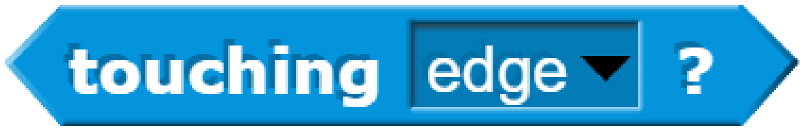


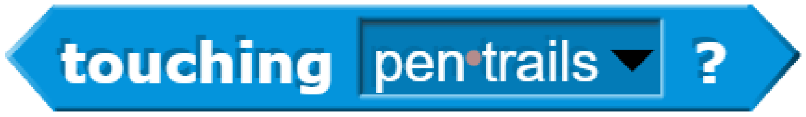
- Doubles the size of the sprite
- Sprite shows if it was hidden
- Sprite hides after 5 seconds
- Repeated keys have no effect

Sensing touch

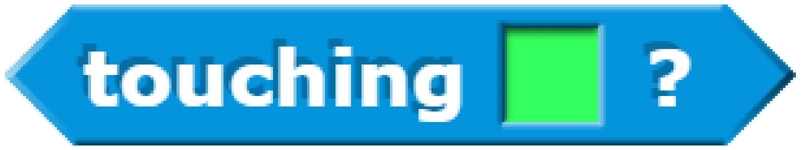
- You can check if things are touching one another, resulting in a Boolean value (TRUE or FALSE)
- The **Boolean** value can be used to control conditional statements like If, Wait until, Repeat until etc.
- The `touching` command checks if the sprite is touching the mouse pointer, the screen edges, a pen drawing or another sprite



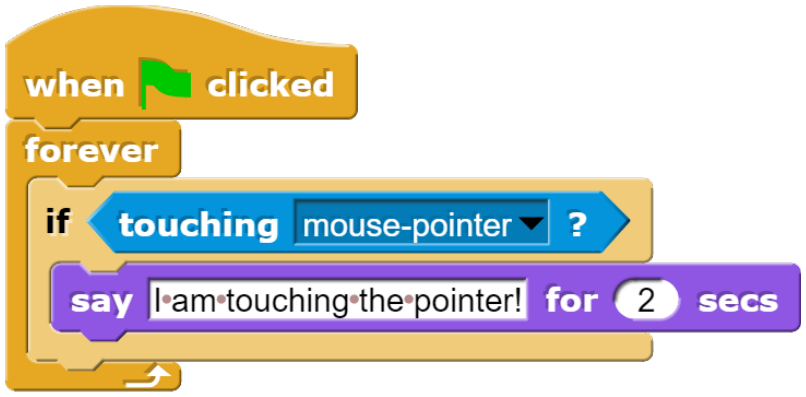




- You can also sense if a sprite is touching a color - the selector leads to a color palette

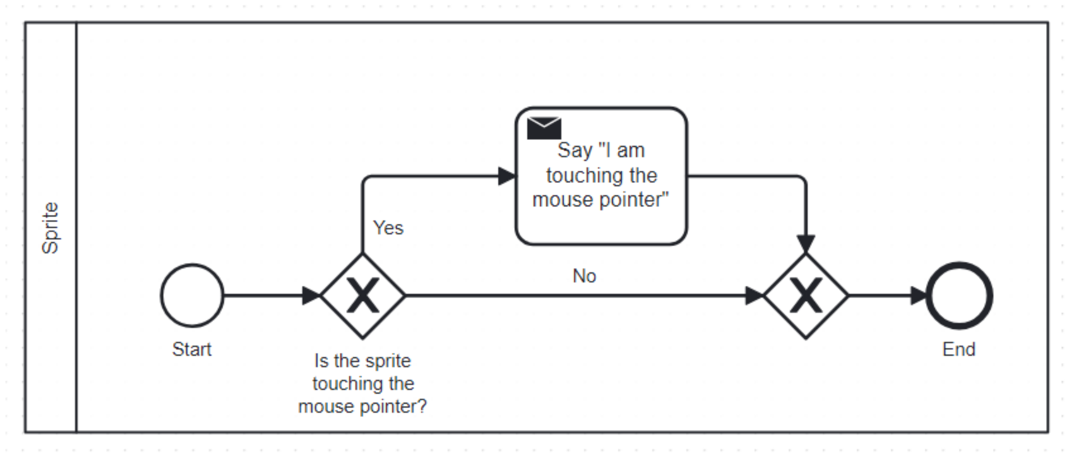


- Create this script, and think first what it might do:

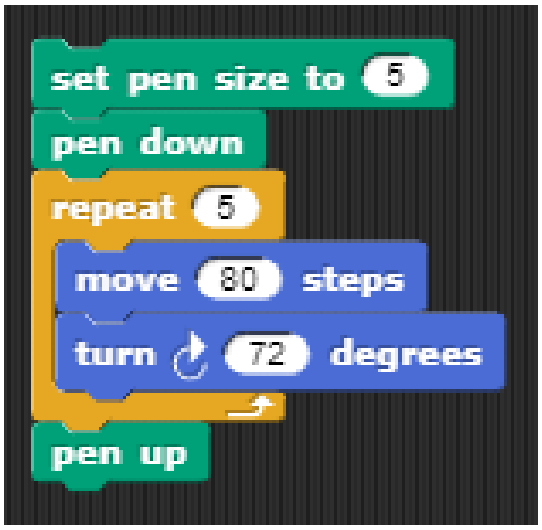

when clicked
forever
if touching mouse-pointer ?
say I am touching the pointer! for 2 secs

  - When starting the script, it waits in the background forever
  - When you touch the sprite with the mouse, you see the text (2sec)

- Here is a BPMN diagram of this decision:

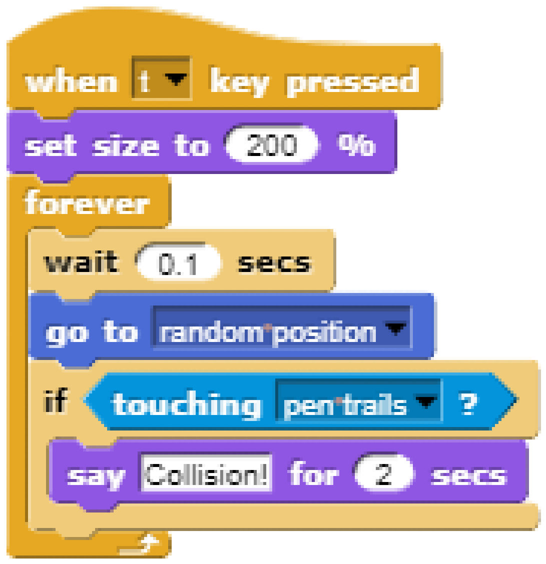

Sprite
Say "I am touching the mouse pointer"
Yes
No
Start
Is the sprite touching the mouse pointer?
End

- Make a pen-trail:



```
set pen size to 5
pen down
repeat 5
    move 80 steps
    turn ↻ 72 degrees
pen up
```

- Write a small program where the turtle goes around and whenever it collides with the pen trail, it changes color or size.

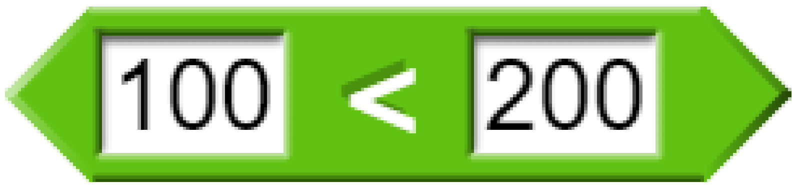

```
when t key pressed
set size to 200 %
forever
    wait 0.1 secs
    go to random position
    if touching pen trails ?
        say Collision! for 2 secs
```

The story behind the IF

- A condition is another name for a logic operation
- Do you know examples for such operations?
- The result of the operation determines its truth or falsehood, represented as `TRUE` or `FALSE`, represented by `1` or `0` in the computer
- An OOP way of thinking about IF statements: the inside of the IF statement is a local environment that interacts with the outside only via the conditional statement
- Another approach views the IF as a decision point - the condition is a question leading in one or the other direction

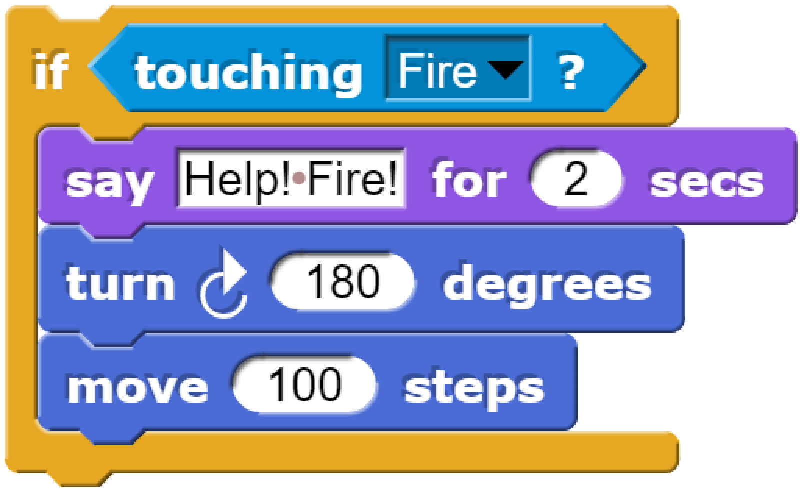
IF statements

- The `Sensing` and `Operators` tab offers several conditions
- This operator checks the condition: `Is 100 less than 200?`



- When you execute the command, the operation is evaluated. You can alter the result of the condition - the computer is not picky, it wants to serve its master!

- Add the following script to your project, think about what it might do and run it



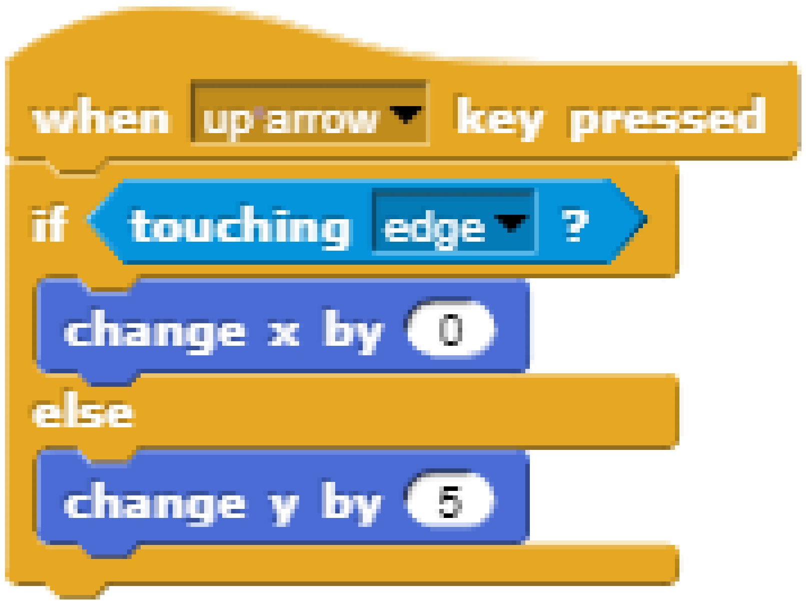
  - Check if sprite touches the sprite called `Fire`
  - If the condition holds, say `Help!Fire!` for 2 seconds
  - Turn in the opposite direction and move away for 100 steps

IF-ELSE statements

- IF-ELSE statements are gateway points of the program flow

- If the IF condition is not met, the code following ELSE is executed

- Example: the following script is run when the up arrow is pressed. The sprite is moved up until it hits an edge:

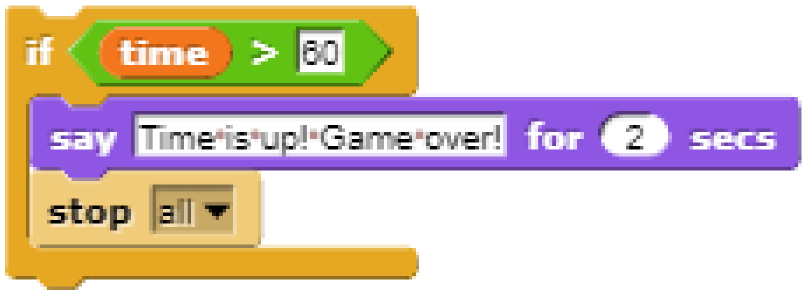


```
when up arrow key pressed
if touching edge ?
    change x by 0
else
    change y by 5
```

- The gateway (or the conditional test) question is: "is the sprite touching any edge?" If the answer is yes, the sprite stops, if it is no, it is moved up in small steps of 5.

Stopping scripts

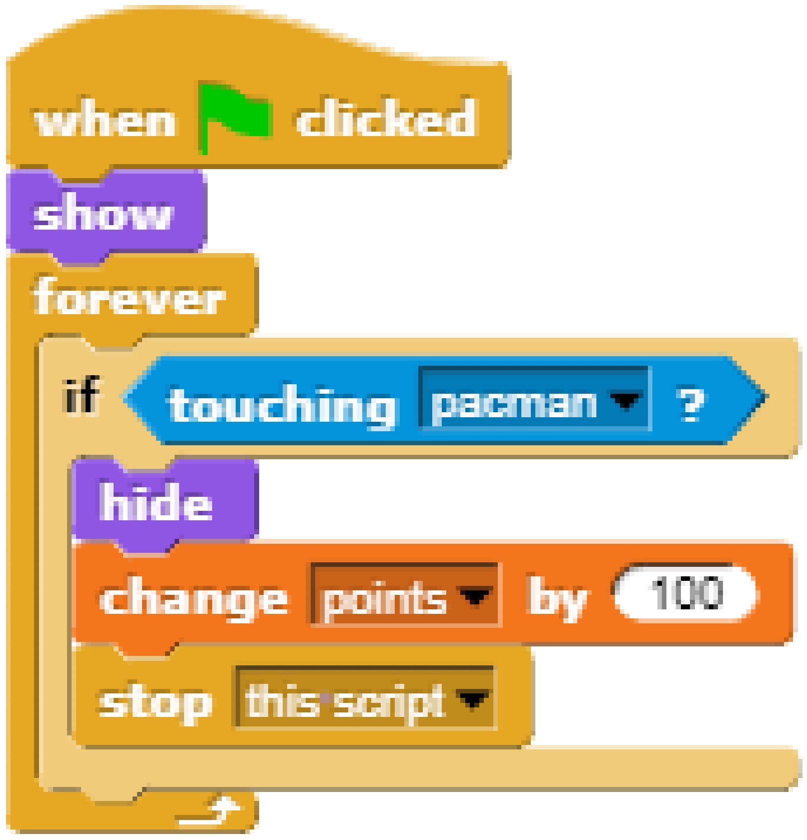
- The `STOP` commands (`Control` tab) permit stopping Snap! scripts
- The following command stops all scripts when the *variable* `time` exceeds the value `60`



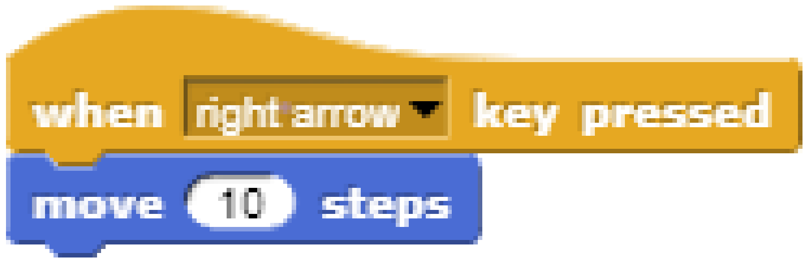
```
if time > 60
    say Time is up! Game over! for 2 secs
    stop all
```

- The command `stop this script` stops only the current script
- Create a new project or reuse an old project:

1. Generate two sprites and name one `pacman`
2. Create the following two scripts
3. Start the script with the green flag
4. Move the sprite towards the `pacman` sprite
5. See what happens when the sprite touches `pacman`

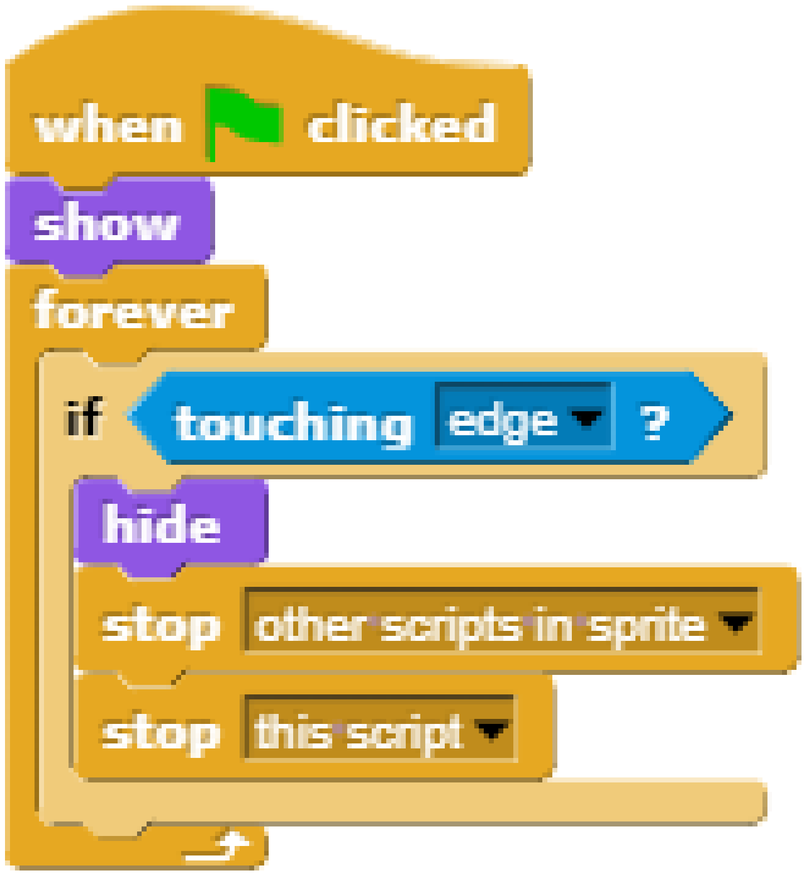


```
when ⚑ clicked
show
forever
    if <touching [pacman ▾] ?>
        hide
        change [points ▾] by (100)
        stop [this script ▾]
```



```
when [right arrow ▾] key pressed
move (10) steps
```

- If you want to stop all script of the current sprite, you need to include the stopping script:



```
when [flag] clicked
show
forever
    if <touching [edge ▾] ?>
        hide
        stop [other scripts in sprite ▾]
        stop [this script ▾]
```

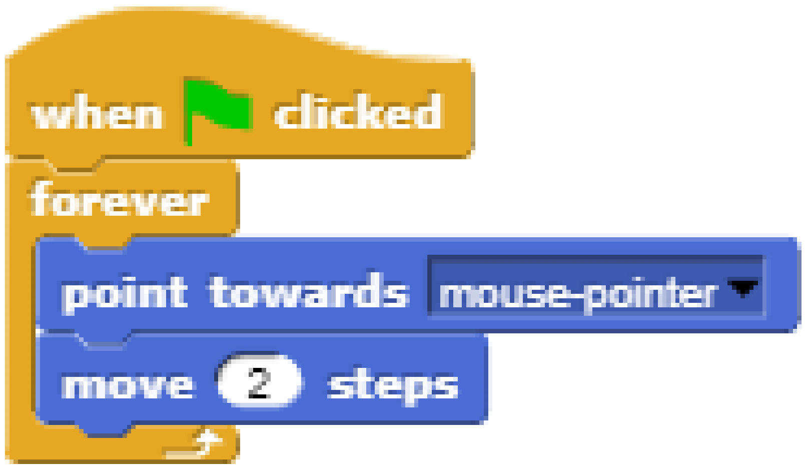
Assignments: ball and paddle / color circles

For this programming assignment, pick either option 1 or 2 below. If you complete both programs, you get up to 10 bonus points (provided the program does what it should).

1. Design a program in which a ball bounces up and down and is deflected by a paddle. The user should be able to move the paddle horizontally using the arrow keys.

   Sample solutions by Bryceton Church (Fall'22) and by Brayden Burrow!

2. Design a program in which there are several separate color circles (red, blue, green etc.) on the screen. When the green flag is clicked, an animal sprite starts following the mouse pointer. Whenever it touches a circle, its color becomes the same as the circle's color. Tip: to make the animal sprite follow the pointer, use the "point towards" and "move" commands in a "forever" loop:



```
when [flag] clicked
forever
    point towards mouse-pointer
    move 2 steps
```

Sample solution by Isaac Rice (Fall'22)

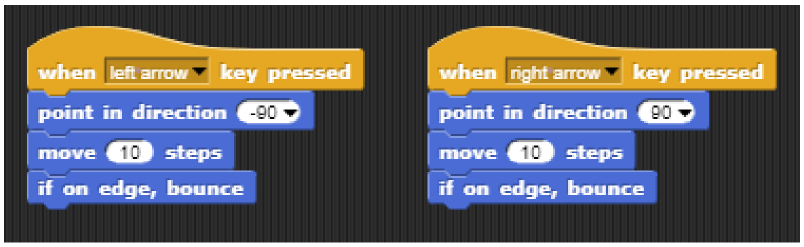
Remember to add notes to your program (or lose points).

Submit the URL(s) of your final, working program in Canvas (if you submit two programs, use the text entry option and paste both URLs in). If Canvas does not let you upload (there has been trouble lately), send me your URLs via email.

If you're curious to try the BPMN process modeling method that I mentioned in class, you can do so for free at bpmn.ioLinks to an external site.. Take a screenshot of your model and send it to me via email for 5 bonus points.

Solutions - ball and paddle

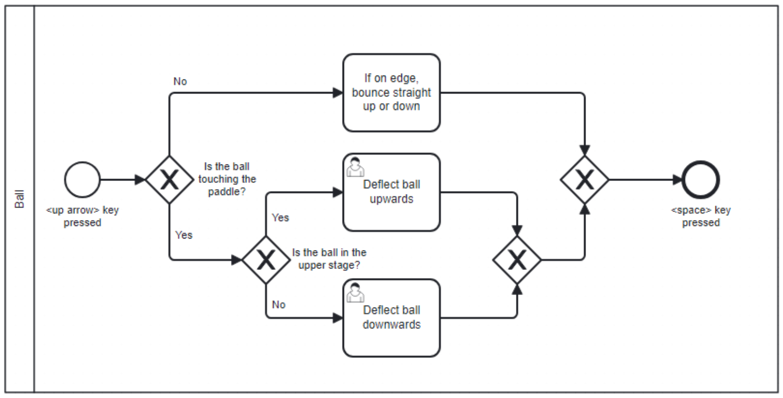
- We have two sprites but only the ball has interesting code. Here's the paddle (costume straight from the Snap! cloud):



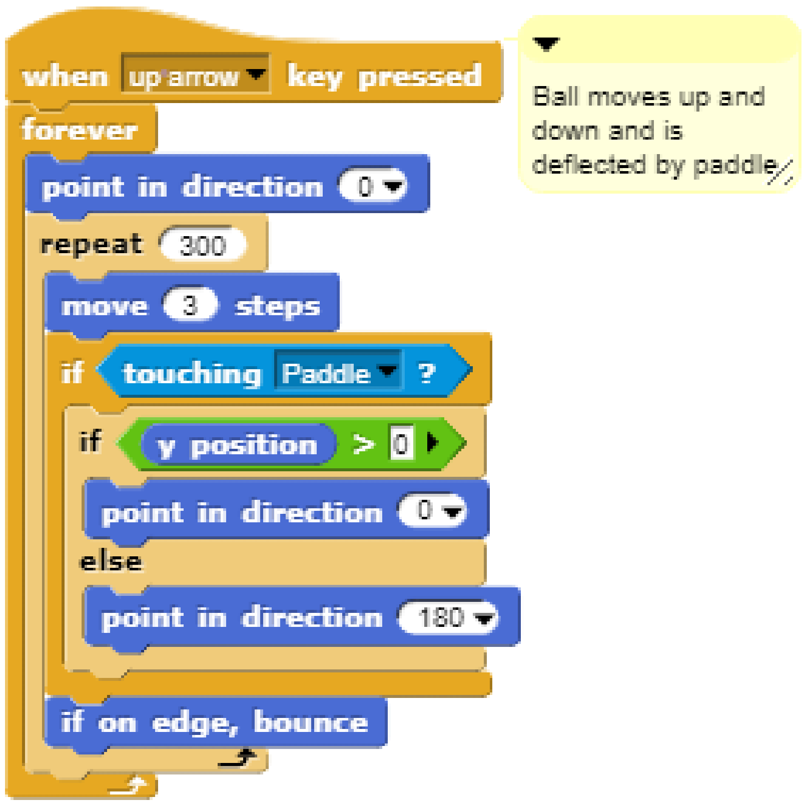
```
when [left arrow] key pressed
point in direction (-90)
move (10) steps
if on edge, bounce
```

```
when [right arrow] key pressed
point in direction (90)
move (10) steps
if on edge, bounce
```

- Here is the flow diagram for the simple solution, a ball that moves up and down and is deflected (opposite) by the paddle: there is a decision implied in the code block.


Ball
<up arrow> key pressed
No
If on edge, bounce straight up or down
Is the ball touching the paddle?
Yes
Deflect ball upwards
Yes
Is the ball in the upper stage?
No
Deflect ball downwards
<space> key pressed

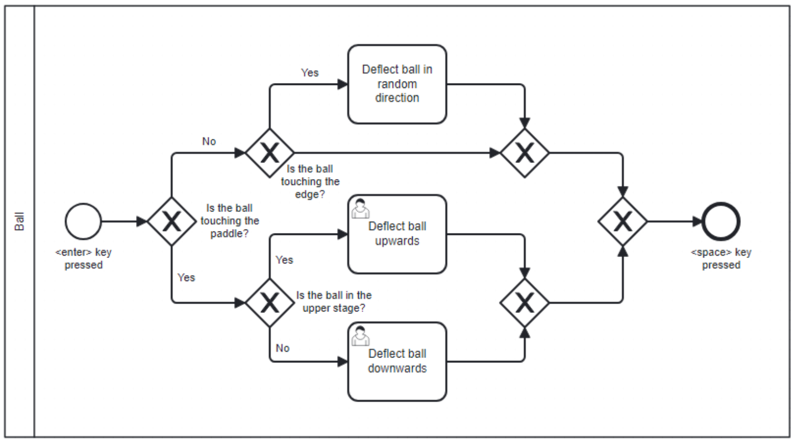
- Here is the code: activated with `<up arrow>`



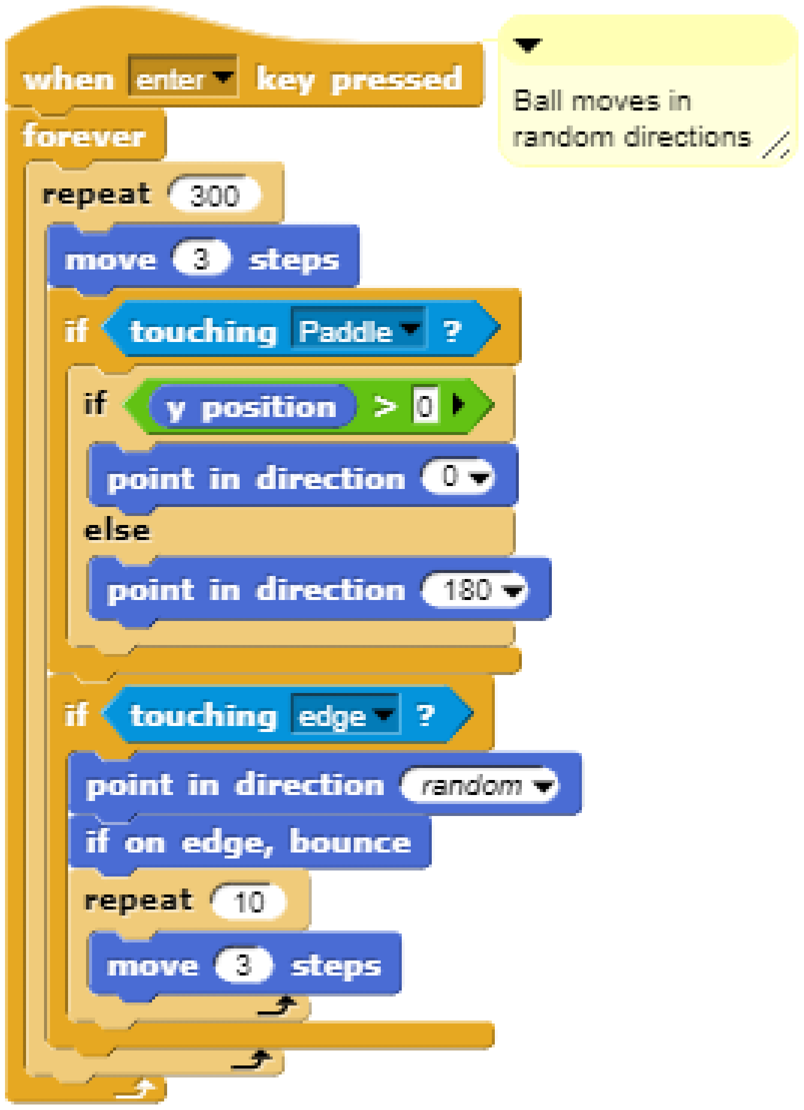
```
when [up arrow] key pressed
forever
  point in direction (0)
  repeat (300)
    move (3) steps
    if <touching [Paddle] ?>
      if <(y position) > [0]>
        point in direction (0)
      else
        point in direction (180)
  if on edge, bounce
```

Ball moves up and down and is deflected by paddle

- See the screenrecording.
- The action is more interesting if the ball bounces around (not just vertically) though this was not asked in the assignment - here we must program the decision explicitly:


Ball
<enter> key pressed
Is the ball touching the paddle?
No
Yes
Is the ball touching the edge?
Yes
Deflect ball in random direction
Is the ball in the upper stage?
Yes
No
Deflect ball upwards
Deflect ball downwards
<space> key pressed

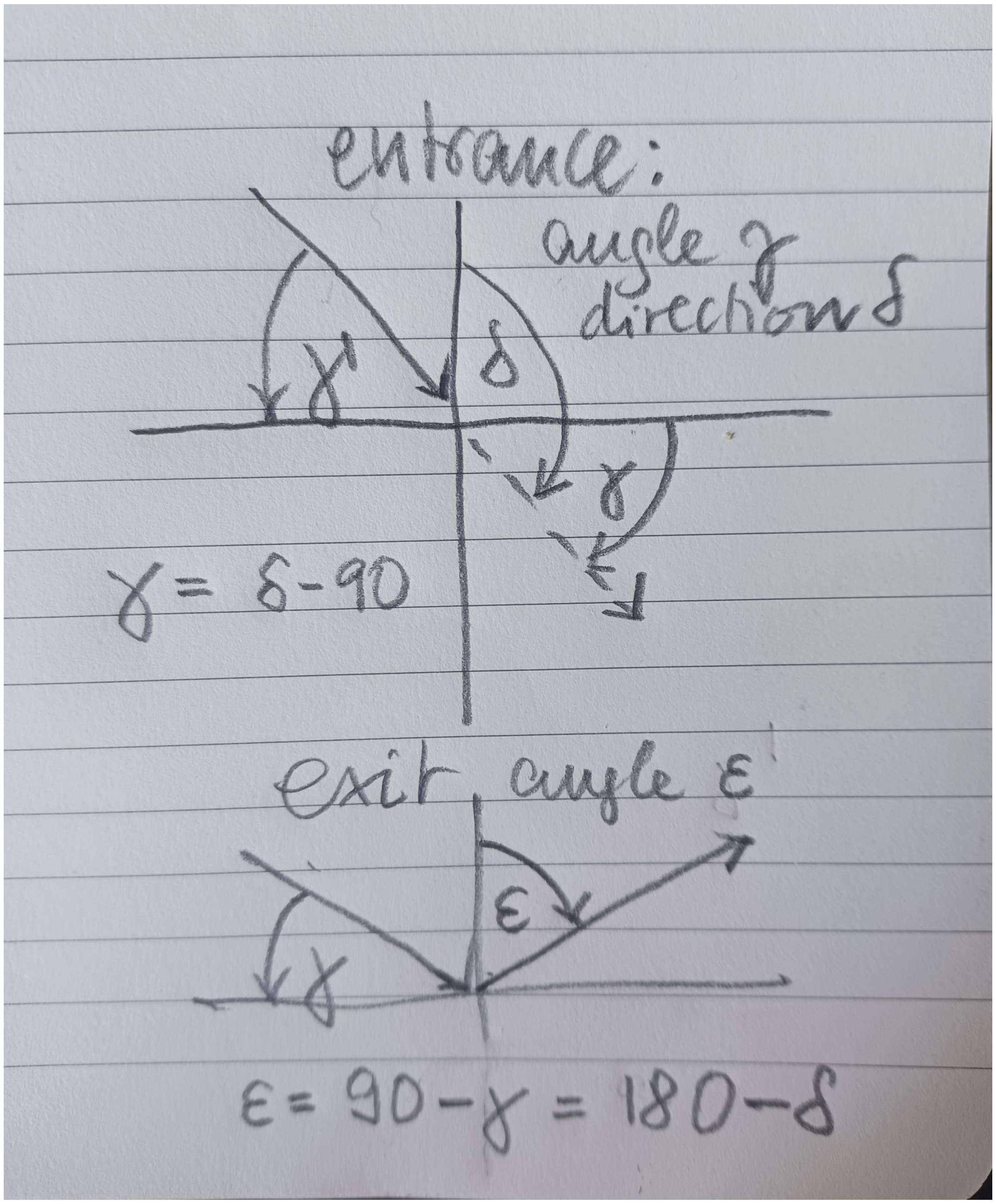
- Here is the code: activated with `<enter>`



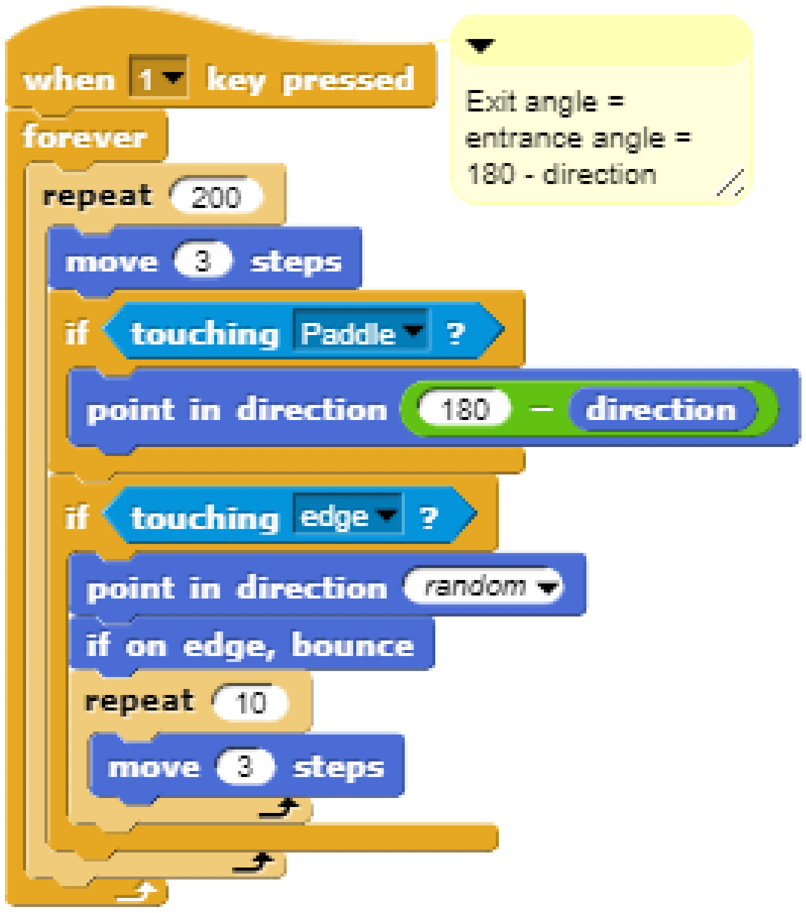
```
when [enter] key pressed
forever
  repeat (300)
    move (3) steps
    if <touching [Paddle] ?>
      if <(y position) > [0]>
        point in direction (0)
      else
        point in direction (180)
    if <touching [edge] ?>
      point in direction (random)
      if on edge, bounce
      repeat (10)
        move (3) steps
```

Ball moves in random directions

- See the screenrecording.
- Finally, let's add some physics to the paddle deflection: the exit angle is opposite the entrance angle (direction): to compute it, we must take a look at the relationship of entrance angle (γ), the direction recorded by Snap! (δ), and the desired exit angle (ϵ):



- Here is the code:



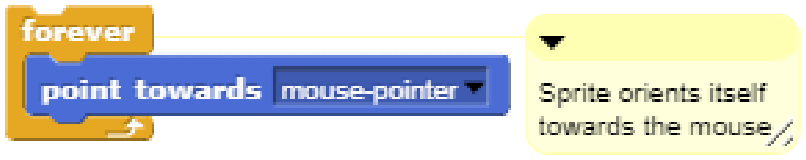
```
when 1 key pressed
forever
    repeat 200
        move 3 steps
        if touching Paddle ?
            point in direction (180 - direction)
        if touching edge ?
            point in direction random
            if on edge, bounce
            repeat 10
                move 3 steps
```

Exit angle = entrance angle = 180 - direction

- Code link

Solutions - color circles

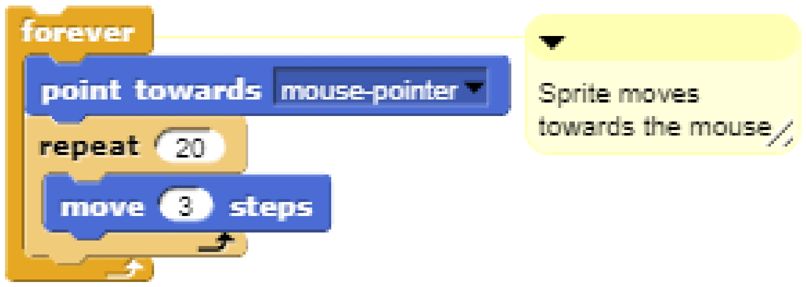
- This script makes the sprite orient itself towards the mouse pointer:



```
forever
    point towards mouse-pointer
```

Sprite orients itself towards the mouse

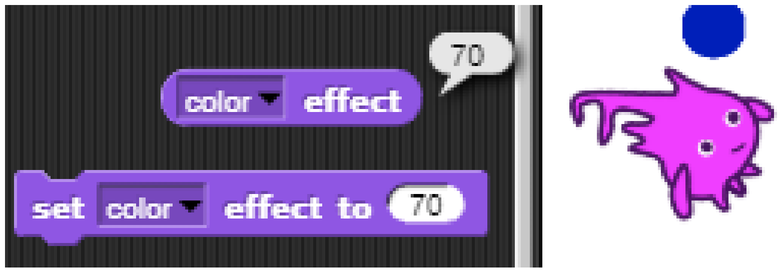
- We can add a smooth motion block to make the sprite follow the mouse pointer:



```
forever
  point towards mouse-pointer
  repeat 20
    move 3 steps
```

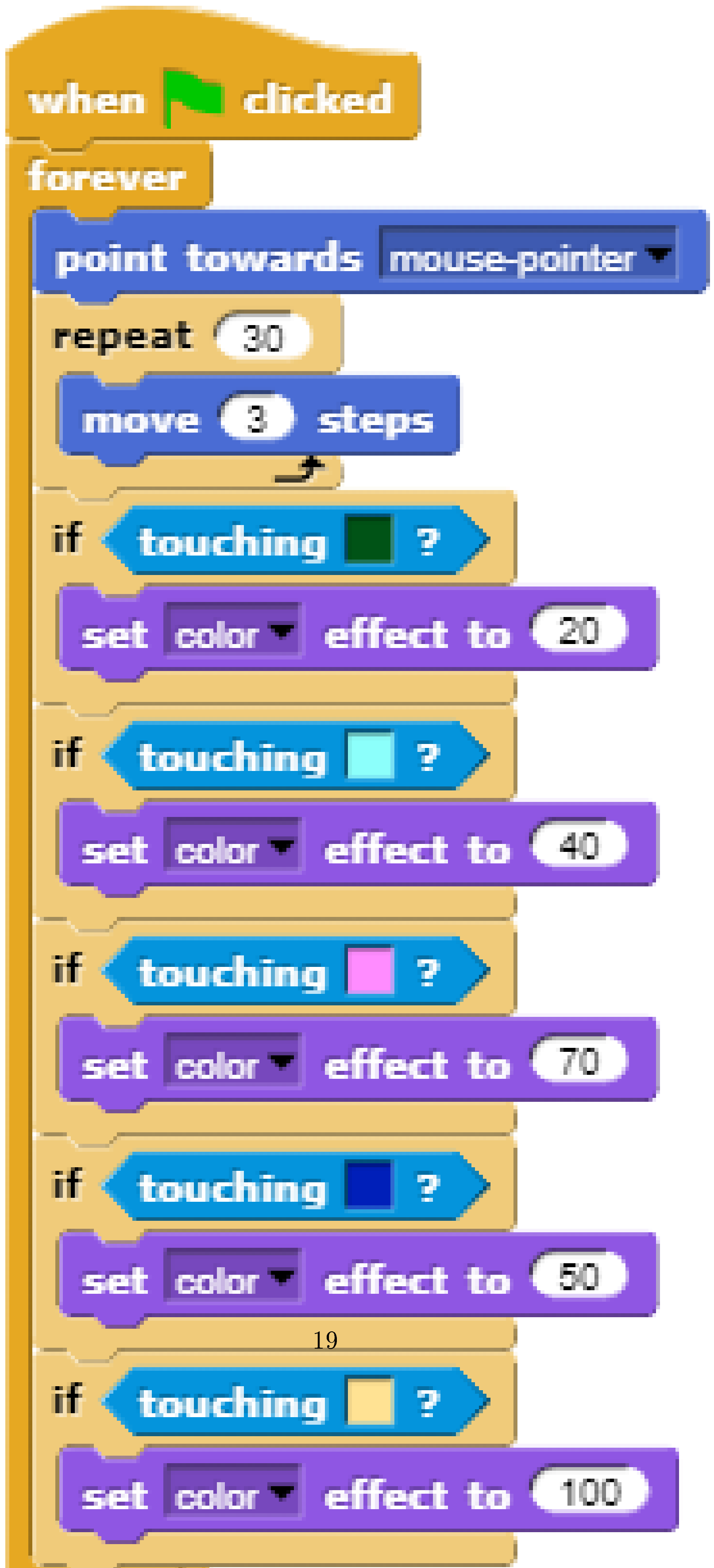
Sprite moves towards the mouse

- To solve the problem, we need color circles to touch.
- My first attempt was to create a background with color circles and use an approximation to the color for a touch-set-color combo:



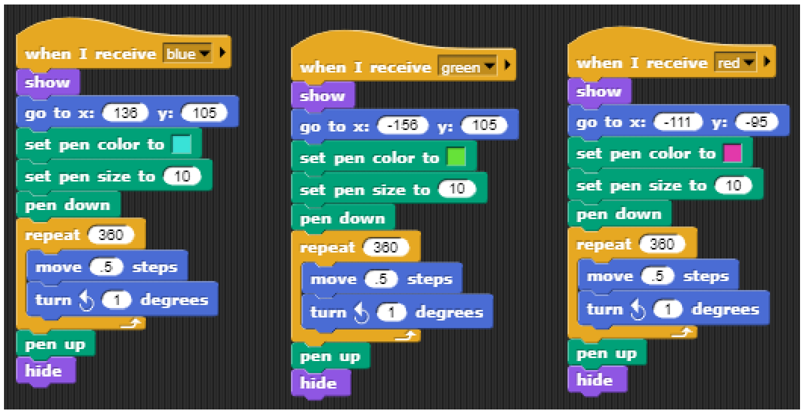
```
(color effect) → 70
set color effect to 70
```

- The resulting script works approximately and not for all colors:



```
when 🏁 clicked
forever
    point towards [mouse-pointer ▾]
    repeat (30)
        move (3) steps
    ↻
    if <touching 🟩 ?>
        set [color ▾] effect to (20)
    if <touching 🟦 ?>
        set [color ▾] effect to (40)
    if <touching 🟪 ?>
        set [color ▾] effect to (70)
    if <touching 🟦 ?>
        set [color ▾] effect to (50)
    if <touching 🟨 ?>
        set [color ▾] effect to (100)
```

- See screencast. Here is the code link.
- I was frustrated with the fact that the color effect picker does not allow you to identify the color. I looked it up in the reference manual and found a color library with a lot of choice that you can import in the `pen` command category.
- In the second solution, I use a pen sprite to draw circles in exactly the color I want and make sure that the sprite changes into that color:

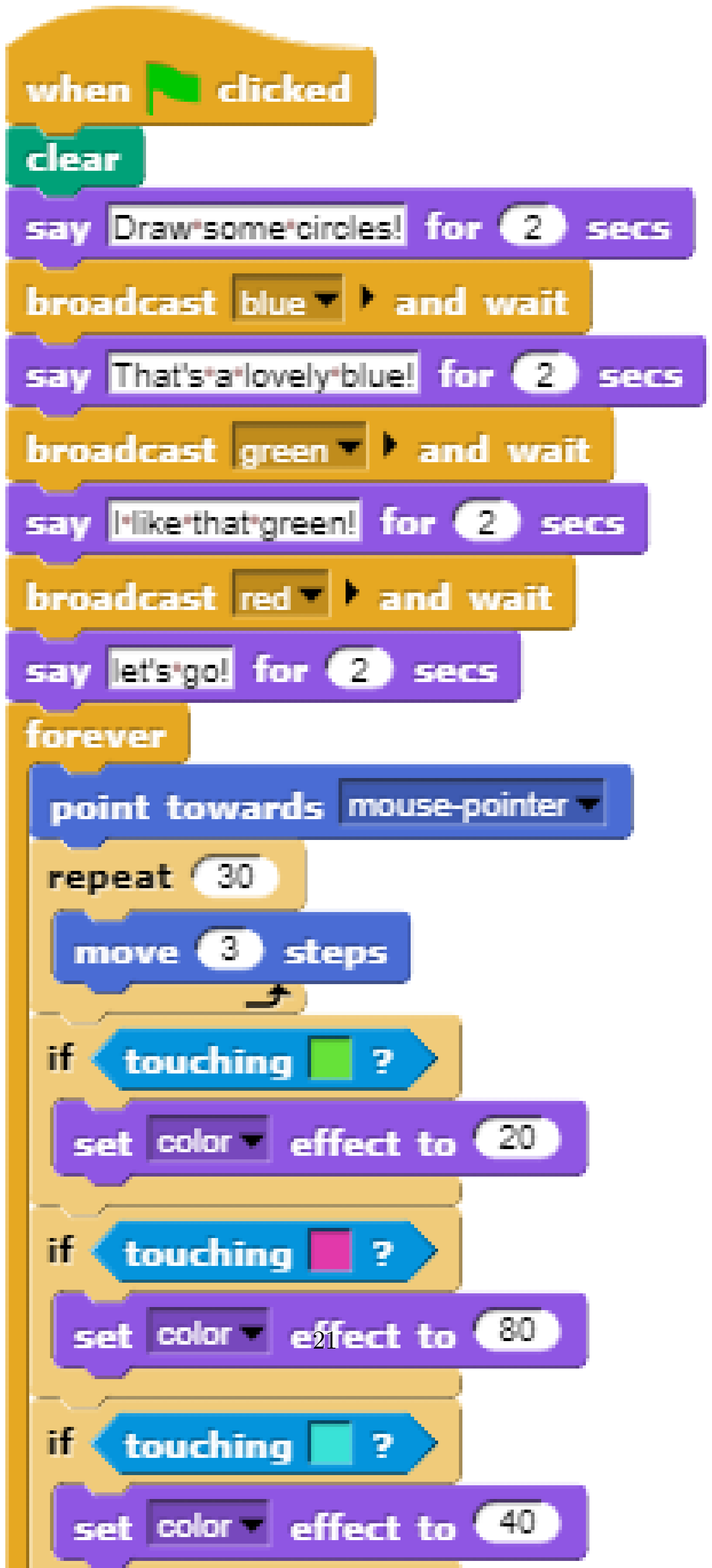


```
when I receive blue
show
go to x: 136 y: 105
set pen color to 🟦
set pen size to 10
pen down
repeat 360
    move .5 steps
    turn ↺ 1 degrees
pen up
hide
```

```
when I receive green
show
go to x: -156 y: 105
set pen color to 🟩
set pen size to 10
pen down
repeat 360
    move .5 steps
    turn ↺ 1 degrees
pen up
hide
```

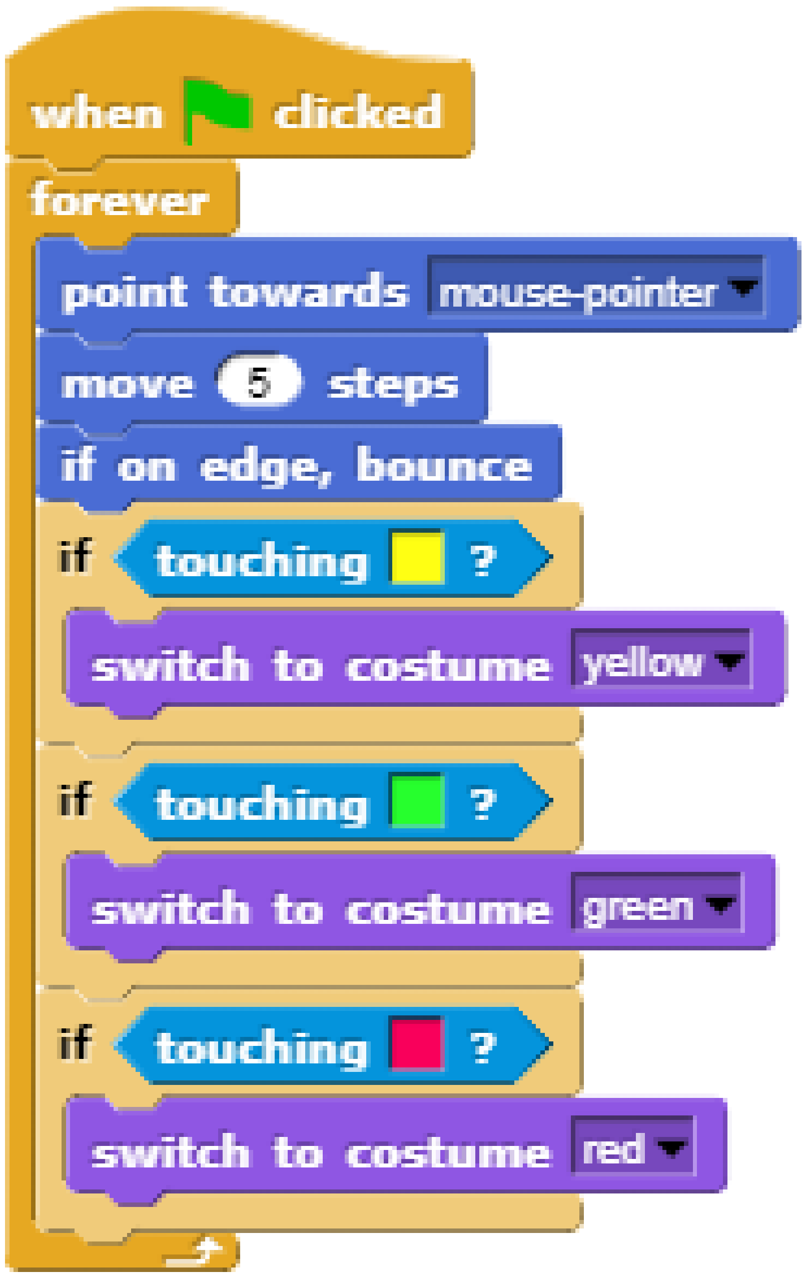
```
when I receive red
show
go to x: -111 y: -95
set pen color to 🟪
set pen size to 10
pen down
repeat 360
    move .5 steps
    turn ↺ 1 degrees
pen up
hide
```

- The full program calls upon the pen to draw the circles and then enables the color change through touching:



```
when [flag] clicked
clear
say [Draw some circles!] for (2) secs
broadcast [blue ▾] and wait
say [That's a lovely blue!] for (2) secs
broadcast [green ▾] and wait
say [I like that green!] for (2) secs
broadcast [red ▾] and wait
say [let's go!] for (2) secs
forever
    point towards [mouse-pointer ▾]
    repeat (30)
        move (3) steps
    if <touching [🟩] ?>
        set [color ▾] effect to (20)
    if <touching [🟪] ?>
        set [color ▾] effect to (80)
    if <touching [🟦] ?>
        set [color ▾] effect to (40)
```

- See screenrecording (45 sec) and code link.
- Another solution is to create sprites that are color balls or circles and then upon touching copy the color of the sprite, Alonzo changes to one of his (colored) pre-programmed costumes:



```
when 🏁 clicked
forever
    point towards mouse-pointer ▾
    move 5 steps
    if on edge, bounce
    if touching 🟨 ?
        switch to costume yellow ▾
    if touching 🟩 ?
        switch to costume green ▾
    if touching 🟥 ?
        switch to costume red ▾
```

- Code link.