Computer Science Concepts in Scratch

(Supplement for Scratch 2.0)

Version 1.0

Michal Armoni and Moti Ben-Ari

Chapter 1

Introduction

This document is a supplement to *Computer Science Concepts in Scratch*. It explains the modifications to Scratch 2 relative to Scratch 1.4. Chapter 1 presents the principles that guided the development of the new version and an overview of the major changes. Chapter 2 explains how to work with the new user interface. The final three chapters discuss new instructions added to Scratch: Chapter 3 on abstraction and defining new blocks, Chapter 4 on cloning sprites and Chapter 5 on integrating images from a video camera into Scratch projects.

The need for a new version arose from a major shift in computing that has occurred in the past few years. Until recently, all personal computers were desktop computers permanently installed at school or home, or laptop computers that were still quite heavy and usually remained on a desk. Software was distributed on disks and installed on the computer; alternatively, a software package could be downloaded from a website, although it still had to be installed.

Now, the range of computing devices includes smartphones, tablets, netbooks and ultrabooks that are much more portable and lack removable disks. Furthermore, wireless communication with the internet through WIFI is available almost everywhere. Files are stored in the *cloud*, that is, on servers of companies like DROPBOX. Software itself need not be installed on your computer. The term *software as a service* is used for software like GOOGLE DRIVE that are accessed through your web browser. Finally, we no longer exchange files like pictures and music on disks; instead, social networks like FACEBOOK are used.

Scratch 2 is based on this new computing environment:

Run in a browser There is no need to install Scratch. Just point your browser to `scratch.mit.edu` and you can immediately start to develop projects.

Store in the cloud All your projects are stored on the Scratch server. You can start working on a project at school and then continue working at home. Projects are stored automatically so no effort is required to back them up or copy them to media like disks.

Sharing components of projects Since all data is stored in the cloud, it is easy to share individual costumes, sounds, scripts and sprites between projects.

Social network In Scratch 2 it is easier to share a project, as well as to comment on it, indicate that you love it, or remix it.

Scratch 2 implements some extensions to the Scratch programming language:

Abstraction by defining new blocks In Scratch 2 you can define a new instruction that is composed of a sequence of existing instructions. Defining new instructions is a significant addition to Scratch because it enables us to implement a central concept in computer science called ***abstraction***: defining a computation that can be used without knowing the details of its implementation.

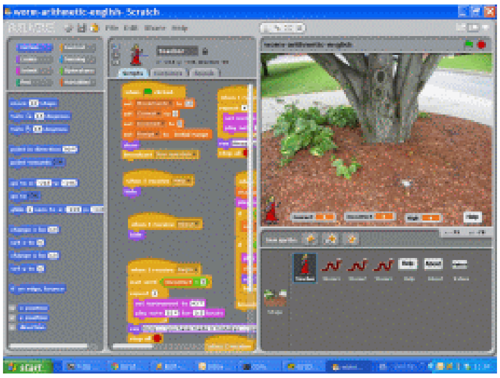
Cloning sprites Many projects such as games and simulations need multiple copies of a single sprite: flocks of birds, batteries of rockets to attack an alien, players on a soccer team, etc. In Scratch 2 you can construct a single sprite and then generate ***clones*** of the sprite. A clone appears as a separate image on the stage but all clones of a sprite share the same scripts.

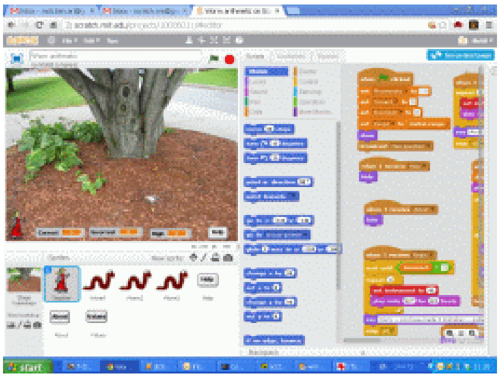
Integrating video Most computing devices include a built-in camera that you can use for creating video clips or for engaging in video chats using services such as SKYPE. In Scratch 2 you can project the output of the camera onto the stage. Sprites on the stage can sense and measure movement in the video image. This features can be used to interact with sprites without using the mouse or keyboard.

Chapter 2

The Scratch 2 User Interface

The pictures below show the Scratch 1.4 interface on the left and the Scratch 2 interface on the right:





There are really no significant differences. The stage and sprite area have moved from the right side of the screen to the left, while the block palettes and the script area have moved to the right. The main difference is that Scratch 1.4 was a separate application that appears in its own window that you can close, expand or resize, while Scratch 2 appears in a *tab* of the web browser. To close Scratch 2, click on the small x in the tab, not on the x in the browser.

Since most of the interface is similar, we will explain only the differences between the two interfaces. Furthermore, we will focus on creating and running projects in Scratch. The social networking aspects are similar to other such networks and present no problems.

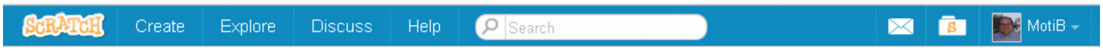
Opening an account

Since Scratch 2 runs in a web server, you have to open a Scratch account if you don't already have one. You will need to choose a user name and a password that must be entered each time that you connect to Scratch. (The browser will suggest that it remember your user name and password and you will have to decide if you want it do so!) Once you log in successfully, you will see the Scratch home page.

The Scratch website

The Scratch website remains at `scratch.mit.edu` but the design of the home page has been significantly changed. Much of the page is devoted to highlighting new and interesting projects and you are invited to explore them.

To create your own projects click Create in the toolbar at the top of the page:



The main Scratch screen (shown at the beginning of this chapter) will appear.

Once you start working with Scratch 2, your projects will be stored in the cloud (the Scratch servers). To continue working on a project, click on the suitcase icon near the right edge of the toolbar.

Instruction palettes

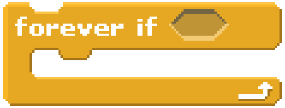
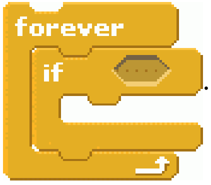
Three changes have been made to the instruction palettes:

- The name of the Variables palette has been changed to Data.
- The blocks in the Control palette have been divided into two palettes.

  There is a new palette Events that contains instructions that start a script like when space ▾ key pressed and when clicked, as well as broadcast ▾ and broadcast ▾ and wait.

  The Control palette retains the instructions such as forever and if that influence the sequence of instructions that is run.

- A new palette More Blocks is used for defining new instructions.

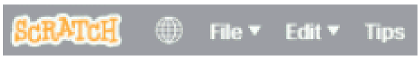
The instruction  no longer exists. It can be replaced by the two nested instructions .

```
forever if < >
```

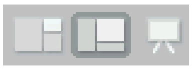
```
forever
    if < >
```

The upper toolbar

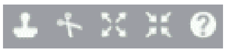
The toolbar at the top of the screen is divided into three areas:

- At the left is the menu . The globe is used to change the language.

  The File menu is described below.

  Edit contains the selection Undelete as well as a selection to display a small and larger script area. To display the stage on the entire screen, click the icon ⛶ just below the menu. In Scratch 1.4, changing the stage size was done by clicking the buttons  at the top right of the screen. Turbo mode makes scripts run faster.

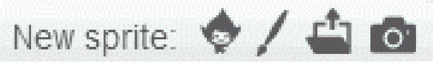
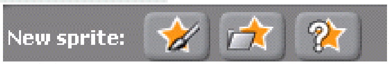
  The Tips menu displays a help frame at the right side of the screen.

- The tools area  contains the icons for stamping images of a sprite, deleting a sprite and making the sprite's image larger or smaller. The question icon displays help about a specific block.[1]

- The area at the right of the toolbar contains two icons. Clicking the suitcase brings up the My Stuff screen that displays the projects you have stored in the cloud. This screen is discussed below. The second icon is your user name; clicking on it displays a menu that enables you to change your personal data, as well as selections to display the My Stuff screen and to Sign out.

[1] As the time this document is written, this feature does not seem to work.

Toolbars for sprites and backdrops

The toolbar  above the sprite area in the lower left of the screen replaces the toolbar . From left to right the icons are: choose a sprite from the library, paint a sprite, choose a sprite from a file, import a a sprite from a camera. It is no longer possible to choose a random sprite.

The same icons for the backdrop appear to the left of the sprite area: .

Note: The terminology has changed: *backgrounds* are now called *backdrops*.

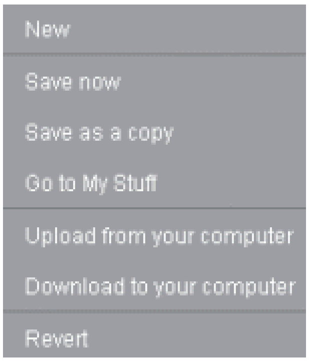
Editing pictures and sounds

The Paint Editor of Scratch 2 is similar to that of Scratch 1.4 although the placement of the icons and buttons has changed. You shouldn't have any difficulty working with the new interface. As mentioned in the introduction, images are now stored as vector graphics, but this is transparent to the user.

Scratch 2 introduces Sound Editor that enables you to cut and paste sections of a sound track and to apply effects to the sound. You should be able to master the editor by experimenting with it.

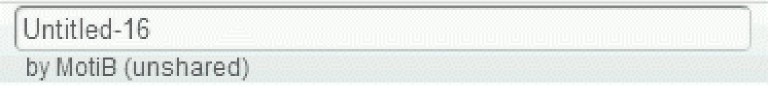
Saving projects in the cloud

The preferred way of saving projects is to upload them to the cloud—the server at the Scratch website. This is done automatically so you should have little need of for using the selections in the File menu.



There are seven selections. New is used to obtain a clean project that contains just the cat sprite and no scripts. Although saving a project is automatic, you may want to Save now, for example, just before terminating a session with Scratch. Save as a copy is useful if you want to make a significant change to a project but you still want to save the current one. It will save the project using the current name to which is added the string copy. Of course, you can change this to a more meaningful name. Finally, Revert throws away all changes since you first opened this project.

When you begin a new project, be sure to give it a meaningful name. Enter the name in the text field above the stage:

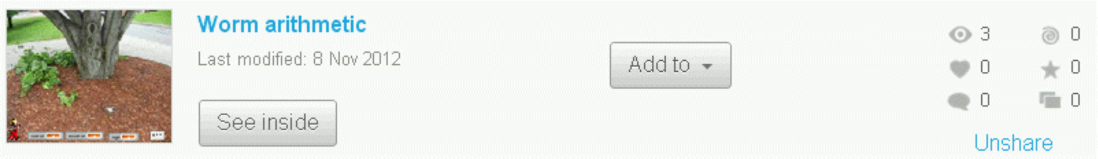


Untitled-16

by MotiB (unshared)

Sharing projects and opening an existing project

Your projects are stored in the cloud. To open one, choose the selection Go to My Stuff in the File menu, or simply click on the suitcase in the upper right corner of the screen. The My Stuff screen will be displayed; it contains an entry for each project:

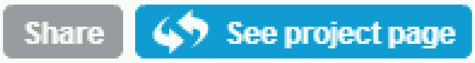


Worm arithmetic

Last modified: 8 Nov 2012

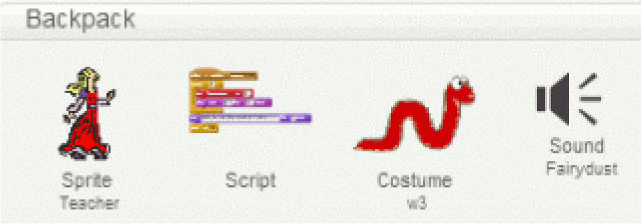
The entry contains: a small image of the stage, the name of the project, buttons See inside, Add to and Share or Unshare, and indicators of the number of times people have commented on, loved, remixed, etc. the project. Click See inside to start working on the project. Click Share to allow other people to see your project. If a project is shared, you can make it private again by clicking Unshare. If the project is not shared, a Delete button will appear.

On the My Stuff page you can define *Studios* to group projects together. The button Add to adds a project to a studio.

There are two buttons  in the upper right corner of the main Scratch screen. Share enables you to share the current project without going to the My Stuff screen. It opens the *project page* where you can add information about the project. The button See project page displays this page so that you can modify the information.

Sharing between projects

You can share sprites, scripts, costumes and sounds between your projects by copying them to the ***backpack***. Click the small upward-pointing triangle at the bottom right of the screen to show the backpack. Now, simply drag-and-drop the things you want to share to the backpack area:

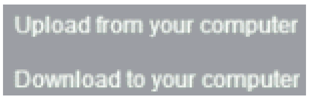


They will be stored in the cloud and you can access them from any project by dragging and dropping from the backpack to the appropriate area: a sprite to the sprite area, a costume to the costume area, and so on. If you need to delete something from the backpack, right-click on it and select Delete.

If you insist on files ...

Saving projects in the cloud is very convenient but there are reasons why you might want to save project in a file on your computer; for example, you have a project that you developed in Scratch 1.4 and you want to continue to work on it in Scratch 2, or you need to submit a project to your teacher without sharing it with everyone in the Scratch world.

There are selections in the File menu for *uploading* a project from a file on your computer to the cloud and for *downloading* a project to a file. Scratch 2 projects use the file extension sb2, but you can also upload files with extension sb that were created by Scratch 1.4.



Warning: When you open files, for example in a word processor, the file you were working on is closed and the new file is opened. However, when you upload a file in Scratch 2, it ***overwrites*** the project that is open. When the project is saved (automatically) you will loose whatever was in that project. Therefore, ***before*** you upload a project, select New from the File menu so that the uploaded project will overwrite a new empty project. Then you can give it a meaningful name and select Save now.

Chapter 3

Abstraction and New Blocks

Abstraction

How does the computer run an instruction like move 10 steps? The size of the stage is 480 units in the x-direction and 240 units in the y-direction. Suppose that the Scratch cat is placed at position $(100, 100)$ pointing in direction 0 (to the right). To run the instruction `move 10 steps`, the computer must: (a) erase the current image of the cat sprite on the stage, (b) calculate the new position of the sprite as $(110, 100)$ (adding 10 to the x-position 100), (c) draw the image of the cat centered at the new position $(110, 100)$.

If the cat is pointing in direction 90 (up), the computer must perform the same operations except that the new position is now $(100, 110)$. If the cat is pointing in direction 45 (northeast), the computer of the new position is more difficult: the new position of the cat is $(107, 107)$.

Aren't we lucky that the Scratch environment lets us use a single simple instruction move 10 steps and performs all the complicated calculations itself?! The Scratch environment ***hides*** the calculations needed to run the instruction and all we had to do was to supply one piece of information, the number of steps that the cat move move. This is an example of **abstraction**: we see a simple description of ***what*** a computation does, but the details of ***how*** the computation is done are hidden from us. Abstraction is the core concept of computer science because computers and their programs are extremely complex and we would not be able to work with them if we had to be aware of all the details all the time.

In fact, abstraction is central to scientific and engineering activity. Most people drive cars but know very little about what happens when they start the car and drive. It is quite simple to drive a car: you only have to master using the steering wheel, the gas and brake pedals, and the gear lever. The complex mechanical details of the engine (the cylinders and pistons), the transmission (gears and clutch), the brakes (disks and shoes) are all hidden from the driver (at least until the car needs to be serviced at a garage!).

Scratch 2 enables us to create new abstractions by defining new instructions that are composed from sequences of instructions (both those supplied by Scratch and those we define ourselves). You can consider a new instruction simply as a shorthand for a sequence of instructions, but you will obtain the greatest benefit from this feature if you can express a new instruction as an abstraction of a computation.

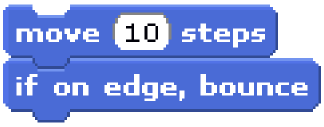
Definition and interface

Every abstraction has two parts: a ***definition*** and an ***interface***. The definition is the hidden part. Continuing our example from the beginning of the chapter, the definition of instruction `move 10 steps` consists of the three steps needed to erase the image of the sprite, calculate its new position and draw the image. The interface is the part of the abstraction that lets us use this computation. The word *interface* itself can be considered at something that enables two "faces"—the computation and the user of the computation—to interact.

New instructions for implementing abstractions

Let us now demonstrate these concepts by defining an abstraction.

In project meet-forever in Chapter 3, the two sprites run the following pair of instructions indefinitely:



```
move 10 steps
if on edge, bounce
```

Since we use this sequence of two instructions frequently, let us create an abstraction that hides the details of the computation and an interface that lets us use the computation (Figure 3.1).

At the top of the diagram is the new instruction represented by a new block called `move ten steps and bounce`. At the bottom of the diagram is the hidden implementation of the instruction consisting of two existing instructions. The dashed box is the interface that connects the two.

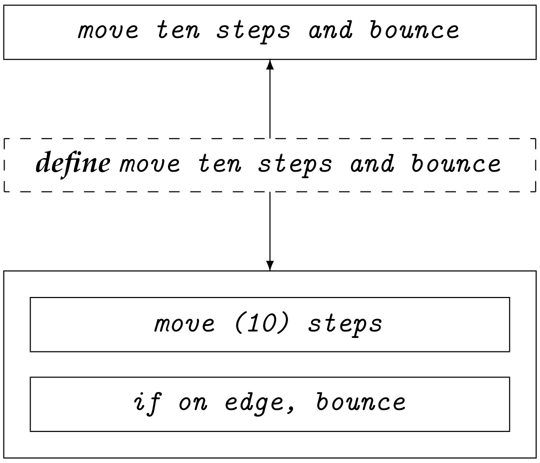
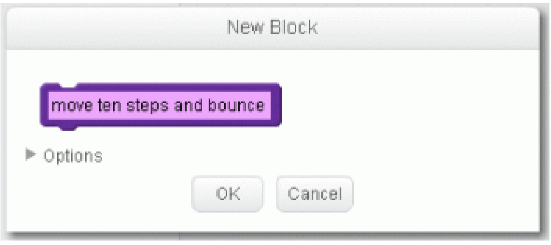


Figure 3.1: Components of an abstraction

Creating a new abstraction

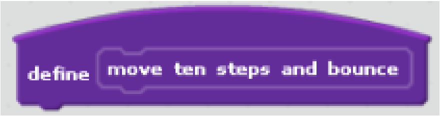
To create a new abstraction, we first define the interface and then construct the instructions that implement the computation:

- Click on More Blocks to display the palette for defining new instructions. Initially, it has only one button displayed: Make a Block. Click on this button to bring up the window used to define new blocks. The cursor is placed within the small purple window; enter the name of the new block and click OK:



move ten steps and bounce

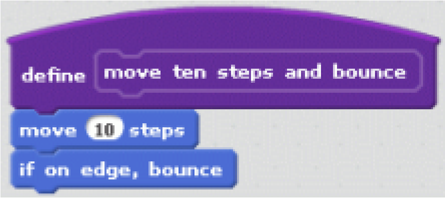
- Within this purple palette, a new block will appear with the name that you gave it move ten steps and bounce. This block can be used like any other block by dragging it and dropping it in a script.
- After creating the block, the first block in a script will appear in the script area:



define move ten steps and bounce

This is the interface block, which contains the name of the new instruction that we are about to define. The interface serves two purposes: it gives the name that will be used by other scripts and it indicates the beginning of the sequence of instructions that implement the new instruction. The curved top of the block means, as usual, that it has to be the first block in a script.

- Now we need to complete the definition of the new instruction by constructing the script as usual—dragging and dropping blocks to form a sequence of instructions:



```
define move ten steps and bounce
move 10 steps
if on edge, bounce
```

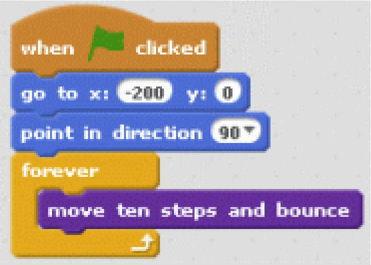
In the project meet-forever there were two sprites: a cat and a dog. For now, create the new abstraction separately in both sprites; later we will see how to copy the new instruction from from sprite to another.

The new instruction move ten steps and bounce defines the interface to an abstraction and means different things from two perspectives:

- From the perspective of the user, the interface defines an abstraction of a computation and gives all the information needed to use the abstraction.
- From the perspective of the implementor of the abstraction, the interface defines what the implementation is willing to expose to the user. This instruction exposed only the name of the abstraction, but later we will see that other information may be necessary.

Using a new instruction

Open the project meet-forever and in each sprite replace the sequence of two instructions `move 10 steps` and `if on edge, bounce` with the new instruction:



```
when [flag] clicked
go to x: -200 y: 0
point in direction 90
forever
    move ten steps and bounce
```

Run the project and check that the behavior of the sprites has not changed.

Exercise 1

> Create abstractions the following actions, use them in projects and check that their behaviors are correct:
>
> (a) *`move 10 steps and turn right`*
>
> (b) *`move 10 steps backwards and turn around`*
>
> (c) *`if you touch the dog, say ''Help'' for two seconds`*

General and specific instructions: Parameters

Most Scratch instructions are not as simple as move ten steps and bounce . Many instructions are general instructions like move ( ) steps that have to be changed into specific instructions like move (10) steps by including values in the instruction. We will now see how to create new general instructions.

An abstraction can contain one or more ***parameters***. A parameter is a way of communicating between the user of an abstraction and the implementation of the abstraction. It functions somewhat like a mail box. The user of the abstraction places a value in the parameter and this value is picked up and used in the implementation. In terms of the instruction, a parameter is then used to change a general instruction into a specific instruction.

Parameters are essential to creating useful abstractions because they enable one abstraction to be used in many contexts. Consider, for example, an abstraction for drawing a square. If we had to create an separate abstraction for many different sizes of squares, we would have to work too hard, because the instructions for drawing a square depend very simply on the length of the side of the square. Here is a description of an appropriate abstraction:

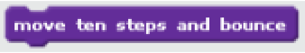
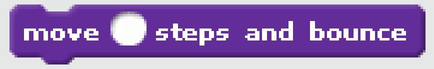
```
define draw a square whose side is N centimeters
    draw a horizontal line of N centimeters
    at one end of the horizontal line
        draw an upwards vertical line of N centimeters
    at the other end of the horizontal line
        draw an upwards vertical line of N centimeters
    join the tops of the two vertical lines with a horizontal line
```

From this example we see that defining an abstraction with parameters involves:

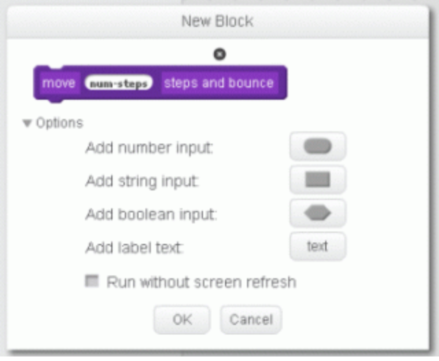
- Giving a name to the parameter in the interface of the abstraction.
- Using the parameter in the instructions that implement the abstraction.

The users of the abstraction still need to know very little: the name of the abstraction and the meaning of its parameter. The details of the implementation remain hidden. There are many other ways of implementing the abstraction, some of them probably better. (For example, draw a square as one continuous line: right, up, left, down.) We can change the implementation without telling the users of the abstraction about the change; they only need to know what the abstraction does, its name and the meanings of its parameters.

New instructions with parameters: the interface

We will change the simple instruction  move ten steps and bounce into a general instruction  move ○ steps and bounce that can be made into a specific instruction by giving a value for the parameter. Note the structure of the instruction: it consists of the word `move` followed by a small window for the value followed by the words `steps and bounce`. To construct this instruction, first click Make a Block and perform the following steps:

- After entering the word `move`, click Options. The window will now display additional fields that can be added to the block (Figure 3.2).
- Add the second field—the numerical parameter—by clicking on the icon to the right of Add a number input. The field will appear with rounded ends meaning that a numerical value must be entered in order to turn the general instruction into a specific instruction. There will be a name (`number1`) in the field and this is the name of the parameter that is used when implementing the abstraction. Of



New Block

move num-steps steps and bounce

▾ Options

Add number input

Add string input

Add boolean input

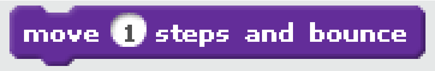
Add label text text

□ Run without screen refresh

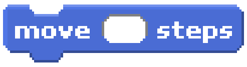
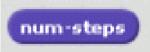
OK Cancel

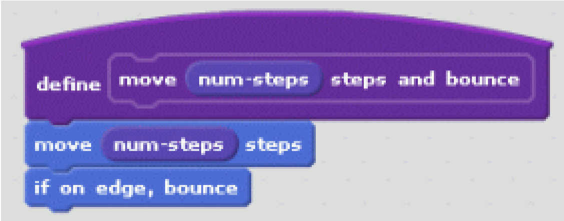
Figure 3.2: Defining a parameter

course you should change the name to a more meaningful one than `number1`; we have entered `num-steps` as the name of the parameter.

- The third field is a text field. Click the icon to the right of Add label text and enter the text *steps and bounce*.
- Click OK to save. The new block  move 1 steps and bounce will appear in the palette.

New instructions with parameters: the implementation

To implement the abstraction we need to add the two instructions *move (...) steps* and *if on edge, bounce* to the abstraction's script. The difference now is what happens with the general instruction  move steps. Previously, we entered the numerical value 10 into the window to make a specific instruction, but now the number of steps must be the value that is entered each time that the instruction is run. The parameter in the interface definition  num-steps is just like a variable reporter and we can drag and drop it into the window of the *move* block:



define move num-steps steps and bounce

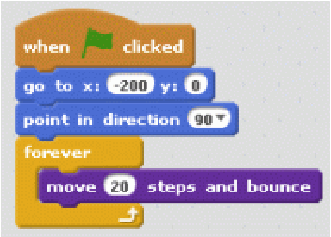
move num-steps steps

if on edge, bounce

The parameter functions as a mail box: whatever value was placed there by the script running the instruction is transfered to the script defining the instruction. Of course, what the script does with the value is hidden from the script that uses the instruction.

New instructions with parameters: using the instruction

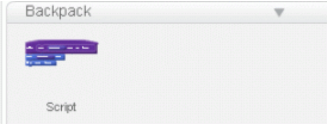
Using the instruction with a parameter is no different than using any similar general instruction. Drag it to a script and enter a numerical value (a number or a variable or an expression) in the small window:



```
when ⚑ clicked
go to x: -200 y: 0
point in direction 90▾
forever
    move 20 steps and bounce
```

Saving abstractions for reuse

The abstractions we created were for a specific sprite and the new instructions appear in the palette of that sprite only, but abstractions are usually useful in a different context than the one where it was created. To save a new instruction, drag and drop its script into the backpack area at the bottom right of the screen:



You may have to click the small triangle in order to see the backpack area.

To use an instruction that has been saved, simply drag its script from the backpack into the script area of another sprite. When you click on the button More Blocks the new instruction will appear in its palette.

Exercise 2

Create abstractions the following actions, use them in projects and check that their behaviors are correct:

(a) *`move (a number of) steps and turn right`*

(b) *`move (a number of) steps and turn (a number of degrees)`*

(c) *`turn right and glide to x: (x-position), y: (y-position)`*

(d) *`turn (a number of) degrees and glide to x: (x-position), y: (y-position)`*

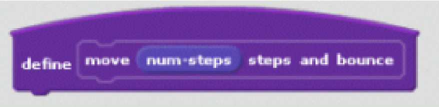
(e) *`glide around a square whose size is (length)`*

New concept: abstraction
An ***abstraction*** is a way to construct a computation that can be used even though its implementation is hidden.
An abstraction is composed of a ***definition*** and an ***interface***. The definition is the implementation of the abstraction, while the interface tells how to use the abstraction.
Parameters in the interface are used to pass information from the user of the abstraction to the implementation.

New construct in Scratch: new instructions

New instructions are created by clicking on Make a Block in the More Blocks palette. The new-block window appears and is used to define the interface of the instruction (its name and the names and types of its parameters). When this block is used, values for the parameters must be supplied to turn the general instruction into a specific instruction.

The definition of the new instruction is given in a script that starts with the define-block; for example:



```
define move (num-steps) steps and bounce
```

Parameters that appear in this block can be used in the script to read the values entered when the block is used.

Chapter 4

Clones

Sprites and clones

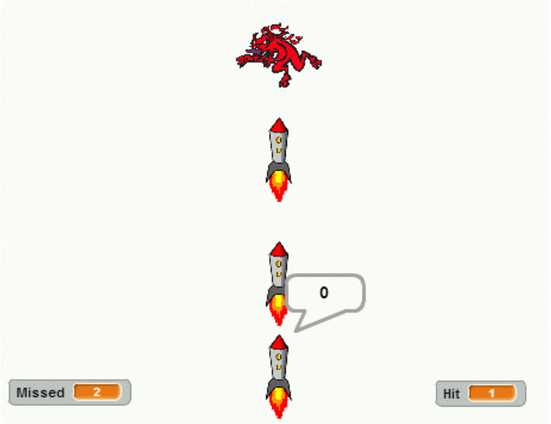
One of the limitations of Sprite 1.4 is that you need to create a separate sprite for each entity. For example, if you are creating a game with ten cars you need ten sprites. In Scratch 2, you can create *clones* of single sprite.

Example 1
Launching multiple rockets

(Based upon Exercise 12(d) in Chapter 10.)

A monster moves from left to right and back at the top of the stage. At the bottom of the stage is a launch pad from which rockets are launched when the space key is pressed. The direction of the rocket can be controlled by pressing the left and right arrow keys, but only until the rocket is launched. After the rocket is launched its direction cannot be changed. If a rocket hits the monster, the monster explodes but immediately returns to life and continues to move. Count the number of times that rockets hit the monster and the number of times that they miss.

Program file name: rocket-clone1



Designing a solution

The monster has two costumes, the scary one and the one where it explodes. Here is a description of its behavior:

```
when the green flag is clicked
    initialize the position, direction and costume
    forever
        move across the stage, changing direction at the edge
        if you are touching a rocket
            change to the exploding costume
            wait a while and then change back to the scary costume
```

Describing the behavior of a *single* rocket is not hard:

```
when the green flag is clicked
    initialize the position and direction
    forever
        when the space key is pressed
            move upwards until touching the monster or the edge of the stage
```

Later we will add counters to count the number of hits and misses, as well as scripts to turn the rocket when the arrow keys are pressed. Now, we have a more important problem to solve.

? How can we modify these descriptions to launch multiple rockets?

One solution is to create multiple sprites, one for each rocket. Aside from the extra work needed to copy the sprites, this solution has the limitation that we need to know *in advance* the maximum number of rockets. Instead, we will use clones.

New concept: clones
A ***clone*** is a *copy* of a sprite. (It is not a sprite itself.) Each clone appears as a separate image on the stage, but all clones run the *same scripts* that were defined for the sprite.

With the concept of clones, we add the following behavior to the rocket sprite:

```
when this clone is created
  forever
    if I am touching the edge of the stage
      add one to Hits
      erase me
    else if I am touching the monster
      add one to Misses
      erase me
    otherwise
      move upwards
```

? How are clones created?

This will be done when the space key is pressed:

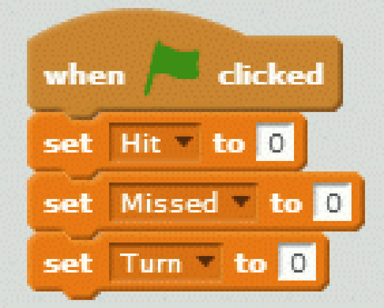
```
when the space key is pressed
  create a clone of the rocket sprite
```

For technical reasons to be explained later, this script will belong to the stage and not to the rocket sprite.

Let us now implement these behaviors in Scratch.

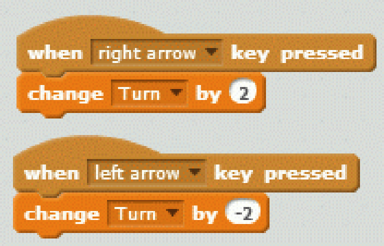
Implementing the project with clones

Initialize the variables in the stage because they are common to several sprites:



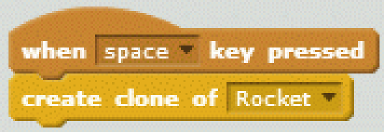
```
when [flag] clicked
set Hit to 0
set Missed to 0
set Turn to 0
```

Similarly, responding to key presses can be done in the stage:



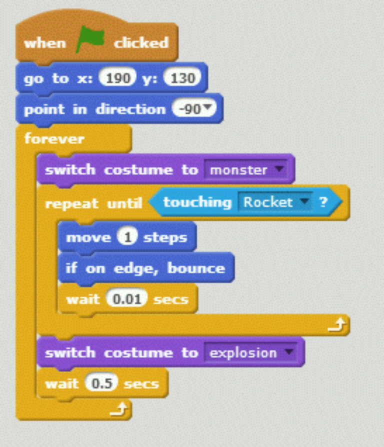
```
when right arrow key pressed
change Turn by 2

when left arrow key pressed
change Turn by -2
```

The new instruction to create a clone is run when the space key is pressed:



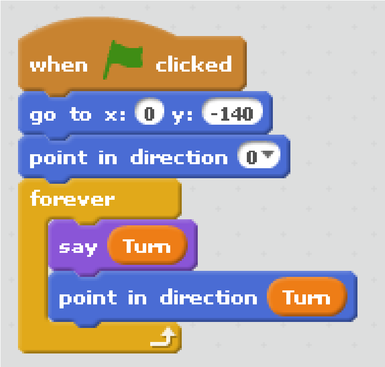
```
when space key pressed
create clone of Rocket
```

There is nothing new in the script for the monster:



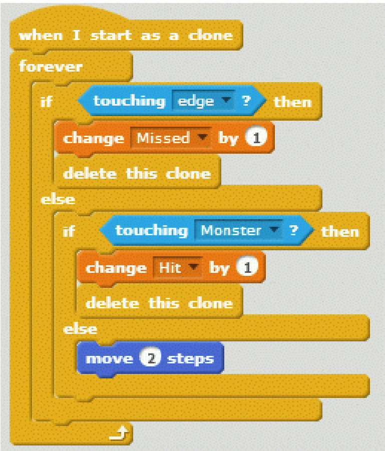
```
when [flag] clicked
go to x: 190 y: 130
point in direction -90
forever
  switch costume to monster
  repeat until touching Rocket ?
    move 1 steps
    if on edge, bounce
    wait 0.01 secs
  switch costume to explosion
  wait 0.5 secs
```

or the first script of the rocket:



```
when ⚑ clicked
go to x: 0 y: -140
point in direction 0▾
forever
    say Turn
    point in direction Turn
```

This script is run *once* when the green flag is clicked, not when each clone is created. There is a separate script that is run when a clone is created:



```
when I start as a clone
forever
    if touching edge▾ ? then
        change Missed▾ by 1
        delete this clone
    else
        if touching Monster▾ ? then
            change Hit▾ by 1
            delete this clone
        else
            move 2 steps
```

New construct in Scratch: clones

The instruction create clone of ▾ creates a clone of the sprite selected in the small window.

The instruction when I start as a clone appears at the beginning of a script and is run when a new clone of the sprite is created. The instruction delete this clone deletes the clone that runs the script in which it appears.

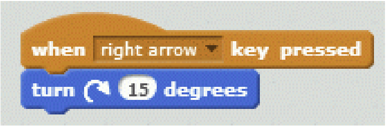
Note that the instruction delete this clone has a flat bottom so it must be the last instruction in a script. Since the instruction is run *by the clone*, this makes sense because the clone no longer exists after running the instructions. However, the *sprite* from which it was cloned continues to exist and to run its scripts.

Guidelines for programming with clones

- When a clone is created its image appears on the stage exactly at the same position as its sprite. Therefore, you won't be able to see it unless you move the clone.

  The first instructions after when I start as a clone should be movement instructions so that the clone will appear.

- All clones run all the scripts of their sprite. If the script:

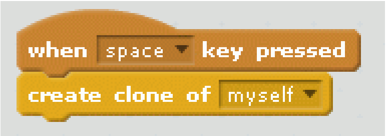
  

  ```
  when right arrow key pressed
  turn ↻ 15 degrees
  ```

  appeared as a script of the rocket, ***all the rocket clones*** would turn when the arrow key is pressed! That is why we placed the script that handles the key events in the stage and used a variable to communicate with the sprite.

- The components of a sprite—its scripts, costumes, sounds, local variables—are common to all the clones. Therefore, you cannot define different behaviors for different clones.

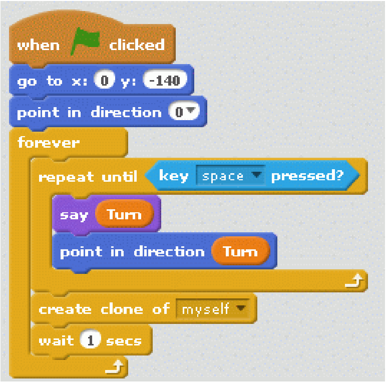
- There is an instruction create clone of myself which creates a clone of a sprite from within a script of the sprite. However, it is somewhat dangerous to use; for example, if the following script:

  ```
  when space key pressed
  create clone of myself
  ```

  were included in the *rocket sprite*, pressing the space key would cause each clone to create an additional clone of itself!

  You can safely use the instruction create clone of myself within the script that is run when the green flag is clicked, because that script will only be run once at the beginning of the program. Here is another version of the project that uses this instruction:



```
when 🏴 clicked
go to x: 0 y: -140
point in direction 0
forever
    repeat until key space pressed?
        say Turn
        point in direction Turn
    create clone of myself
    wait 1 secs
```

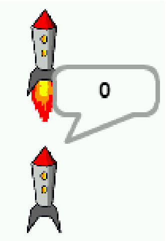
Program file name: rocket-clone2

Exercise 1

(Challenging!) Explain what happens if the instruction `wait (1) secs` is left out of the script.

Exercise 2

Modify the project so that the rocket has two costumes: one with flames coming out of the rear nozzle and one without the flames. Initially, the rocket sprite is displayed without flames, while the clones that are launched are displayed with flames. Additionally, the rocket on the launch pad is not displayed until the clone that is launched has cleared the pad so that its display does not cover that of the sprite.



Program file name: rocket-clone3

Chapter 5

Integrating Video into Projects

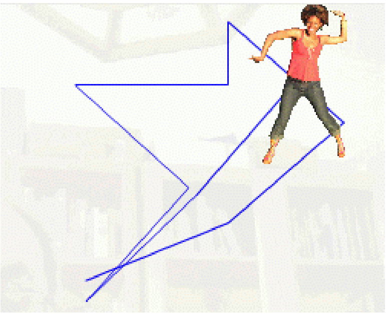
Digital devices such as smartphones use touch screens and motion sensors. Scratch 2 can detect motion on the computer's camera and use it to control an animation. The support is rather basic: the video image is displayed on the backdrop and a sprite can sense if motion exists in the image and the direction of the motion.

> You must experiment with the commands because every camera will return different values for the motion sensor. The values may also depend on the lighting in the room and even on the clothes you are wearing.

Example 1
Wave to dance

(Based on Example 3 in Chapter 11.)

Define dancing steps by waving your hand in front of the camera. After N waves, the dancer will move in the *directions* of each wave.



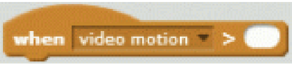
Program file name: choreography

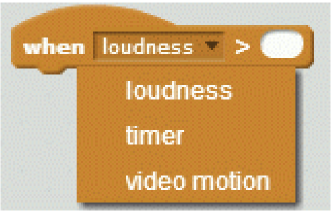
Designing a solution

In the solution of Example 3 in Chapter 11 we used two lists for each dancer, one for the x-positions and one for the y-positions. Here we only need one list for the dancer's *directions*. In that example, data was stored in the lists in response to the event of clicking a mouse button; here, the event will be sensing motion.

```
when motion in front of the camera is sensed
    add the direction of the motion to the list
    if the number of directions saved is N
        notify the sprite to start dancing

when notified to start dancing
    for each number I from 1 to N
        turn to the I'th direction in the list
        move forward
```

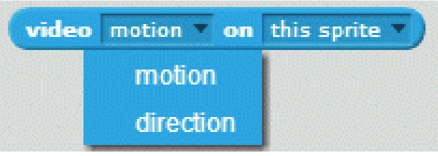
Implementing the solution in Scratch 2

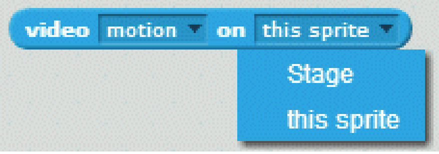
Scratch can sense an *amount* of motion on the video. It doesn't really matter what this amount actually is, only that the faster the movement, the higher the value that is sensed. When the amount of motion is large enough it can cause a script to begin running. The script begins with the instruction , where you have to decide what "large enough" is by entering a number in the small window. This instruction appears as one selection of the following event block:



The steps of the solution are as follows:

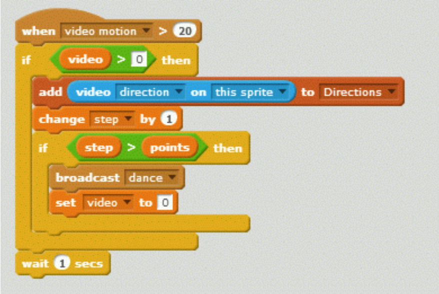
- When motion is detected the script stores the direction in the list `Directions`. A reporter is used to obtain the direction of the motion over the sprite.





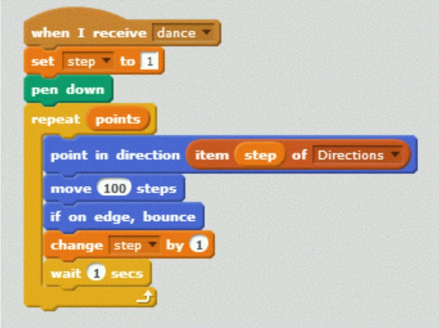
There are also selections for reporting the amount of motion over the sprite, and the direction or amount of motion on the stage.

- When N (we choose 8) directions have been stored the message `dance` is sent:



```
when video motion > 20
if video > 0 then
    add video direction on this sprite to Directions
    change step by 1
    if step > points then
        broadcast dance
        set video to 0
wait 1 secs
```

- The script run when the message is received is straightforward:



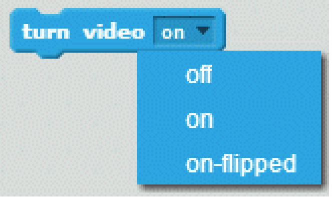
```
when I receive dance
set step to 1
pen down
repeat points
    point in direction item step of Directions
    move 100 steps
    if on edge, bounce
    change step by 1
    wait 1 secs
```

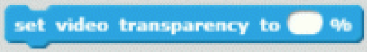
- An additional variable `video` is used to ensure that we don't measure and store the video data until initialization has been terminated, and, furthermore, that we stop storing data when N directions have been saved. The variable is initialized to 0 and set to 1 only when we are ready to start measuring.

Controlling the video camera

There are two instructions for controlling the video camera:

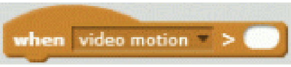
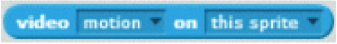
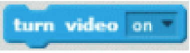
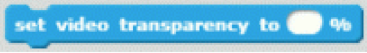
- You can turn the camera on or off (or flip its state to the other state):



- You can change the *transparency*  of the video image. The higher the transparency, the fainter the image appears on the stage, so it interferes less with the image of the sprites. If it is important to see the video image, use a lower transparency.

New concept: video
A ***video image*** can be displayed on the screen.
The ***amount and direction of motion*** can be measured.

New construct in Scratch: video

The instruction  starts a script when the amount of motion is larger than the value in the window.
The reporter  gives the amount or direction of motion over the sprite or the backdrop.
The video camera and be turned on and off .
The transparency of the image can be set .