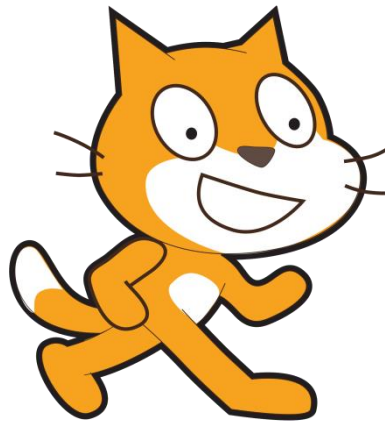


Starting from

SCRATCH



2013 edited version

An Introduction to Computing Science

by Jeremy Scott

LEARNER NOTES

3: A Mazing Game

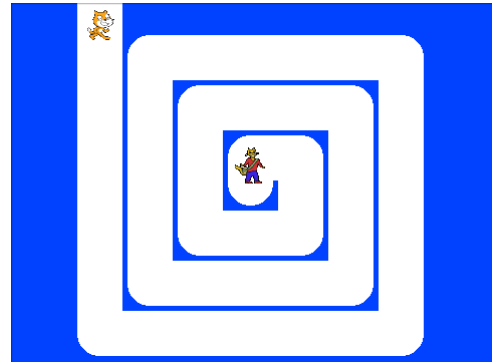
This lesson will cover

- Game creation
- Collision detection

Introduction

You are going to create a simple game where the player guides an “explorer” character around a maze using the arrow keys.

The game will end when the explorer rescues its friend in the middle.



Introduction

Watch screencast **Maze** to learn how to create the Maze game.

Task 1: Setting the scene

Set up the game by importing the stage costume (Maze) and two sprites – an explorer and a friend for the explorer to rescue. **Don't do any more at this point.**



The Importance of Design

Before we make anything – a house, a dress or a computer program – we should start with a **design**. Because there are two important parts to most programs – the **interface** (how it looks) and the **code** – we design these separately.

- The easiest way to design the **interface** is by sketching it out on paper.
- To design the **code**, write out a list of steps it will have to perform **in English**. This is known as an **algorithm** and is just like the steps in a food recipe.

Solving problems like this is what programming is *really* about, rather than entering commands on the computer.

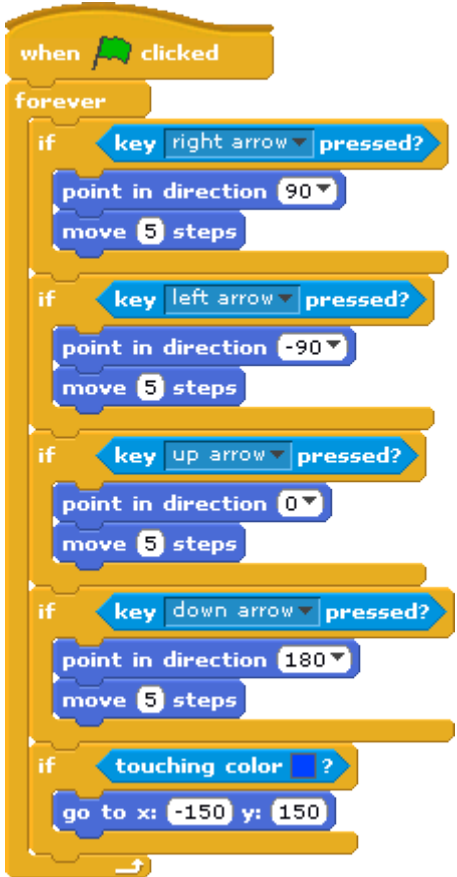
All good programmers design algorithms before starting to code!

Task 2: Designing the solution

Let's look again at the two main things we need to code in our game:

1. moving the explorer
2. reaching centre of the maze (and rescuing the explorer's friend)

The table below shows an **algorithm** for moving the explorer and Scratch **code** that does the same thing.

Algorithm for moving explorer	Code
<p>when the flag is clicked</p> <p>repeat forever</p> <p style="padding-left: 40px;">if right arrow key is pressed</p> <p style="padding-left: 80px;">point right</p> <p style="padding-left: 80px;">move 5 steps</p> <p style="padding-left: 40px;">if left arrow key is pressed</p> <p style="padding-left: 80px;">point left</p> <p style="padding-left: 80px;">move 5 steps</p> <p style="padding-left: 40px;">if up arrow key is pressed</p> <p style="padding-left: 80px;">point up</p> <p style="padding-left: 80px;">move 5 steps</p> <p style="padding-left: 40px;">if down arrow key is pressed</p> <p style="padding-left: 80px;">point down</p> <p style="padding-left: 80px;">move 5 steps</p> <p style="padding-left: 40px;">if explorer touches the same colour as the maze wall</p> <p style="padding-left: 80px;">go back to starting position</p>	

Algorithms let programmers concentrate on what the program has to do instead of how to do it on the computer. Once the algorithm is worked out, writing the code is easy!



Notice how an algorithm is **indented** to show which parts belong **inside** other parts e.g.

repeat forever

- if right arrow key is pressed goes inside **repeat forever**
 - point right goes inside **if right arrow key is pressed**
 - move 5 steps goes inside **if right arrow key is pressed**

Task 2: Designing the solution (continued)

The table below shows an algorithm for the explorer's friend sprite.

From this algorithm, see if you can create the code yourself. **Remember to put it in the friend sprite!**

Algorithm for reaching centre of maze	Code for friend sprite
when the flag is clicked show sprite repeat forever if touching explorer sprite say "Thank you!" hide sprite stop all scripts	Code this one yourself!

Now test your game to see if it works.

Extension 1: Getting in tune

Add a background tune to your game (sound "xylo1" seems to suit, but choose what you think sounds best).



Think about the following:

- Where would be the best place to store this, since it applies to the whole game?
- How will you get the music to keep playing?
- Should you use a **play sound** or **play sound until done** block to play the music?

Extension 2: Add an enemy



Add a sprite that constantly moves back and forth across the stage. If your explorer touches the enemy, the explorer should go back to the start.

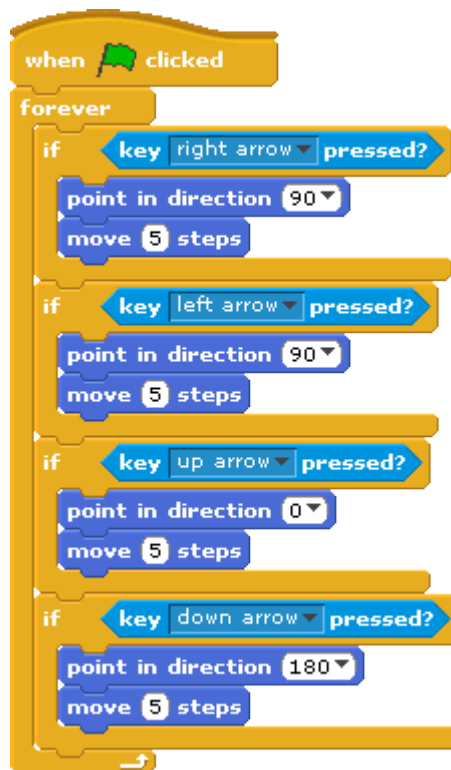
Hint: set your enemy sprite to move only left & right.

The **if on edge, bounce** block is useful to bounce back and forth off the edge of the stage.



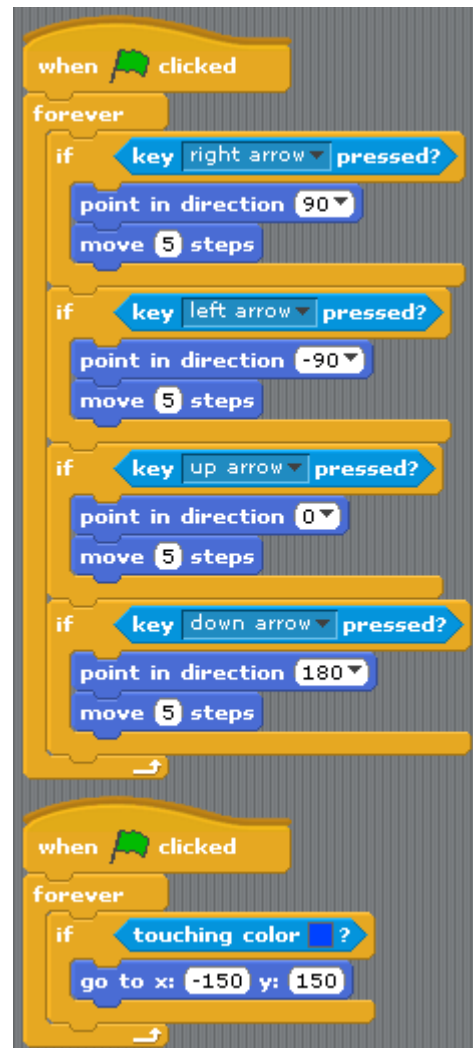
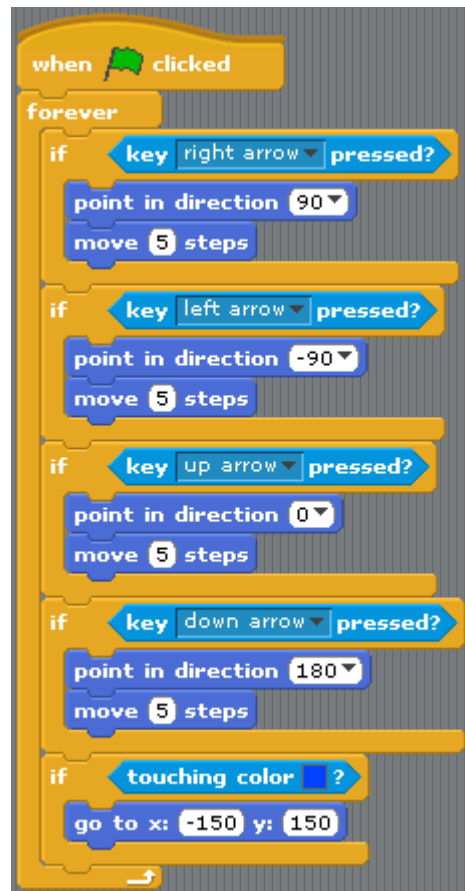
Did you understand?

- 3.1 A programmer creates a maze game like the one you've just created. Unfortunately, her character doesn't move as expected.



What mistake has she made?

3.2 Look at the examples of code below.



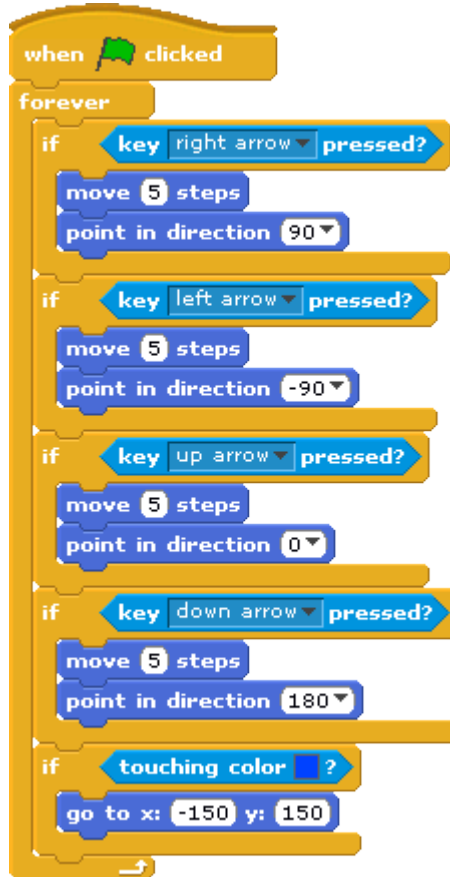
Do they perform the same task? _____

Explain your answer _____



- 3.3 The code below controls a sprite going round a maze. If the sprite touches the side of the maze (the colour blue), it returns to its starting position of -150, 150.

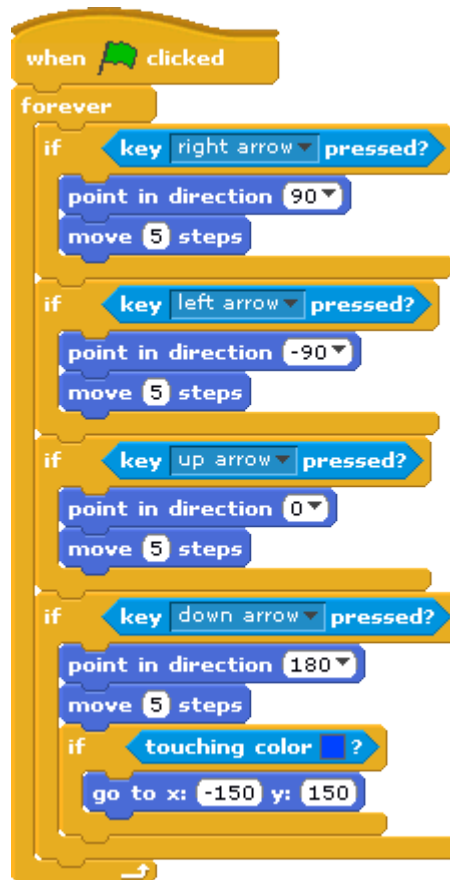
Unfortunately, the sprite sometimes touches the walls of the maze and returns to the start when the player doesn't expect.



What mistake has the programmer made?



- 3.4 In this example, the sprite is supposed to return to the centre of the maze when it touches the sides (coloured blue); however, it only does this sometimes.



What mistake has the programmer made?



- 3.5 In this example, the sprite **never** returns to starting position, even if it touches the walls of the maze (coloured blue).

```

when green flag clicked
  forever loop
    if key right arrow pressed?
      point in direction 90
      move 5 steps
    if key left arrow pressed?
      point in direction -90
      move 5 steps
    if key up arrow pressed?
      point in direction 0
      move 5 steps
    if key down arrow pressed?
      point in direction 180
      move 5 steps
  
```

```

when green flag clicked
  if touching color blue?
    go to x: -150 y: 150
  
```

What mistake has the programmer made?

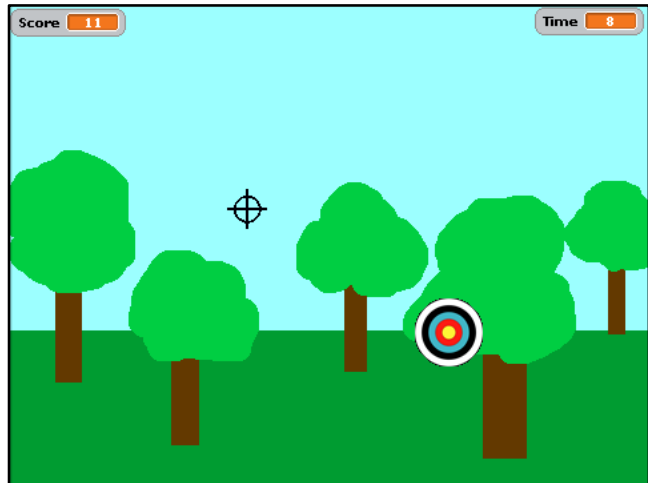
[illegible]

- 3.6 Now make up a buggy question of your own and pass it to your neighbour.

5: Forest Archery Game

This lesson will cover

- Decision statements
- conditional loops
- variables
- random numbers
- animation
- sound



Introduction

Watch screencast **ForestArchery** to see how to create this game.

Task 1: Designing the solution

Let's look again at the two main things we need to code in our game:

1. moving the target
2. shooting the target

Try to code your program from the algorithms given overleaf, rather than looking at the screencast again.

Algorithm to move target (in Target sprite)

when flag is clicked

repeat forever

glide in 1 second to a random position*

* *x is a random number from -240 to 240*

y is a random number from -180 to 180

Algorithm to move sight and shoot (in Sight sprite) /...

Algorithm to move sight and shoot (in Sight sprite)

```
when flag is clicked
repeat forever
  go to mouse location (the mouse x and mouse y positions)
  if the mouse button is down (the user has clicked the mouse)
    if the sprite is touching the target sprite
      add 1 to score variable
      play Pop! sound
      Say "Hit!" for 0.5 seconds
```

Task 2: Hit and miss

Change your code to make the program count **misses** as well as **hits (taking off 1 point from the score)**:

```
If touching target
  change score by 1
  play Pop! sound
  Say "Hit!" for 0.5 seconds
else
  change score by -1
  play sound
  say "Miss!" for 0.5 seconds
```

Task 3: Against the clock

Add a timer **variable** to your program which makes the game last 30 seconds. Make the variable appear on the screen as it counts down from 30 to 0.

```
when flag is clicked
repeat 30 times
  wait 1 second
  change time by -1
stop all scripts
```

Task 4: Bullseye!

Using **if** and **touching colour** blocks, change the program so that when the target is hit, it adds the following to the score:

- White – 1 point
- Black – 2 points
- Blue – 3 points
- Red – 4 points
- Gold – 5 points (and says “Bullseye!”)

Task 5: Stay positive!

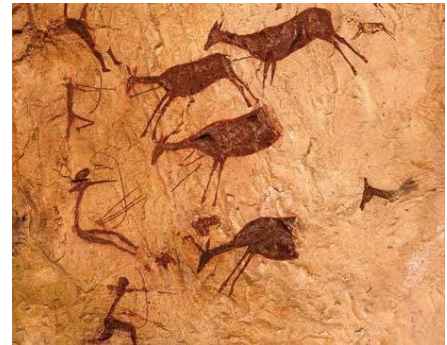
Adapt the program so that the user will **never** get a negative score.

Hint: take off a point only if the score is above zero.

Did you know...? Humans are known to have practised archery for at least 10,000 years. It was first used for hunting (see cave painting opposite¹), then in warfare.

In medieval England, it was compulsory for all men to practise archery regularly, so they would be skilled if required to go to war.

Nowadays, archery is a popular leisure activity enjoyed by people all around the world.



¹© Instituto de Turismo de España (TURESPAÑA). Image of cave painting from Cova dels Cavalls remains the exclusive property of Turespaña and cannot be used or reproduced without Turespaña's prior written consent.



Variables

In this game, we introduced the idea of keeping a score using a variable block.

A **variable** is a space in a computer's memory where we can hold information used by our program – just like storing things in a box.

We should always give a variable a sensible **name** that tells us **what kind of information is stored in it** – just like putting a label on the box to tell us what's inside.

To create a variable in Scratch, we **make a variable** block.

Once a variable is created, the information stored inside it can be **set** or **changed** (that is, varied – hence the word “variable”).



Extension 1: A Mazing cool feature

We're now going to add a new feature to your **Maze** game from lesson 3 – a **timer** that gives the user 30 seconds to finish the game.

To do this, add a variable called **time** and create a new script that does the following:

When green flag is clicked

set variable (time) to 30

repeat until time = 0

 wait 1 second

 subtract 1 from variable (time)

say "You Lose"

stop all scripts

Before you write this script, think about where might be the best place to put it.

Hint: is it something that applies to a single sprite or the whole game?

Extension 2: A Harder Maze

Now create a maze of your own which has more than one route to the middle.

Hint: Just create a new stage background for this.

Extension 3: Do I get a prize?

Create new sprites in your Mazing game to act as bonuses along the way.

These should disappear (hide) when the explorer touches them and add to a score variable. Be sure to place some of them **away** from the quickest route around the maze to make it more challenging!

Extension 4: Now you see it...

Add some code to your Mazing game that shows and hides your bonus sprites after random times e.g. between 1 and 5 seconds (but experiment to see what works best).



Did you understand?

5.1 Look at the script below to make a timer variable count down from 30 to 0.



Will it work? _____

Explain your answer _____

5.2 Now make up a buggy question of your own and pass it to your neighbour.

Summary

Programming structures/commands

In this course, you have used the following programming features:

- Reacting to events
- Decision-making
 - if
 - if...else
- Variables – for example
 - scores
 - timers
- Loops
 - fixed (repeat, forever)
 - conditional (forever if)
- Collision detection
 - if ... touching
 - if ... touching colour

Scratch has many more commands, but you have now learned enough to go on to the next stage.

Scratch features

You have also learned about the following features of Scratch:

- Sprites & stage
- Properties
 - Scripts
 - Costumes/backgrounds
 - Sounds
- Animation
- Graphics tools

You now have all the skills you need to create some really amazing Scratch projects!

Scratch Project

Working in a pair or group, you are now going to **create a Scratch project of your own!**

You may have some ideas already, but programs are normally created in a series of stages:

1. Analyse
2. Design
3. Implement
4. Test
5. Document
6. Evaluate
7. Maintain



Or... A Dance In The Dark Every Midnight!

Analyse



Working in pairs or small groups, **brainstorm three ideas for your project**. Think of how it might link in with other subject areas you're studying.

Think of the areas you've covered so far...

Is it going to be music or graphics-based? A story? A game?

The Scratch gallery at <http://scratch.mit.edu> might give you some ideas.

1. _____

2. _____

3. _____



Now discuss your ideas with your teacher.

Once you have agreed on your project, describe what it will do below.



Design (Screen)

Make a storyboard of your project.

Your sketch should be labelled to show what is happening and what each sprite does.



Design (Code)

Design the steps for your code (algorithm):

- Think about the steps **each sprite or the stage** will have to perform. Write them in English.
- Think about **variables** your project will use.

Sprite/Stage	Algorithm

Sprite/Stage	Algorithm

- Think about **variables** your project will use.

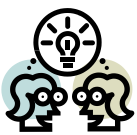
Variable name	What it will store



Implement

Now create your project!

- Gather the **sprites**, **costumes**, **sounds** and **backgrounds**
Remember to give them sensible names.
- Then create the **scripts**
Make sure you have your algorithms in front of you!



Test

Test your project to make sure it works.

Let your classmates test it too and note their comments below:

Good points: _____

Bad points: _____

Describe bugs that were found (by you or by testers) and how you fixed them:

Bug: _____

Solution: _____

Bug: _____

Solution: _____



Document

Let's imagine that you're going to post your project on the Scratch website.

Write down below a brief description (50 words max.) of:

- your project's **main features** and
- **how to use them.**

Remember – you want to get people to try out your project!

Once you have written the description, enter it into your project's notes (**File→Project notes...**).

Evaluate

How did the project turn out **compared to how you originally planned it**?

What **mistakes** did you make on the way?

If you were to start again from the beginning, what would you **do differently**?

Look at your **code** again.

Is there anywhere you could have taken a shortcut to make it “slicker”?

Maintain

What **additional features** would make your project better?



Congratulations

You have now completed this introduction to Computing Science in Scratch!

Remember that you can download and use Scratch at home, so there's no need for this to be the end of your time as a programmer.

<http://scratch.mit.edu>