**Teacher's Guide for:** Simple Studio for Character Design & Animation

# **Purpose of Project:**

This project aims to explore concepts related to "Lists" in a hands-on and fun manner. Most students enjoy drawing and almost all are fascinated by the animated works of studios like Pixar. This project hopes to bring home the idea of using lists to store and transform data. It also hopes to be a first stepping stone to more advanced character design and animation techniques students may pursue in the future.

#### Time Commitment:

About 4 days.

First day: Watching the introduction video, playing with the Snap! program provided and starting the project design in partner groups.

Second day: Coding. Using the hint questions, and Hints video as needed.

Third day: Finishing the project, exploring some extensions. If stuck, watching the step-by-step videos of the Drawing and Animation phases.

Fourth day: Exploring more extensions and bonus opportunities. Catch up day for those really behind. Classroom sharing.

### Tips:

Be sure to do this lab and some of its extensions on your own prior to introducing it to the students.

Make sure that, while playing with the Snap! program to understand the project, students are <u>not</u> accessing the code. (Hopefully UC Berkeley folks will soon create such a capability: to use a program without access to the code.)

Make sure that, students spend a good amount of time coming up with their own design ideas prior to reading the hints questions and watching the Hints video. Make the step-by-step solution videos on the Drawing and Animation phases available only to students who are really stuck.

Encourage the students who are ahead of schedule to come up with their own extensions prior to reading suggested extensions.

### **Correlation with AP CS Principles Framework:**

EU 5.1 Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).

LO 1.2.2 Create a computational artifact using computing tools and techniques to solve a problem.

EK 5.1.3B Collaboration facilitates multiple perspectives in developing ideas for solving problems by programming.

EK 5.3.1L Using lists and procedures as abstractions in programming can result in programs that are easier to develop and maintain.

# **Possible Code Solutions:**

Code for Initialization & Drawing

```
when elicked
clear
pen up
set xList to list
set yList to list
set step to 25
set pen color to
set pen size to 5
forever
if mouse down?
  go to x: (mouse x) y: (mouse y
  pen down
  add mouse x to xList
  add mouse y to yList
 else
  move 0001 steps
  pen up
```

Possible Code for Movement with UP arrow:

```
when up arrow * key pressed

warp

clear

set i to 1

repeat length of xList

replace item i of yList with item i of yList + step

go to x: item i of xList y: item i of yList

pen down

move 0.0001 steps

pen up

change i by 1
```

Can you guess how can the remaining key movements be coded?

```
when down arrow key pressed

warp

clear

set to 1

repeat length of xList

replace item i of yList with item i of yList - step

go to x: item i of xList y: item i of yList

pen down

move 0.0001 steps

pen up

change by 1
```

```
when nght arrow key pressed

warp

clear

set i to 1

repeat length of xList

replace item i of xList with item i of xList + step

go to x: item i of xList y: item i of yList

pen down

move 0.0001 steps

pen up

change i by 1
```

```
when left arrow key pressed

warp

clear

set to 1

repeat length of xList

replace item i of xList with item i of xList - step

go to x: item i of xList y: item i of yList

pen down

move 0.0001 steps

pen up

change by 1
```

Possible single block of to unify behavior from the up/down right/left keys:

```
when any key key pressed
set xStep v to 0
set yStep v to 0
if key up arrow pressed?
set yStep v to step
If key down arrow pressed?
set yStep to 1 x step
if key right arrow pressed?
set xStep v to step
if key lettarrow pressed?
set xStep to 1 x step
warp
clear
set 🔽 to 1
 repeat length of xList
  replace item ( ) of xList with ( item ( ) of xList
                                                   + xStep
  replace item (i) of (yList) with (item (i)
                                         of yList + yStep
  go to x: item (i) of xList y; item (i) of yList
  pen down
  move 0.0001 steps
  pen up
  change | by 1
```

Hint for the possible to use a single list of lists to keep track of mouse positions:

```
add (list (mouse x) mouse y) () to (locationList)
```