SnapPy Documentation

The purpose of this document is to describe the available SnapPy functions and how they are used.

SnapPy Program Structure

All SnapPy programs should start with "`import snappy`" on the first line of the file, and should have "`snappy.start()`" as the last line in the file. Currently the stage is a fixed size of 480x320, with the same coordinate system as BYOB/Snap!. Locations are given as pairs (x,y).

Costumes

A Costume object is a picture that can be used by a sprite when it is drawn. The costume can be loaded from an image file (all common formats supported, but typically you will want to use an image format that supports transparency like PNG or GIF), or you can use one of the built-in costumes. To create a costume object initialized from a file, you use the `snappy.CostumeFile(filename)` function, like

```
costume = snappy.CostumeFile('joey.png')
```

To create a costume object from the built-in costumes, you use `snappy.CostumeBuiltIn(name)`, where the name argument is one of the following:

Clock-hand
alonzo
dragon1-a
dragon1-b
car2
car-blue
car-bug
car-cow
bomb
wizardhat

Sprites

The basic graphical object in SnapPy is an object of type `snappy.Sprite`. The constructor for a sprite takes 4 arguments, which are, in order, location, costume, proto, and hidden. They all have default values, so can be specified out of order. A typical sprite creation will specify the first two parameters (location and costume), but you can use just the proto argument if you want to copy/clone an existing object. For example, to create an alonzo sprite with the standard costume at the center of the screen, you could use

```
alonzo = snappy.Sprite((0,0), snappy.CostumeBuiltIn('alonzo'))
```

To clone an arrow sprite named aiming to create a new sprite named shot, you could use

```
shot = snappy.Sprite(proto=aiming)
```

Sprites have the following attributes that can be read (do not assign values to these, but they can be read at any time):

```
sprite.location
sprite.size
sprite.direction
sprite.rotationStyle
sprite.costume_num
```

The following methods are available for any sprite – the functions parallel the similarly-named blocks in BYOB, so I don't provide any more detailed explanation here.

Motion methods

```
sprite.move(steps)
sprite.turnCW(degrees)
sprite.turnCCW(degrees)
sprite.pointInDirection(dir)
sprite.goTo(pos)
sprite.glideForTo(duration, to)
sprite.changeXBy(deltaX)
sprite.setXTo(newX)
sprite.changeYBy(deltaY)
sprite.setYTo(newY)
sprite.if_on_edge_bounce()
```

Looks methods

```
sprite.switchToCostume(number)
sprite.sayFor(text, duration)
sprite.say(text)
sprite.changeSizeBy(delta)
sprite.setSize(size)
sprite.show()
sprite.hide()
sprite.goToFront()
sprite.goBackLayers(num)
```

Sensing methods

```
sprite.touching_edge()
sprite.touching(other, offset=(0,0))
```

Miscellaneous methods

```
sprite.loadCostume(costume)
sprite.setRotStyle(style)
sprite.wait(duration)
```

Global functions (not Sprite methods)

```
snappy.launch(fn)
snappy.broadcast(signalName)
snappy.stop_all_scripts()
```

Event Handlers

Functions can be used as event handlers – code that will be executed whenever a certain event occurs, such as a specific key being pressed, the green flag being clicked, or a signal received. A signal handler is defined just like any other function, but should not take any arguments. If you need to pass data into an event handler, the only way to do that at this time is to use global variables.

To turn a function into an event handler, you precede the function "`def`" line with a line that starts with "`@snappy.`" – here are the options:

`@snappy.startOnGreenFlag()`

Runs the function whenever the green flag in the SnapPy run window is clicked

`@snappy.startOnKey(key)`

Runs whenever the given key is pressed. key can be either a single letter or symbol key (such as 'a', ' ', or '@') or it can be one of the following words, representing the arrow keys: 'left', 'right', 'up', or 'down'.

`@snappy.startOnReceiving(signalName)`

Runs whenever any sprite or function broadcasts the signal named in this line. Names are arbitrary strings. To broadcast a signal use `snappy.broadcast(signalName)`. For example, if you wanted a signal to represent whenever something was hit, you could name the signal 'hit' and broadcast it using `snappy.broadcast('hit')`. Then any function definition that was preceded with `@snappy.startOnReceiving('hit')` would be executed.

Example

The following is an example of a complete SnapPy program.

```
import snappy

alonzo = snappy.Sprite((-70,0), snappy.CostumeBuiltIn('alonzo'))
dragon = snappy.Sprite((120,0), snappy.CostumeBuiltIn('dragon1-a'))
dragon.setRotStyle(2)
dragon.pointInDirection(-90)
dragon.loadCostume(snappy.CostumeBuiltIn('dragon1-b'))

@snappy.startOnGreenFlag()
    def DragonScript1():
    sprite = dragon
    sprite.goTo((120,2))
    while True:
        sprite.glideForTo(0.5, (120,12))
        sprite.glideForTo(0.5, (120,2))

@snappy.startOnGreenFlag()
def DragonScript2():
    sprite = dragon
    sprite.sayFor("Hello!", 1)
    sprite.sayFor("I'm a dragon", 1)
    snappy.broadcast('Dragon Done')

@snappy.startOnReceiving('Alonzo Done')
def DragonScript3():
    sprite = dragon
    sprite.sayFor("Want to see me breathe fire?", 1)
    sprite.switchToCostume(1)
    sprite.wait(1)
    sprite.switchToCostume(0)
    sprite.sayFor("Isn't that cool?", 1)

@snappy.startOnReceiving('Dragon Done')
def AlonzoScript2():
    sprite=alonzo
    sprite.sayFor("Hi, I'm Alonzo", 1)
    sprite.sayFor("I'm a...  well, I'm not sure", 1)
    snappy.broadcast('Alonzo Done')

snappy.start()
```