Snap! Mouse interaction, push-buttons on stage

UBMS Snap! Programming Summer 2023

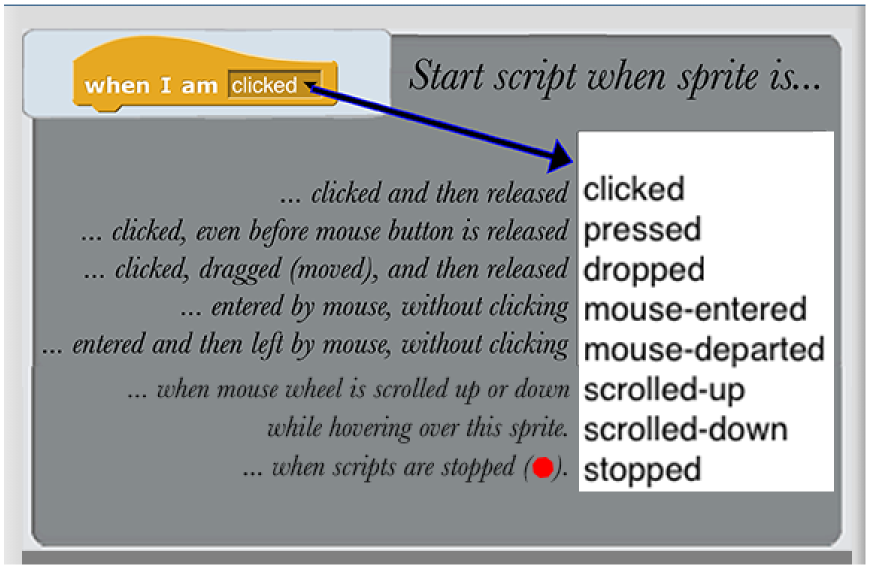
June 27, 2023

Game design

- ☐ User interaction using mouse pointer
- ☐ Conditionals (Wait until)
- ☐ Numeric variables (numbers)
- ☐ Variables as sliders
- ☐ Keyboard events (polling)
- ☐ User input with buttons

Mouse interaction

Mouse events for sprite and stage



when I am clicked

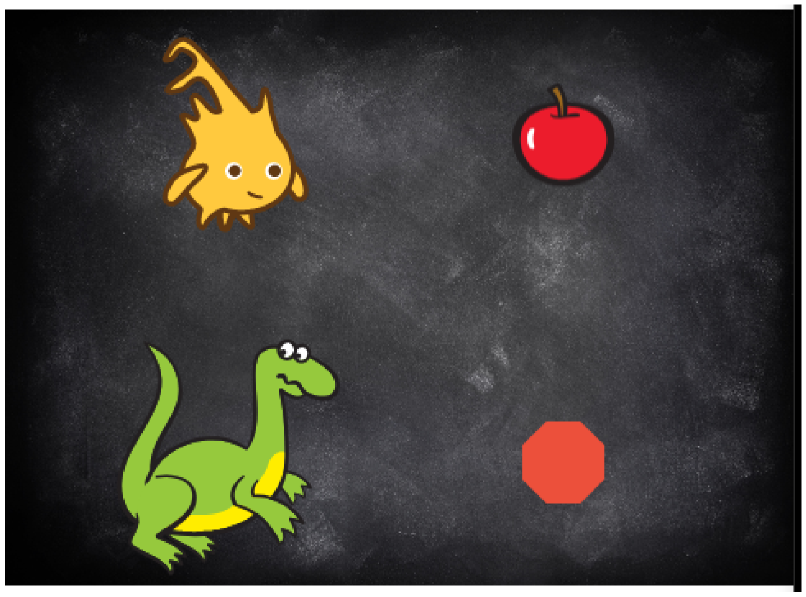
Start script when sprite is...

... clicked and then released	clicked
... clicked, even before mouse button is released	pressed
... clicked, dragged (moved), and then released	dropped
... entered by mouse, without clicking	mouse-entered
... entered and then left by mouse, without clicking	mouse-departed
... when mouse wheel is scrolled up or down	scrolled-up
while hovering over this sprite.	scrolled-down
... when scripts are stopped (●).	stopped

- Write a short script for each of these interactions:

1. create a new project `MouseMoves`,
2. import the `reset` script
3. write the first script, duplicate and alter accordingly.
4. Distinguish the examples with an action and a sound.
5. Add a short note on the purpose of this project for later.

- My sample solution:



1. Alonzo: mouse clicked/pressed/dropped.
2. Apple: moused-entered/mouse-departed.
3. Dino: scrolled-up/scrolled-down.
4. STOP: stopped.

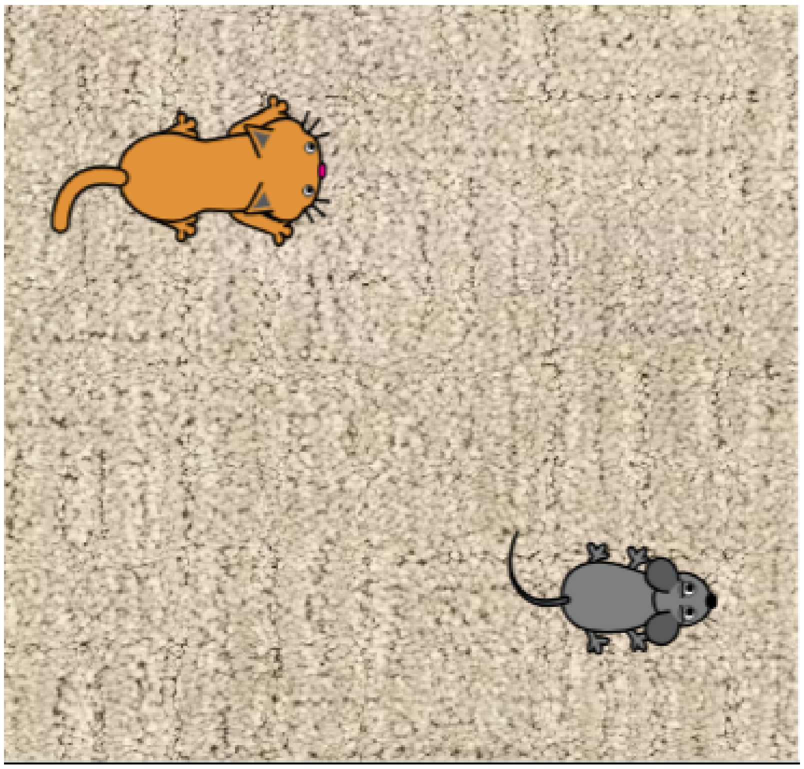
- The `When I am stopped` event is a little tricky: in particular, the `say...` commands do not work with this event (I don't know why).

Cat-and-mouse

Use the mouse interaction events to make one sprite follow another:

1. create a new sprite `hunter` and another sprite `prey`

2. add costumes `cat` and `mouse` - initially both are looking to the right.



3. import reset script for quick experimentation (add `stop all`)
4. remember that you need a reset script for both sprites
5. write the code and test the script for these actions:
    (a) Start the script with the `Green Flag`.
    (b) When mouse is `dropped` somewhere: cat pounces and sits on mouse! (Tip: the sprites on the stage are 'layered'.)
    (c) Reset with `r key`.
    (d) When `scrolled-up`, mouse turns to cat and glides towards it. When it is close, the mouse says "Hello". Then the cat turns to the mouse, says "Go away", and the mouse slowly disappears.
    (e) Reset with `stop` button.

See: sample solution

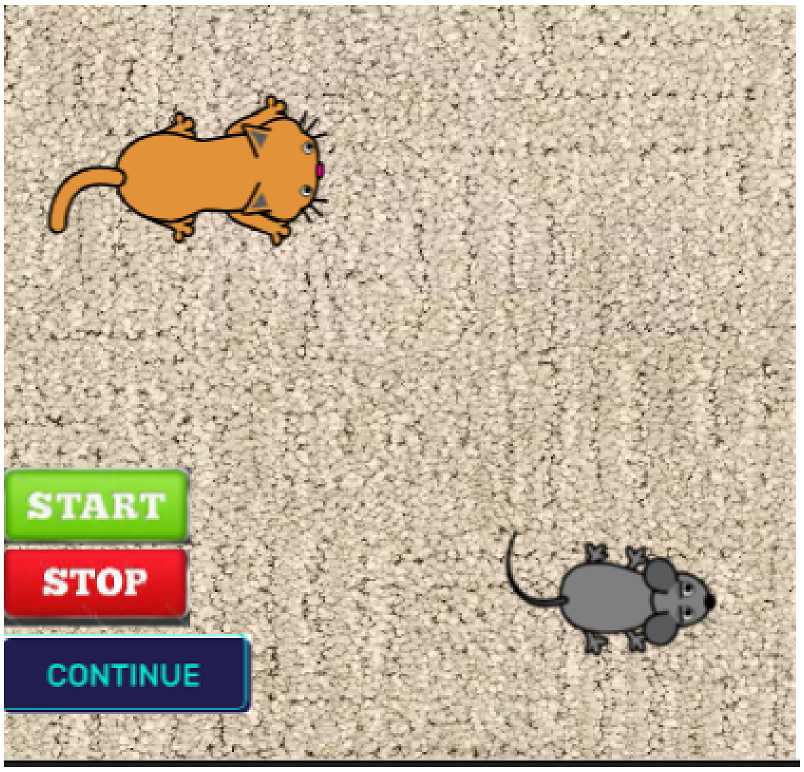
Use push-buttons on the stage

- Click or push-buttons have a specific shape and a label:

- Copy the Cat-and-Mouse animation project and implement three click-buttons:

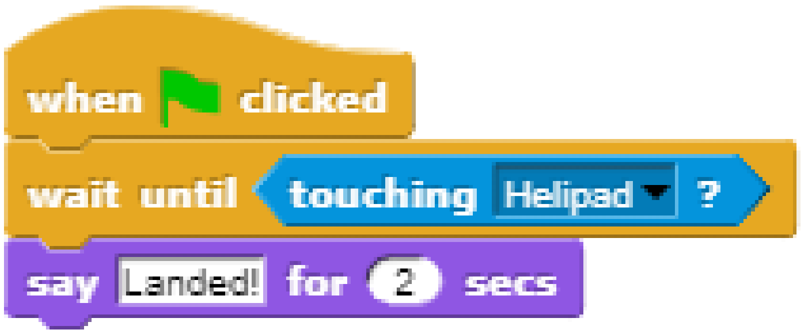
  1. Make three buttons: START, STOP and CONTINUE (you can copy images or - better - make your own).
  2. Use `START` sprite and the `When sprite clicked` event to activate the cat-pounces-and-sits-on-mouse action.
  3. Use the `STOP` sprite to stop the script and reset all scripts.
  4. Use the `CONTINUE` sprite to run the mouse-says-hello-and-disappears action.

- The action should look like shown in this screencast.
- Link to sample solution:

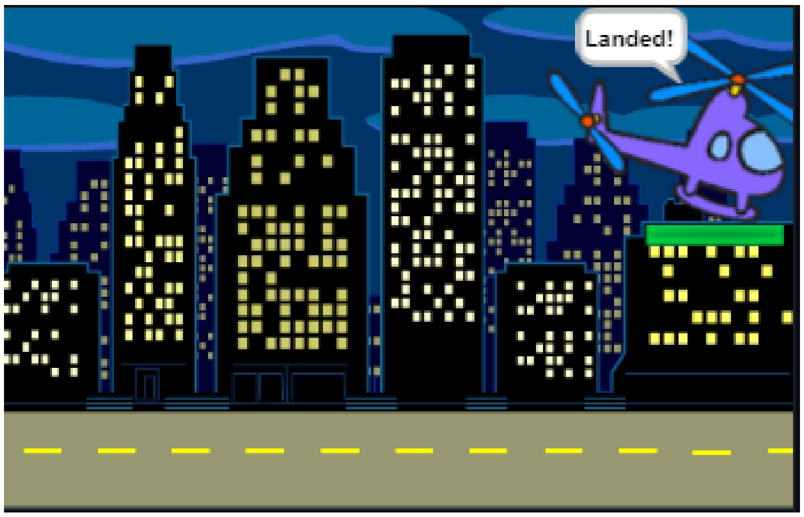


`Wait until...` command

- Use this command if you want a sprite to wait for a condition to become true. It waits as long as the condition is false.
- For example, this block in the minimal helicopter project will be activated only when the helicopter has landed on the helipad:

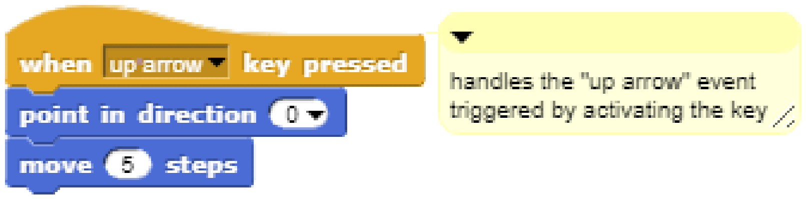


```
when flag clicked
wait until <touching Helipad ?>
say Landed! for 2 secs
```



Keyboard events ('polling')

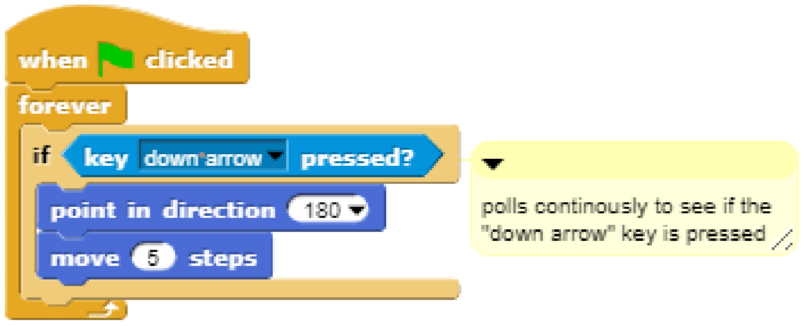
- The CPU has two ways to control events: 'handling' or 'polling'.
- When it handles an event, it starts a process when the event has been triggered:



```
when [up arrow] key pressed
point in direction (0)
move (5) steps
```

handles the "up arrow" event triggered by activating the key

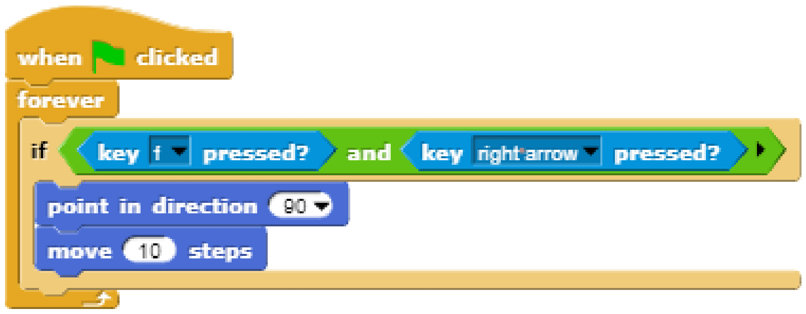
- This is easy on the CPU but it is less responsive than 'polling' where the CPU runs continuously waiting for a signal:



```
when flag clicked
forever
    if <key [down arrow] pressed?>
        point in direction (180)
        move (5) steps
```

polls continously to see if the "down arrow" key is pressed

- The condition has to be wrapped in a `forever` loop to be tested continuously.
- Unlike event handlers (which always start a script), polling commands can be compounded. For example, this script checks if both the 'f' and the 'right arrow' key are pressed before it runs:

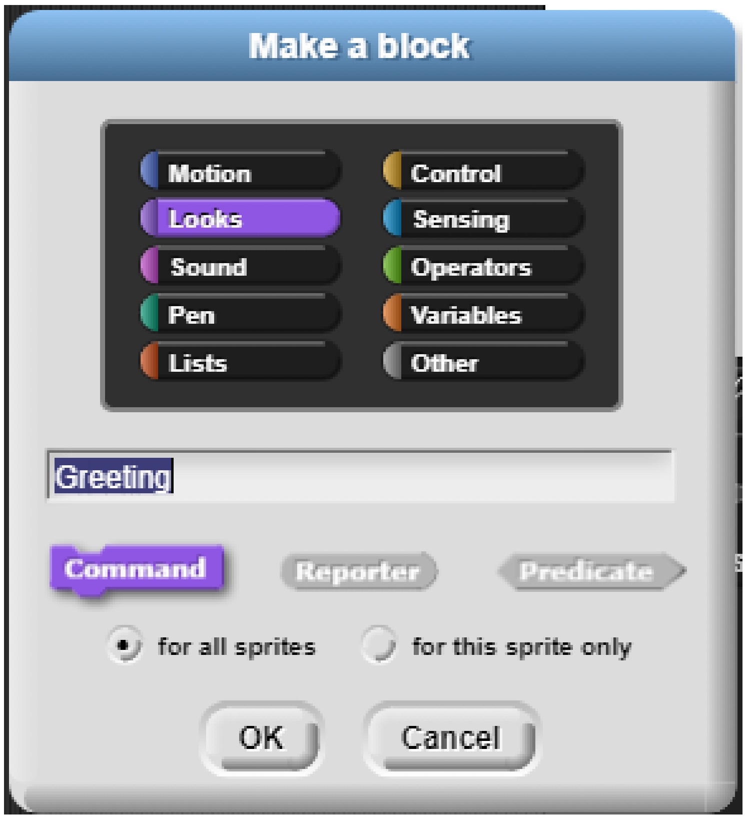


Making blocks

- Snap allows you to define your own procedures using the `Make a block` command (in the sidebar or via right-click in the script area).
- In the dialog, you can specify:

  1. `Command` = procedure without a return value, like a greeting that only prints a string like "hello" on the screen.
  2. `Reporter` = function with a return value, like a computation of F = m * a that takes mass (m) and acceleration (a) as parameters and returns the value of F.
  3. `Predicate` = function that return Boolean (True or False) values only.

  – Make a `Command` block, name it `Greeting` and add a `Looks` command:

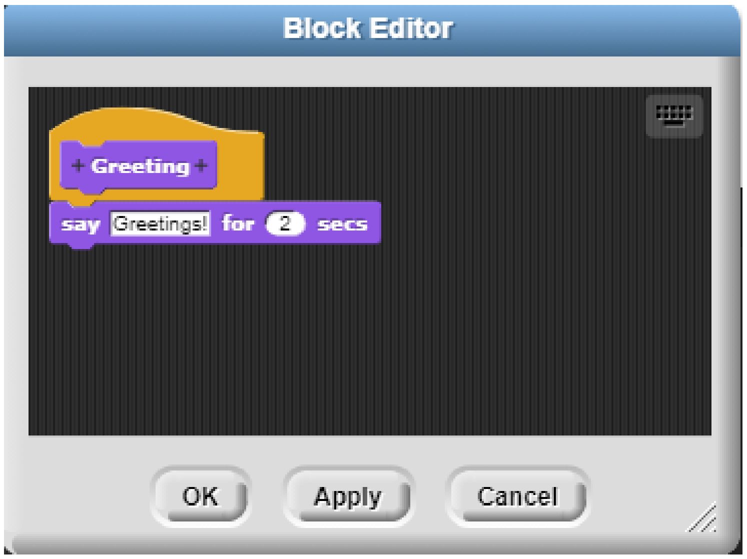


[✓ Looks]

Greeting

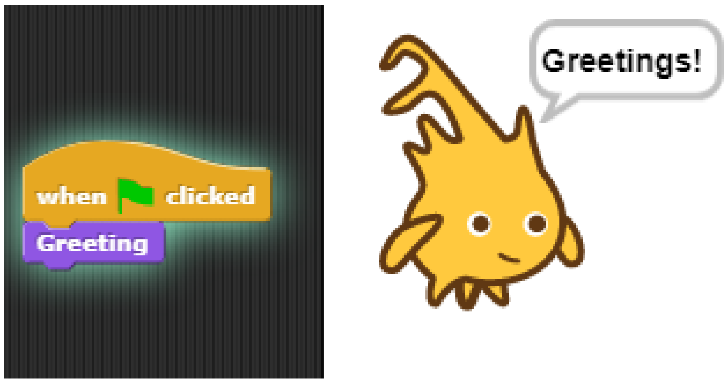
[✓ Command]

[✓ for all sprites]

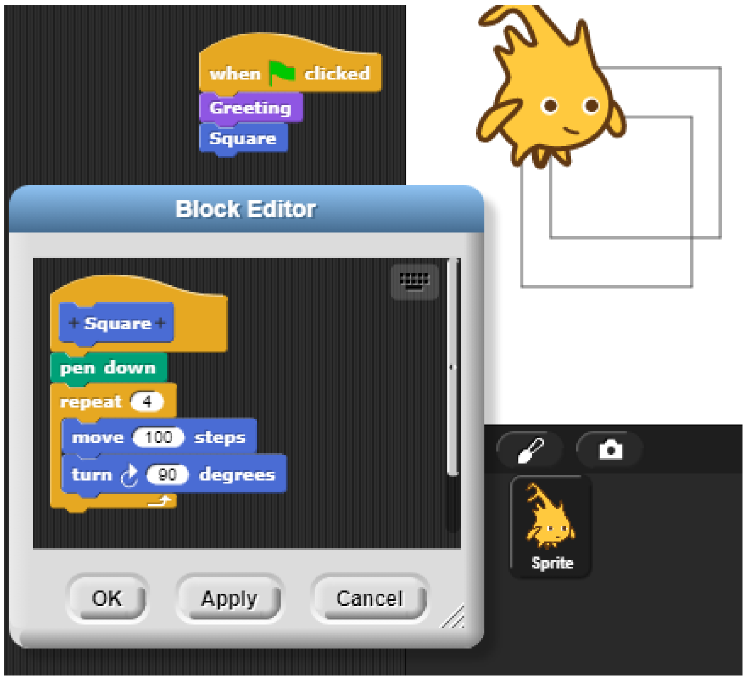


```
+ Greeting +
say Greetings! for 2 secs
```

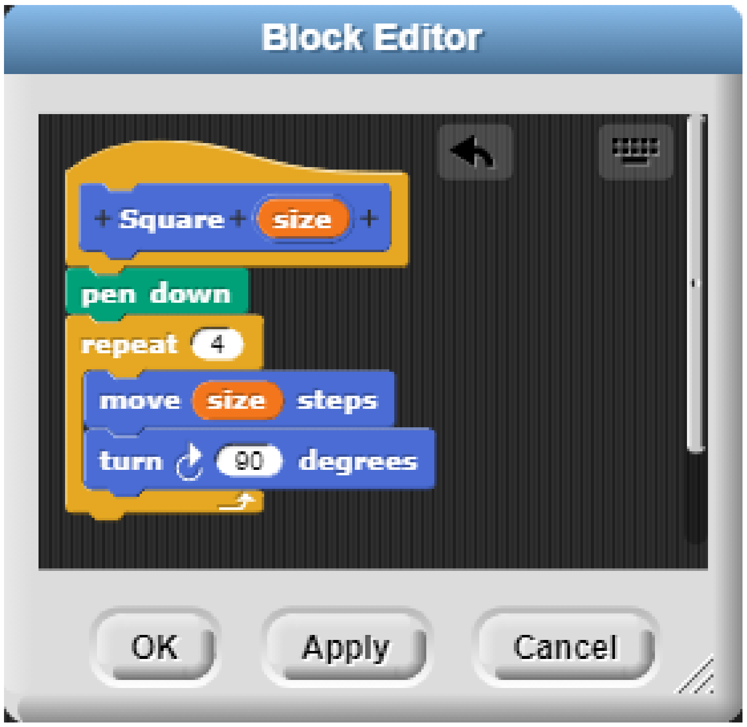
– You can now use the block, which appears in the `Looks` menu, anywhere in the project:



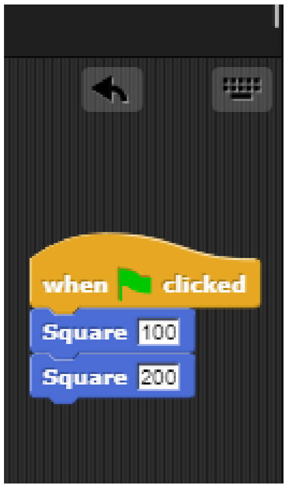
– Another example: make a block `Square` in the `Motion` category and instruct it to draw a square:



– If you want to draw squares of different sizes, you can use the `Input name` dialog: click on the plus on the right of the block name and enter the variable `size`, then exchange the constant 100 steps by the variable `size`:



```
+ Square + (size) +
pen down
repeat 4
    move (size) steps
    turn ↻ 90 degrees
```



```
when 🏁 clicked
Square 100
Square 200
```