Programming Games with Scratch

Teacher's Guide

Contents

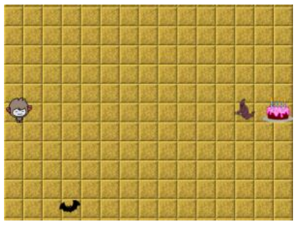
About

"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

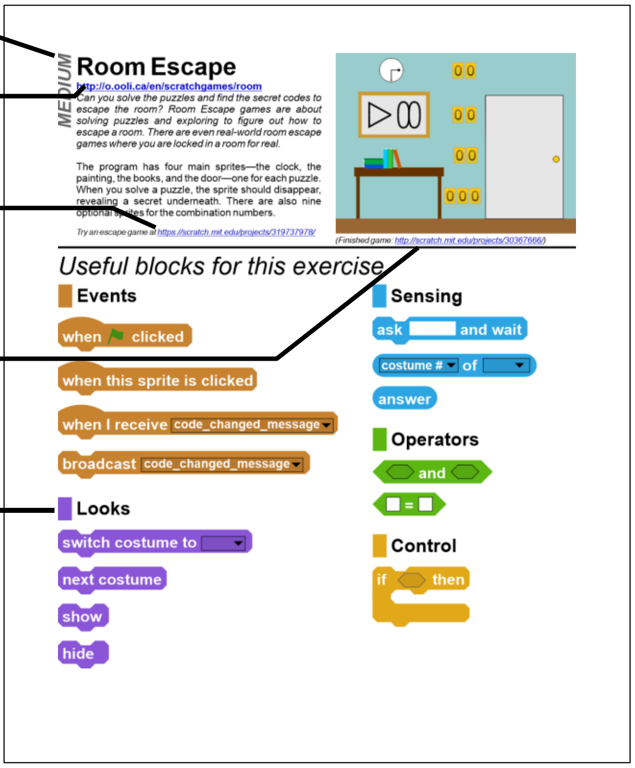
Overall difficulty

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

If you aren't familiar with this genre of game, there is sometimes a sample game you can try

If the instructions are unclear about how the game is supposed to work, play the finished game to find out

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.



MEDIUM

Room Escape

http://o.ooli.ca/en/scratchgames/room

Can you solve the puzzles and find the secret codes to escape the room? Room Escape games are about solving puzzles and exploring to figure out how to escape a room. There are even real-world room escape games where you are locked in a room for real.

The program has four main sprites—the clock, the painting, the books, and the door—one for each puzzle. When you solve a puzzle, the sprite should disappear, revealing a secret underneath. There are also nine optional sprites for the combination numbers.

Try an escape game at https://scratch.mit.edu/projects/319737978/

(Finished game: http://scratch.mit.edu/projects/30367666/)

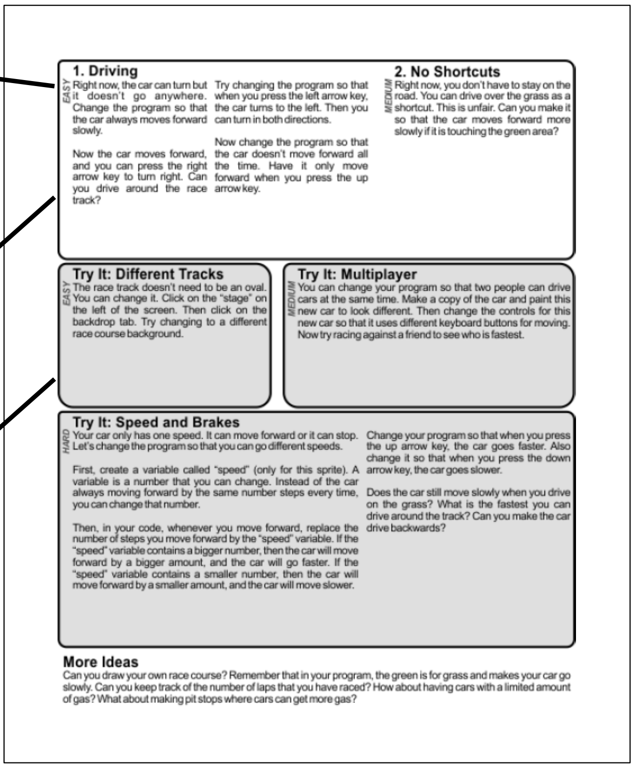
Useful blocks for this exercise

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.



1. Driving

EASY

Right now, the car can turn but it doesn't go anywhere. Change the program so that the car always moves forward slowly.

Now the car moves forward, and you can press the right arrow key to turn right. Can you drive around the race track?

Try changing the program so that when you press the left arrow key, the car turns to the left. Then you can turn in both directions.

Now change the program so that the car doesn't move forward all the time. Have it only move forward when you press the up arrow key.

2. No Shortcuts

MEDIUM

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?

Try It: Different Tracks

EASY

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.

Try It: Multiplayer

MEDIUM

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.

Try It: Speed and Brakes

HARD

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.

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.

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.

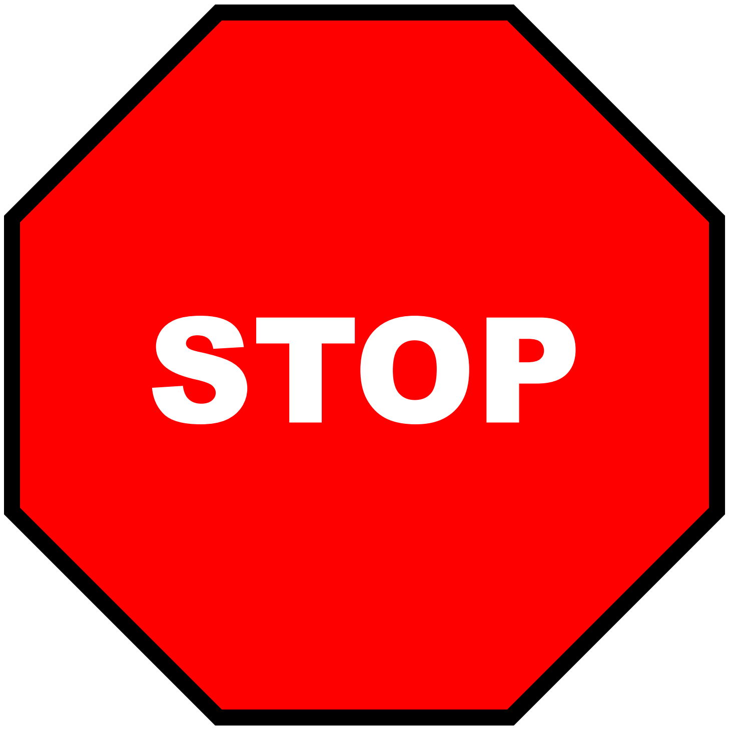
Change your program so that when you press the up arrow key, the car goes faster. Also change it so that when you press the down arrow key, the car goes slower.

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?

More Ideas

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?

Solutions



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

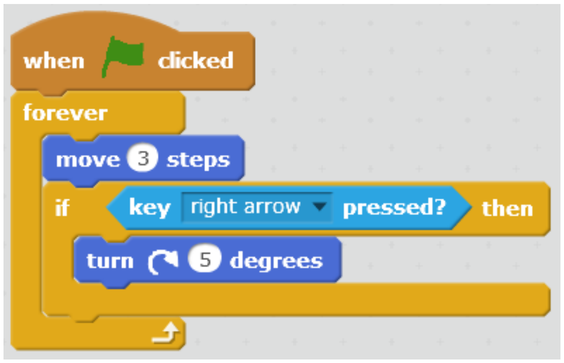
There are no “correct” or “perfect” solutions to the handouts. There are many ways to program the same game. It is also possible (and even encouraged) for students to write their own variants of the games that play differently and use completely different techniques.

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.

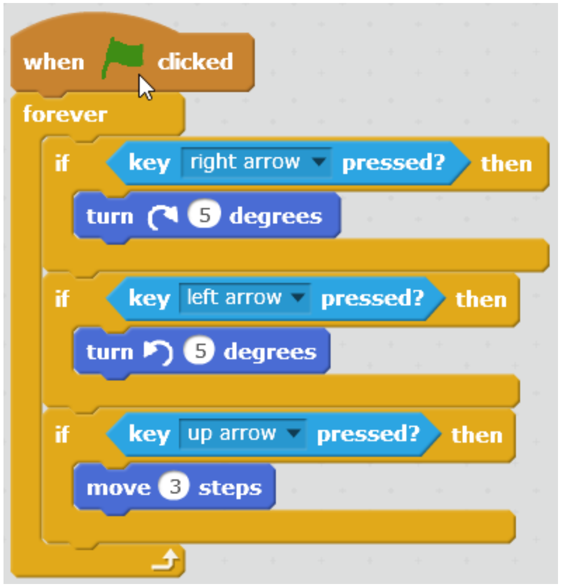
Driving Game

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



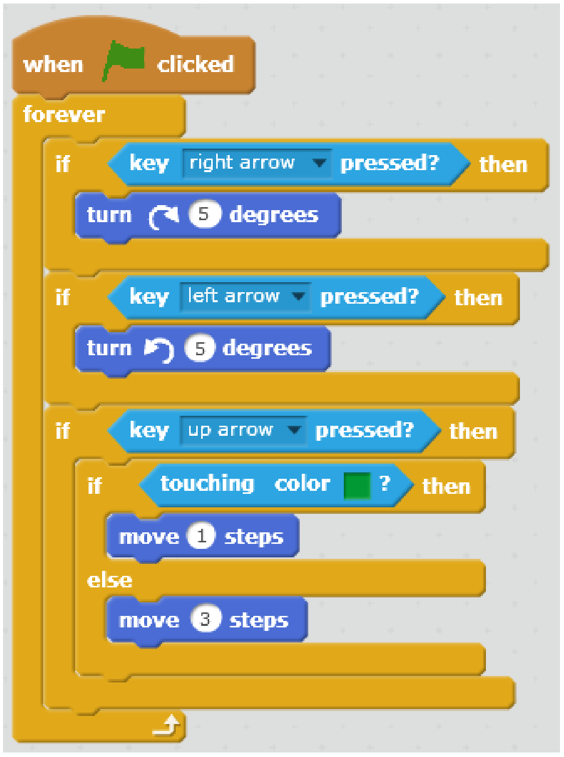
```
when [flag] clicked
forever
    move 3 steps
    if key right arrow pressed? then
        turn ↻ 5 degrees
```

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.



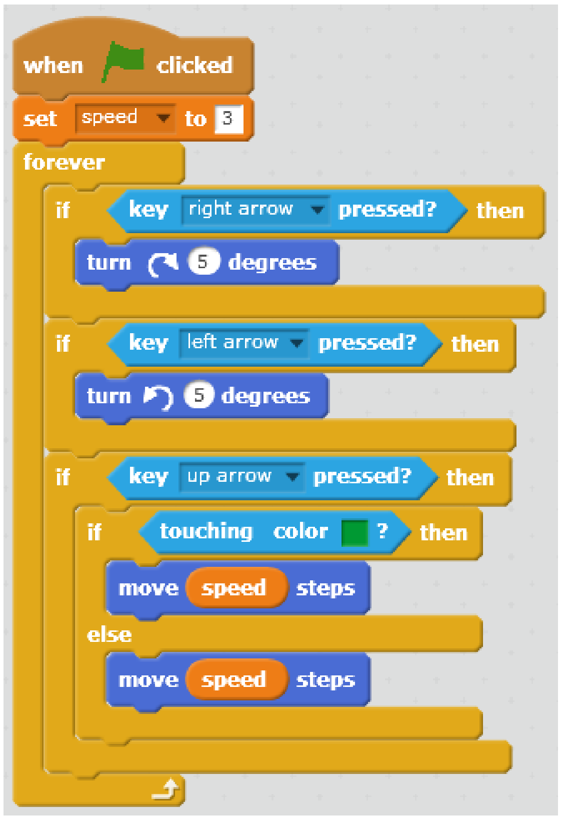
```
when [flag] clicked
forever
    if key right arrow pressed? then
        turn ↻ 5 degrees
    if key left arrow pressed? then
        turn ↺ 5 degrees
    if key up arrow pressed? then
        move 3 steps
```

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.



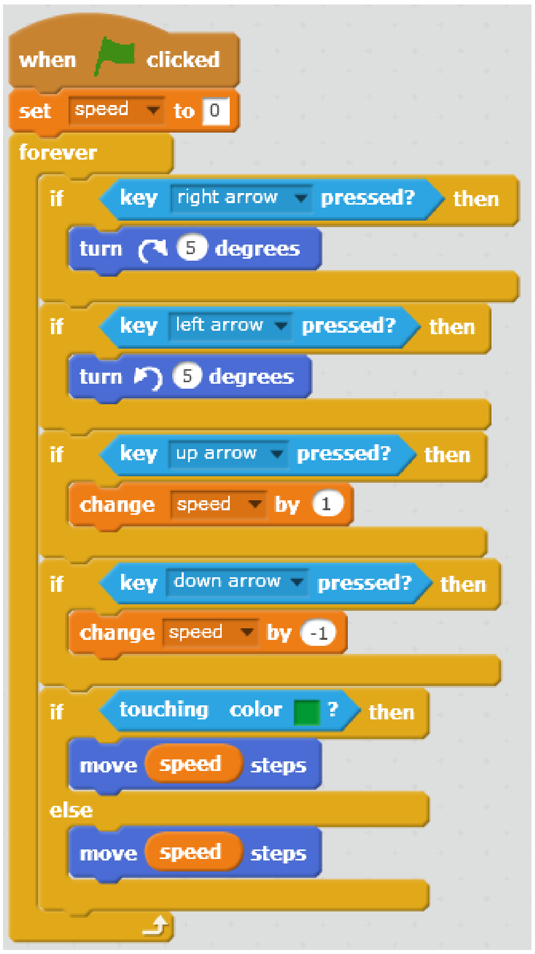
```
when [flag] clicked
forever
    if key right arrow pressed? then
        turn ↻ 5 degrees
    if key left arrow pressed? then
        turn ↺ 5 degrees
    if key up arrow pressed? then
        if touching color 🟩 ? then
            move 1 steps
        else
            move 3 steps
```

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.



```
when ⚑ clicked
set speed ▾ to 3
forever
    if <key right arrow ▾ pressed?> then
        turn ↻ 5 degrees
    if <key left arrow ▾ pressed?> then
        turn ↺ 5 degrees
    if <key up arrow ▾ pressed?> then
        if <touching color 🟩 ?> then
            move (speed) steps
        else
            move (speed) steps
```

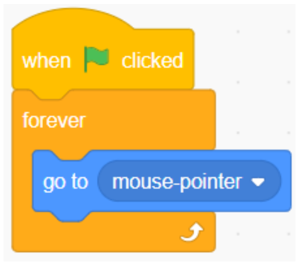
Try It: Speed and Brakes Part 2: Then, we can start playing with adjusting the speed. The speed will initially be set to 0. When the up arrow is pressed, the car should go faster. That means the speed should be increased. Do something similar when the down arrow is pressed for slowing the car down. Lastly,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



```
when ⚑ clicked
set speed ▾ to 0
forever
    if <key right arrow ▾ pressed?> then
        turn ↻ 5 degrees
    if <key left arrow ▾ pressed?> then
        turn ↺ 5 degrees
    if <key up arrow ▾ pressed?> then
        change speed ▾ by 1
    if <key down arrow ▾ pressed?> then
        change speed ▾ by -1
    if <touching color 🟩 ?> then
        move (speed) steps
    else
        move (speed) steps
```

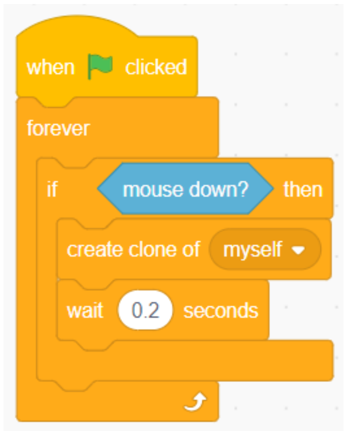
Super Fashion Dress-Up

3. Moving the Pants: You should make the "Clothes" sprite keep moving to the mouse pointer in a forever loop.



```
when 🏁 clicked
forever
    go to mouse-pointer ▾
```

4. Adding the Pants to the Doll: In the "Clothes" sprite, you should keep checking if the mouse button is pressed down, and create a clone of the sprite if the mouse button is down. You can create a new forever loop for this, or reuse the loop used in the previous step. Since clones will be continually created while the mouse button is held down, you should wait a small amount of time (0.2 seconds works well) after creating the clone to prevent too many clones from being created.

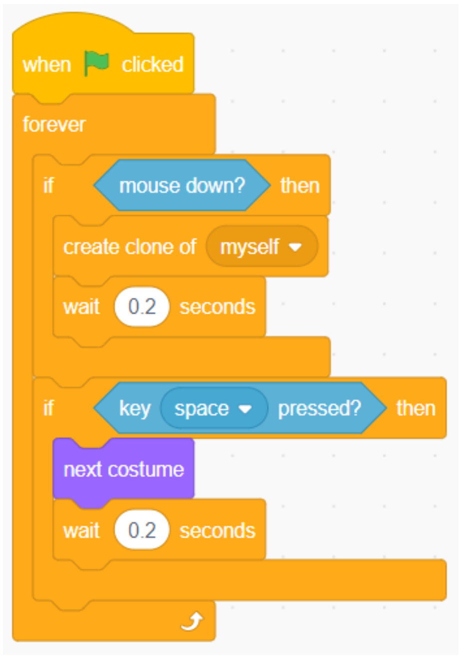


```
when 🏁 clicked
forever
    if mouse down? then
        create clone of myself ▾
        wait 0.2 seconds
```

5. More than Just Pants: Although it's possible to use a "when space key pressed" event block to check when a key is pressed, it is recommended that you use an "if ___ then" block instead.

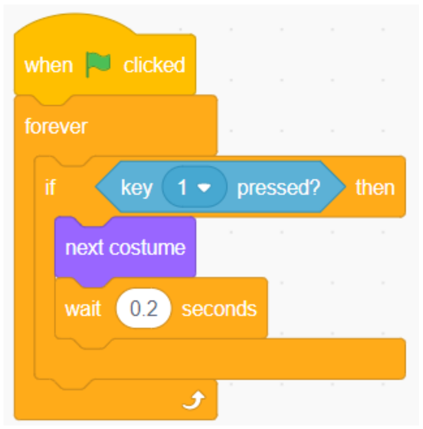
The clones of a sprite will respond to events just like the original sprite. When a "when space key pressed" event block is used, all the clothes added to the doll will change when you press space because they are clones. You don't want that. By using "if ___ then" blocks in a forever loop that is started when the green flag is clicked, you avoid this problem.

The code for the "if ___ then" block is similar to the code from the previous section, except that it will change the costume instead of creating a clone.



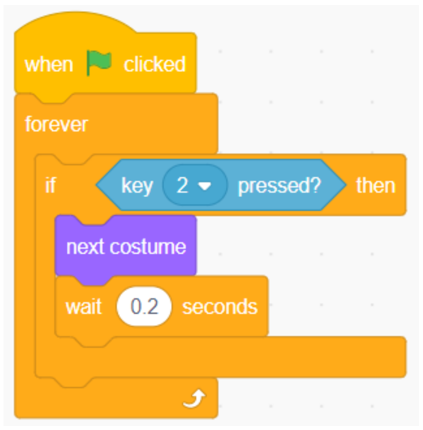
```
when 🏁 clicked
forever
    if mouse down? then
        create clone of myself ▾
        wait 0.2 seconds
    if key space ▾ pressed? then
        next costume
        wait 0.2 seconds
```

6. Body Styles: In the "Body" sprite, you should add code that is almost identical to the code you wrote in the previous section. It's also possible to use a "when ___ key pressed" event block, and it is safe to do so since you are not making any clones of the "Body" sprite, just the "Clothes" sprite.



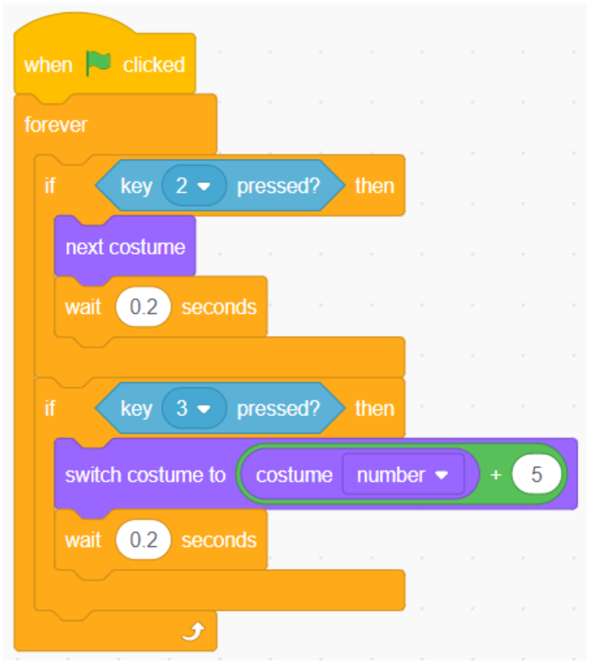
```
when 🏳 clicked
forever
    if <key (1 ▾) pressed?> then
        next costume
        wait (0.2) seconds
    end
end
```

7. New Hair at the Press of a Key: The code for the hair is the same as the code that you wrote for the body.



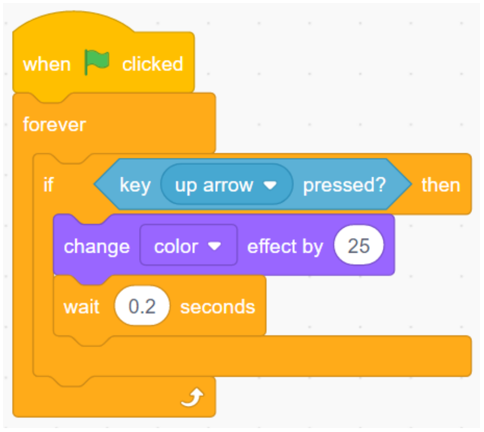
```
when 🏳 clicked
forever
    if <key (2 ▾) pressed?> then
        next costume
        wait (0.2) seconds
    end
end
```

8. Hair Styles: The "switch costume to" block can be used to switch to a costume # and not just a costume name. Since each hair style has five colours, you can change the hair style by skipping over five costumes.



```
when 🏳 clicked
forever
    if <key (2 ▾) pressed?> then
        next costume
        wait (0.2) seconds
    end
    if <key (3 ▾) pressed?> then
        switch costume to ((costume (number ▾)) + (5))
        wait (0.2) seconds
    end
end
```

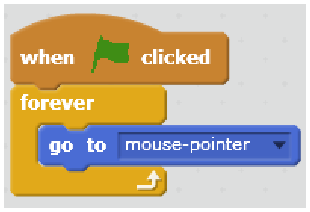
Try It: Clothing Colours: When a key is pressed, you can use the "change color effect by ___" block to change the colour of a sprite.



```
when 🏳 clicked
forever
    if <key (up arrow ▾) pressed?> then
        change (color ▾) effect by (25)
        wait (0.2) seconds
    end
end
```

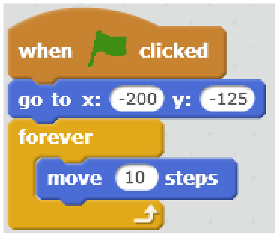
Clay Shooting

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



```
when green flag clicked
forever
    go to mouse-pointer
```

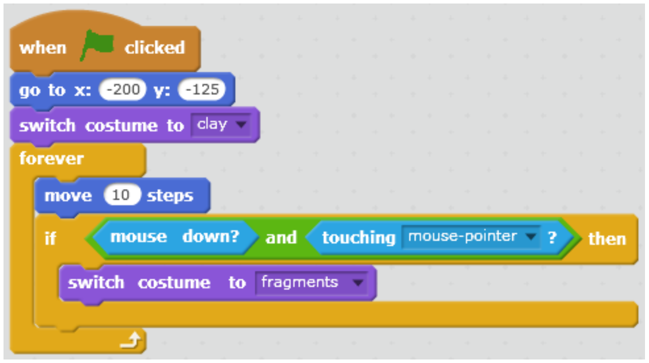
2. Flying Clay: On the clay sprite, move it to the starting position. Then have it just continually move.



```
when green flag clicked
go to x: -200 y: -125
forever
    move 10 steps
```

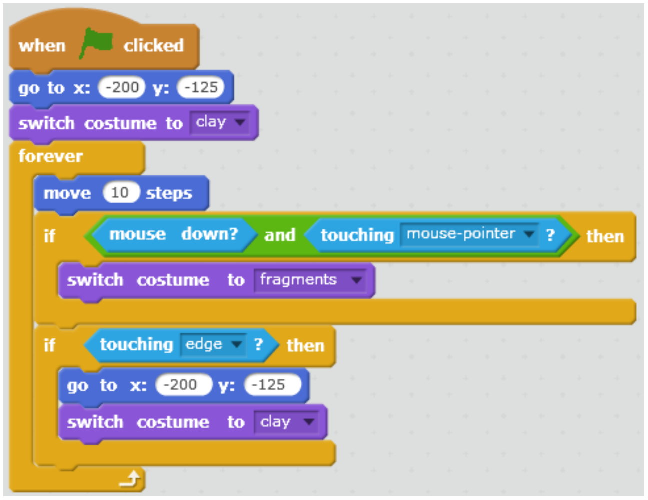
3. Shooting: Write your code on the Clay sprite.

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.



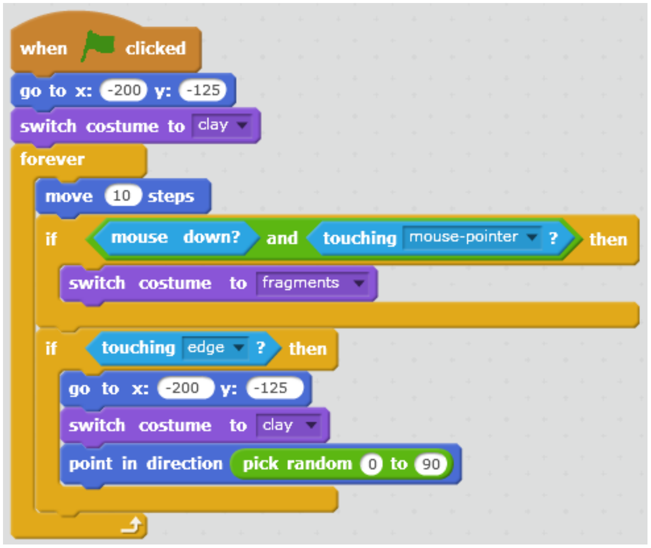
```
when green flag clicked
go to x: -200 y: -125
switch costume to clay
forever
    move 10 steps
    if mouse down? and touching mouse-pointer ? then
        switch costume to fragments
```

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.



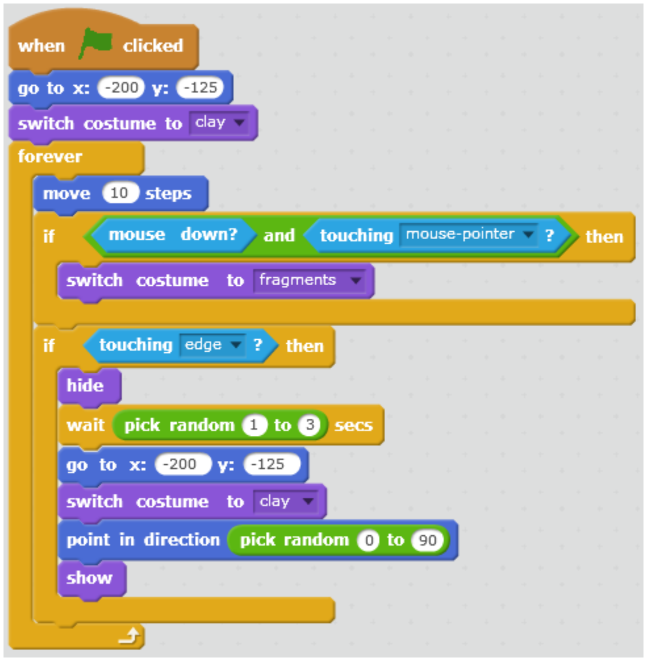
```
when green flag clicked
go to x: -200 y: -125
switch costume to clay
forever
    move 10 steps
    if mouse down? and touching mouse-pointer ? then
        switch costume to fragments
    if touching edge ? then
        go to x: -200 y: -125
        switch costume to clay
```

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.



```
when green flag clicked
go to x: -200 y: -125
switch costume to clay
forever
    move 10 steps
    if mouse down? and touching mouse-pointer ? then
        switch costume to fragments
    if touching edge ? then
        go to x: -200 y: -125
        switch costume to clay
        point in direction pick random 0 to 90
```

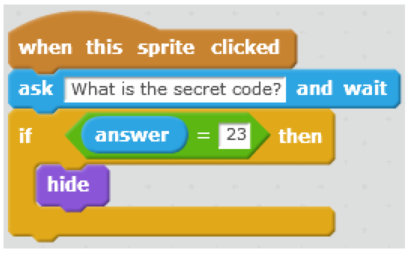
6. No Cheating: Hide the clay when it touches the edge. Wait a random amount. Show the sprite again after the wait.



```
when [flag] clicked
go to x: -200 y: -125
switch costume to clay
forever
    move 10 steps
    if <<mouse down?> and <touching mouse-pointer ?>> then
        switch costume to fragments
    if <touching edge ?> then
        hide
        wait (pick random 1 to 3) secs
        go to x: -200 y: -125
        switch costume to clay
        point in direction (pick random 0 to 90)
        show
```

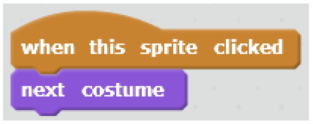
Room Escape

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.



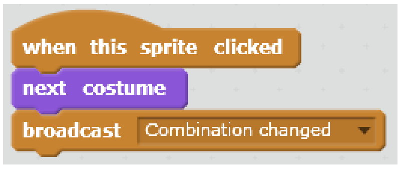
```
when this sprite clicked
ask What is the secret code? and wait
if answer = 23 then
    hide
```

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



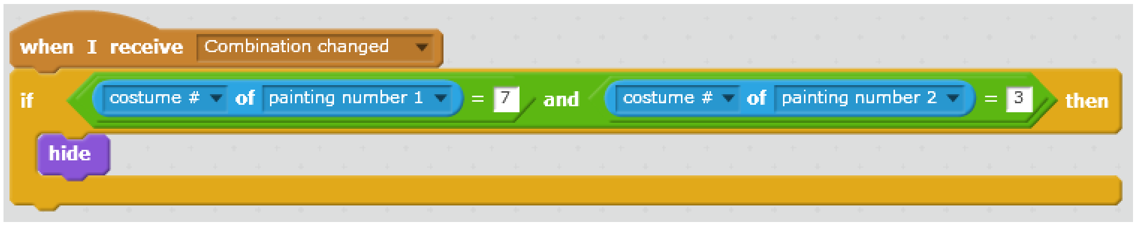
```
when this sprite clicked
next costume
```

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



```
when this sprite clicked
next costume
broadcast Combination changed
```

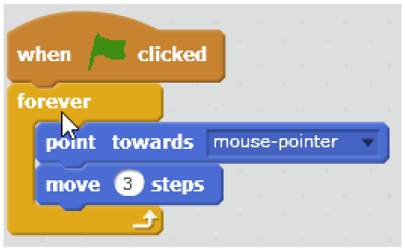
Try It: Checking the Combination Part 2: On the books, painting, clock, and door, there needs to be code that is run whenever one of the numbers on the combinations are changed. It should then check if the code is correct. Since the costume # is the same as the number being shown, it's possible to just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



```
when I receive Combination changed
if costume # of painting number 1 = 7 and costume # of painting number 2 = 3 then
    hide
```

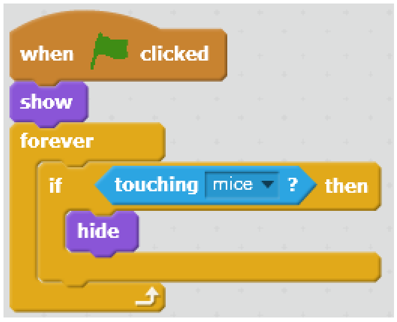
Cat and Mouse

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



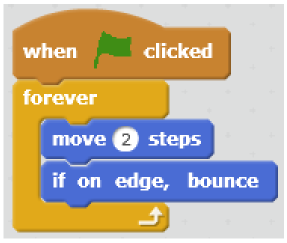
```
when [flag] clicked
forever
    point towards mouse-pointer
    move 3 steps
```

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



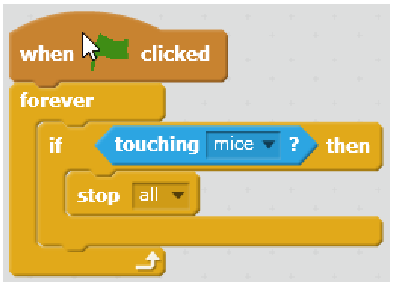
```
when [flag] clicked
show
forever
    if touching mice ? then
        hide
```

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



```
when [flag] clicked
forever
    move 2 steps
    if on edge, bounce
```

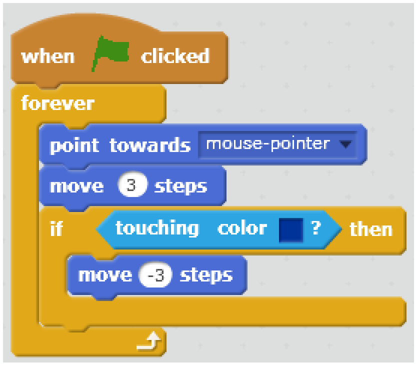
4. Caught!: The code below is put on the cat, but similar code can be put on the mouse instead. In this case, the code is put in a new, separate forever loop, but it's also possible to put the code in forever loop that you already have.



```
when [flag] clicked
forever
    if touching mice ? then
        stop all
```

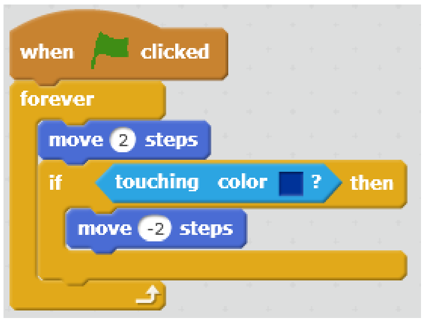
5. Walls: This can be a little tricky because you have to be sure that if you touch a wall, you move back to exactly where you started. Often, the logic is a little off, and the mouse doesn't move back all the way. In that case, the mouse might end up still touching the wall afterwards, leading to it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



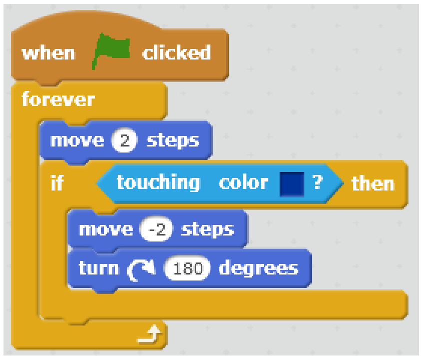
```
when [flag] clicked
forever
    point towards mouse-pointer
    move 3 steps
    if touching color ■ ? then
        move -3 steps
```

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



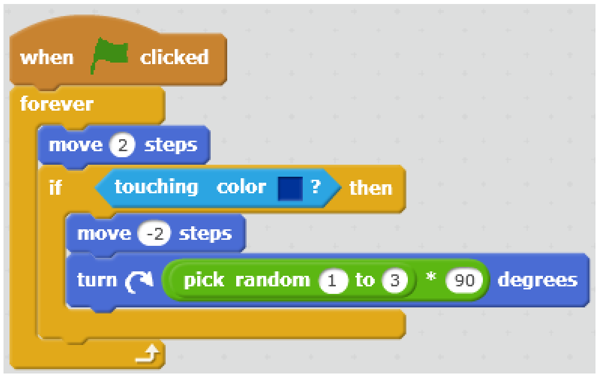
```
when ⚑ clicked
forever
    move 2 steps
    if touching color 🟦 ? then
        move -2 steps
```

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



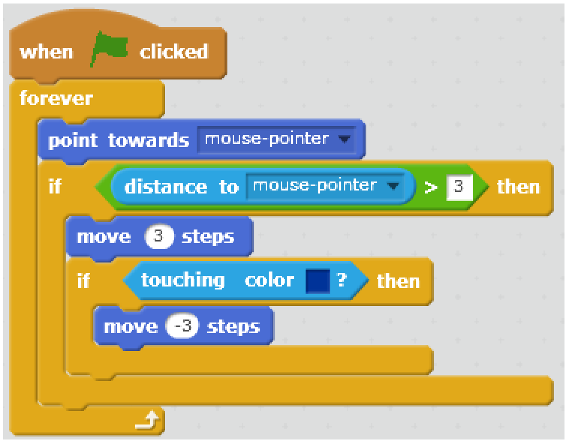
```
when ⚑ clicked
forever
    move 2 steps
    if touching color 🟦 ? then
        move -2 steps
        turn ↻ 180 degrees
```

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



```
when ⚑ clicked
forever
    move 2 steps
    if touching color 🟦 ? then
        move -2 steps
        turn ↻ (pick random 1 to 3) * 90 degrees
```

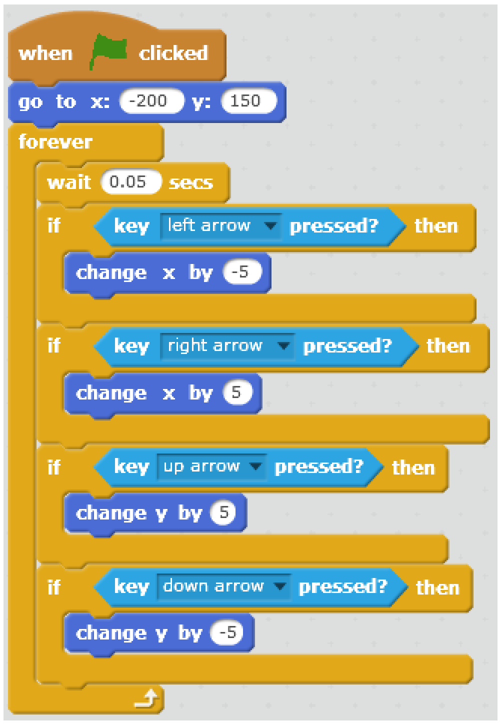
Try It: Less Jumpy: If the mouse is too close to the mouse pointer, it will "overshoot" the mouse-pointer when it moves towards it. To avoid this problem, don't move if the mouse is already close to the mouse pointer.



```
when ⚑ clicked
forever
    point towards mouse-pointer ▾
    if distance to mouse-pointer ▾ > 3 then
        move 3 steps
        if touching color 🟦 ? then
            move -3 steps
```

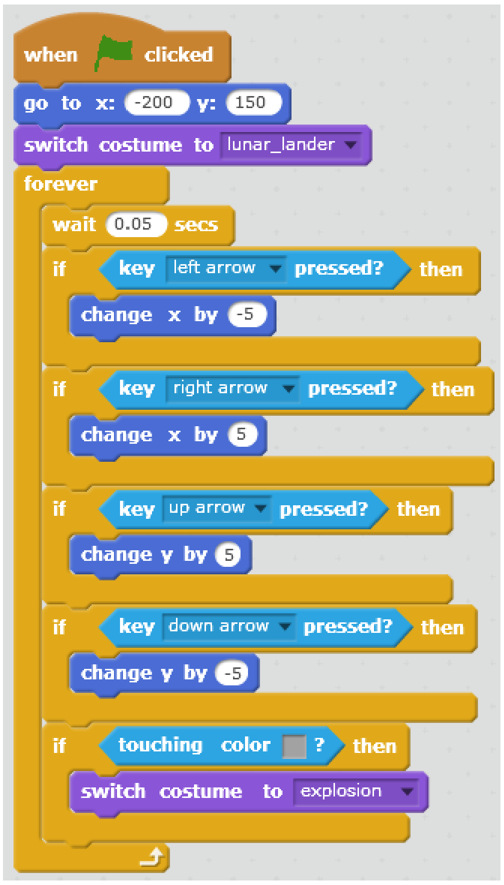
Moon Landing

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



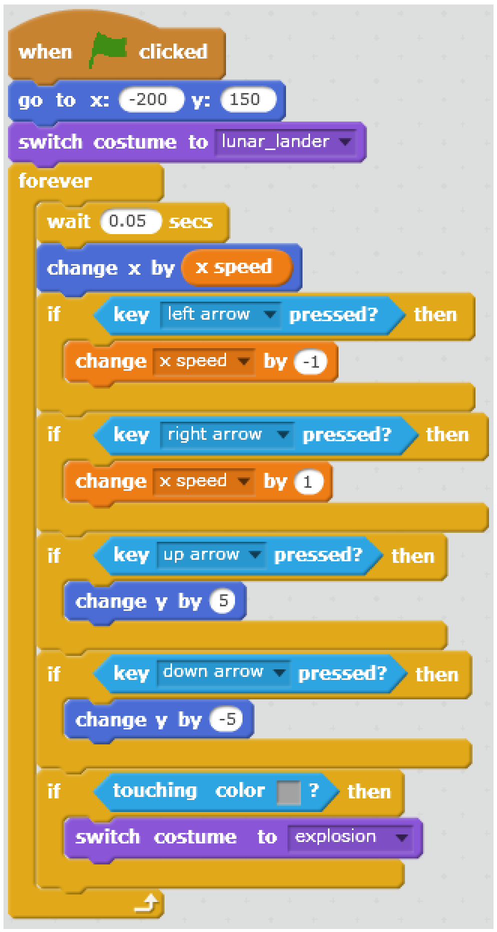
```
when [flag] clicked
go to x: -200 y: 150
forever
    wait 0.05 secs
    if key left arrow pressed? then
        change x by -5
    if key right arrow pressed? then
        change x by 5
    if key up arrow pressed? then
        change y by 5
    if key down arrow pressed? then
        change y by -5
```

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



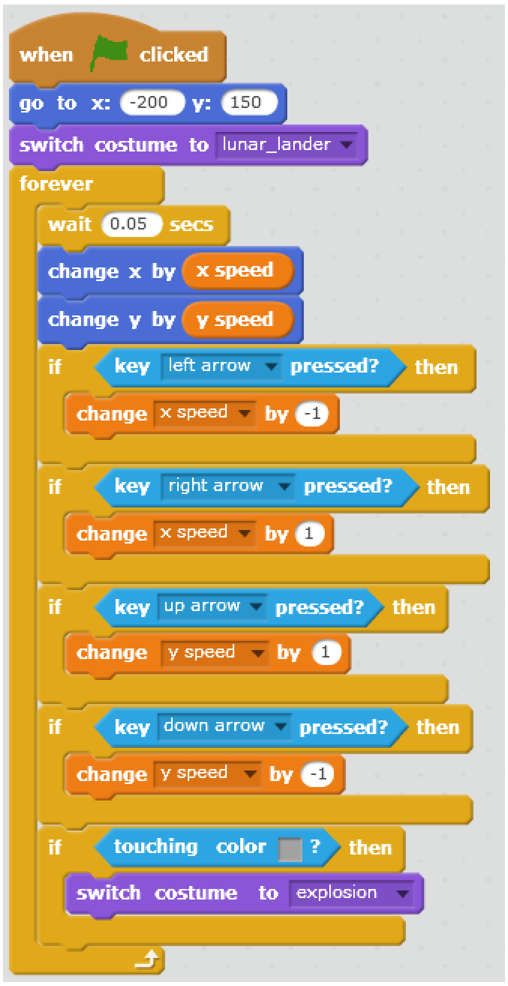
```
when [flag] clicked
go to x: -200 y: 150
switch costume to lunar_lander
forever
    wait 0.05 secs
    if key left arrow pressed? then
        change x by -5
    if key right arrow pressed? then
        change x by 5
    if key up arrow pressed? then
        change y by 5
    if key down arrow pressed? then
        change y by -5
    if touching color ⬜ ? then
        switch costume to explosion
```

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.



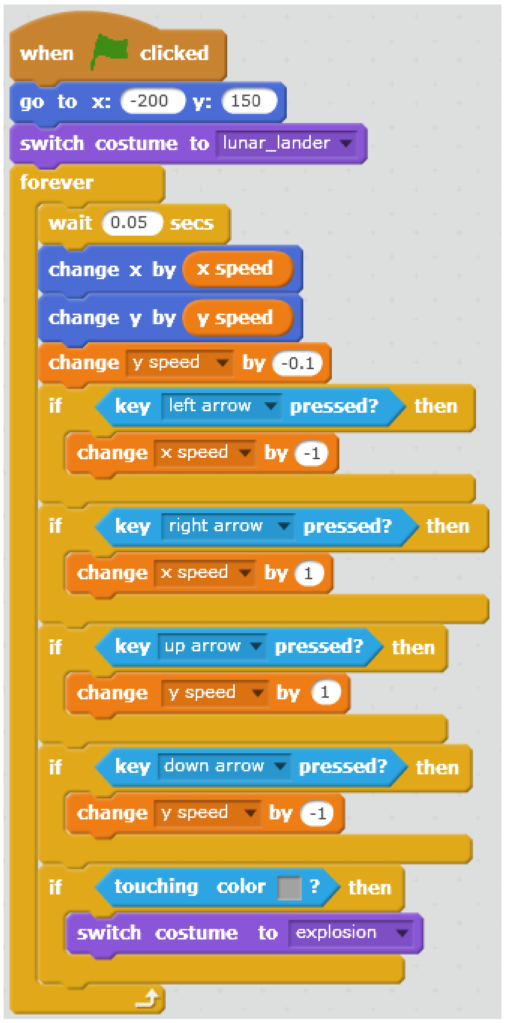
```
when [flag] clicked
go to x: -200 y: 150
switch costume to lunar_lander
forever
    wait 0.05 secs
    change x by (x speed)
    if <key left arrow pressed?> then
        change x speed by -1
    if <key right arrow pressed?> then
        change x speed by 1
    if <key up arrow pressed?> then
        change y by 5
    if <key down arrow pressed?> then
        change y by -5
    if <touching color [ ]?> then
        switch costume to explosion
```

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



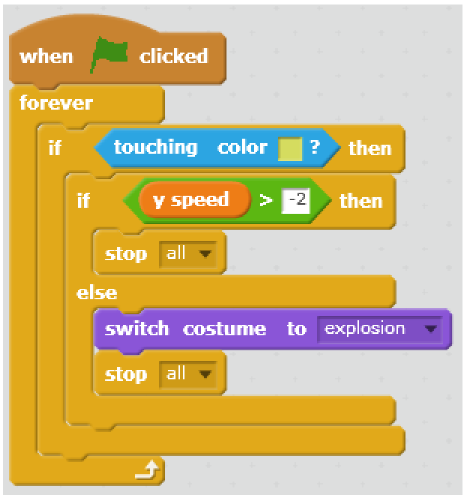
```
when [flag] clicked
go to x: -200 y: 150
switch costume to lunar_lander
forever
    wait 0.05 secs
    change x by (x speed)
    change y by (y speed)
    if <key left arrow pressed?> then
        change x speed by -1
    if <key right arrow pressed?> then
        change x speed by 1
    if <key up arrow pressed?> then
        change y speed by 1
    if <key down arrow pressed?> then
        change y speed by -1
    if <touching color [ ]?> then
        switch costume to explosion
```

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



```
when 🏴 clicked
go to x: -200 y: 150
switch costume to lunar_lander ▾
forever
    wait 0.05 secs
    change x by x speed
    change y by y speed
    change y speed ▾ by -0.1
    if key left arrow ▾ pressed? then
        change x speed ▾ by -1
    if key right arrow ▾ pressed? then
        change x speed ▾ by 1
    if key up arrow ▾ pressed? then
        change y speed ▾ by 1
    if key down arrow ▾ pressed? then
        change y speed ▾ by -1
    if touching color ■ ? then
        switch costume to explosion ▾
```

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.



```
when 🏴 clicked
forever
    if touching color 🟨 ? then
        if y speed > -2 then
            stop all ▾
        else
            switch costume to explosion ▾
            stop all ▾
```