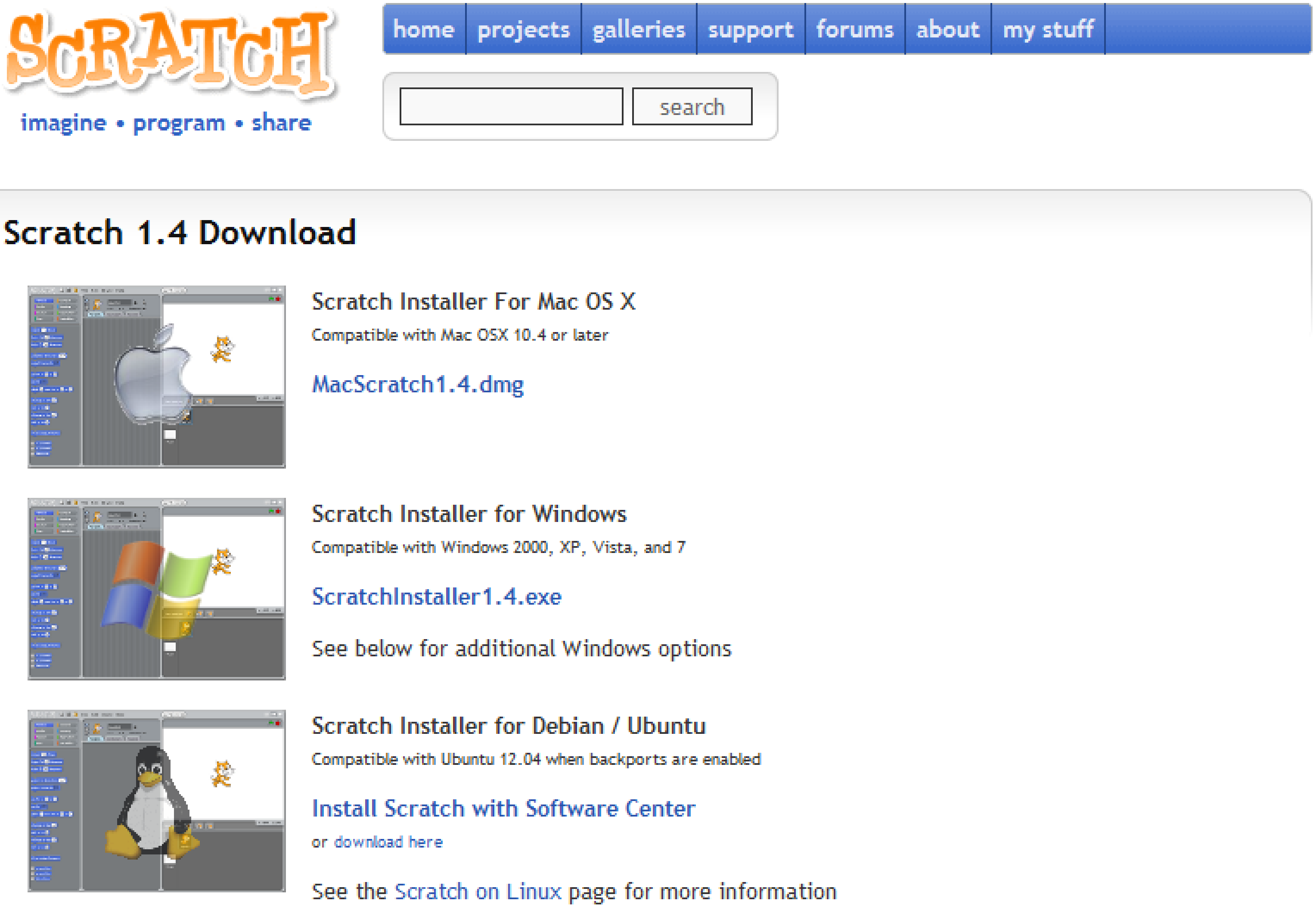
5. Visual Programming 3

What did we do last time?

SCRATCH is a new programming language that let you create your own interactive stories, animations, games, music and art.

SCRATCH – visual programming



Scratch 1.4 Download

Scratch Installer For Mac OS X

Compatible with Mac OSX 10.4 or later

MacScratch1.4.dmg

Scratch Installer for Windows

Compatible with Windows 2000, XP, Vista, and 7

ScratchInstaller1.4.exe

See below for additional Windows options

Scratch Installer for Debian / Ubuntu

Compatible with Ubuntu 12.04 when backports are enabled

Install Scratch with Software Center

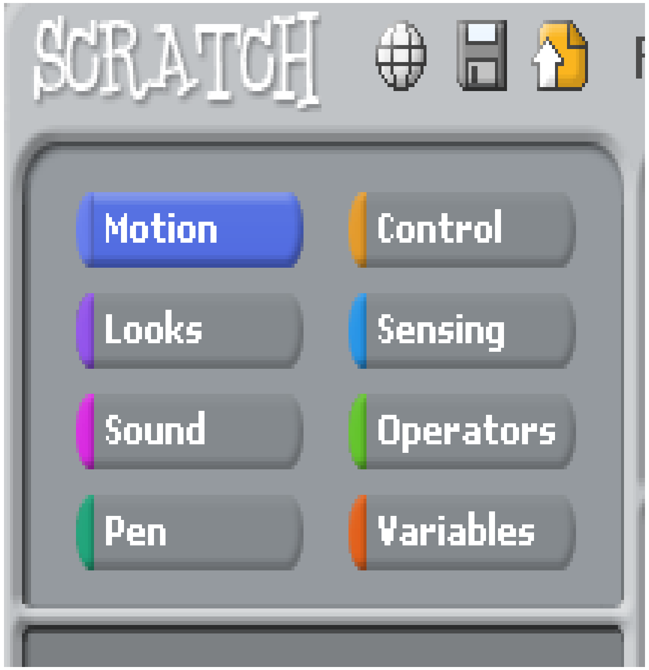
or download here

See the Scratch on Linux page for more information

http://info.scratch.mit.edu/Scratch_1.4_Download

About SCRATCH

- Scratch allows the user to write programs by dragging and connecting simple programming instructions.
- The programming instructions resemble puzzle pieces and will only “fit” together in ways that make semantic sense.
- The instruction pieces are also color-coded according to what type of instruction they represent.
- The program that the user creates controls one or more objects, or sprites.

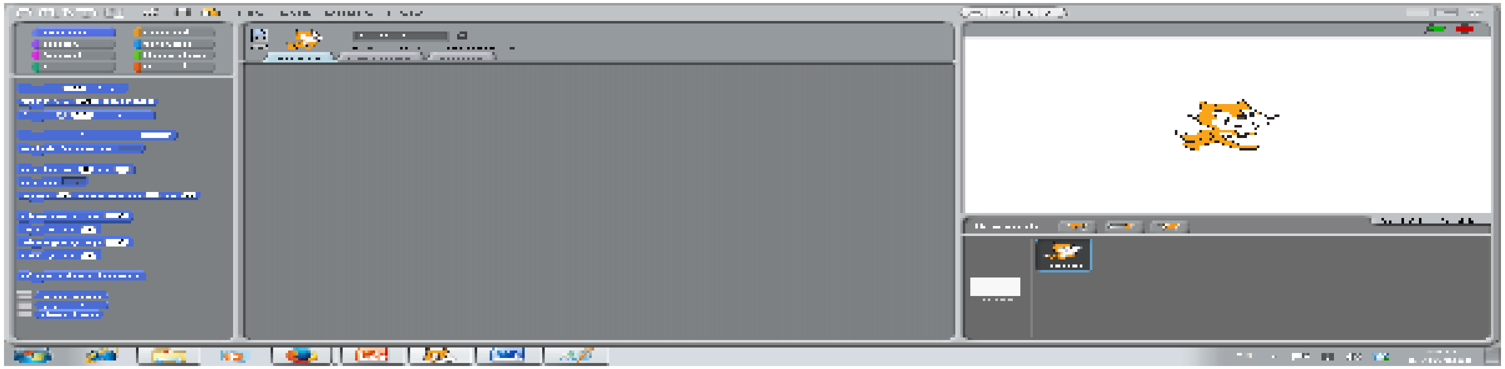


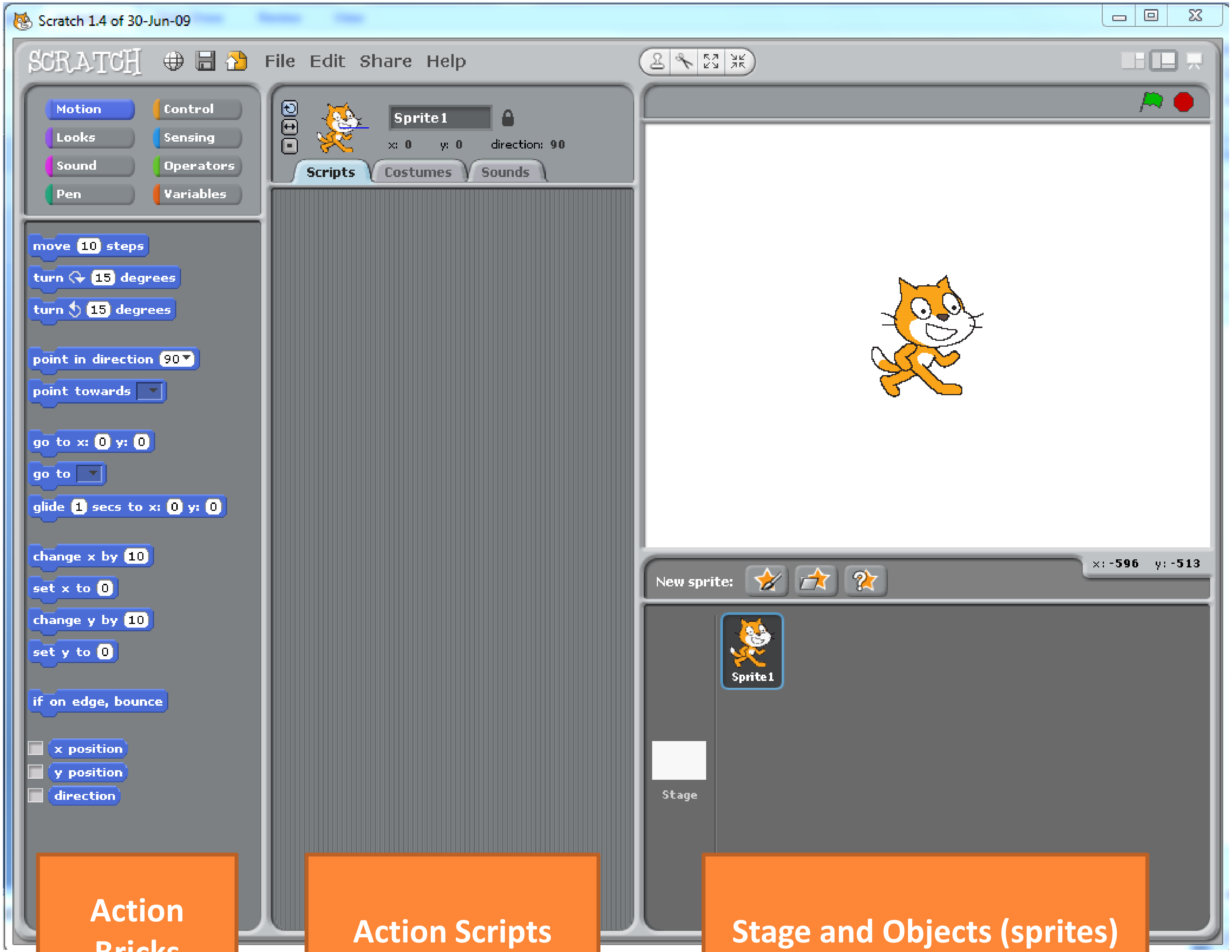
Eight categories of programming institutions

- **Motion:** move and rotate.
- **Looks:** changing a sprite's costume and colour, and “say” and “think” messages to the user.
- **Sound:** playing drum sounds as well as 128 different instruments and sound affects.
- **Pen:** ability to draw lines under program control.
- **Control:** control structure such as while loops and if statements.
- **Sensing:** allow the user's program to test the location of a sprite or the mouse pointer.
- **Operators:** arithmetic, boolean, and string operators that can be combined to form complex expressions.
- **Variables:** allow the user to create, display and manipulate scalar and list variables.

SCRATCH interface breaks out into 3 columns

- The **left column** contains the various instructions that the user can choose from to build a program.
- The **right column** is divided into two parts. The top part is the "stage" where all of the action takes place. The bottom part contains one or more sprites that are used in the program.
- The **center column** is where the actual programming takes place. The user simply drags programming instructions from the pallet into the center column and connects them together to build up one or more programs that control the current sprite.

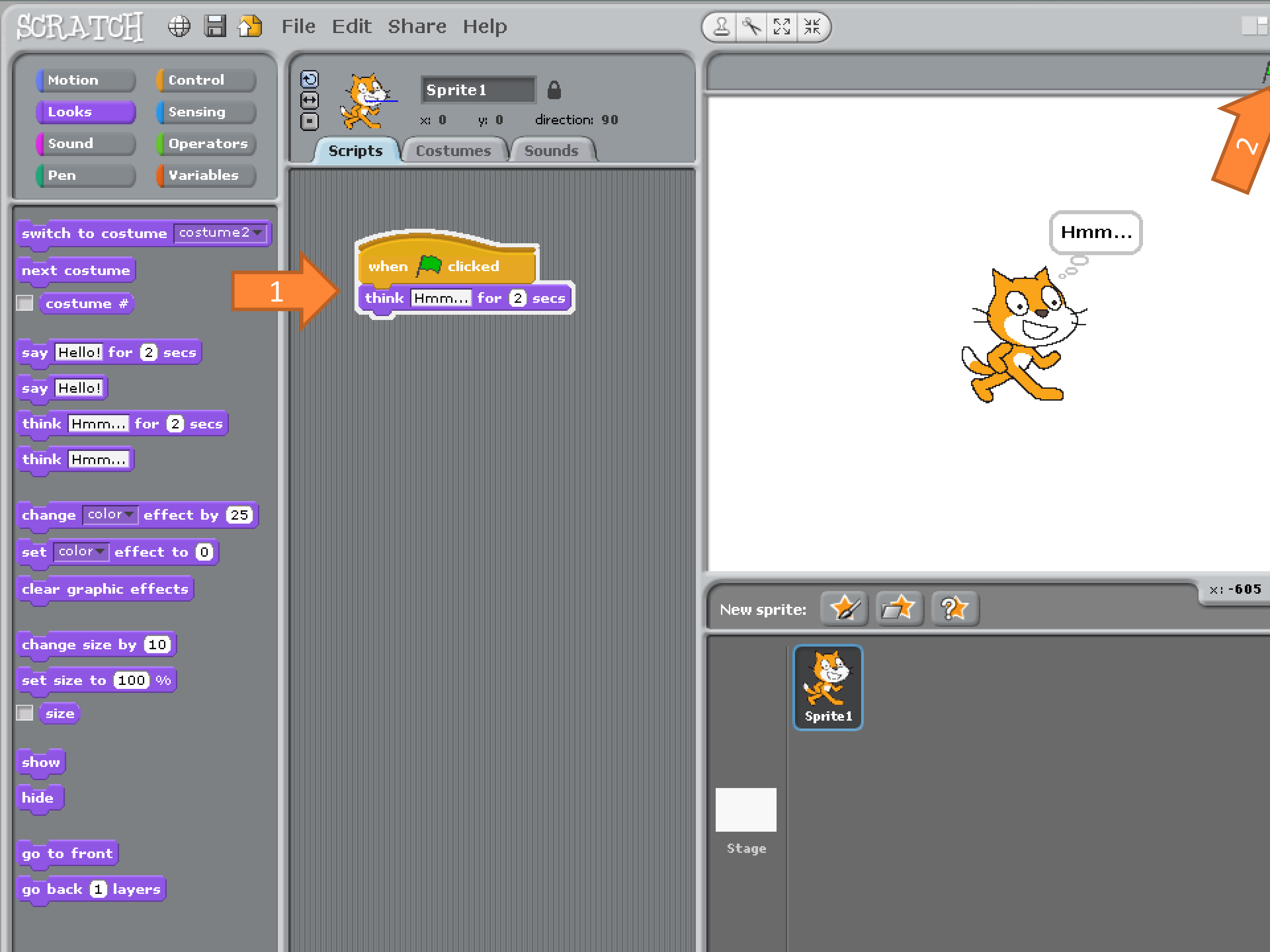




Action Bricks

Action Scripts

Stage and Objects (sprites)



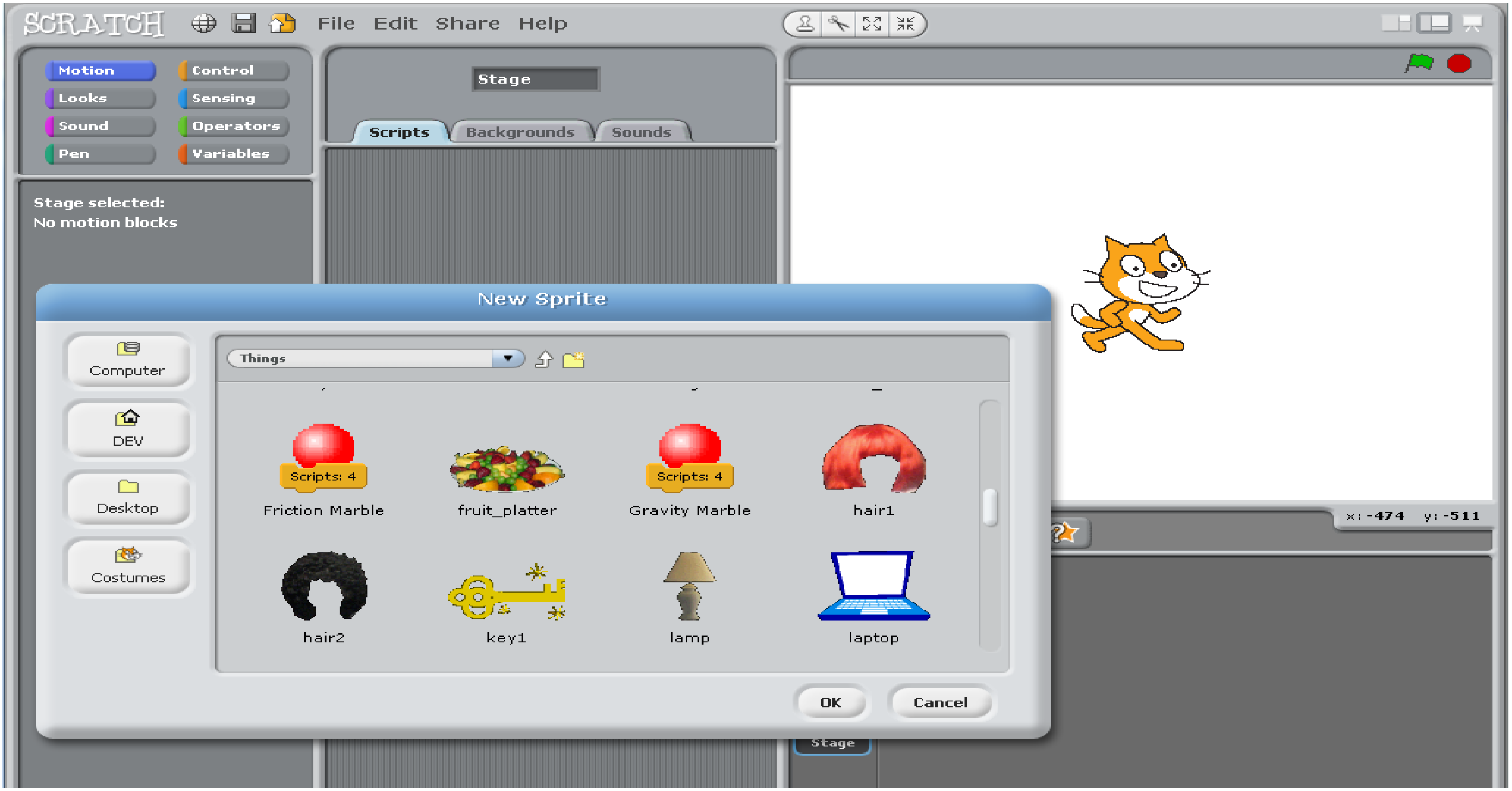
1

```
when [green flag] clicked
think [Hmm...] for (2) secs
```

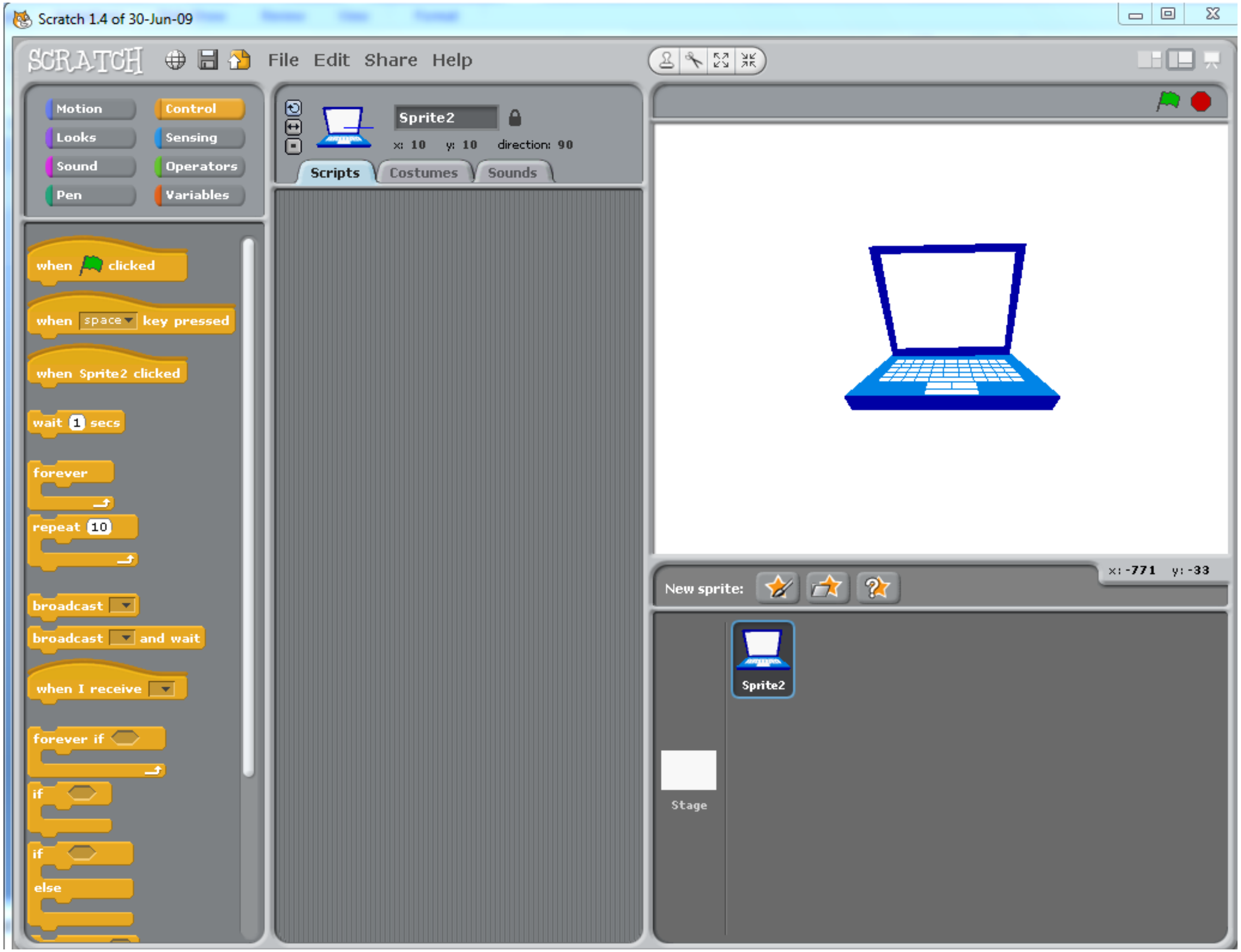
2

Change Sprite

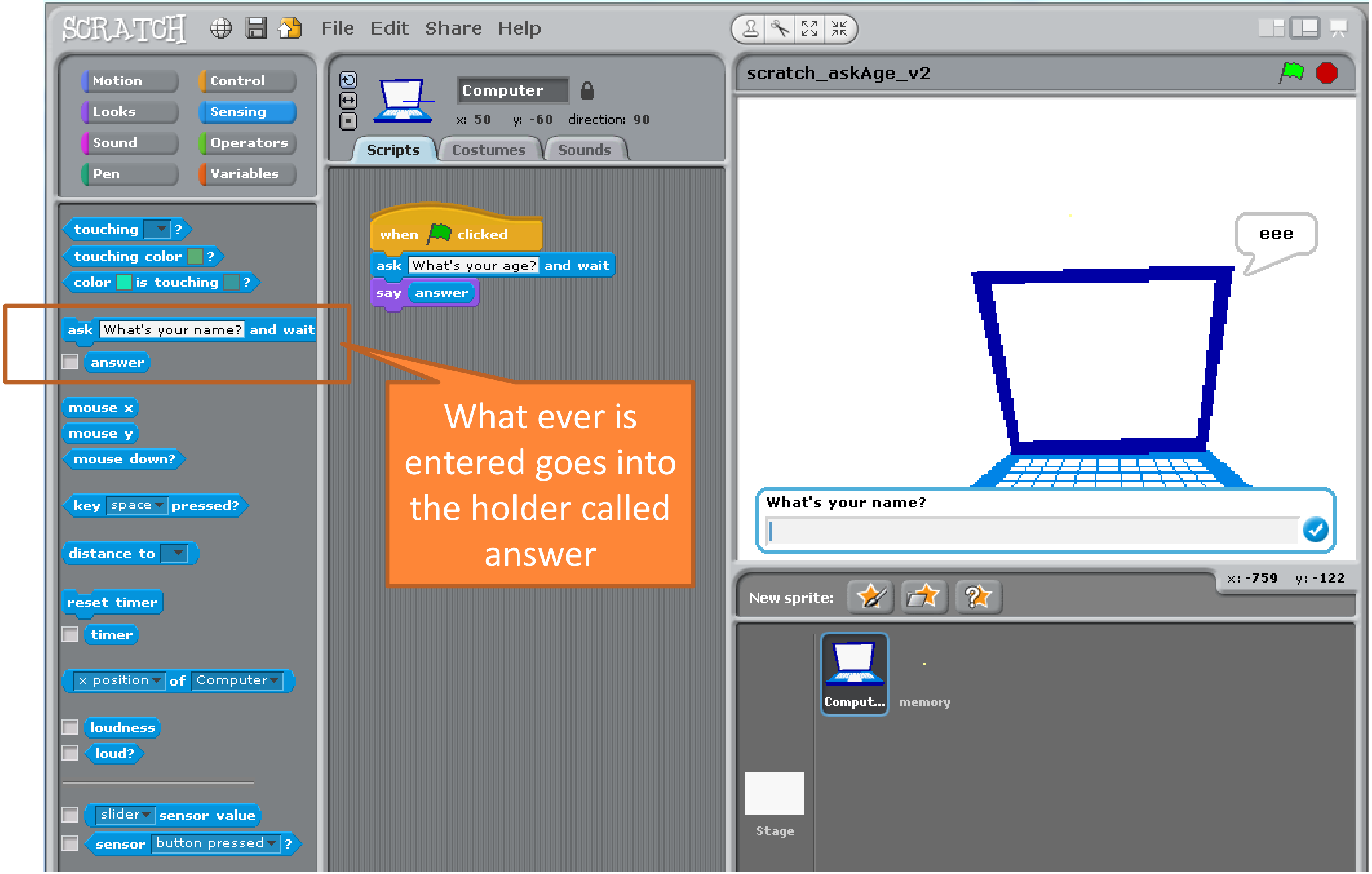
- Scratch includes several different sprites in quite a few different categories. However, the user can also import their own graphics or use the built-in sprite editor.



Want to ask for your age and display answer



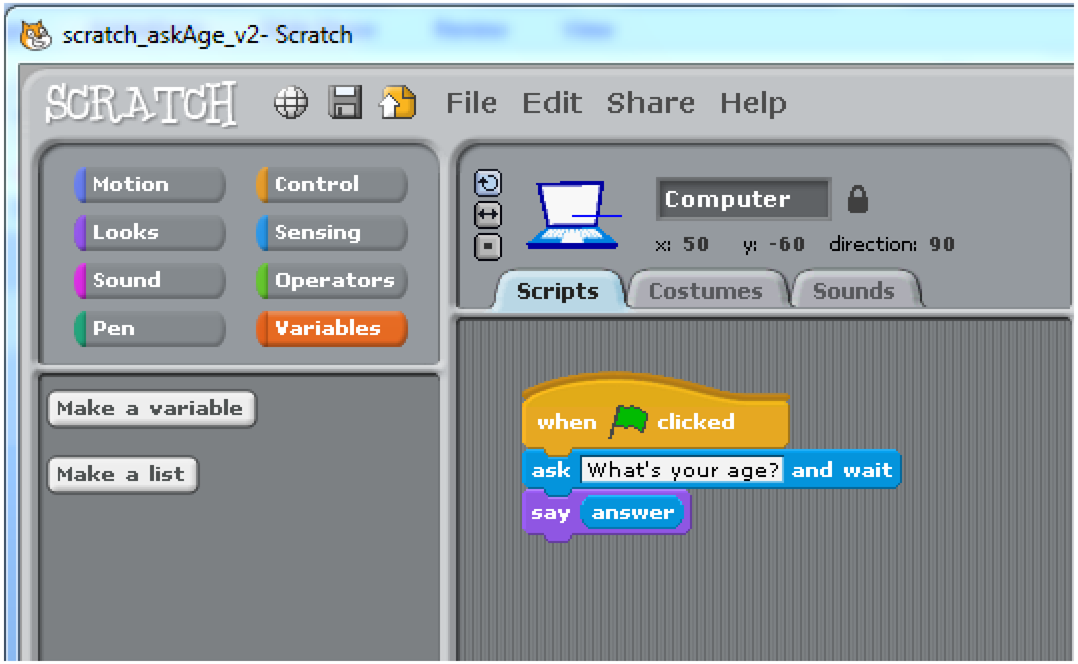
Ask Age and Show it



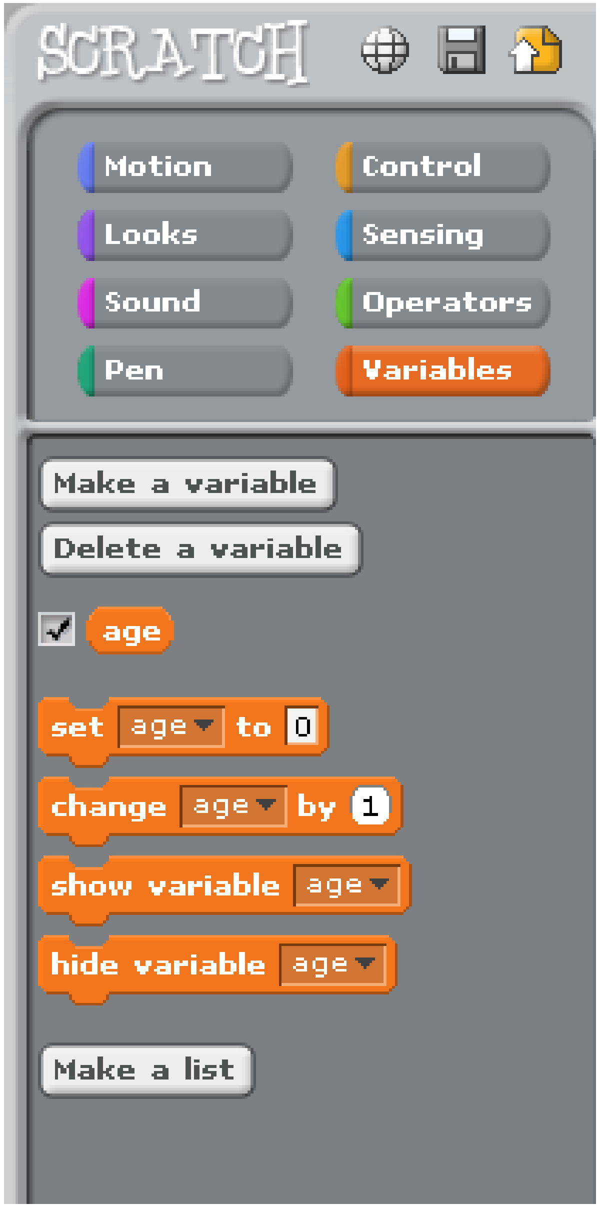
```
when 🏁 clicked
ask What's your age? and wait
say answer
```

What ever is entered goes into the holder called answer

Lets create a variable

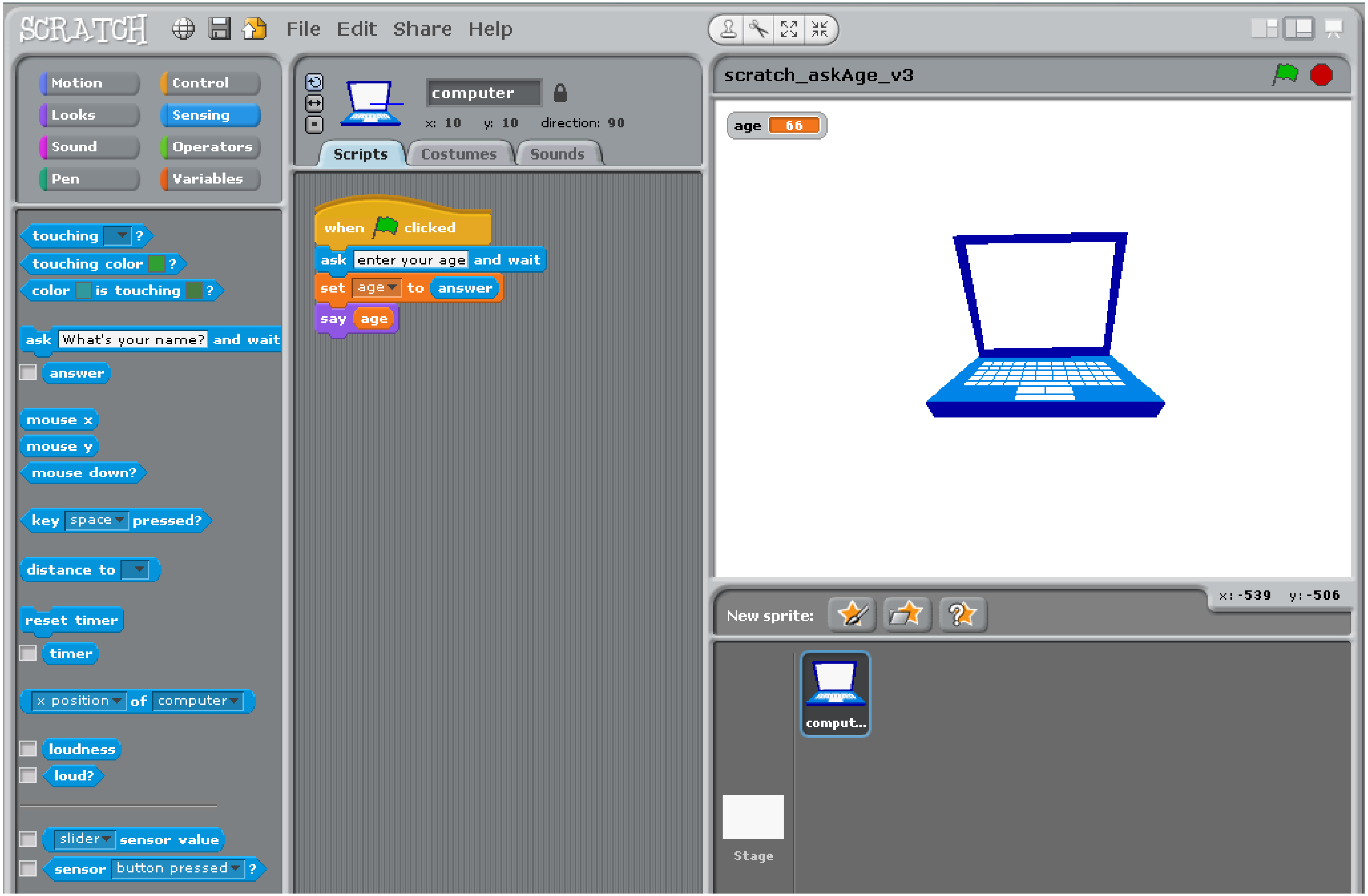


```
when [flag] clicked
ask What's your age? and wait
say answer
```



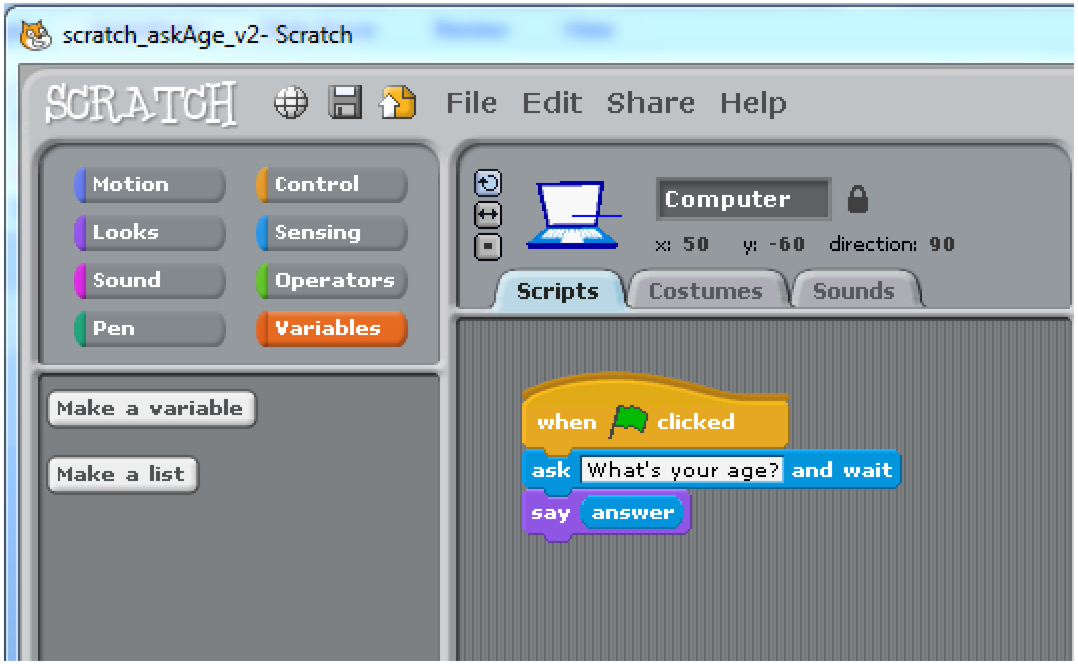
Writing Pad

Use the variable

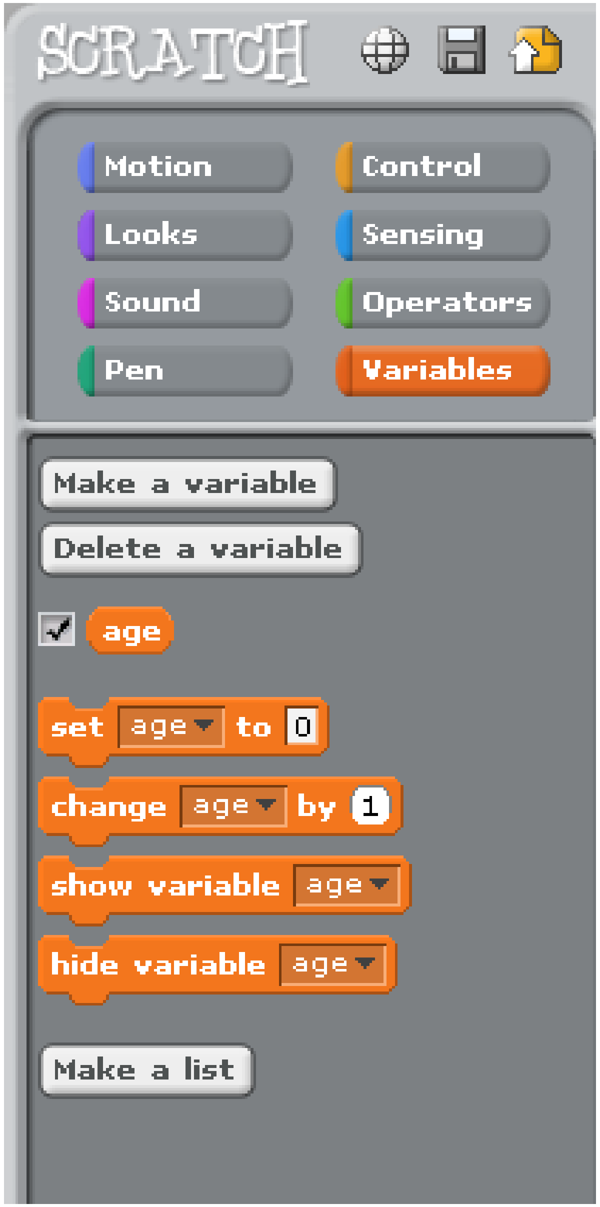


```
when 🟩 clicked
ask [enter your age] and wait
set [age ▾] to (answer)
say (age)
```

Lets create a variable



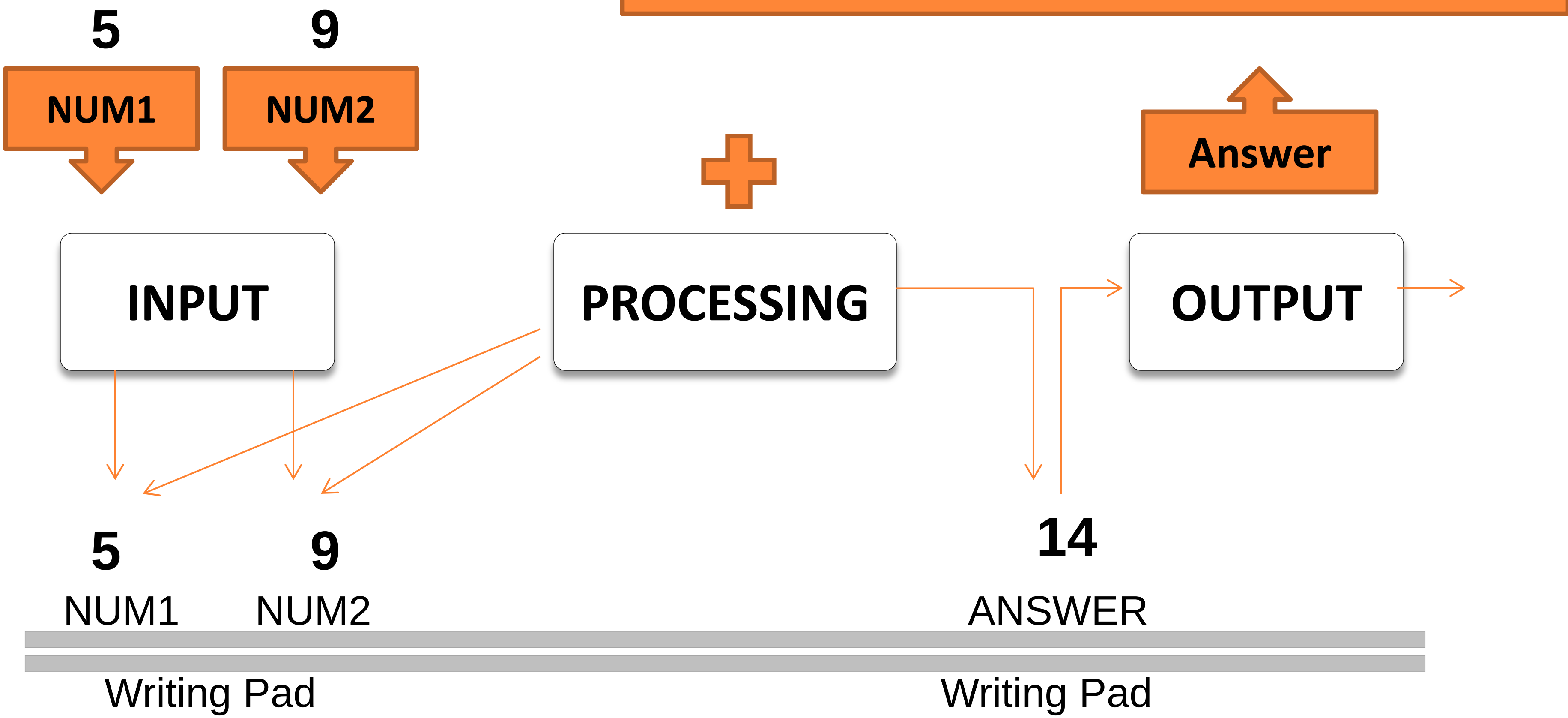
```
when [flag] clicked
ask What's your age? and wait
say answer
```



Writing Pad

Want to write this program

What do I need?



5

9

NUM1

NUM2

Answer

INPUT

PROCESSING

OUTPUT

5

9

14

NUM1

NUM2

ANSWER

Writing Pad

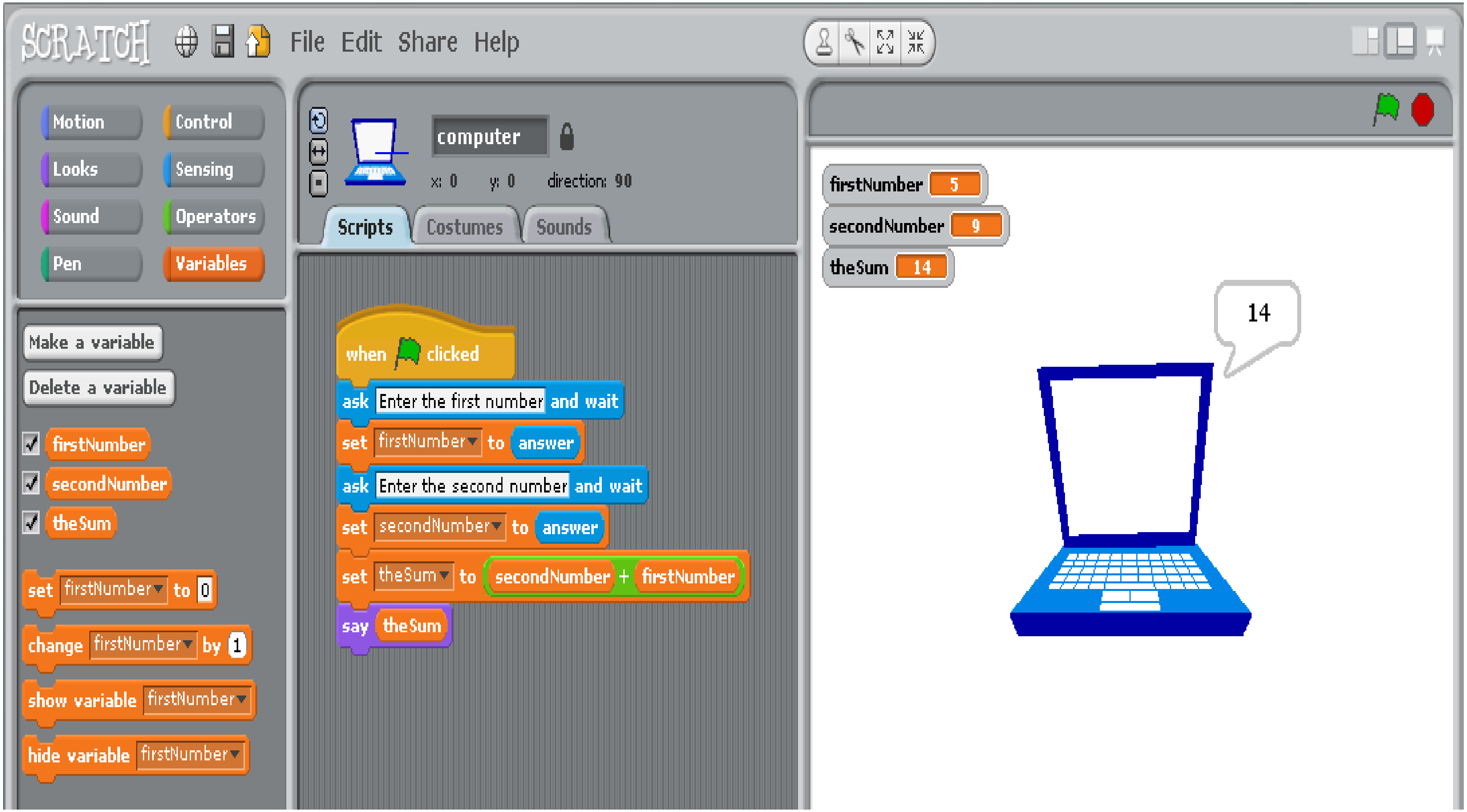
Writing Pad

Program Design Process

1. Problem Definition
   - What is the objective
   - What is the program to do
2. Design
   - Draw a picture or the execution steps
   - Write down in words the execution steps
3. Test Cases (how will you test it)
   - Write what you will use for testing that it runs and creates the right answer
     - Test Case 1 : 1 + 1 = 2 Simple Case
     - Test Case 2: 5 + 9 = 14 Normal Case
     - Test Case 3: 0 + 9 = 9 Edge condition
     - Test Case 4 : 5 + 0 = 5 Edge condition

       If this was division we could have division by zero issues and very small answers
     - Test Case 5 : 166666666666 + 788777777777777 = 788944444444443 test the very big numbers
4. Write Code
   - Step by step, on piece of functionality at a time, get it working, save a copy of that working version , then add the next bit of functionality.
5. Test Code with test cases
   - Debug the code, change ONLY ONE thing at a time, KEEP SAVING VERSIONS

Run the program



x: 0 y: 0 direction: 90

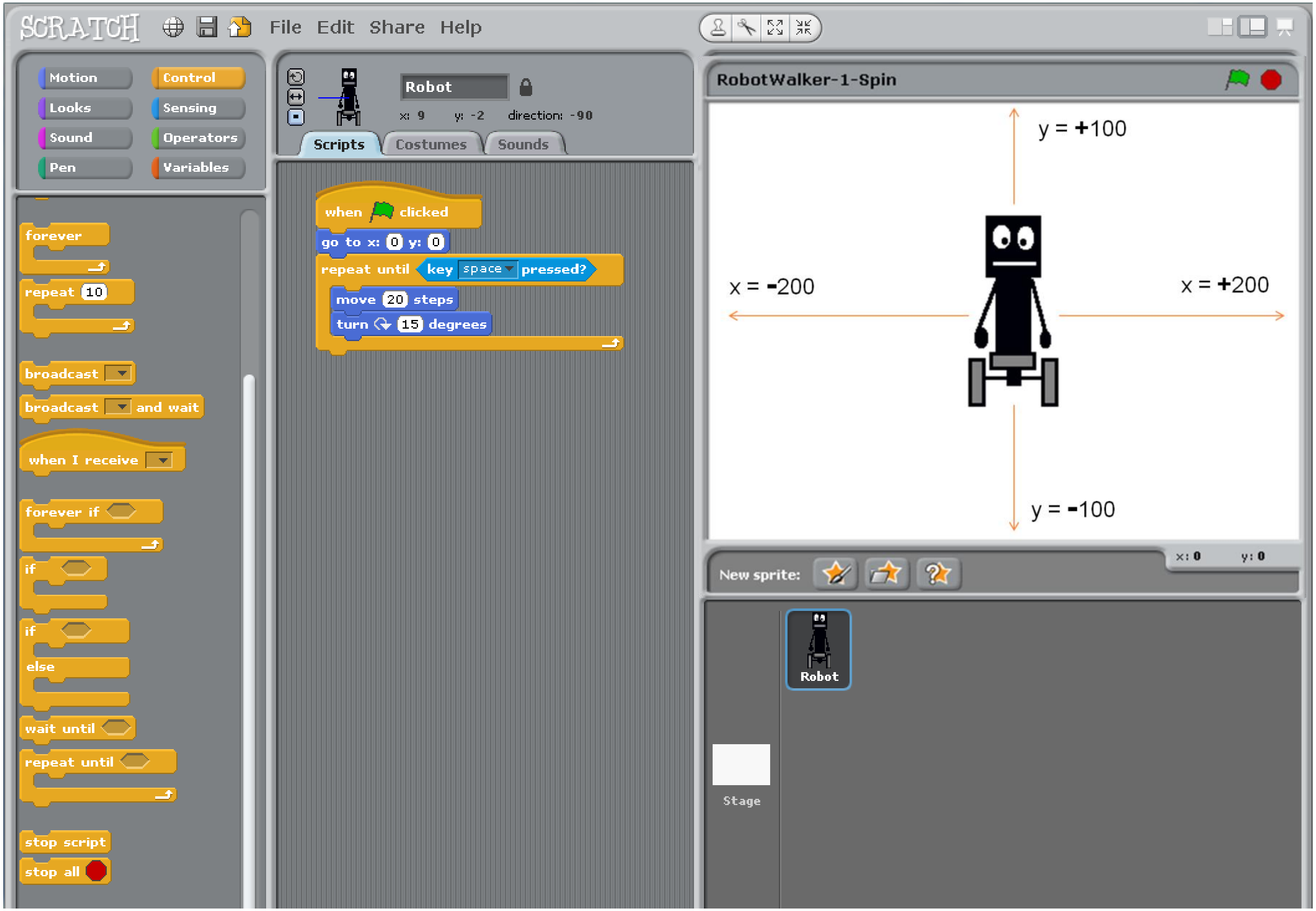
☑ firstNumber
☑ secondNumber
☑ theSum

```
set firstNumber to 0
change firstNumber by 1
show variable firstNumber
hide variable firstNumber
```

```
when [green flag] clicked
ask Enter the first number and wait
set firstNumber to answer
ask Enter the second number and wait
set secondNumber to answer
set theSum to secondNumber + firstNumber
say theSum
```

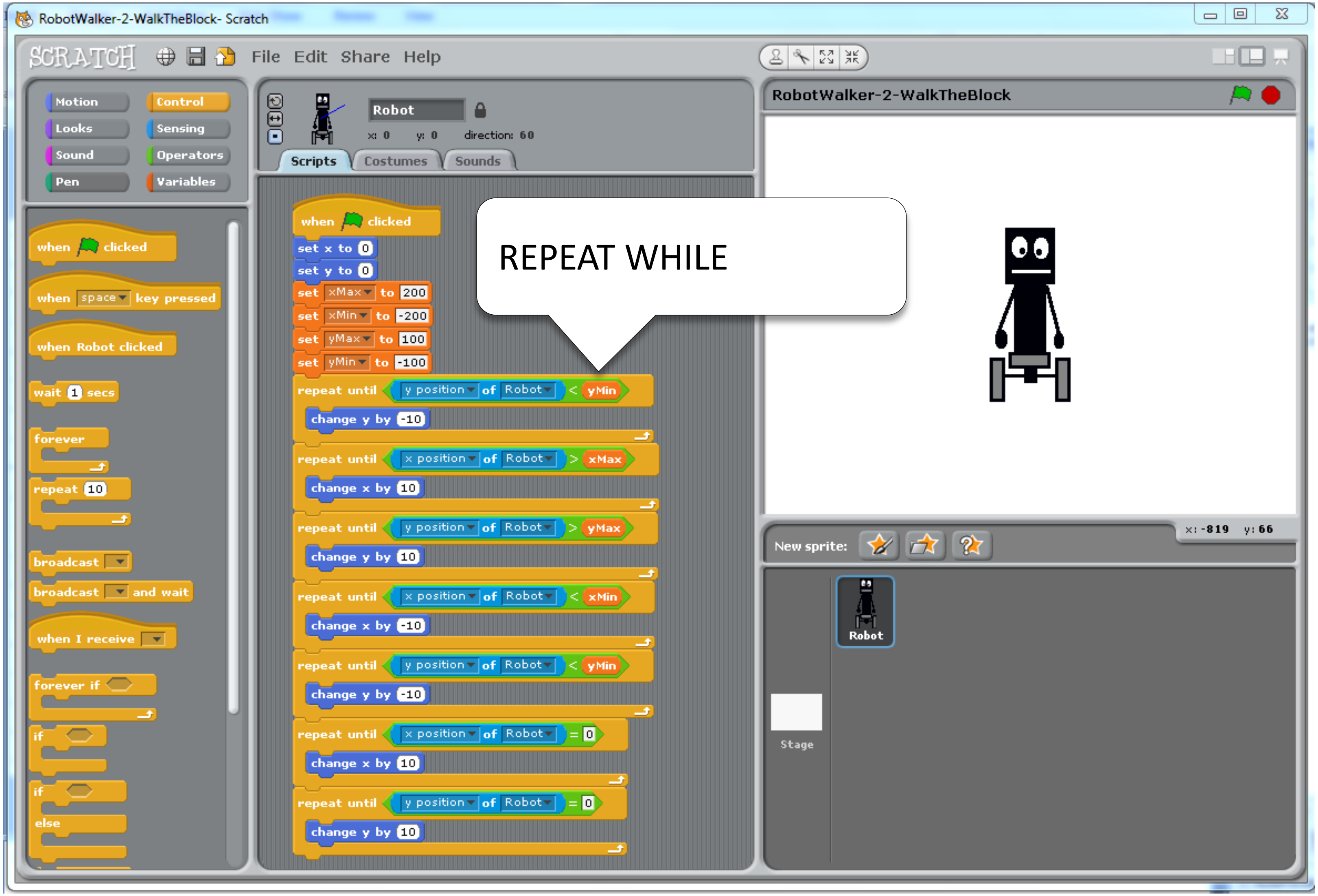
firstNumber	5
secondNumber	9
theSum	14

Moving Scooter around the screen, forever



```
when [flag] clicked
go to x: 0 y: 0
repeat until <key space pressed?>
    move 20 steps
    turn ↻ 15 degrees
```

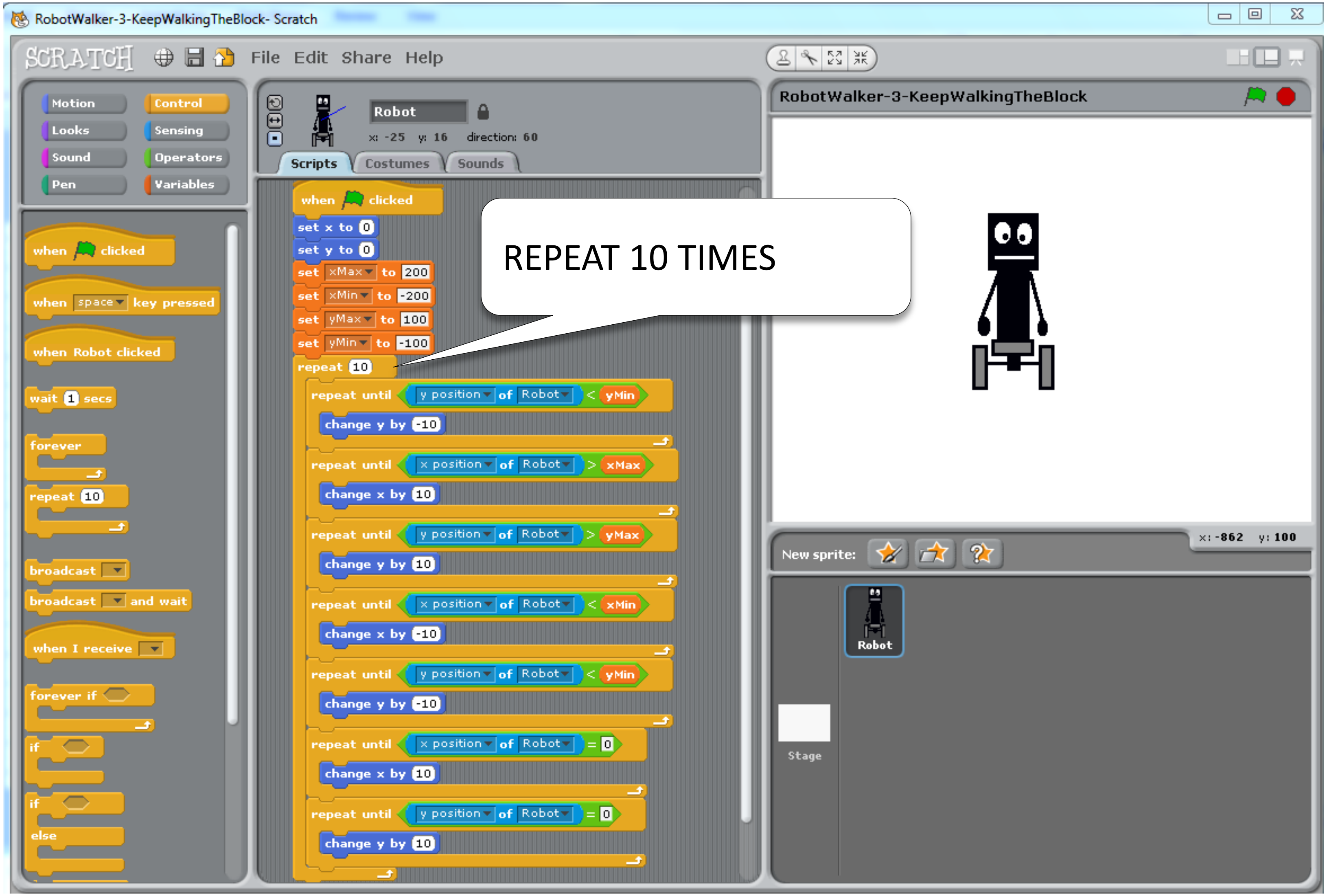
Walking the block and not falling off the edge



```
when [flag] clicked
set x to 0
set y to 0
set xMax to 200
set xMin to -200
set yMax to 100
set yMin to -100
repeat until < y position of Robot < yMin >
    change y by -10
repeat until < x position of Robot > xMax >
    change x by 10
repeat until < y position of Robot > yMax >
    change y by 10
repeat until < x position of Robot < xMin >
    change x by -10
repeat until < y position of Robot < yMin >
    change y by -10
repeat until < x position of Robot = 0 >
    change x by 10
repeat until < y position of Robot = 0 >
    change y by 10
```

REPEAT WHILE

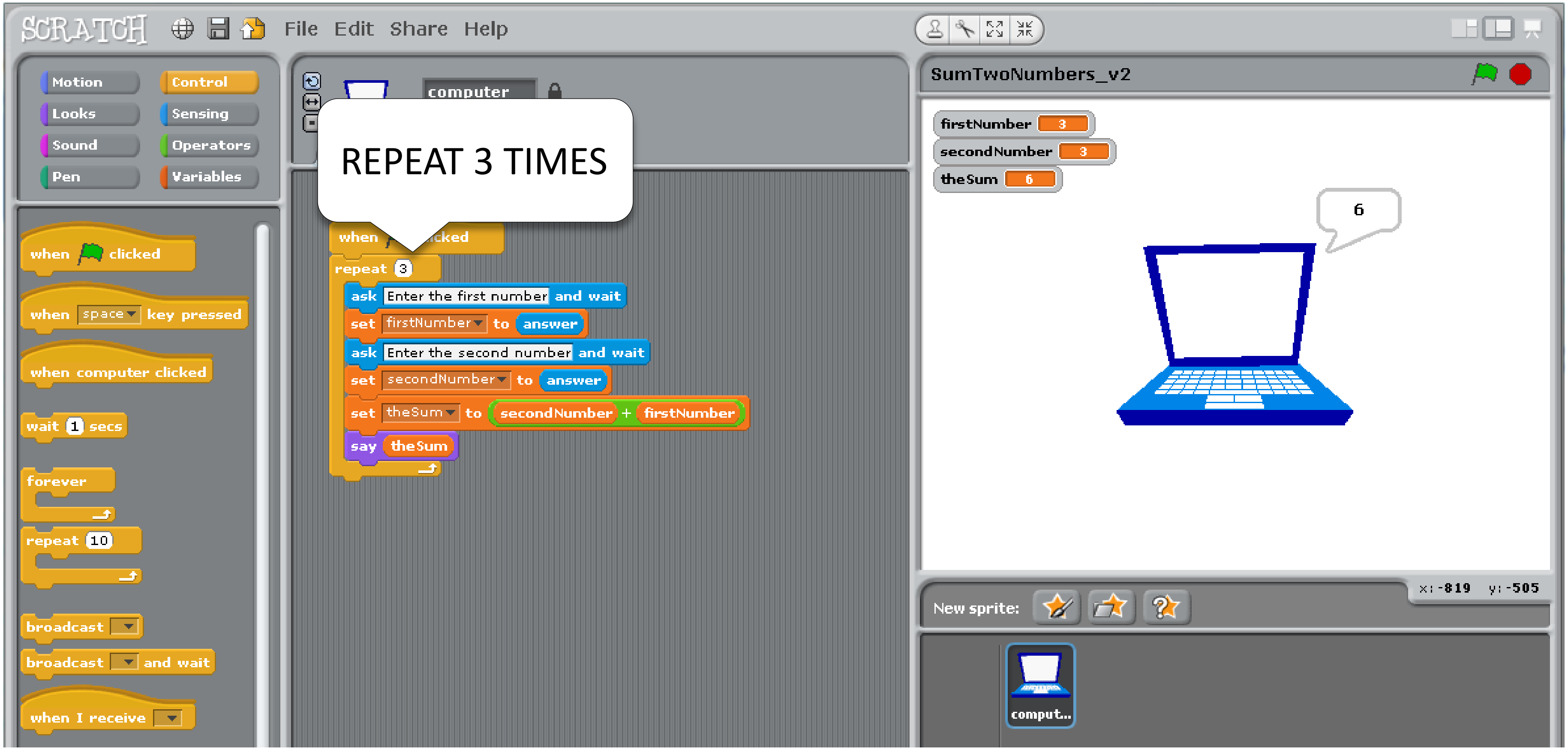
Keep walking the block



```
when [flag] clicked
set x to 0
set y to 0
set xMax to 200
set xMin to -200
set yMax to 100
set yMin to -100
repeat 10
    repeat until <(y position of Robot) < yMin>
        change y by -10
    repeat until <(x position of Robot) > xMax>
        change x by 10
    repeat until <(y position of Robot) > yMax>
        change y by 10
    repeat until <(x position of Robot) < xMin>
        change x by -10
    repeat until <(y position of Robot) < yMin>
        change y by -10
    repeat until <(x position of Robot) = 0>
        change x by 10
    repeat until <(y position of Robot) = 0>
        change y by 10
```

REPEAT 10 TIMES

If we use repeat with our sum two numbers



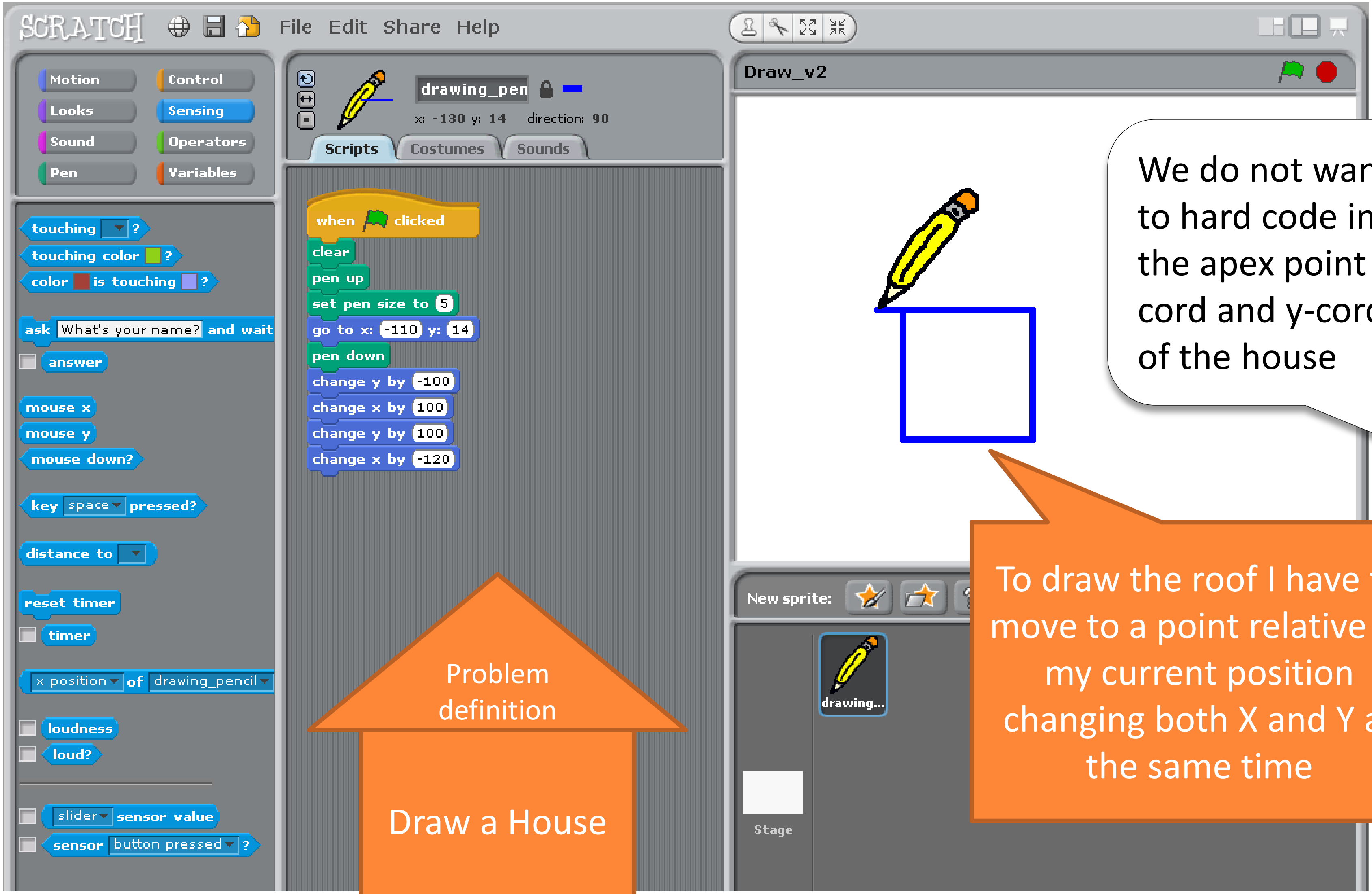
REPEAT 3 TIMES

```
when clicked
repeat 3
    ask Enter the first number and wait
    set firstNumber to answer
    ask Enter the second number and wait
    set secondNumber to answer
    set theSum to secondNumber + firstNumber
    say theSum
```

firstNumber 3
secondNumber 3
theSum 6

6

Want to write code that will work no matter where the pen is first placed on the screen



```
when [green flag] clicked
clear
pen up
set pen size to 5
go to x: -110 y: 14
pen down
change y by -100
change x by 100
change y by 100
change x by -120
```

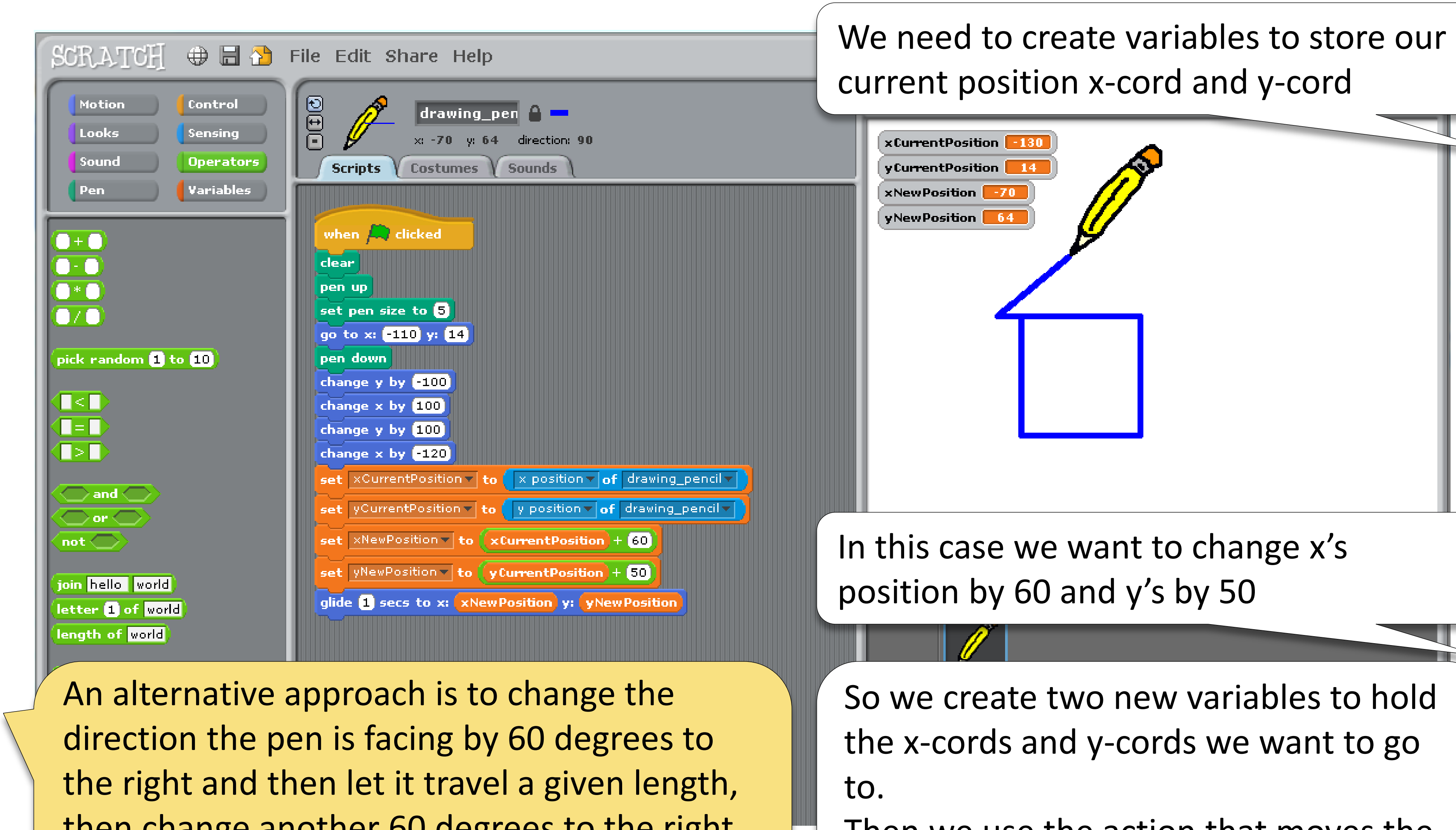
We do not want to hard code in the apex point x-cord and y-cord of the house

To draw the roof I have to move to a point relative to my current position changing both X and Y at the same time

Problem definition

Draw a House

Need to use variables – for relative positions



drawing_pen
x: -70 y: 64 direction: 90

```
when [flag] clicked
clear
pen up
set pen size to 5
go to x: -110 y: 14
pen down
change y by -100
change x by 100
change y by 100
change x by -120
set xCurrentPosition to (x position of drawing_pencil)
set yCurrentPosition to (y position of drawing_pencil)
set xNewPosition to (xCurrentPosition + 60)
set yNewPosition to (yCurrentPosition + 50)
glide 1 secs to x: xNewPosition y: yNewPosition
```

xCurrentPosition	-130
yCurrentPosition	14
xNewPosition	-70
yNewPosition	64

We need to create variables to store our current position x-cord and y-cord

In this case we want to change x's position by 60 and y's by 50

So we create two new variables to hold the x-cords and y-cords we want to go to.
Then we use the action that moves the pen to a given set of x-cords and y-cords

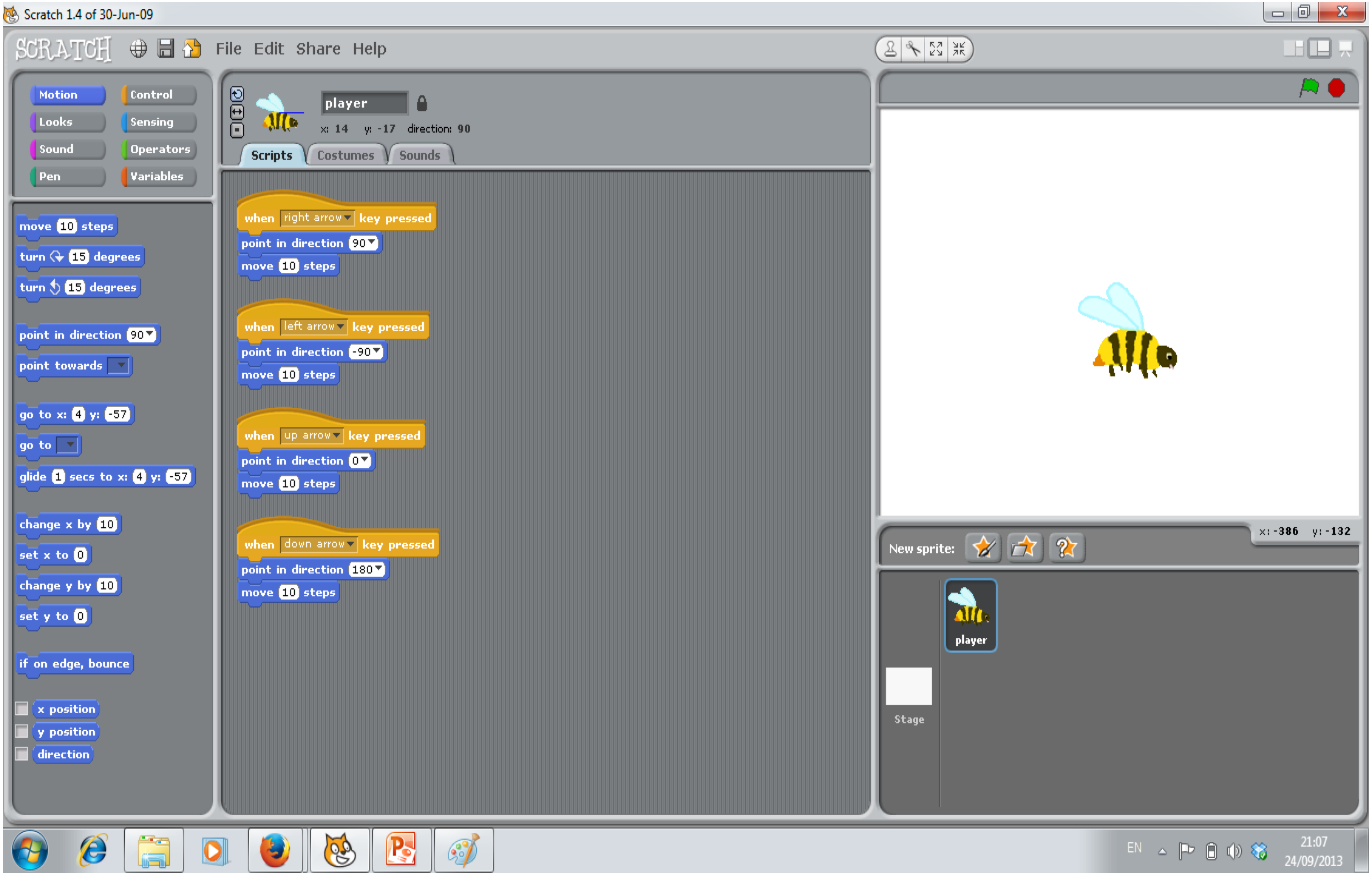
An alternative approach is to change the direction the pen is facing by 60 degrees to the right and then let it travel a given length, then change another 60 degrees to the right and travel the same distance, turn again 60 degrees and travel 20

Algorithms

- An algorithm is a well-developed, organized approach to solving a complex problem
- It is repeatable
  - no hard coding i.e. Of co-ordinates, or values
  - can work based on current position or user inputs
- The solution for walking the block could be turned into drawing a square around the edges of the stage
- Maths equations and ideas start becoming useful for doing things – co-ordinates, the sum of all angles on a triangle : ESPECIALLY FOR GAMES – 2D and 3D

Using SCRATCH to build a maze game

Step 1: Build our player

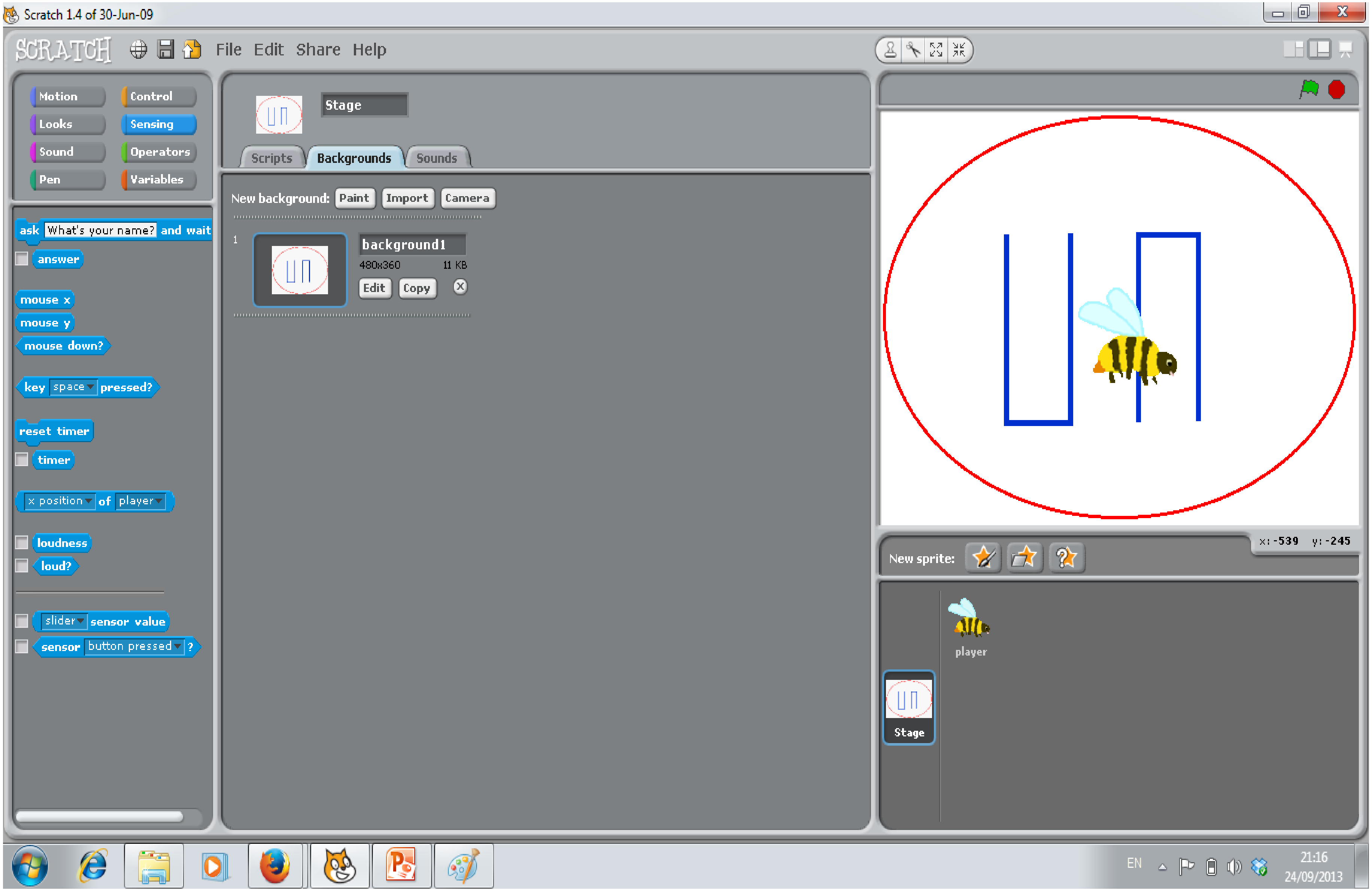


player

x: 14 y: -17 direction: 90

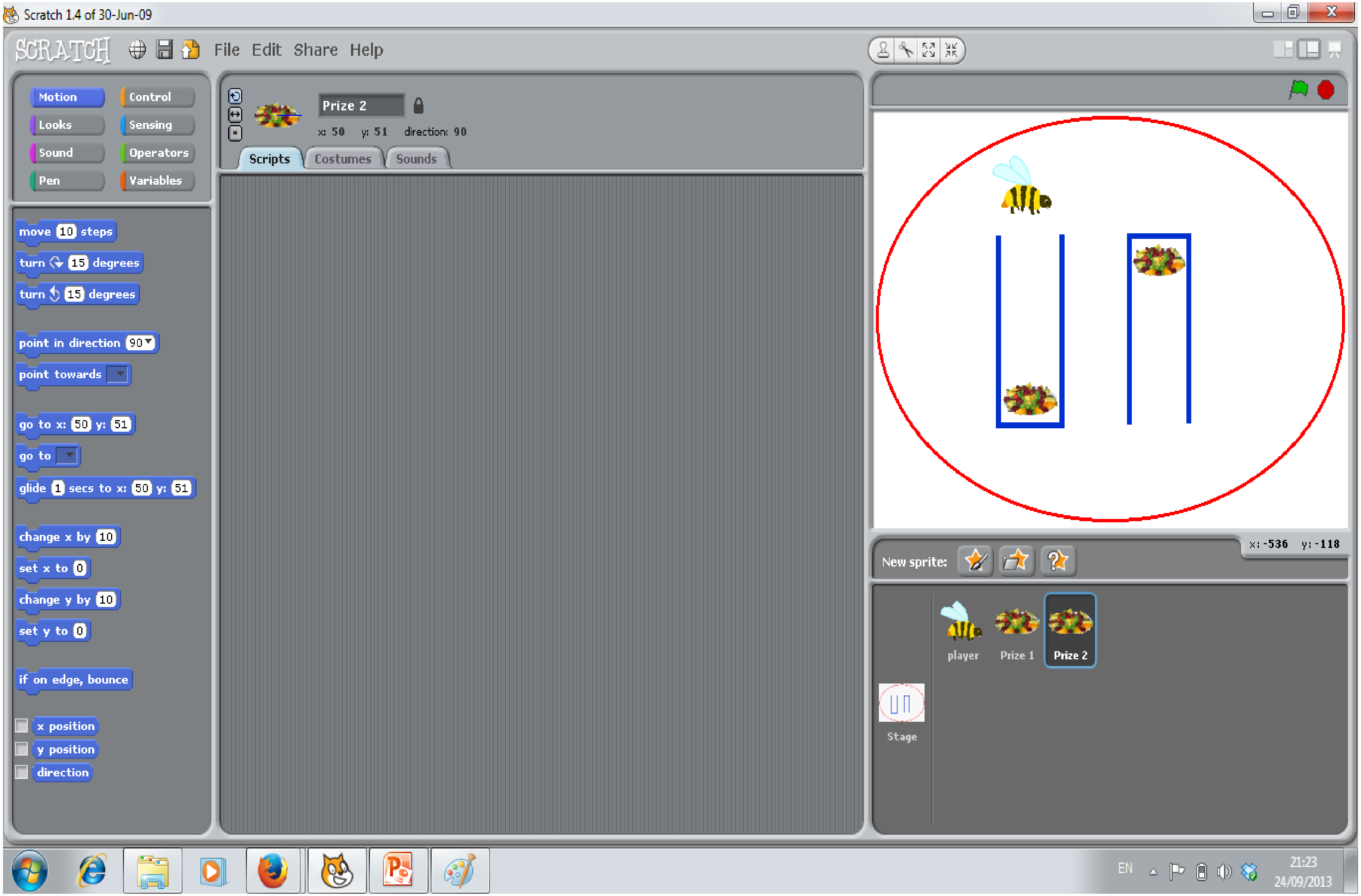
```
when [right arrow] key pressed
point in direction (90)
move (10) steps

when [left arrow] key pressed
point in direction (-90)
move (10) steps

when [up arrow] key pressed
point in direction (0)
move (10) steps

when [down arrow] key pressed
point in direction (180)
move (10) steps
```

Step 2: Draw the stage

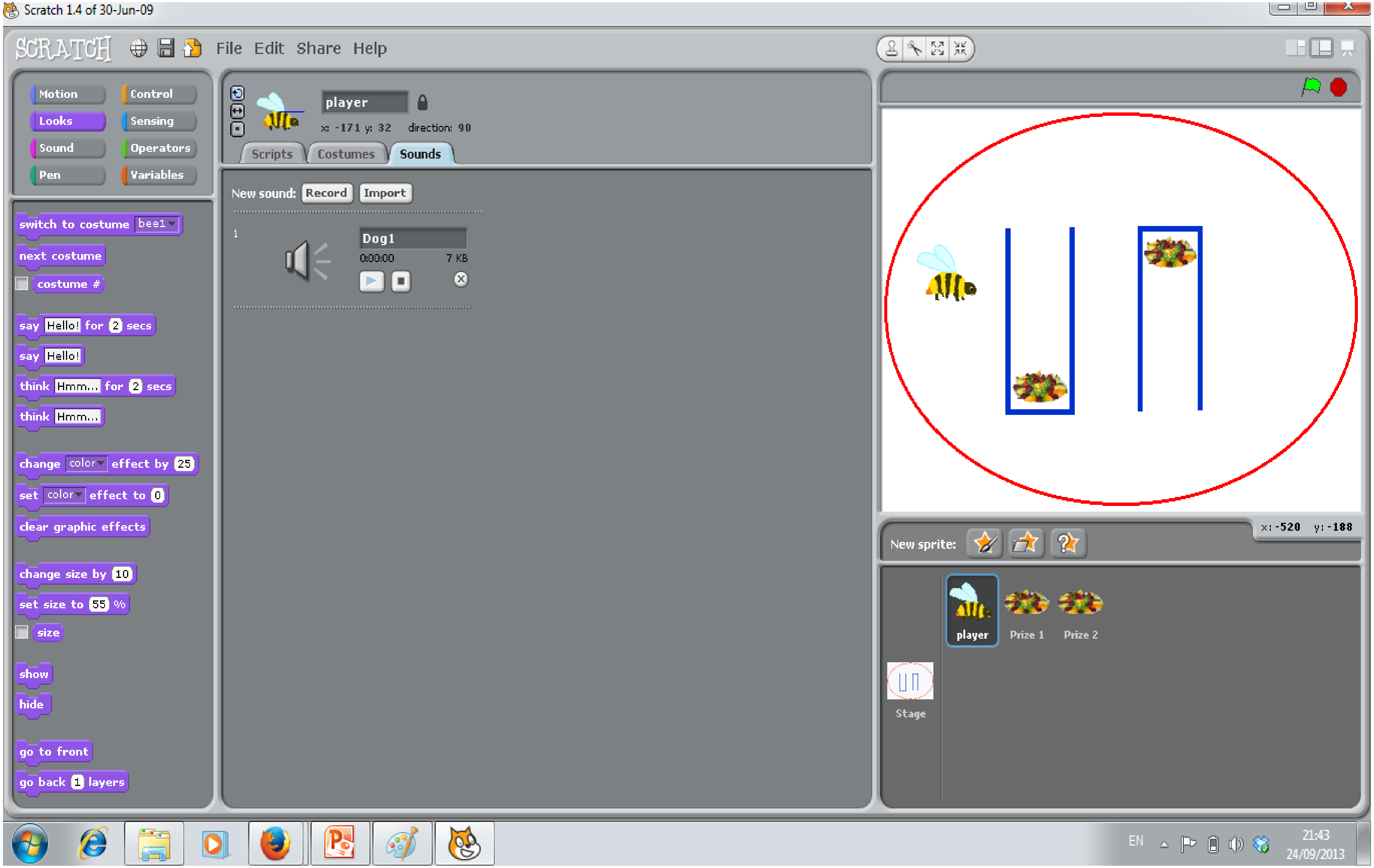


Scratch 1.4 of 30-Jun-09

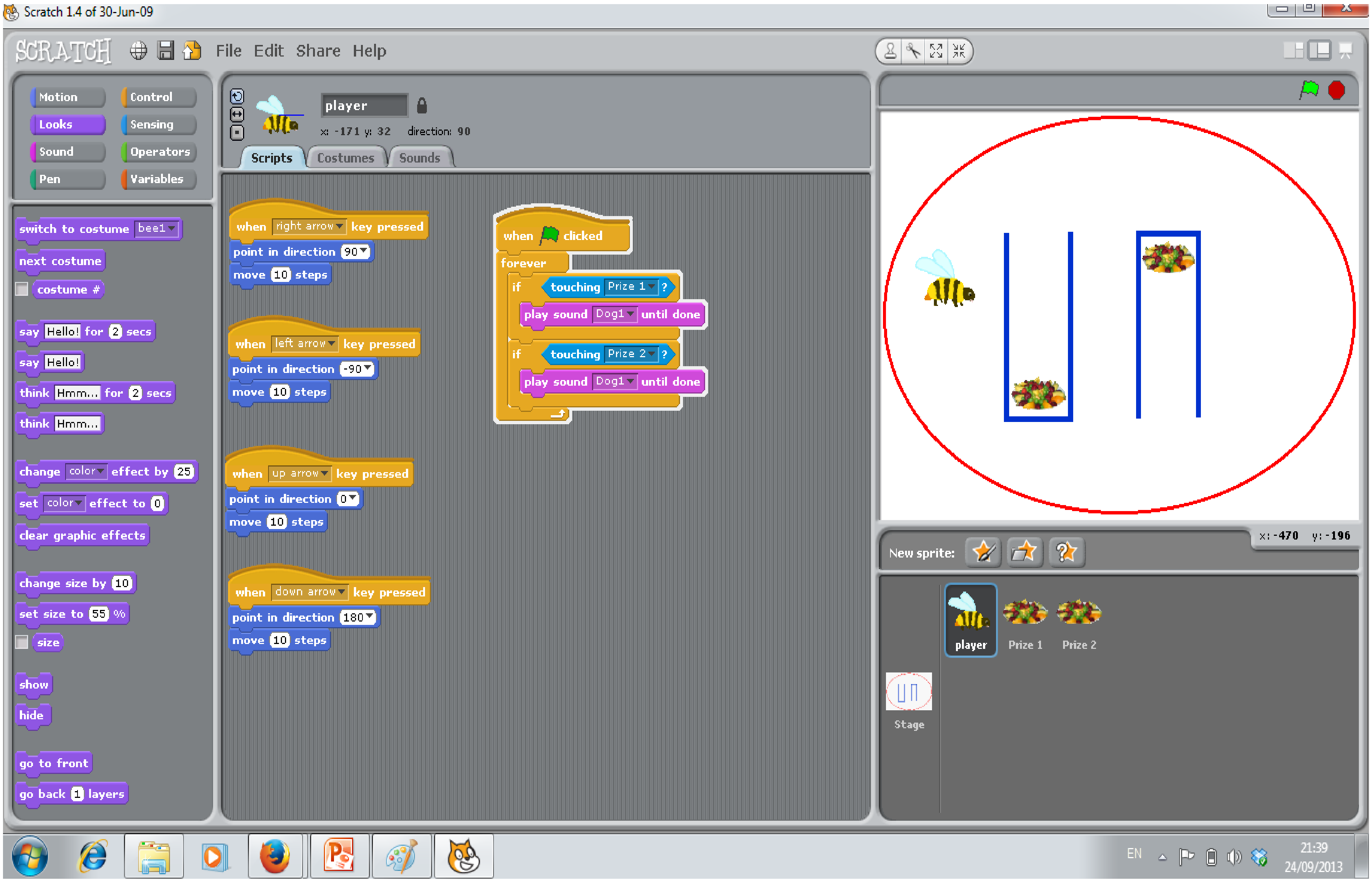
Step 3: Create some prizes



Step 4: Add sound to the player



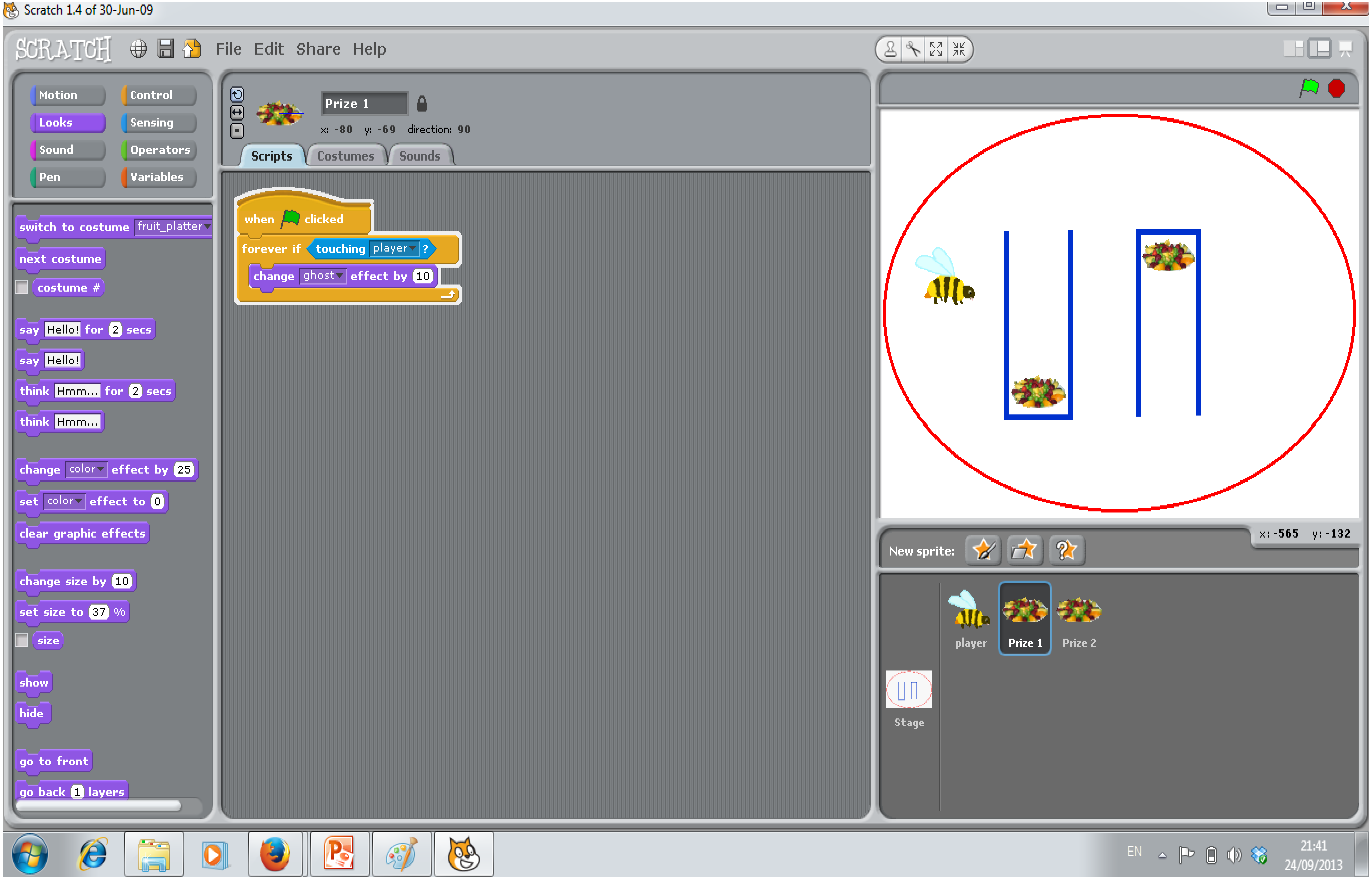
Step 5: Program the player



player
x: -171 y: 32 direction: 90

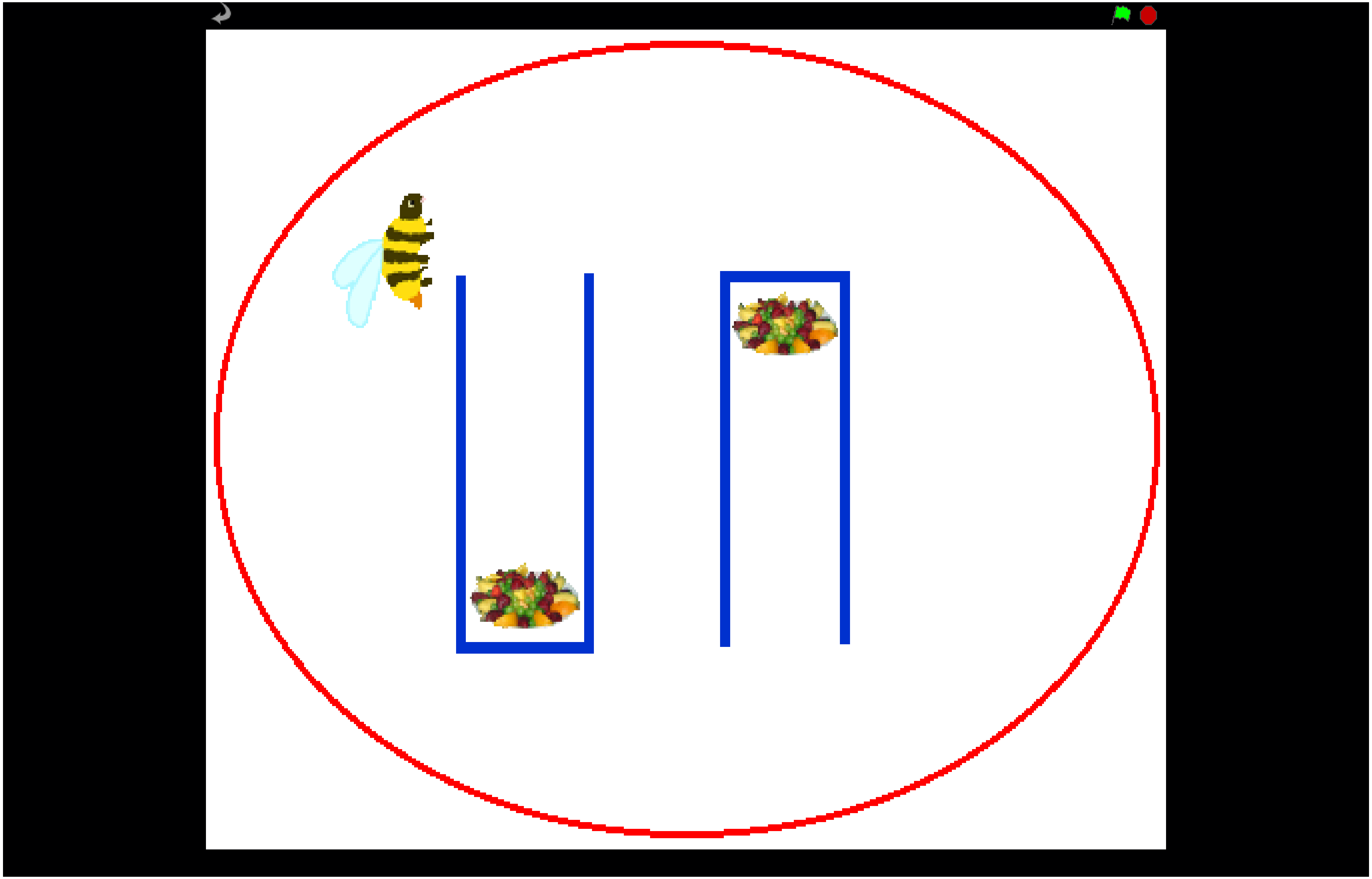
```
when [right arrow] key pressed
point in direction (90)
move (10) steps

when [left arrow] key pressed
point in direction (-90)
move (10) steps

when [up arrow] key pressed
point in direction (0)
move (10) steps

when [down arrow] key pressed
point in direction (180)
move (10) steps

when green flag clicked
forever
    if touching [Prize 1] ?
        play sound [Dog1] until done
    if touching [Prize 2] ?
        play sound [Dog1] until done
```

Step 6: Program the prizes



```
when [flag] clicked
forever if touching player ?
    change ghost effect by 10
```

Step 7: Game completed!



References

- 2009, Barry, Paul and Griffiths, David; Head First Programming, O’Reilly Media Inc.
- 2009, Pine,Chris ; Learn to Program, 2nd Edition, The Pragmatic Programmers