

Programming Games with Scratch

Teacher's Guide

Contents

About.....	1
Suggested Activity	1
Additional Details	2
Handout Overview	3
Solutions.....	4
Driving Game.....	5
Super Fashion Dress-Up	7
Clay Shooting.....	8
Room Escape.....	10
Cat and Mouse	11
Moon Landing.....	13

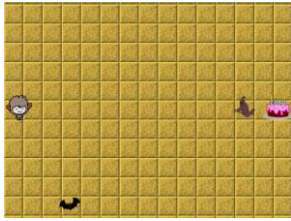
About

“Programming Games in Scratch” is a series of handouts that can be used to teach programming and computational thinking. The handouts guide students through the logic and algorithms underlying computer games. They cover many different types of games and different experience levels, so it is easy to customize your lessons to the interests and abilities of different students. All the necessary instructions are included on the handouts themselves, so students can often follow them without guidance from a teacher. As such, teachers do not necessarily need a programming background to use the handouts in classrooms; nevertheless, a programming background is useful for teachers to help students contextualize the lessons, to help students when they encounter difficulties, and to direct students to supplementary material for further learning.

The handouts make use of the Scratch programming language from MIT. Scratch is a free programming language that can be used from most PCs without the need to install any software. It is designed to expose programming to students in a friendly environment that encourages creativity and independent learning.

Suggested Activity

The handouts contain all the necessary instructions to use them. As such, students can follow the handouts themselves individually or in small groups. Younger students may need adult guidance to read through the handouts and to stay focused on the tasks.



Unless students have experience using the Scratch language, they should all start with the introductory handout “Making a Basic Game in Scratch.” They simply visit the website listed on the first page of the handout and then following the instructions on the handout. The introductory handout provides an overview of the basic functionality and features of the Scratch language by guiding students through the making of a simple game. The handout is written in a “walkthrough”-style: all steps are described in exact detail with no occasions for deviating from the suggested solution.



The other handouts are more open-ended and require more reflection by students to complete. Instructions are described in general terms, requiring students to figure out the exact steps needed to solve the problems themselves. Each handout is self-contained, so students can choose which handouts to complete based on their interests and abilities. The handouts also often contain optional sections of varied difficulties, which students can optionally complete based on their own interests and abilities.

Additional Details

Below is a list of the handouts:

Handout	Difficulty	Topics
Making a Basic Game in Scratch	Introduction	Introduction
Driving	Easy	Movement
Super Fashion Dress-Up	Easy-Medium	Costumes
Clay Shooting	Easy-Medium	Costumes, Random
Room Escape	Medium	Messages
Cat and Mouse	Hard-Medium	Collision Detection, Artificial Intelligence
Moon Landing	Hard	Variables

If students want to save their games, they can either

1. Choose “Download to your computer” from the “File” menu at the top of the website
2. After making some changes, sign in to Scratch, click on the orange “remix” button in the upper-right of the screen, then choose “Save now” from the “File” menu.

Printable copies of the handouts and downloadable versions of the project files can be found at the website <http://o.ooli.ca/en/scratchgames>

Do not be constrained by the handouts. Feel free to let students to choose a different direction. Help foster

an independent interest in programming. Encourage them to change the artwork, add their own sounds, and to experiment with their own ideas. Have them add more enemies, write a story, or adjust the speeds of things to change the difficulty level.

Handout Overview

Overall difficulty

When you start, go to this website for the project files

If you aren't familiar with how a game is supposed to play, try out the finished game first

These are the blocks used in the sample solution. Use this list as a clue about how to solve each step. If your solution is different though, that's ok. The important thing is figuring out how to solve problems, not getting the right answer.

Difficulty levels of each step are listed at the side. These levels are just guesses. They might be easier or harder for different people.

Steps needed to make a basic game are in white

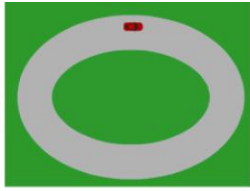
Optional improvements to the game are in grey

Although the handouts are in colour, you can print them in black & white without any problems.

EASY Driving Game
<http://o.ooll.ca/en/scratchgames/car>
 Drive around a track in your fast red car!

The game uses an oval race track as the background. There is one sprite in the program, a red car. If you click on the red car sprite, you will see that there is already code for turning the car to the right when you press the right arrow key on the keyboard. Try it out. Click on the green flag to start the program, and then press the right arrow key. The car will turn.

(Finished game: <http://scratch.mit.edu/projects/20356000>)

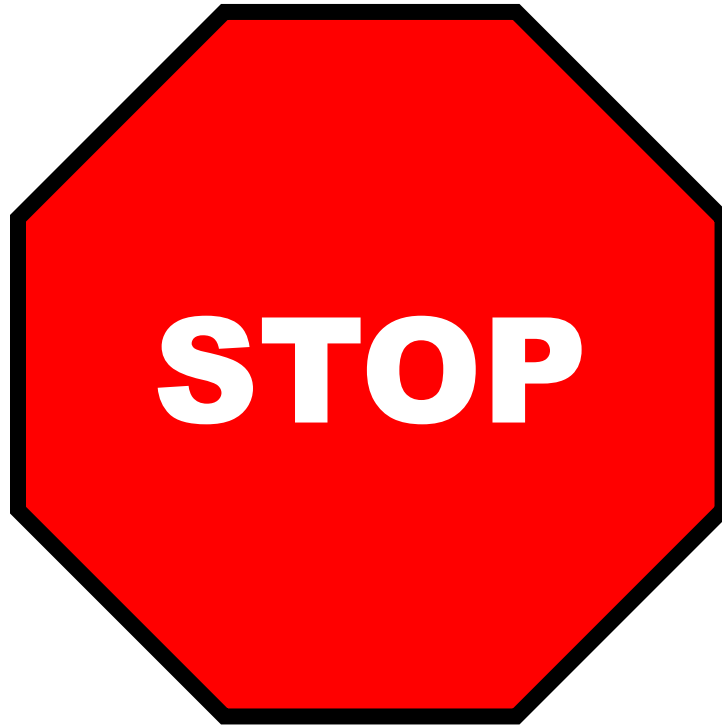


Useful blocks for this exercise

<p>Motion</p> <p>move <input type="text"/> steps</p> <p>turn <input type="text"/> degrees</p> <p>Sensing</p> <p>key <input type="text"/> pressed?</p> <p>touching color <input type="text"/></p> <p>Data</p> <p>set speed <input type="text"/> to <input type="text"/></p> <p>change speed <input type="text"/> by <input type="text"/></p> <p>speed <input type="text"/></p>	<p>Events</p> <p>when <input type="text"/> clicked</p> <p>Control</p> <p>forever</p> <p>if <input type="text"/> then</p> <p>Operators</p> <p><input type="text"/> and <input type="text"/></p> <p><input type="text"/> < <input type="text"/></p>
--	---

<p>1. Driving EASY</p> <p>Right now, the car can turn but it doesn't go anywhere. Change the program so that the car always moves forward slowly.</p> <p>Now the car moves forward, and you can press the right arrow key to turn right. Can you drive around the race track?</p>	<p>2. No Shortcuts MEDIUM</p> <p>Right now, you don't have to stay on the road. You can drive over the grass as a shortcut. This is unfair. Can you make it so that the car moves forward more slowly if it is touching the green area?</p>
<p>Try It: Different Tracks EASY</p> <p>The race track doesn't need to be an oval. You can change it. Click on the "stage" on the left of the screen. Then click on the backdrop tab. Try changing to a different race course background.</p>	<p>Try It: Multiplayer MEDIUM</p> <p>You can change your program so that two people can drive cars at the same time. Make a copy of the car and paint this new car to look different. Then change the controls for this new car so that it uses different keyboard buttons for moving. Now try racing against a friend to see who is fastest.</p>
<p>Try It: Speed and Brakes HARD</p> <p>Your car only has one speed. It can move forward or it can stop. Let's change the program so that you can go different speeds.</p> <p>First, create a variable called "speed" (only for this sprite). A variable is a number that you can change. Instead of the car always moving forward by the same number steps every time, you can change that number.</p> <p>Then, in your code, whenever you move forward, replace the number of steps you move forward by the "speed" variable. If the "speed" variable contains a bigger number, then the car will move forward by a bigger amount, and the car will go faster. If the "speed" variable contains a smaller number, then the car will move forward by a smaller amount, and the car will move slower.</p> <p>Change your program so that when you press the up arrow key, the car goes faster. Also change it so that when you press the down arrow key, the car goes slower.</p> <p>Does the car still move slowly when you drive on the grass? What is the fastest you can drive around the track? Can you make the car drive backwards?</p>	
<p>More Ideas</p> <p>Can you draw your own race course? Remember that in your program, the green is for grass and makes your car go slowly. Can you keep track of the number of laps that you have raced? How about having cars with a limited amount of gas? What about making pit stops where cars can get more gas?</p>	

Solutions



Although sample solutions are provided here, please avoid using them.

There are no “correct” or “perfect” solutions to the handouts. There are many ways to program the same game. It is also possible (and even encouraged) for students to write their own variants of the games that play differently and use completely different techniques.

The most important part of learning to program is to develop problem solving skills. Learners often give up and want to see the “answers” too early. In programming, the “answers” are actually unimportant. The important part of programming is struggling with problems and coming up with solutions. Some programming problems have no solutions. Some require a lot of reflection and ingenuity to find a solution. Some problems exceed a programmer’s skill level and require programmers change their programs to behave in a way that’s easier to program.

The solutions are mainly provided to clarify some unclear descriptions in the handouts and to help instructors provide hints to guide students.

Driving Game

1. *Driving Part 1:* Add a block for moving forward into the forever loop



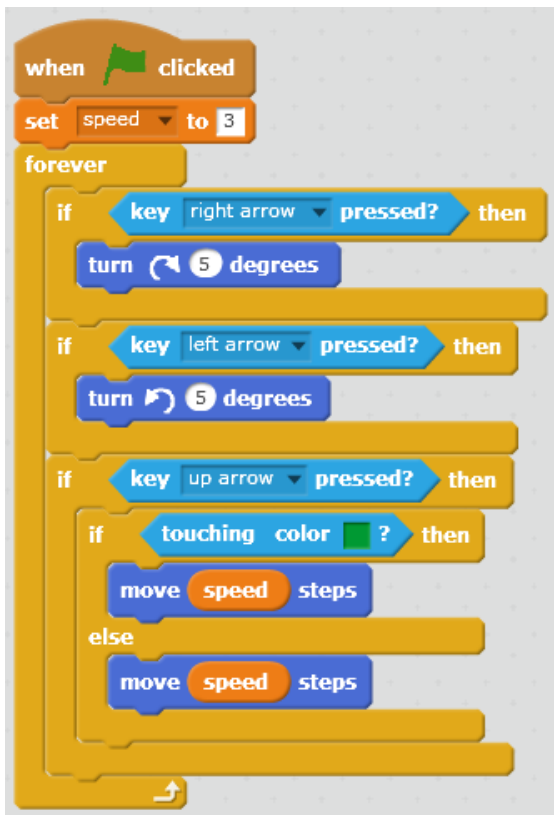
1. *Driving Part 2:* The blocks for turning right can be repeated for turning left and moving forward. In this case, they were put into a single forever loop so that the game will continually check if the keys are pressed forever. The old code for moving forward is removed.



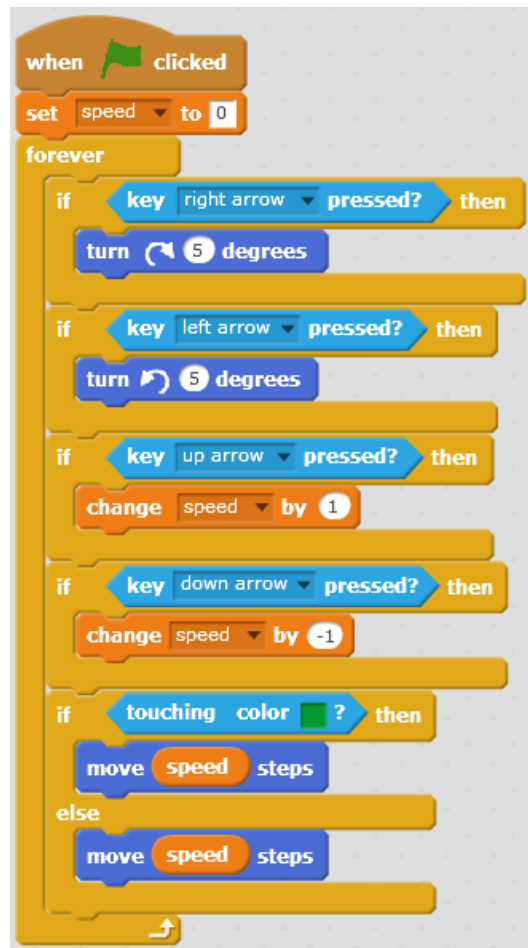
2. *No Shortcuts:* When the up arrow key is pressed, the game should check whether the car is touching green (i.e. the grass) and move forward a smaller amount if that is the case.



Try It: Speed and Brakes Part 1: It's the same code as in step 2 except that a new speed variable is substituted in for the number of steps to move forward. That speed variable is initially set to 3.



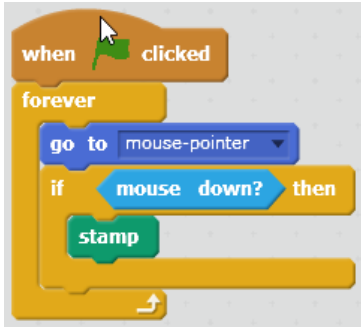
Try It: Speed and Brakes Part 2: Then, we can start playing with adjusting the speed. The speed will initially be set to 0. When the up arrow is pressed, the car should go faster. That means the speed should be increased. Do something similar when the down arrow is pressed for slowing the car down. Lastly, the car should always move forward by the speed regardless of what keys are being pressed.



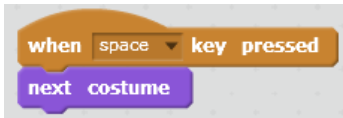
Super Fashion Dress-Up

2. Position the Pants: On the Decoration sprite, you should first add a block to move the decoration to wherever the mouse is pointing. That way, when you move your mouse, the pants is always there.

Then, it should continually check to see if the mouse button is being pressed. If it is, the “stamp” command is used to stamp the decoration onto the picture.



3. Different Clothes: On the Decoration sprite, you should move to the next costume whenever the space key is pressed.



4. Different Sizes and Angles:



5. Doll Styles Part 1: This is a little tricky in that it looks like you can only change the backdrop to a name, but you can actually change it to a backdrop # too. If the first arrow is meant for changing the hairstyle, the backdrop number only needs to change by 1.



5. Doll Styles Part 2: For the second arrow, changing the skin color requires skipping ahead by 2.

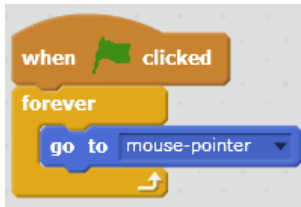


5. Doll Styles Part 3: For the third arrow, changing the hair color requires skipping ahead by 8.



Clay Shooting

1. *Move the Crosshair:* On the crosshair sprite, have it continuously move itself to where the mouse pointer is.

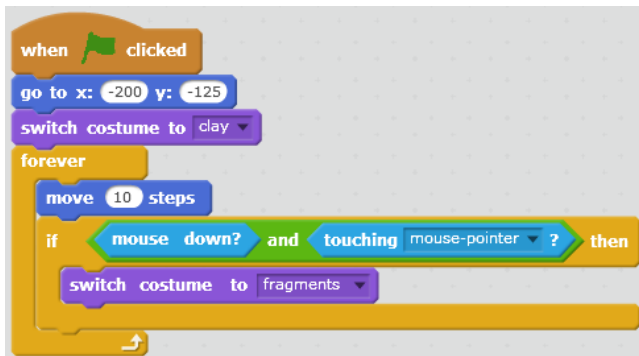


2. *Flying Clay:* On the clay sprite, move it to the starting position. Then have it just continually move.

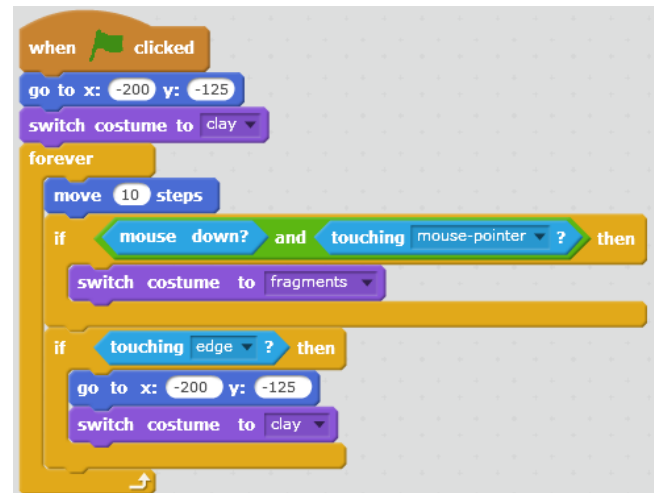


3. *Shooting:* Write your code on the Clay sprite.

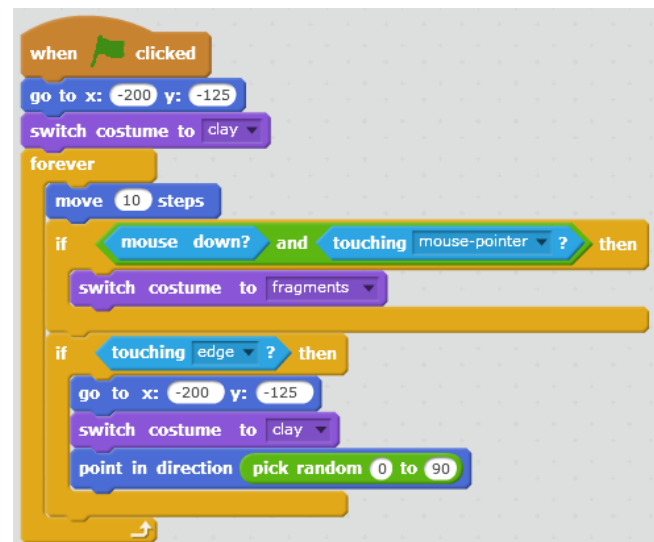
Although it's possible to use the "When sprite is clicked" event block, younger kids often have trouble with that block because the sprite has to be clicked quickly for the event to trigger (Scratch will think you're trying to drag the block), and they lack the coordination needed to do so. Instead, it's more reliable to continually check if the mouse button is pressed down and the sprite is touching the mouse pointer. Switch costumes as appropriate.



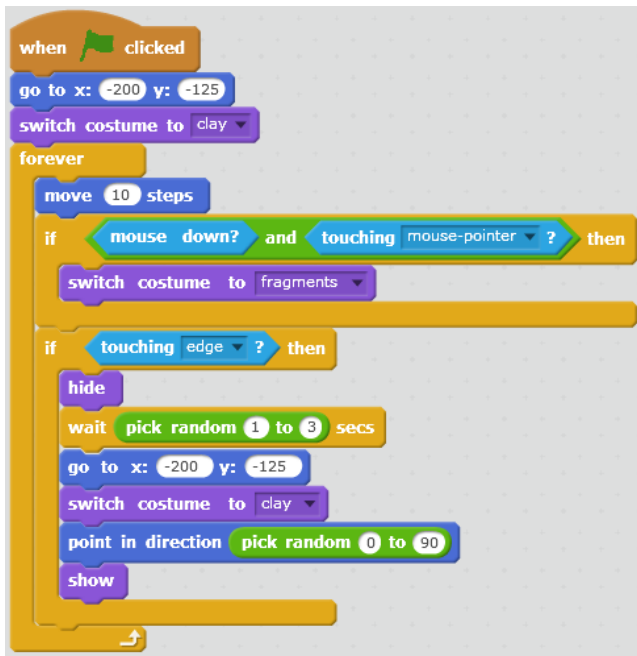
4. *Play Again:* After the clay moves, it should check if it's touching the edge and then reset its position and costume if it is.



5. *Different Directions:* When hitting the edge, the clay will not just reset its position and costume. It will also choose a random direction to move in.

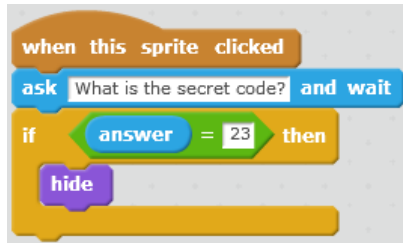


6. *No Cheating*: Hide the clay when it touches the edge. Wait a random amount. Show the sprite again after the wait.



Room Escape

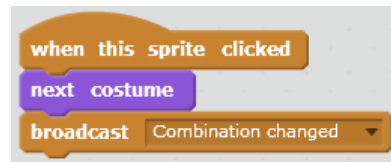
1-4. Books, Painting, Clock, Door: The code should be put on the book, painting, clock, and door respectively. The code is similar for all three. When the sprite is clicked, they should ask for the code, and then hide themselves if the right answer is given.



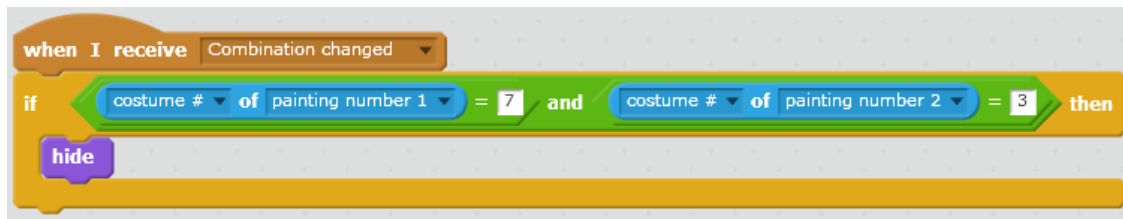
Try It: Combination Lock: On every single number, it should advance to the next number when it is clicked.



Try It: Checking the Combination Part 1: On each of the numbers, it should announce that it has changed whenever its costume changes.

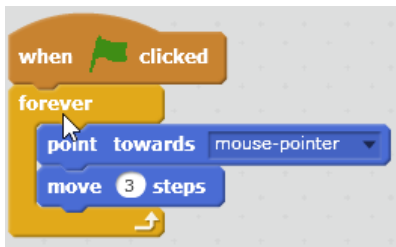


Try It: Checking the Combination Part 2: On the books, painting, clock, and door, there needs to be code that is run whenever one of the numbers on the combinations are changed. It should then check if the code is correct. Since the costume # is the same as the number being shown, it's possible to just compare the costume #s to see if they match the desired code.

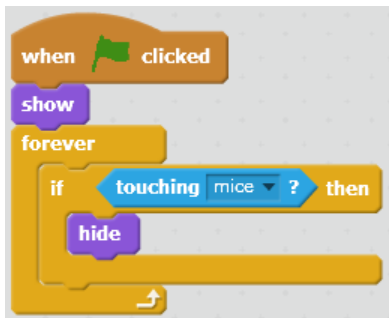


Cat and Mouse

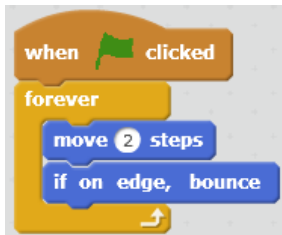
1. *The Mouse*: Put this code on the mouse. It should only “point towards” the mouse-pointer and then move forward.



2. *Cheese*: On the cheese sprite, continually check if the mouse is touching it, and hide itself if that happens.



3. *The Cat*: On the cat, just put in some blocks for moving and bouncing off edges.

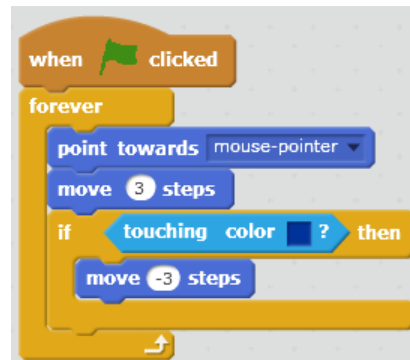


4. *Caught!*: The code below is put on the cat, but similar code can be put on the mouse instead. In this case, the code is put in a new, separate forever loop, but it's also possible to put the code in forever loop that you already have.

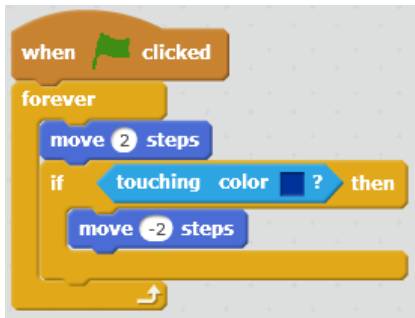


5. *Walls*: This can be a little tricky because you have to be sure that if you touch a wall, you move back to exactly where you started. Often, the logic is a little off, and the mouse doesn't move back all the way. In that case, the mouse might end up still touching the wall afterwards, leading to it getting stuck or to other erratic behavior.

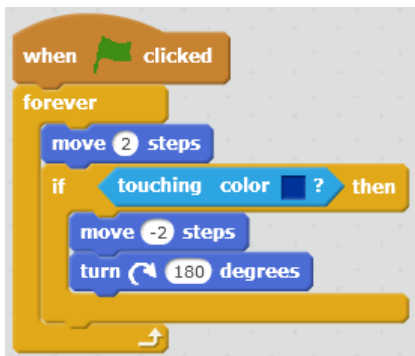
The easiest way to make sure that the behavior is right, is that ANY time the mouse moves, you should immediately check if it has a wall and immediately move it back if it is.



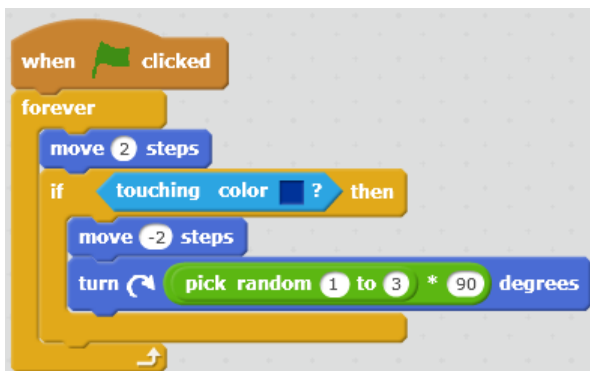
6. *The Cat and Walls Part 1*: The code for moving the cat is similar to that of the mouse.



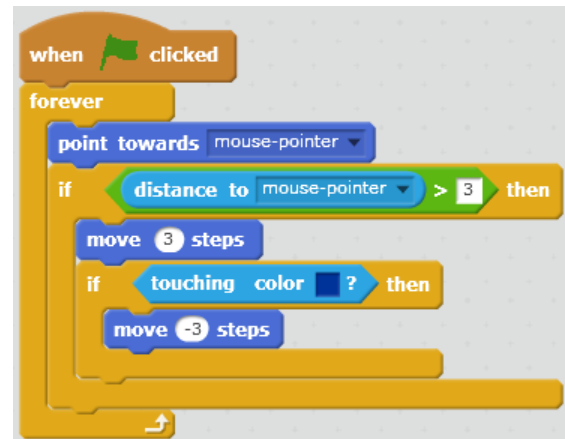
6. *The Cat and Walls Part 2*: Here, it also turns around when it hits the wall.



7. *Walking the Maze*: Instead of turning 180 degrees, the cat should turn a multiple of 90 degrees instead.



Try It: Less Jumpy: If the mouse is too close to the mouse pointer, it will “overshoot” the mouse-pointer when it moves towards it. To avoid this problem, don’t move if the mouse is already close to the mouse pointer.



Moon Landing

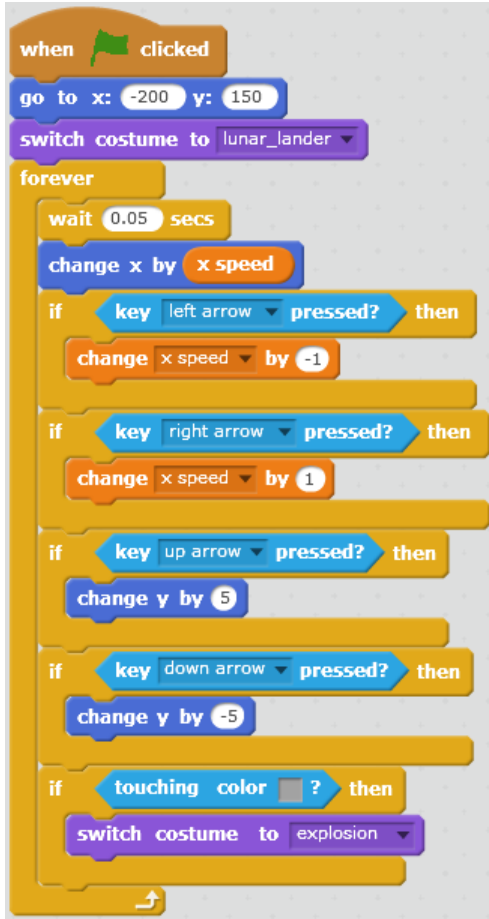
1. *Basic Movement*: Just copy the code for moving left and right in order to move up and down.



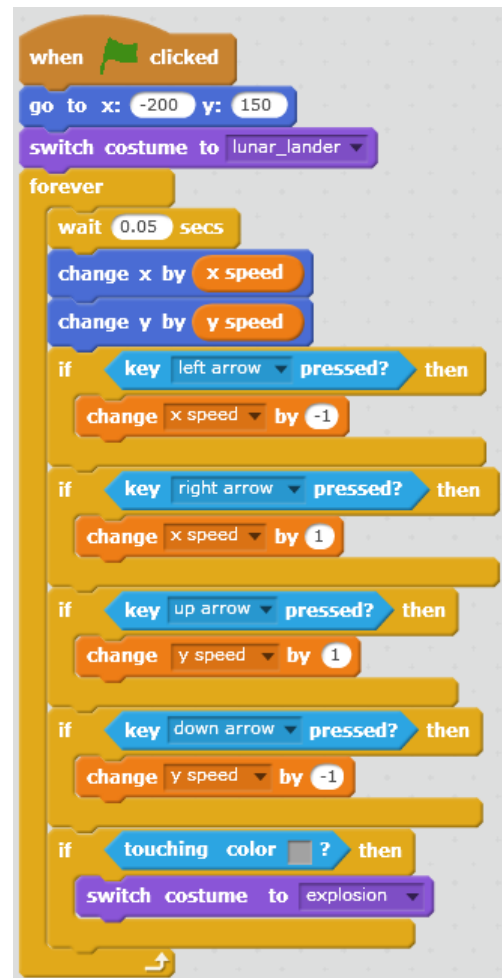
2. *Crashing*: Add some code to the “forever” loop that checks if the ground is hit or not.



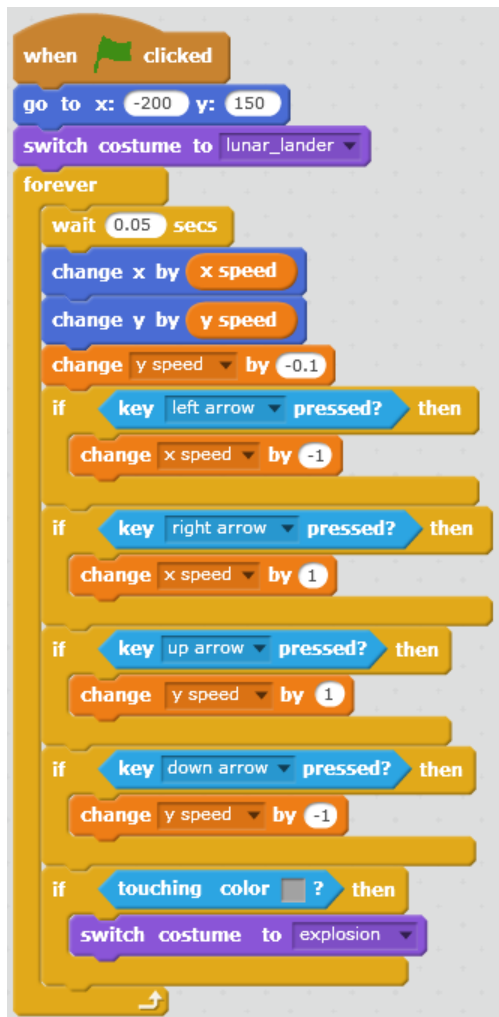
3. *Space Movement*: Instead of having the left and right arrow keys change the x position directly, these keys will change the x speed. The x position will be continuously changed by the x speed regardless of whether the arrow keys are pressed or not.



4. *Up and Down*: Do a similar thing with the up and down arrow keys.



5. *Moon Gravity*: Gravity is simulated with just an extra block that changes the y speed every turn.



6. *Landing*: Since the previous code stack is getting a little long, here the code for landing is put in a separate stack, but it can also be merged into the existing forever loop.

