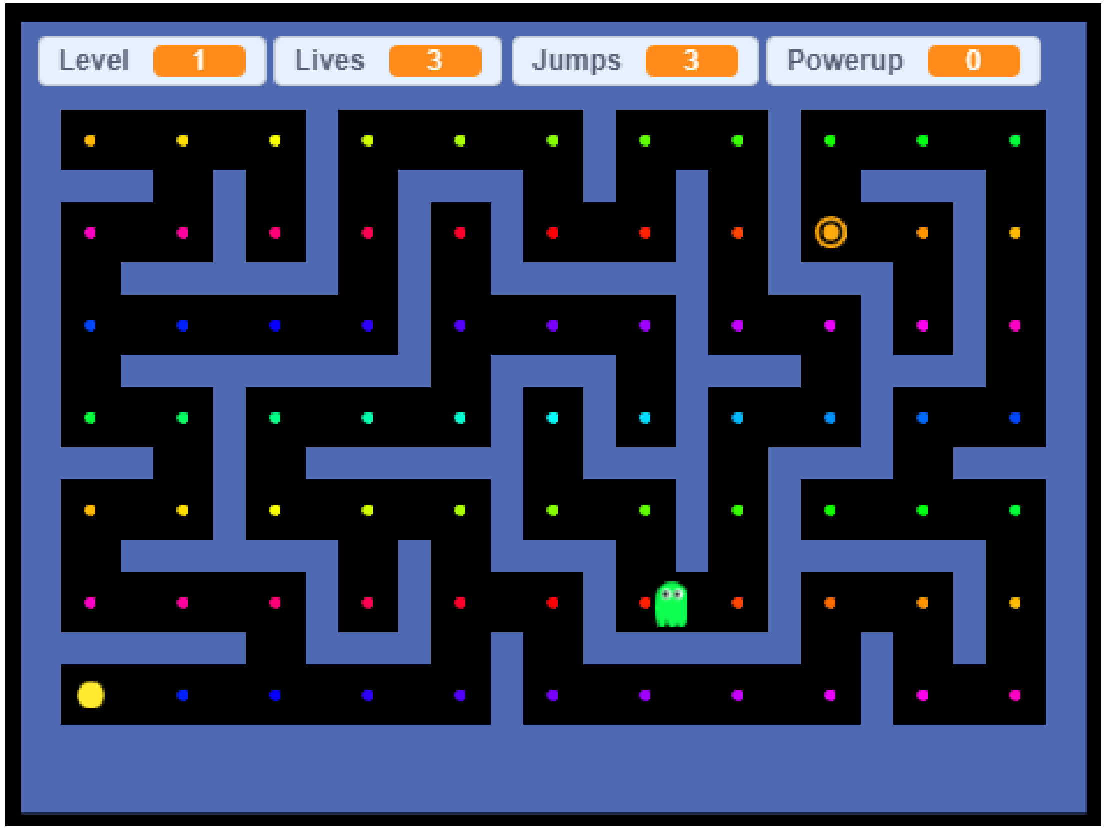
Scratch PacMan

Catford Coding Club

Project structure:

- ❑ Game design
- ❑ Sprites and backdrops
- ❑ Maze generation
- ❑ Motion, control and actions
- ❑ Game features
- ❑ Testing and feedback

The scratch project is called CCC Pacman and can be found here: https://scratch.mit.edu/projects/1256540582



Scratch PacMan

Over 5-6 sessions we will:

- ❑ Think about the design for our game
- ❑ Design sprites for the characters
- ❑ Generate a random maze for PacMan to explore
- ❑ Code our sprites so they behave as we want
- ❑ Add game features like levels, scores, powerups, sound effects
- ❑ Test each others’ games and give feedback

Scratch PacMan

Game design

Scratch PacMan

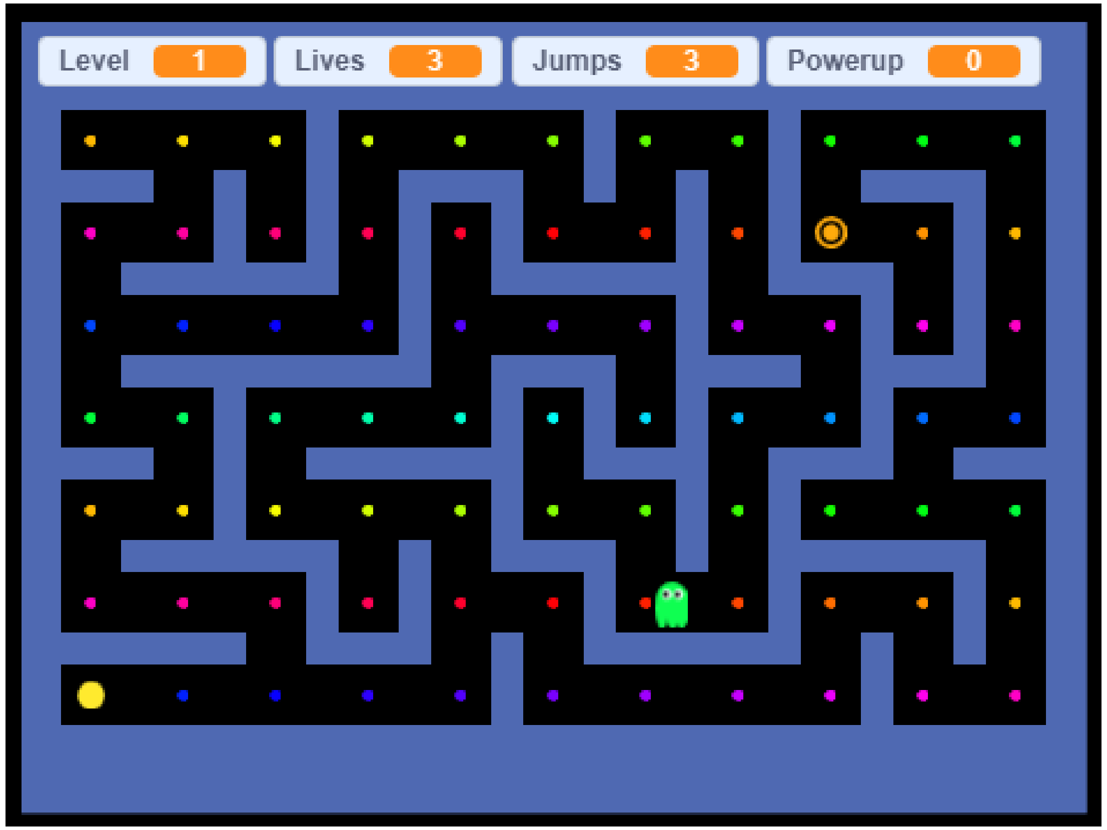
Catford Coding Club

Game design:

Before we start to design and code, we should think about how we want our game to play.

What sprites do we need? What can they do? How will they interact with each other? How do you win the game or move through levels?

Always remember to save your work as you go!



Scratch PacMan

Sprites and Backdrop

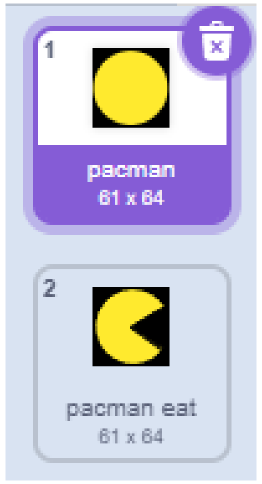
Scratch PacMan

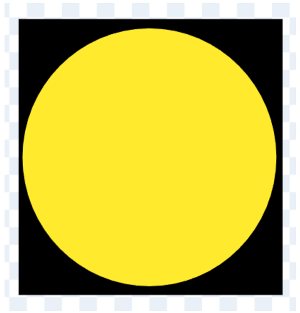
Sprites: PacMan

You can design you own sprites. My PacMan has two costumes for closed and open mouth.

I used a square black box as a background for the PacMan sprite to help move through the maze.

The paths in my maze are black so the square won't be visible when PacMan is moving through the maze.





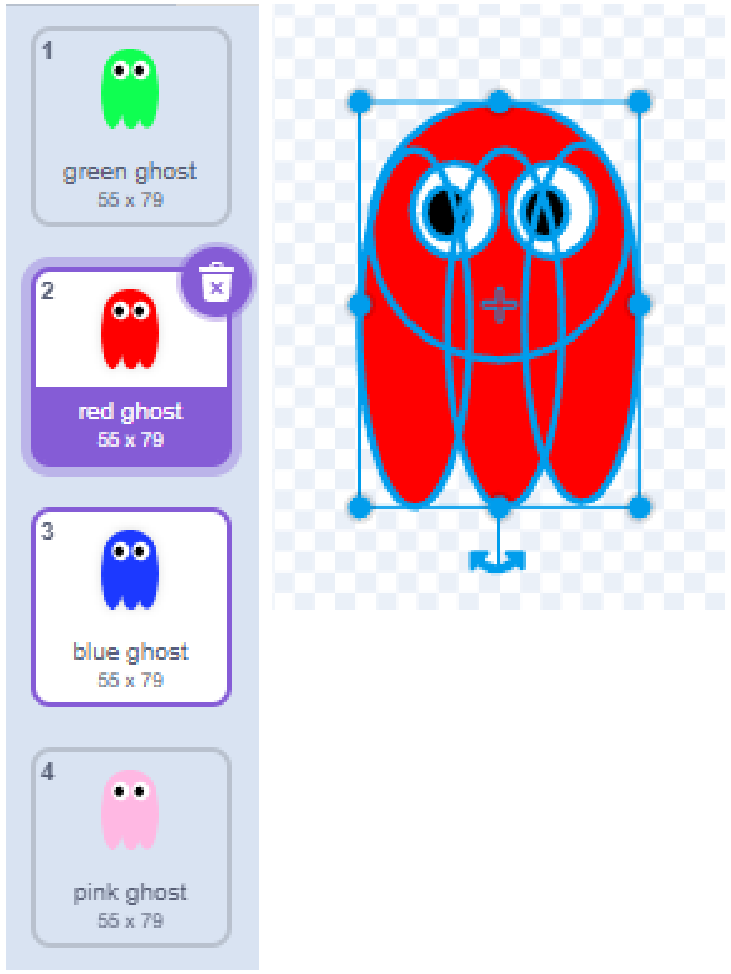
Scratch PacMan

Catford Coding Club

Sprites: Ghosts

I used circle and ellipse (oval) shapes to make my ghosts and used layers to make the eyes appear on top.

Ghosts have four costumes for different colours. We can make them behave differently in the game if we want.

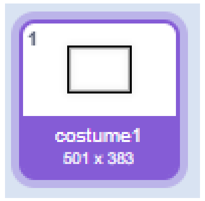


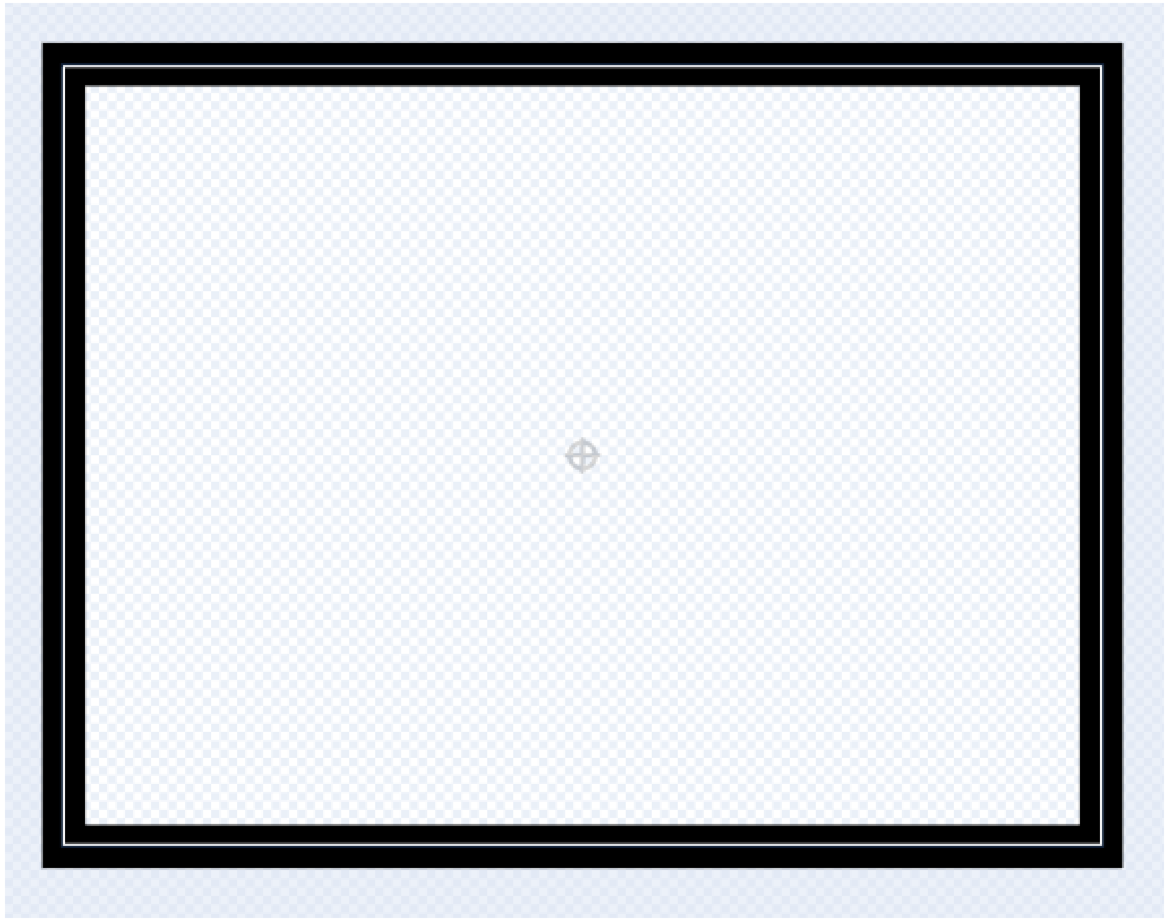
Scratch PacMan

Catford Coding Club

Sprites: Frame

Next make a black frame (same colour as the maze paths). The maze will fit inside the frame.

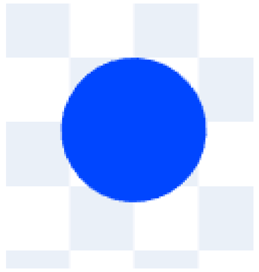




Scratch PacMan

Sprites: Food

My food sprite is just a very simple filled circle. We can make the food change colour when we place it in the maze.

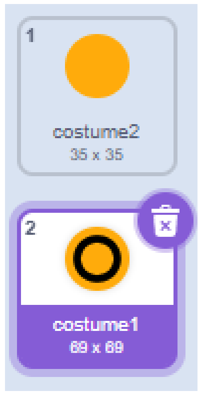


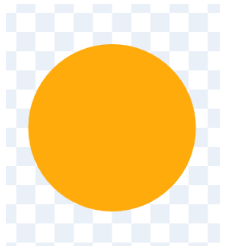
Scratch PacMan

Sprites: Powerup

We can add a powerup sprite which PacMan can eat to get extra powers. This is made with circles and two costumes so we can make a pulsing effect.

The black circle is the same colour as the maze path.





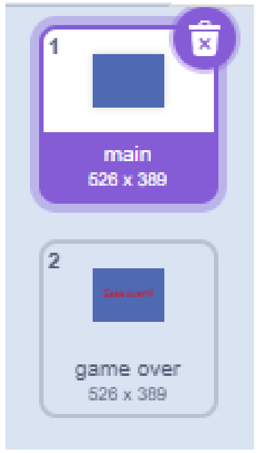
Catford Coding Club

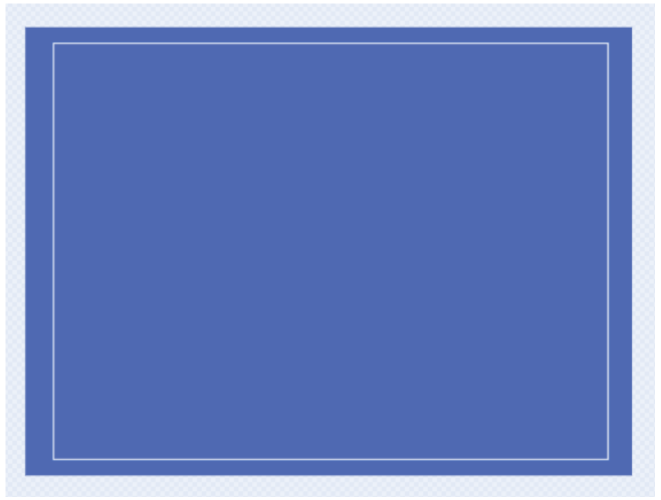
Scratch PacMan

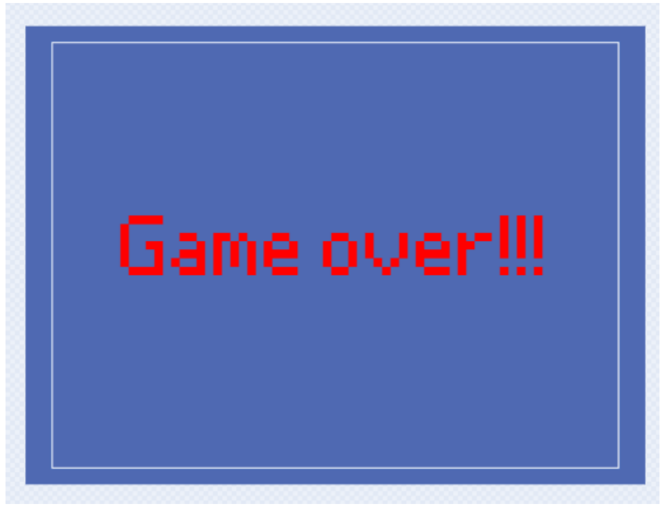
Backdrop

We also need to make two backdrops for the Stage.

I have one plain blue one and one that we will use when the game is over.







Scratch PacMan

Maze Generator

Scratch PacMan

Generating a Maze

We need different mazes for each level. We could do this by just drawing a maze sprite for each level. But we are going to try something a bit more challenging!

In this section, we are going to make a maze generator that will build a new maze for each level for PacMan to explore. This is a little complicated so let's break it down…

Credit: Driple Studios https://www.youtube.com/watch?v=YDCNovY74U4

Scratch PacMan

Catford Coding Club

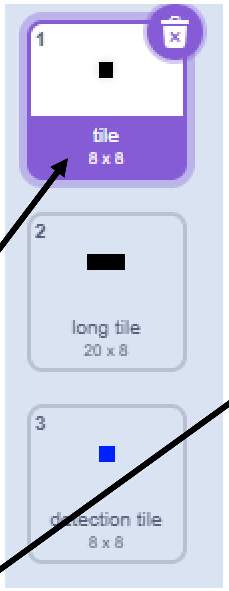
Maze generator sprites

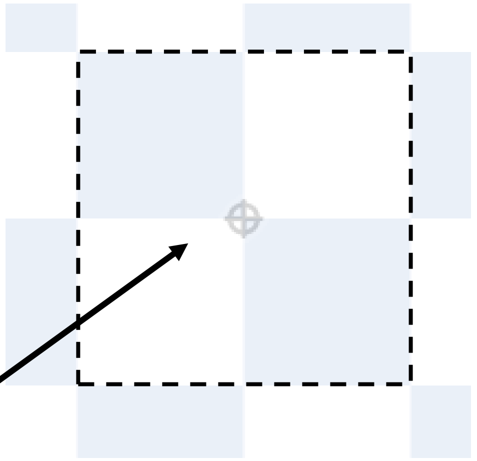
We need a special sprite for the maze generator. Mine is called **Tile**. It has three costumes. We need to make these very carefully.

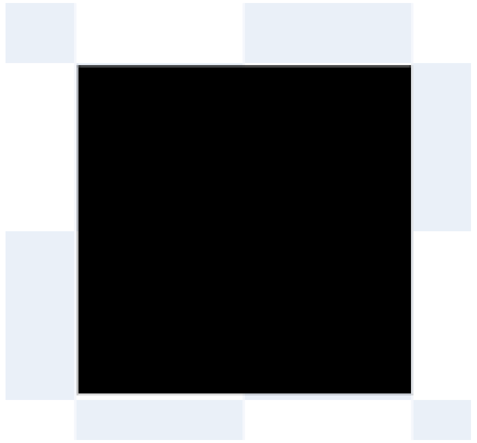
The first costume is called **tile**. We zoom all the way into the costume view and make a black square across 2x2 of the background guide squares.

If we've done it right, the size of the tile costume will be 8x8 (the guide squares are each 4 pixels wide).

The tile costume should be centred using the centre marker in the costume view.







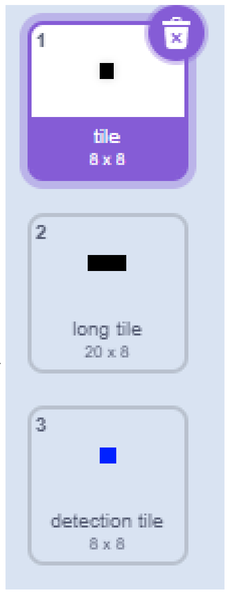
Scratch PacMan

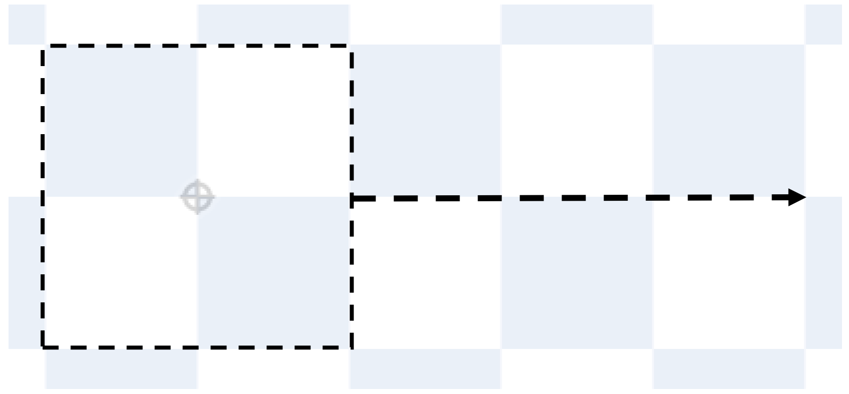
Catford Coding Club

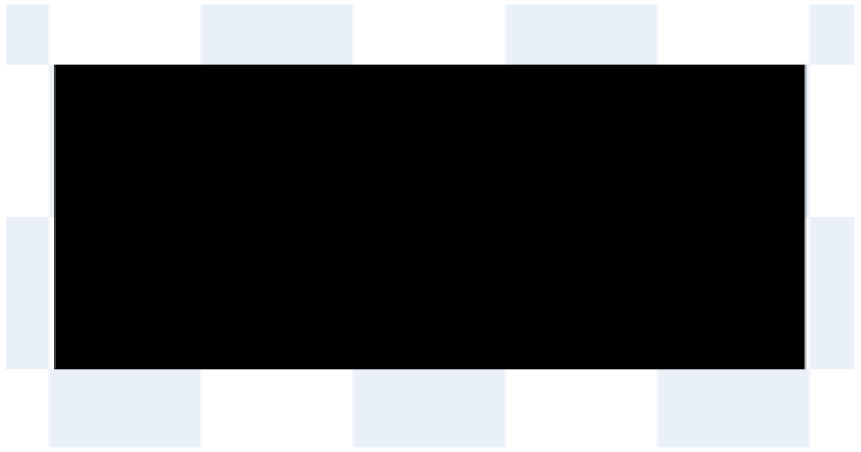
Maze generator sprites

The second costume for the Tile sprite is called **long tile**. We can duplicate the tile costume and stretch the square into a rectangle that covers 5x2 of the background guide squares.

If we've done it right, the size of the tile sprite will be 20x8.







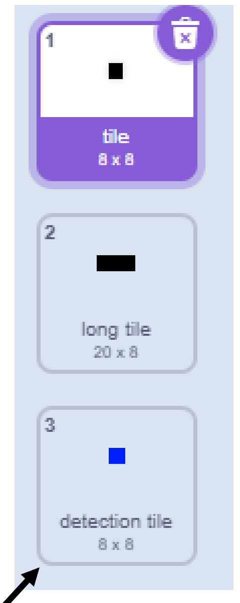
Scratch PacMan

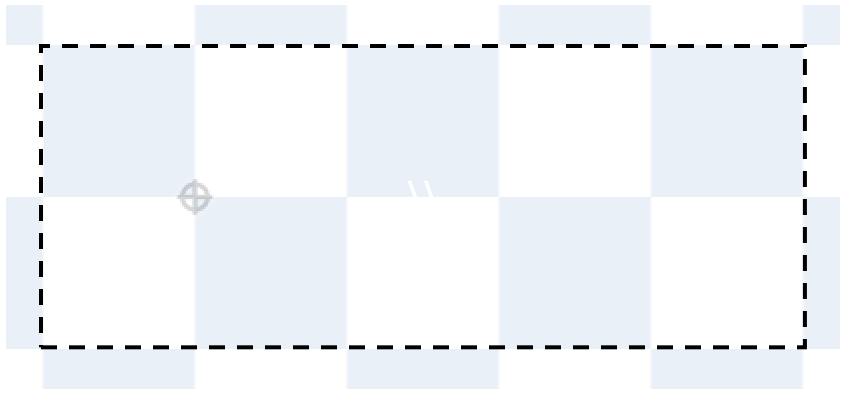
Maze generator sprites

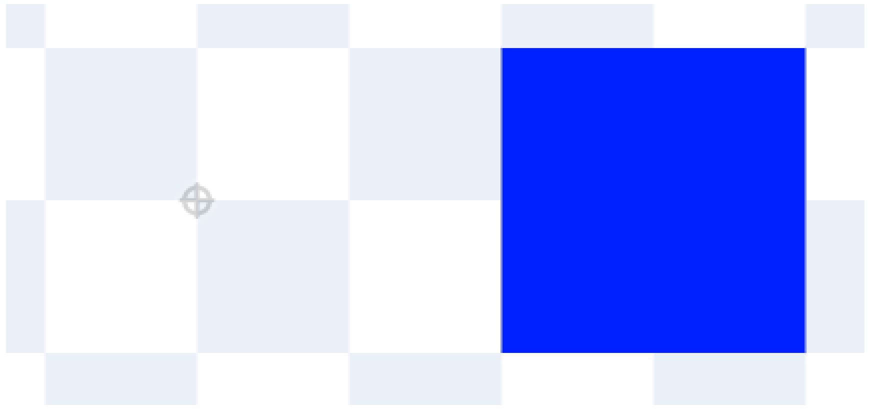
The third costume for the tile sprite is called **detection tile**. We can make this a different colour.

We can duplicate the tile costume and move it to the right so it is in the right 2x2 part of the long tile.

If we've done it right, the size of the tile sprite will be 8x8, the same as the first tile costume.







Scratch PacMan

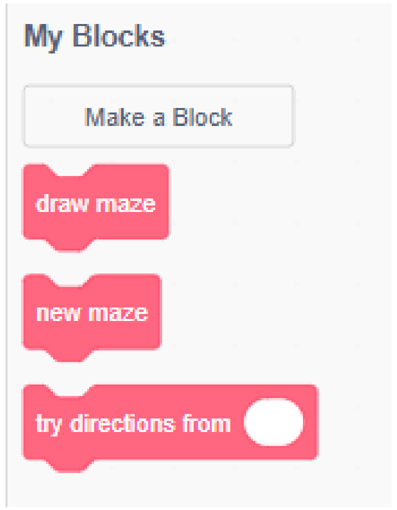
Maze generator code

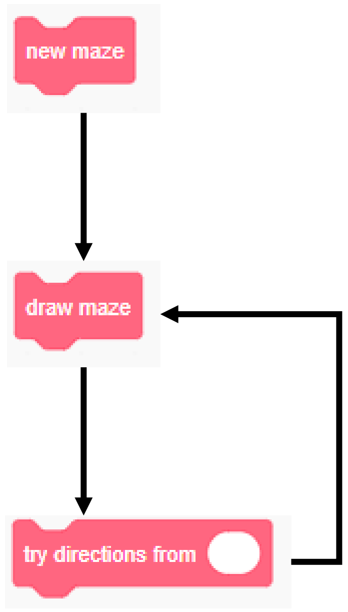
Now we can write the code for the maze generator.

We need to add the Pen extension so we can use the Pen code blocks .

We are going to use code blocks so we can organise the maze generator code.

We will create three code blocks: **try directions from**, **draw maze**, and **new maze**. These blocks will interact to draw a different maze each time!





Scratch PacMan

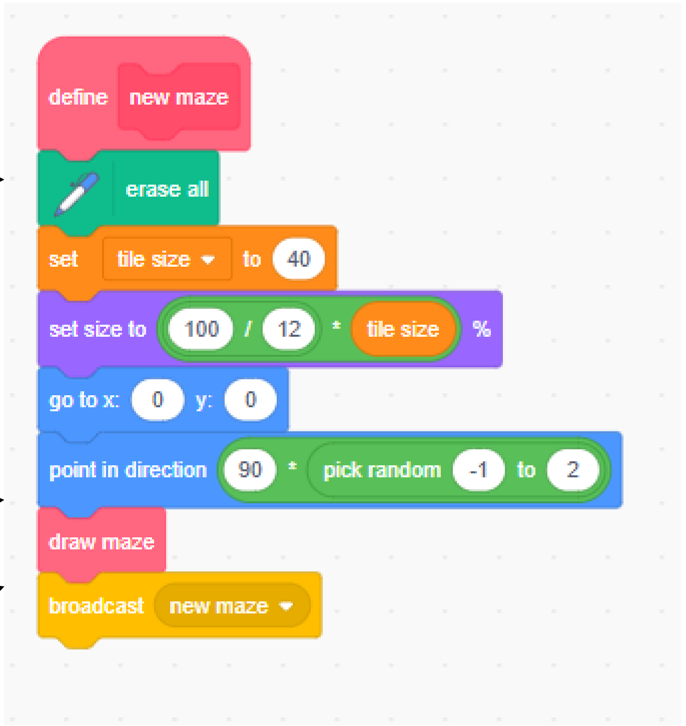
New maze code block

We start by clearing any old mazes using erase all

We create a new variable called **tile size**, then we can set it in the new maze code block. In my game I've used 40 as the tile size. Now we set the size of the tile sprite using the **tile size** variable. This sets the scale of our maze correctly.

Now we choose a random direction to point in. We do this by multiplying 90 degrees by one of -1, 0, +1 or +2 to give us the directions -90. 0, +90 and +180.

Now we call the **draw maze** code block to start drawing the maze. When the maze is finished, we can broadcast a message to tell the sprites that the new maze is ready to play.



```
define new maze
erase all
set tile size to 40
set size to ((100 / 12) * (tile size)) %
go to x: 0 y: 0
point in direction (90 * (pick random -1 to 2))
draw maze
broadcast new maze
```

Scratch PacMan

Draw maze code block

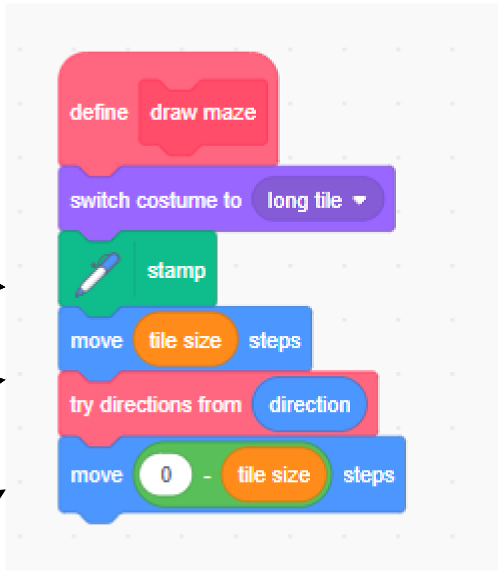
The **draw maze** code block uses the long tile costume.

We use the Pen extension Stamp command to draw a long tile.

We use a variable called **tile size** which is set in the **new maze** code block.

We then move forward by the size of the tile (tile size) and call the try direction code block.

Once the try direction code block has completed, we move back by the size of the tile (0 – tile size)



```
define draw maze
switch costume to (long tile ▾)
stamp
move (tile size) steps
try directions from (direction)
move ((0) - (tile size)) steps
```

Scratch PacMan

Try directions from code block

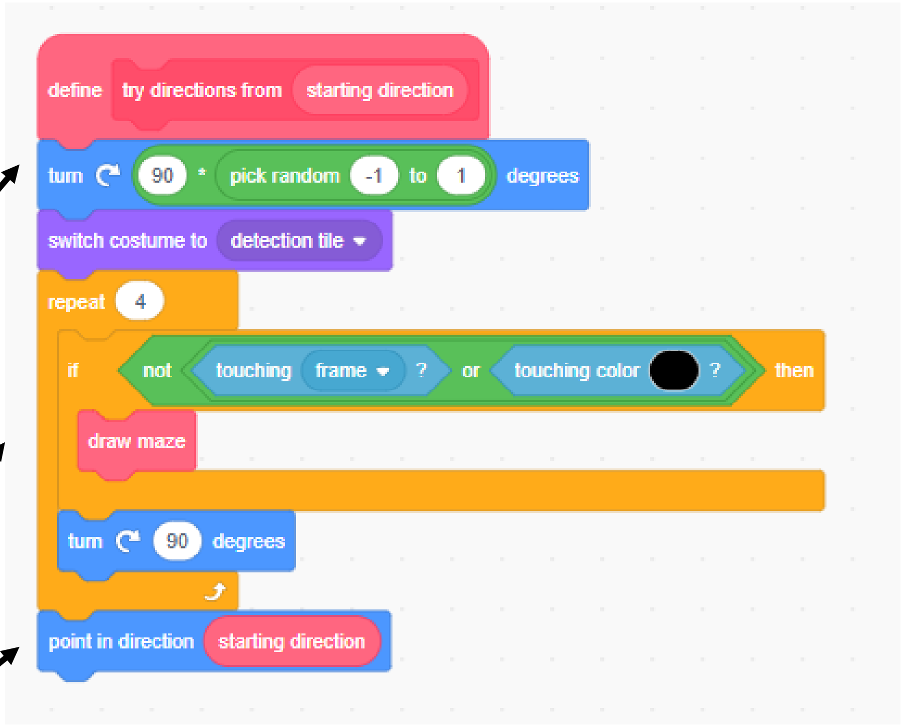
The **try directions from** code block uses the detection tile costume.

From its start direction, we turn either to the right or to the left by 90 degrees.

We then use a repeat block to test the four directions up, down, left and right.

If we find that the detector tile isn't touching the frame or a piece of the maze we have already placed, we call the **draw maze** block then turn 90 degrees to the right.

Finally, we point in the direction we started from.



```
define try directions from (starting direction)
turn ↻ ((90) * (pick random (-1) to (1))) degrees
switch costume to (detection tile ▾)
repeat (4)
    if <not <<touching (frame ▾) ?> or <touching color ● ?>>> then
        draw maze
    end
    turn ↻ (90) degrees
end
point in direction (starting direction)
```

Scratch PacMan

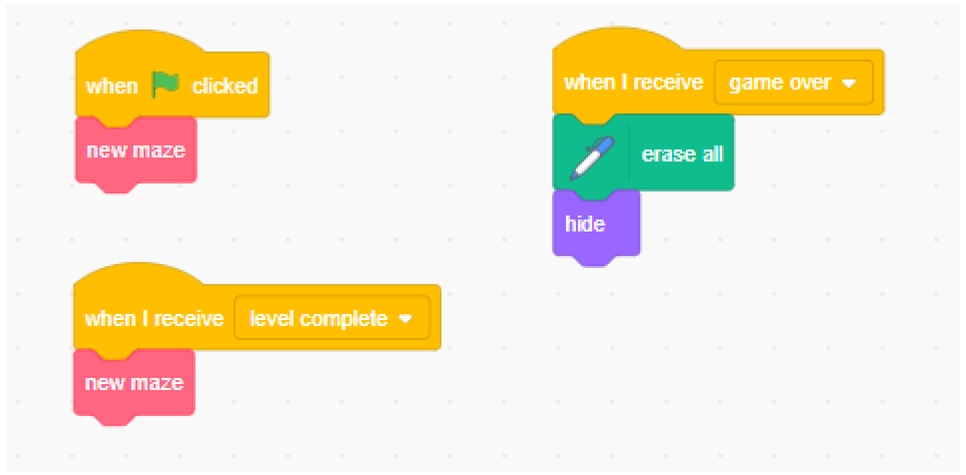
Drawing the maze

We want to draw a maze when one of these things happens:

- At the start of the game
- When a level is complete

When the game is over, we want to delete the maze using erase all.

We need to set up messages for these events that can be broadcast. My messages are **level complete** and **game over.**



```
when ⚑ clicked
new maze

when I receive [level complete ▾]
new maze

when I receive [game over ▾]
erase all
hide
```

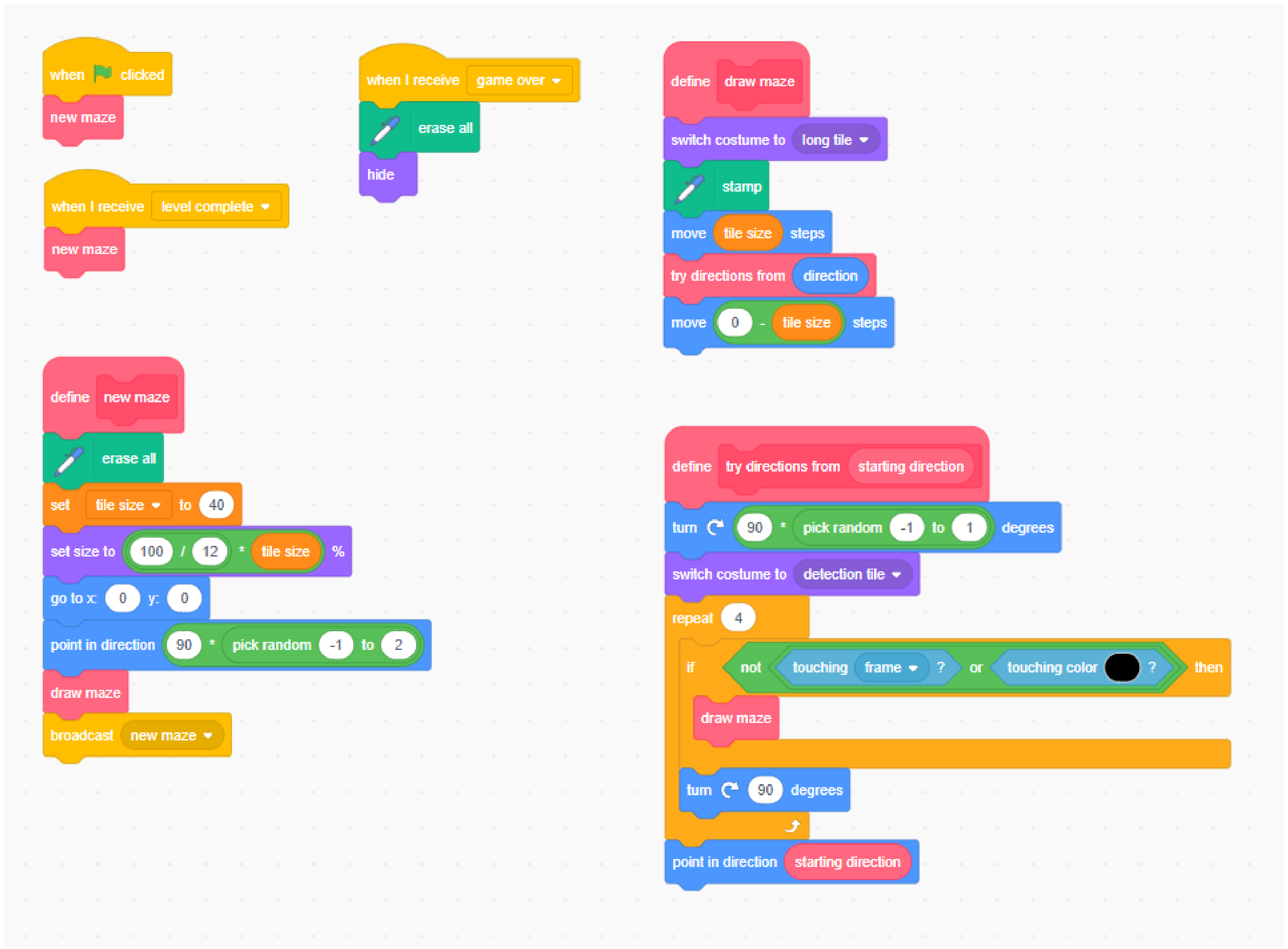
Catford Coding Club

Scratch PacMan

Drawing the maze

The complete code for my maze generator looks like this.

Now we can generate a new maze when we need it.



```
when 🏴 clicked
new maze

when I receive [level complete ▾]
new maze

when I receive [game over ▾]
erase all
hide

define new maze
erase all
set [tile size ▾] to 40
set size to ((100 / 12) * (tile size)) %
go to x: 0 y: 0
point in direction (90 * (pick random -1 to 2))
draw maze
broadcast [new maze ▾]

define draw maze
switch costume to [long tile ▾]
stamp
move (tile size) steps
try directions from (direction)
move (0 - (tile size)) steps

define try directions from (starting direction)
turn ↻ (90 * (pick random -1 to 1)) degrees
switch costume to [detection tile ▾]
repeat 4
    if <not <<touching [frame ▾] ?> or <touching color ● ?>>> then
        draw maze
    end
    turn ↻ 90 degrees
end
point in direction (starting direction)
```

Catford Coding Club

Scratch PacMan

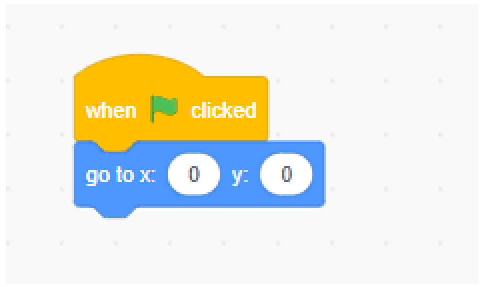
Coding sprites

Scratch PacMan

Catford Coding Club

Coding sprites: Frame

First let’s place our Frame sprite in the middle of the Stage.



```
when 🏴 clicked
go to x: 0 y: 0
```

Scratch PacMan

Coding sprites: Food

Now let's add food to the maze for PacMan to eat.

The code starts when a new maze is created and we receive the **new maze** broadcast.

We are going to place clones of the food sprite in the maze, checking that we are only placing food in a path and not on walls. We'll count how much food we place using a variable **food** so we know when PacMan has eaten all the food for the level!

Because we know the size of our Stage and our maze, we know which rows (y) and column (x) positions where we can try to place food. If we change the size of the maze, we would have to adjust the x and y values we'll use to place food.

Scratch PacMan

Catford Coding Club

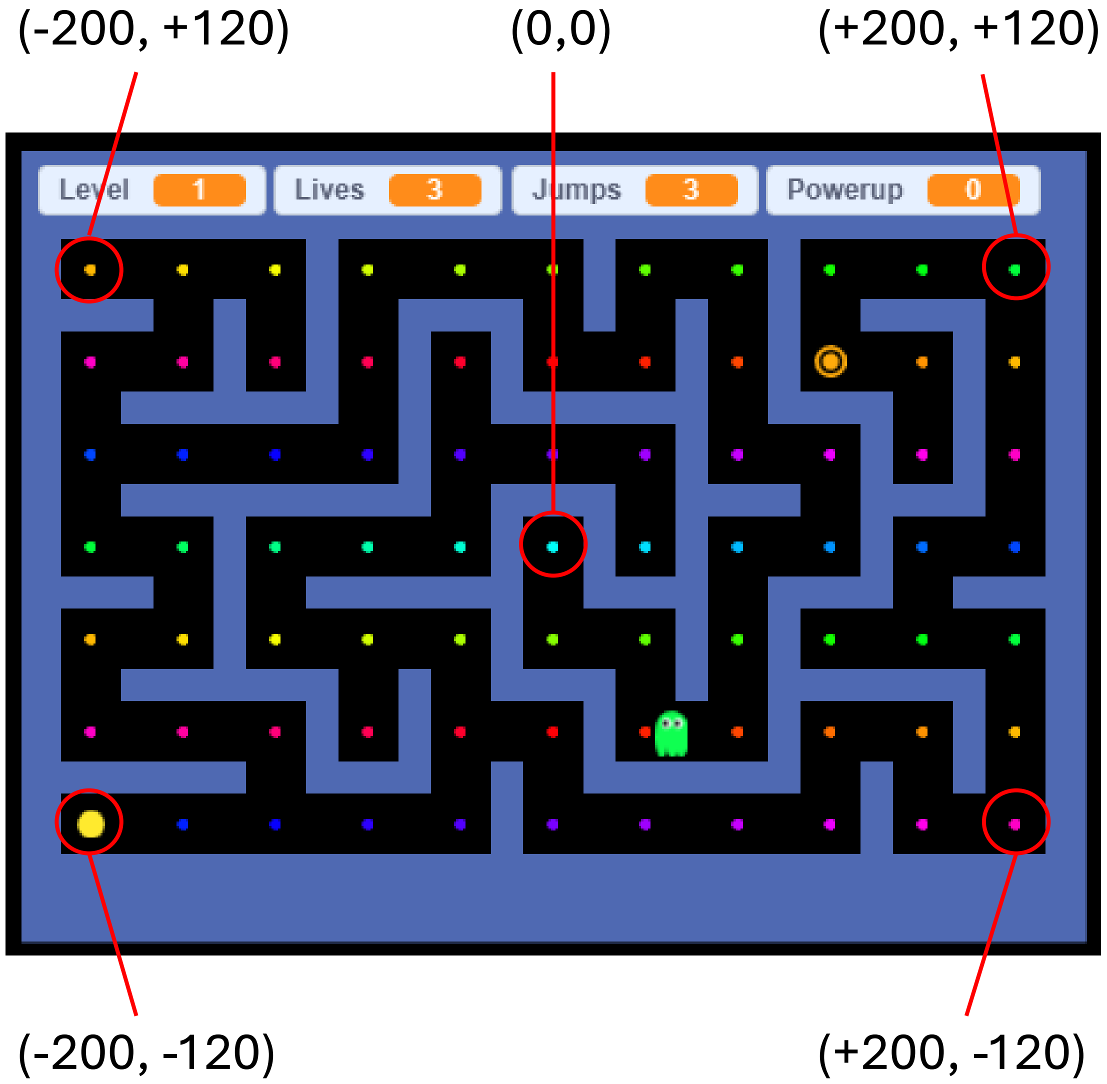
Coding sprites: Food

Remember in the Scratch Stage, (0,0) is in the middle.

The values for y are {-120, -80, -40, 0, +40, +80, +120}.

The values for x are {-200, -160, -120, -80, -40, 0, +40, +80, +120, +160, +200}

We'll use repeat loops to work through the maze for the values of x and y, placing clones of the food sprite.



Catford Coding Club

Scratch PacMan

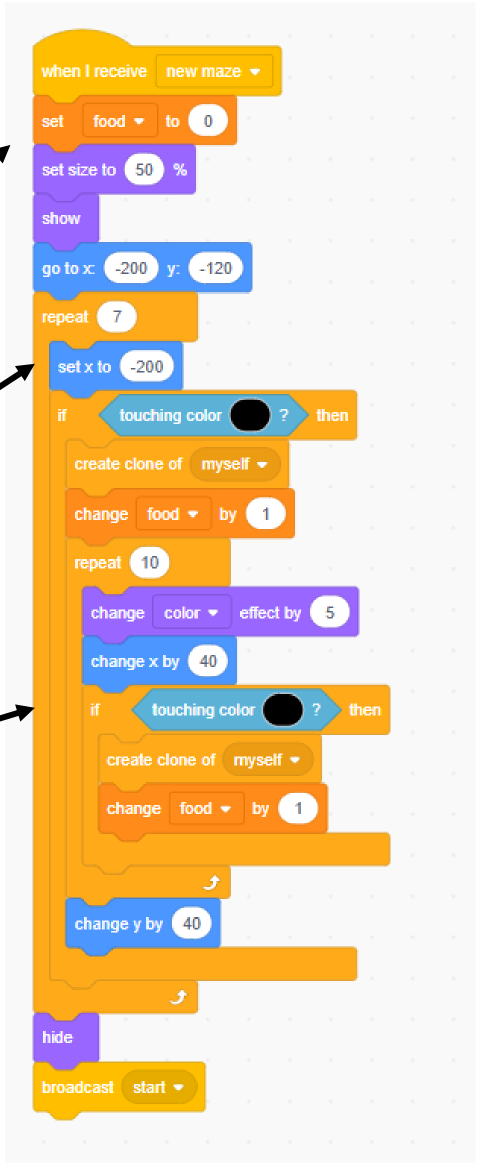
Coding sprites: Food

When a new maze is generated, we set the **food** variable to 0 and set the size of our food sprite to fit nicely in the maze. We show the sprite and go to the first (x,y) coordinate which is (-200,-120).

Now we set up two repeat loops, one to move in the x direction (across) and one to move in the y direction (up). We will move by the space between points which for us is 40. These are called nested loops as one is inside the other. We'll place all the food for a row then move up to the next row until we have covered the whole maze.

When we move to a new position, we check if we are in a maze path by sensing for the black colour. If we are, we place food by creating a clone and adding 1 to the food variable.

When all the food is placed we hide the food sprite (leaving just the clones visible) and broadcast a **start** message so the other sprites know it's time to play!



```
when I receive new maze
set food to 0
set size to 50 %
show
go to x: -200 y: -120
repeat 7
    set x to -200
    if touching color ● ? then
        create clone of myself
        change food by 1
        repeat 10
            change color effect by 5
            change x by 40
            if touching color ● ? then
                create clone of myself
                change food by 1
        change y by 40
hide
broadcast start
```

Scratch PacMan

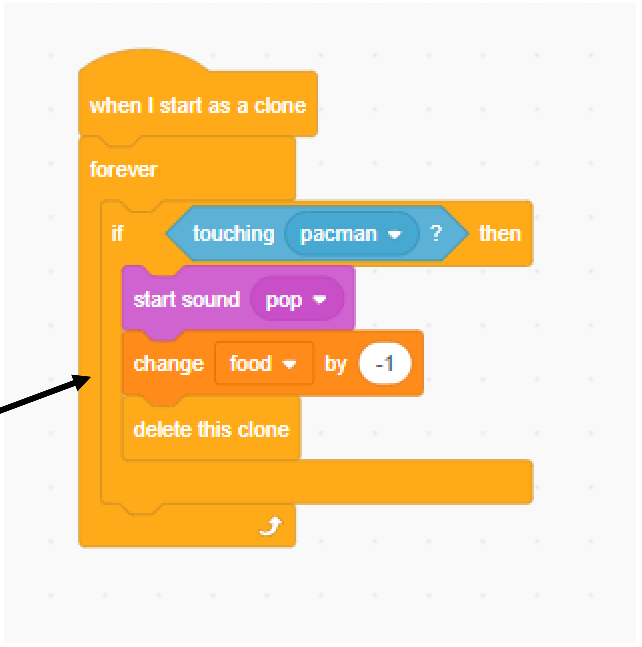
Catford Coding Club

Coding sprites: Food

In our nested loop, when we land on a maze path we create a clone of the food sprite.

Once started as a clone, each piece of food waits until PacMan eats it. When it is eaten it makes a sound (choose one!) and change the value of the **food** variable by -1. Once eaten by PacMan, the clone is deleted so it disappears.

The **food** variable ticks down as the food is eaten and reaches 0 when all the food has been eaten. We'll use this to decide when the level is complete.



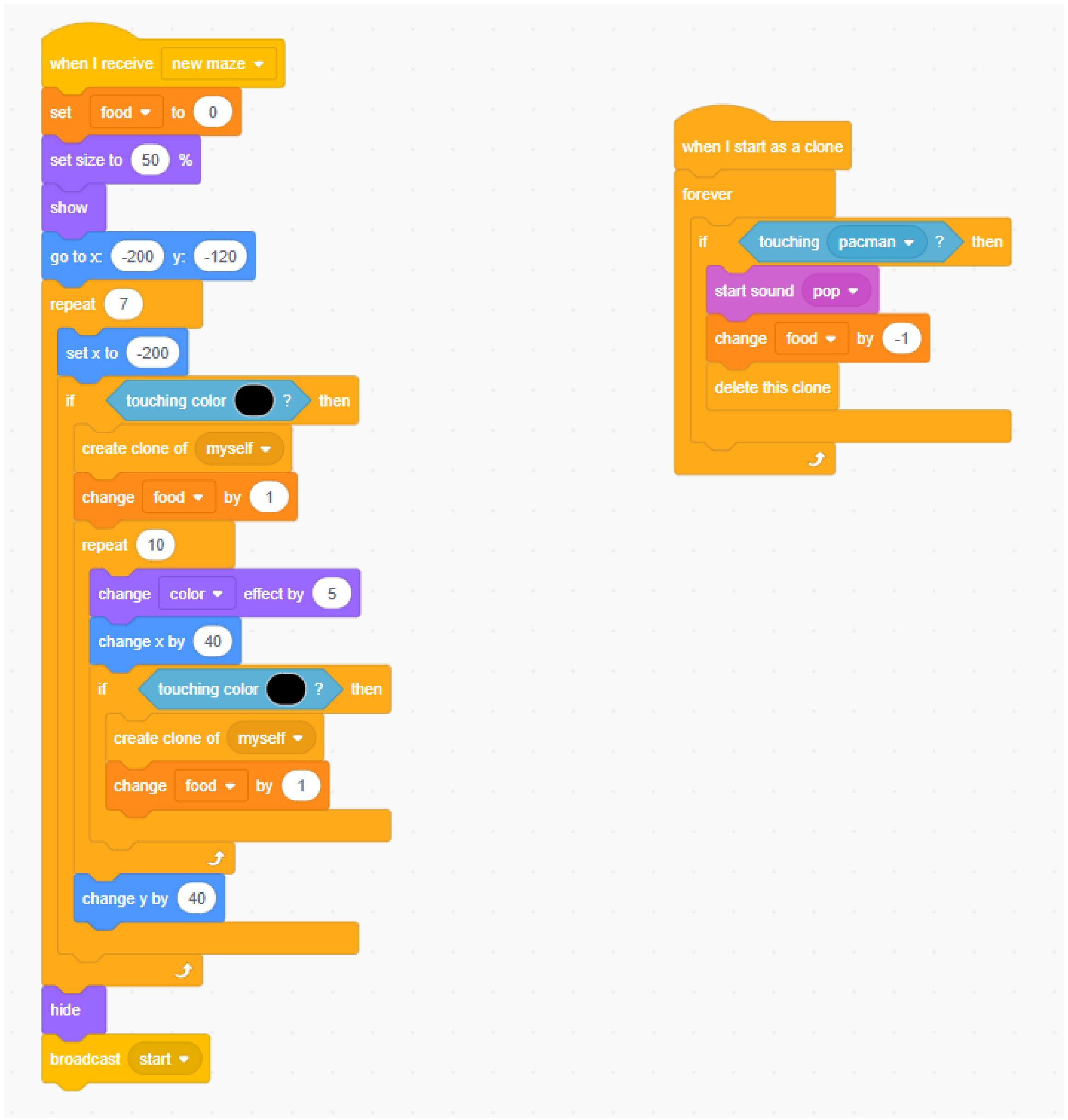
```
when I start as a clone
forever
    if <touching (pacman ▾) ?> then
        start sound (pop ▾)
        change [food ▾] by (-1)
        delete this clone
```

Catford Coding Club

Scratch PacMan

Coding sprites: Food

The complete code for my Food sprite looks like this.



```
when I receive [new maze ▾]
set [food ▾] to (0)
set size to (50) %
show
go to x: (-200) y: (-120)
repeat (7)
    set x to (-200)
    if <touching color [●] ?> then
        create clone of [myself ▾]
        change [food ▾] by (1)
        repeat (10)
            change [color ▾] effect by (5)
            change x by (40)
            if <touching color [●] ?> then
                create clone of [myself ▾]
                change [food ▾] by (1)
            end
        end
        change y by (40)
    end
end
hide
broadcast [start ▾]
```

```
when I start as a clone
forever
    if <touching [pacman ▾] ?> then
        start sound [pop ▾]
        change [food ▾] by (-1)
        delete this clone
    end
end
```

Catford Coding Club

Scratch PacMan

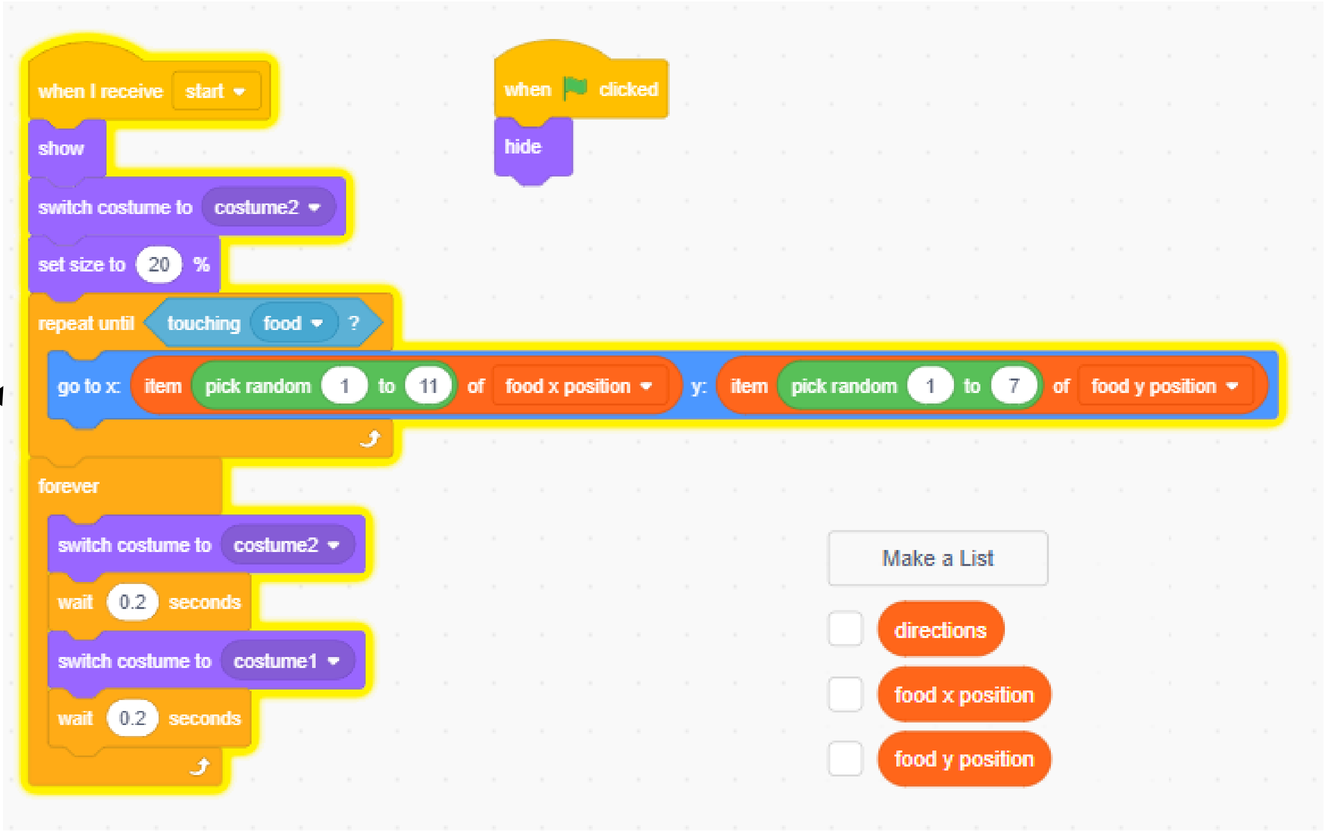
Coding sprites: Powerup

Now we've placed all the food, we can place the Powerup sprite for PacMan to find.

I created two lists, one with the **food x positions** and one with the **food y positions**. In fact, we could have used these lists to place the food earlier instead of changing x and y in our repeat loops.

Now we can add code to select one (x,y) position at random. We make sure it's a valid position by checking if there is food there.

Once placed we make the Powerup sprite flash between its costumes.



```
when I receive [start ▾]
show
switch costume to [costume2 ▾]
set size to (20) %
repeat until <touching [food ▾] ?>
    go to x: (item (pick random (1) to (11)) of [food x position ▾]) y: (item (pick random (1) to (7)) of [food y position ▾])
end
forever
    switch costume to [costume2 ▾]
    wait (0.2) seconds
    switch costume to [costume1 ▾]
    wait (0.2) seconds
end
```

```
when 🏴 clicked
hide
```

The values for **food y positions** are {-120, -80, -40, 0, +40, +80, +120}.

The values for **food x positions** are {-200, -160, -120, -80, -40, 0, +40, +80, +120, +160, +200}

Catford Coding Club

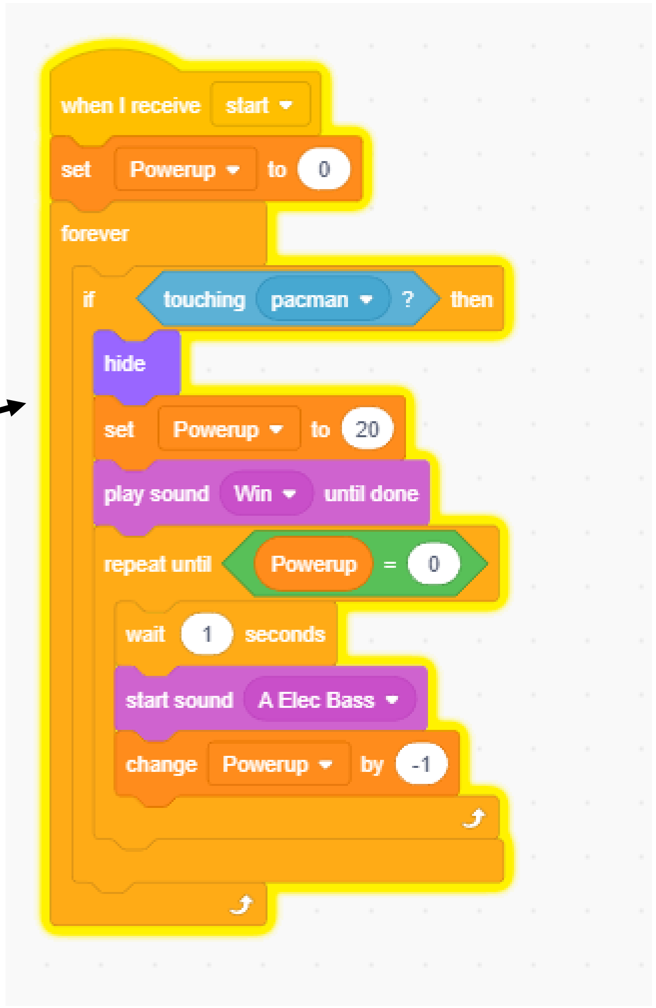
Scratch PacMan

Coding sprites: Powerup

When the game starts, we set a **Powerup** variable to 0 to show that PacMan doesn't have the Powerup.

If PacMan touches the Powerup sprite, the **Powerup** variable is set to 20. Powerup last for 20 seconds, with the variable counting down by one every second using the Wait instruction.

We can check if Pacman has powerup by testing the **Powerup** variable. If it's more than 0 he does, if its 0 then he doesn't!



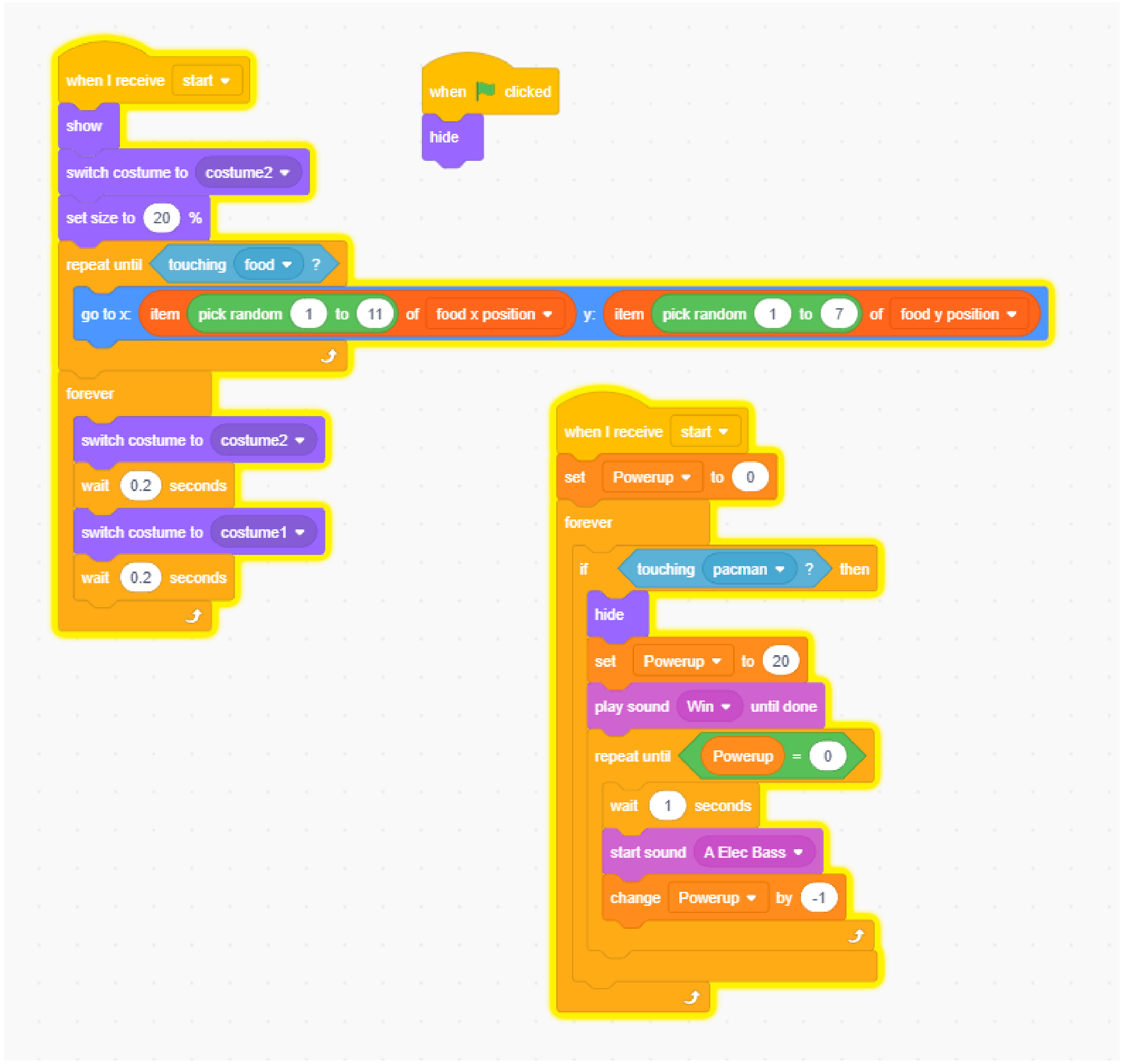
```
when I receive start
set Powerup to 0
forever
    if touching pacman ? then
        hide
        set Powerup to 20
        play sound Win until done
        repeat until Powerup = 0
            wait 1 seconds
            start sound A Elec Bass
            change Powerup by -1
```

Catford Coding Club

Scratch PacMan

Coding sprites: Powerup

The complete code for my Powerup sprite looks like this.



```
when I receive [start ▾]
show
switch costume to [costume2 ▾]
set size to (20) %
repeat until <touching [food ▾] ?>
    go to x: (item (pick random (1) to (11)) of [food x position ▾]) y: (item (pick random (1) to (7)) of [food y position ▾])
end
forever
    switch costume to [costume2 ▾]
    wait (0.2) seconds
    switch costume to [costume1 ▾]
    wait (0.2) seconds
end
```

```
when 🏁 clicked
hide
```

```
when I receive [start ▾]
set [Powerup ▾] to (0)
forever
    if <touching [pacman ▾] ?> then
        hide
        set [Powerup ▾] to (20)
        play sound [Win ▾] until done
        repeat until <(Powerup) = (0)>
            wait (1) seconds
            start sound [A Elec Bass ▾]
            change [Powerup ▾] by (-1)
        end
    end
end
```

Scratch PacMan

Coding sprites: PacMan

Now for our main sprite!

We need to add code to PacMan so that:

- PacMan changes costume to look like he's eating
- We can move PacMan around the maze
- PacMan loses a life when he touches a ghost

Catford Coding Club

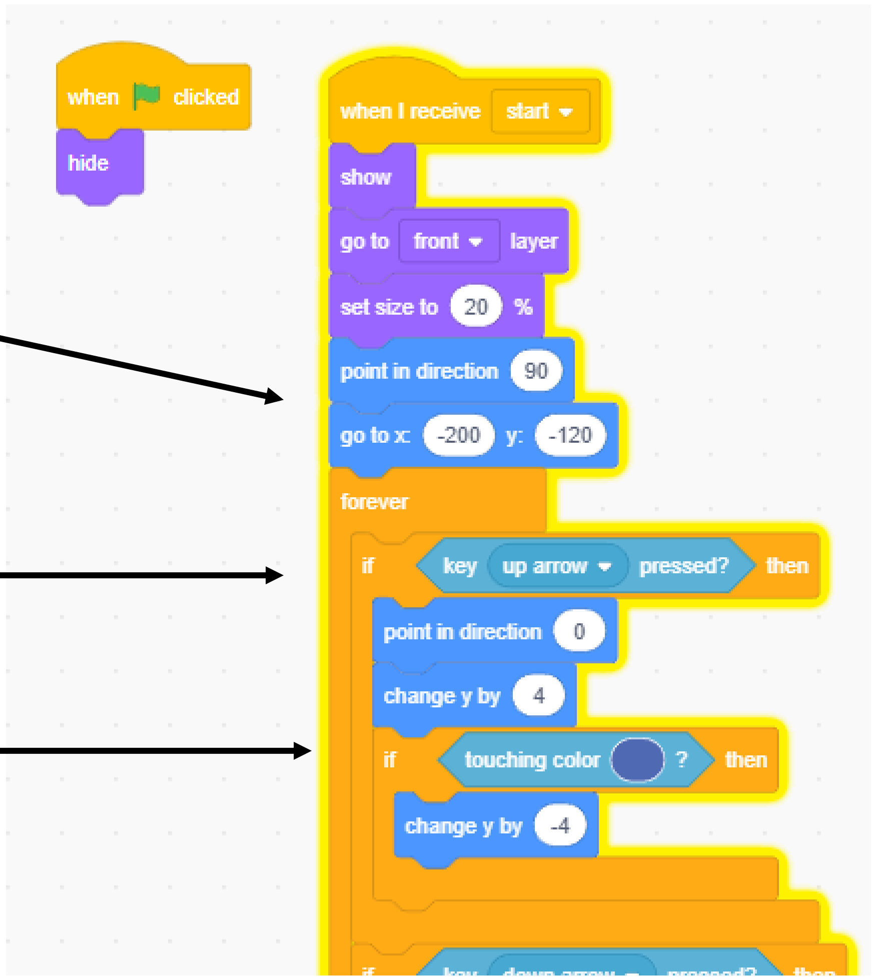
Scratch PacMan

Coding sprites: PacMan

When a new level is ready, PacMan receives the **start** broadcast message. We put him in a starting position and point to the right, making sure he's on the front layer.

We'll use a forever loop and the arrow keys to move PacMan up, down, left and right.

If PacMan hits a maze wall, he jumps back until the player tries another direction. We test for a wall using the sensing block and the maze wall blue colour.



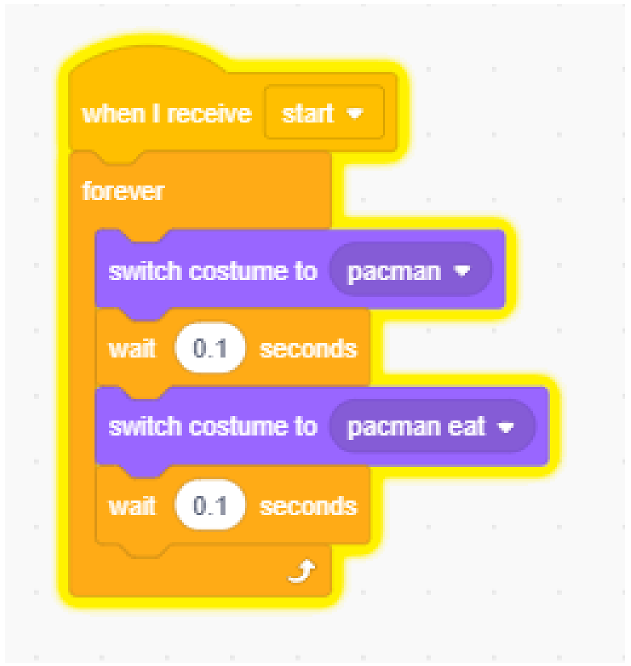
```
when 🏁 clicked
hide

when I receive start ▾
show
go to front ▾ layer
set size to 20 %
point in direction 90
go to x: -200 y: -120
forever
    if key up arrow ▾ pressed? then
        point in direction 0
        change y by 4
        if touching color ● ? then
            change y by -4
    if key down arrow ▾ pressed? then
```

Repeat for down, left and right...

Scratch PacMan

Coding sprites: PacMan

Once the game starts, PacMan switches between the two costumes we created in a forever loop.



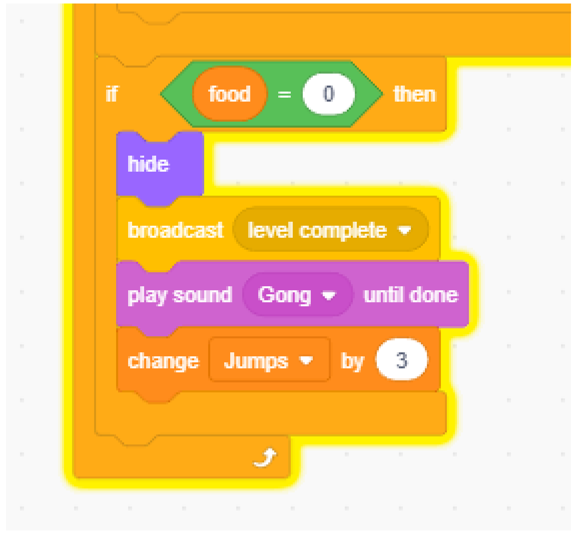
```
when I receive start
forever
    switch costume to pacman
    wait 0.1 seconds
    switch costume to pacman eat
    wait 0.1 seconds
```

Scratch PacMan

Coding sprites: PacMan

We want PacMan to eat all the food to complete the level. In our forever loop we continually check the **food** variable which is reduced by 1 every time PacMan eats a piece of food.

When no food is left, the **food** variable equals 0 and we can broadcast a message to say that the level is complete.



```
if <(food) = (0)> then
    hide
    broadcast [level complete v]
    play sound [Gong v] until done
    change [Jumps v] by (3)
```

Scratch PacMan

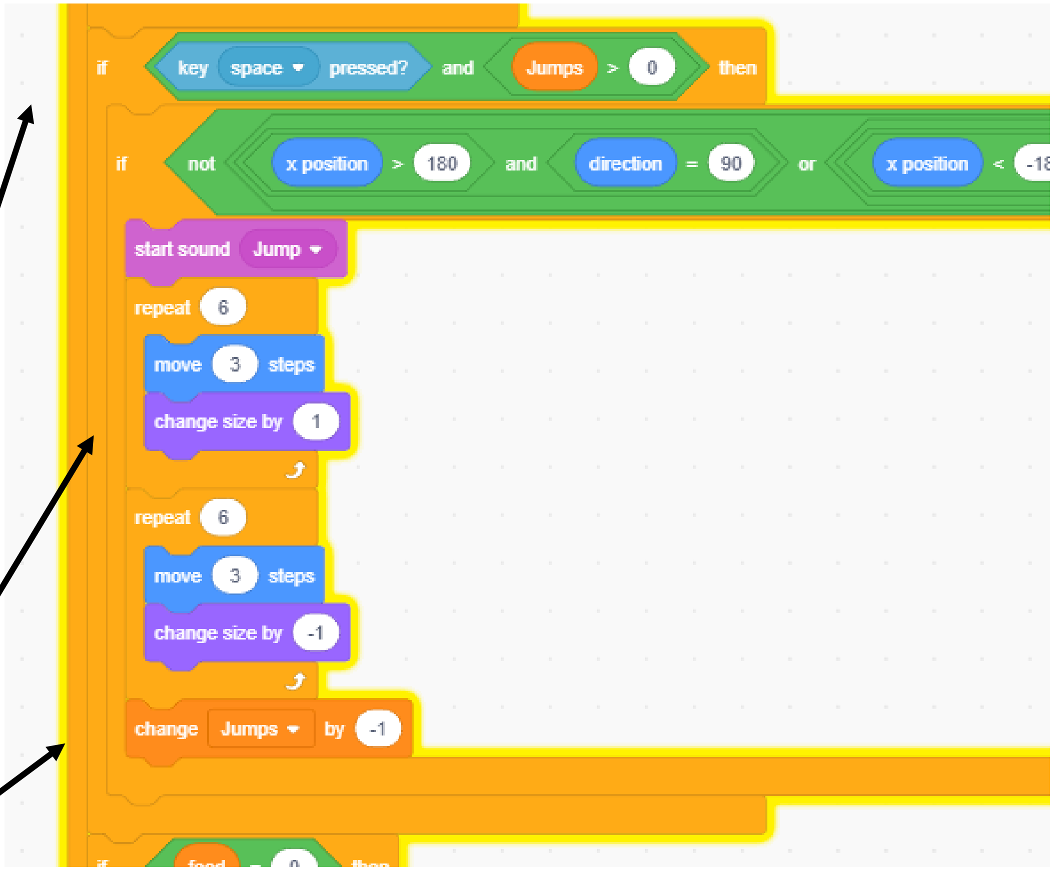
Coding sprites: PacMan

Because of the way we have built our maze, we give PacMan the ability to jump over walls to help avoid ghosts.

PacMan jumps if we press the space key. We use a new variable called **jump** to keep track of how many jumps PacMan has. If PacMan runs out of jumps, nothing happens when the player hits space!

We use a repeat loop to make the jump, and change the size of pacman as he jumps up and then down from the wall.

Each time PacMan jumps, the **jump** variable goes down by 1 to keep track.

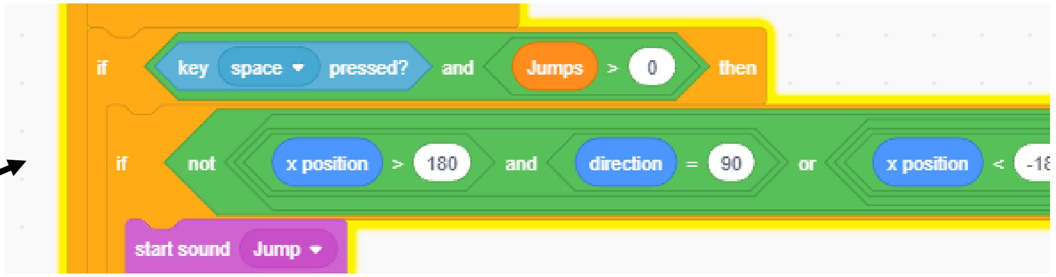


```
if key space pressed? and Jumps > 0 then
  if not x position > 180 and direction = 90 or x position < -18
    start sound Jump
    repeat 6
      move 3 steps
      change size by 1
    repeat 6
      move 3 steps
      change size by -1
    change Jumps by -1
if food = 0 then
```

Catford Coding Club

Scratch PacMan

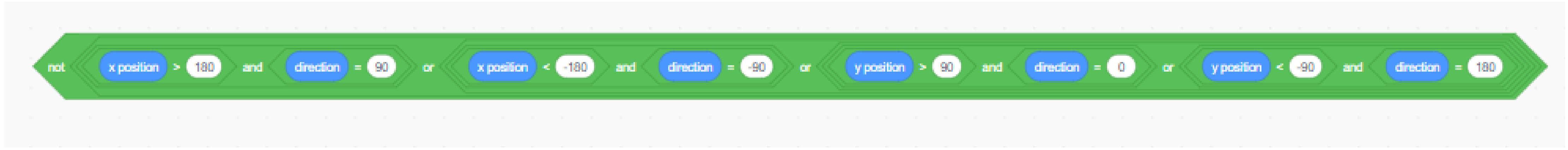
Coding sprites: PacMan



```
if <<key (space) pressed?> and <(Jumps) > (0)>> then
  if <not <<<(x position) > (180)> and <(direction) = (90)>> or <<(x position) < (-18
    start sound (Jump)
```

You might have noticed an if statement in the jump code. This is needed to make sure Pacman doesn't jump out of the maze.

The whole test for the if statement looks like this:



```
not <<<<(x position) > (180)> and <(direction) = (90)>> or <<(x position) < (-180)> and <(direction) = (-90)>>> or <<<(y position) > (90)> and <(direction) = (0)>> or <<(y position) < (-90)> and <(direction) = (180)>>>>
```

We use not, and and or operators and the **x position**, **y position** and **direction** of the PacMan sprite to make the test. So, for example, if we at the top of the maze (**y position** > 90) and pointing up (**direction** = 0), a jump is not allowed.

Scratch PacMan

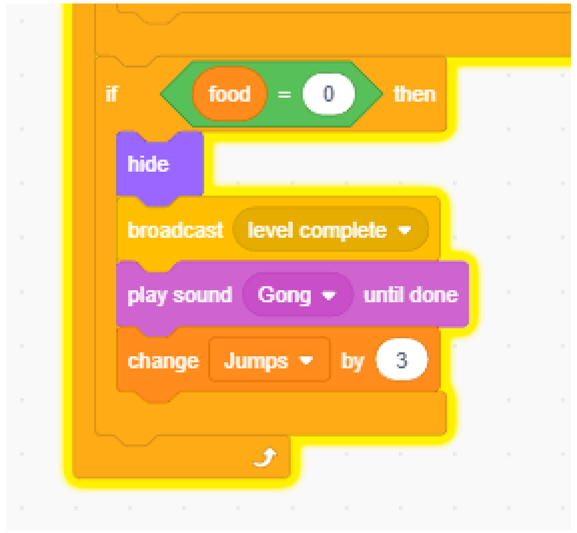
Catford Coding Club

Coding sprites: PacMan

We want PacMan to eat all the food to complete the level. We continually check the **food** variable which is reduced by 1 every time PacMan eats a piece of food.

When no food is left, the **food** variable equals 0 and we can broadcast a message to say that the level is complete.

When the level is complete we give PacMan some new jumps for the next level.



```
if <(food) = (0)> then
    hide
    broadcast [level complete v]
    play sound [Gong v] until done
    change [Jumps v] by (3)
```

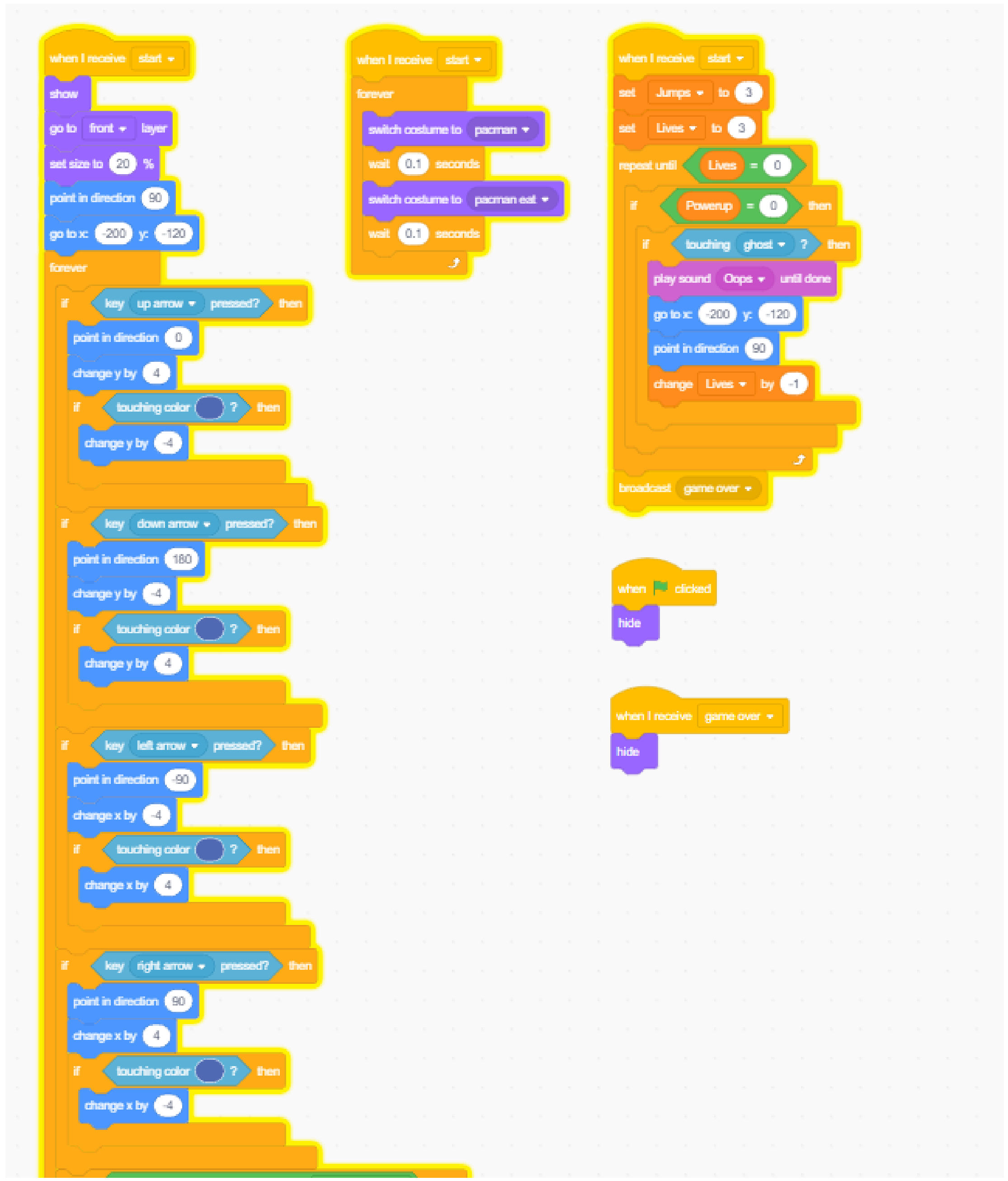
Repeat for down, left and right…

Scratch PacMan

Coding sprites: PacMan

The complete code for my PacMan sprite looks like this.

Main forever loop continues on following page...



```
when I receive [start ▾]
show
go to [front ▾] layer
set size to (20) %
point in direction (90)
go to x: (-200) y: (-120)
forever
    if <key [up arrow ▾] pressed?> then
        point in direction (0)
        change y by (4)
        if <touching color ( )?> then
            change y by (-4)
    if <key [down arrow ▾] pressed?> then
        point in direction (180)
        change y by (-4)
        if <touching color ( )?> then
            change y by (4)
    if <key [left arrow ▾] pressed?> then
        point in direction (-90)
        change x by (-4)
        if <touching color ( )?> then
            change x by (4)
    if <key [right arrow ▾] pressed?> then
        point in direction (90)
        change x by (4)
        if <touching color ( )?> then
            change x by (-4)
```

```
when I receive [start ▾]
forever
    switch costume to [pacman ▾]
    wait (0.1) seconds
    switch costume to [pacman eat ▾]
    wait (0.1) seconds
```

```
when I receive [start ▾]
set [Jumps ▾] to (3)
set [Lives ▾] to (3)
repeat until <(Lives) = (0)>
    if <(Powerup) = (0)> then
        if <touching [ghost ▾] ?> then
            play sound [Oops ▾] until done
            go to x: (-200) y: (-120)
            point in direction (90)
            change [Lives ▾] by (-1)
broadcast [game over ▾]
```

```
when 🏴 clicked
hide
```

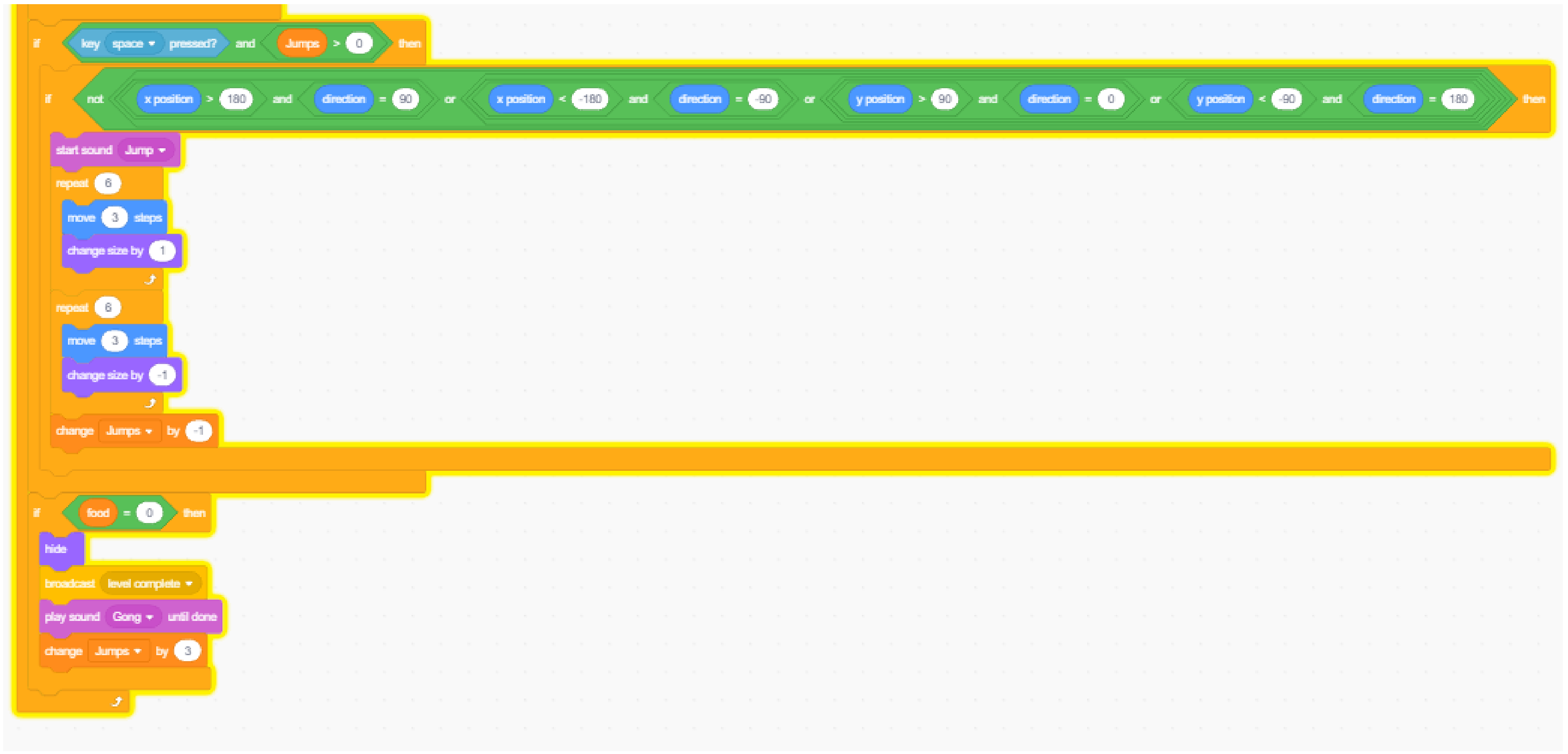
```
when I receive [game over ▾]
hide
```

Scratch PacMan

Coding sprites: PacMan

The complete code for my PacMan sprite looks like this.

Main forever loop continues from previous page



```
if <<key (space) pressed?> and <(Jumps) > (0)>> then
    if <not <<<(x position) > (180)> and <(direction) = (90)>> or <<(x position) < (-180)> and <(direction) = (-90)>> or <<(y position) > (90)> and <(direction) = (0)>> or <<(y position) < (-90)> and <(direction) = (180)>>>> then
        start sound (Jump)
        repeat (6)
            move (3) steps
            change size by (1)
        end
        repeat (6)
            move (3) steps
            change size by (-1)
        end
        change [Jumps] by (-1)
    end
end
if <(food) = (0)> then
    hide
    broadcast (level complete)
    play sound (Gong) until done
    change [Jumps] by (3)
end
```

Scratch PacMan

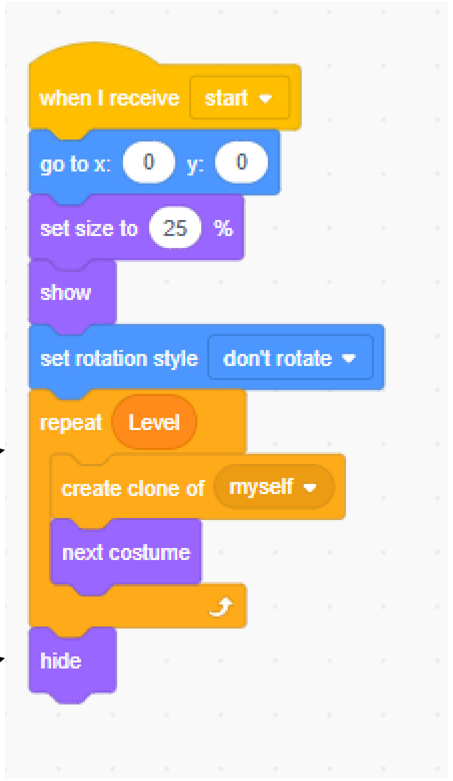
Coding sprites: Ghosts

Our ghosts will start in the centre of the maze at (0,0) when we receive the broadcast message start.

The first level will have one ghost, and we'll add more ghosts as we move into higher levels.

We'll create a new variable to keep track of the level, and use it to create as many ghost clones that we need using a repeat loop. Each ghost clone will use a different costume, so we'll need to add more ghosts if we want more levels.

Finally we hide the ghost sprite, leaving just the correct number of clones.



```
when I receive [start ▾]
go to x: (0) y: (0)
set size to (25) %
show
set rotation style [don't rotate ▾]
repeat (Level)
    create clone of [myself ▾]
    next costume
end
hide
```

Catford Coding Club

Scratch PacMan

Coding sprites: Ghosts

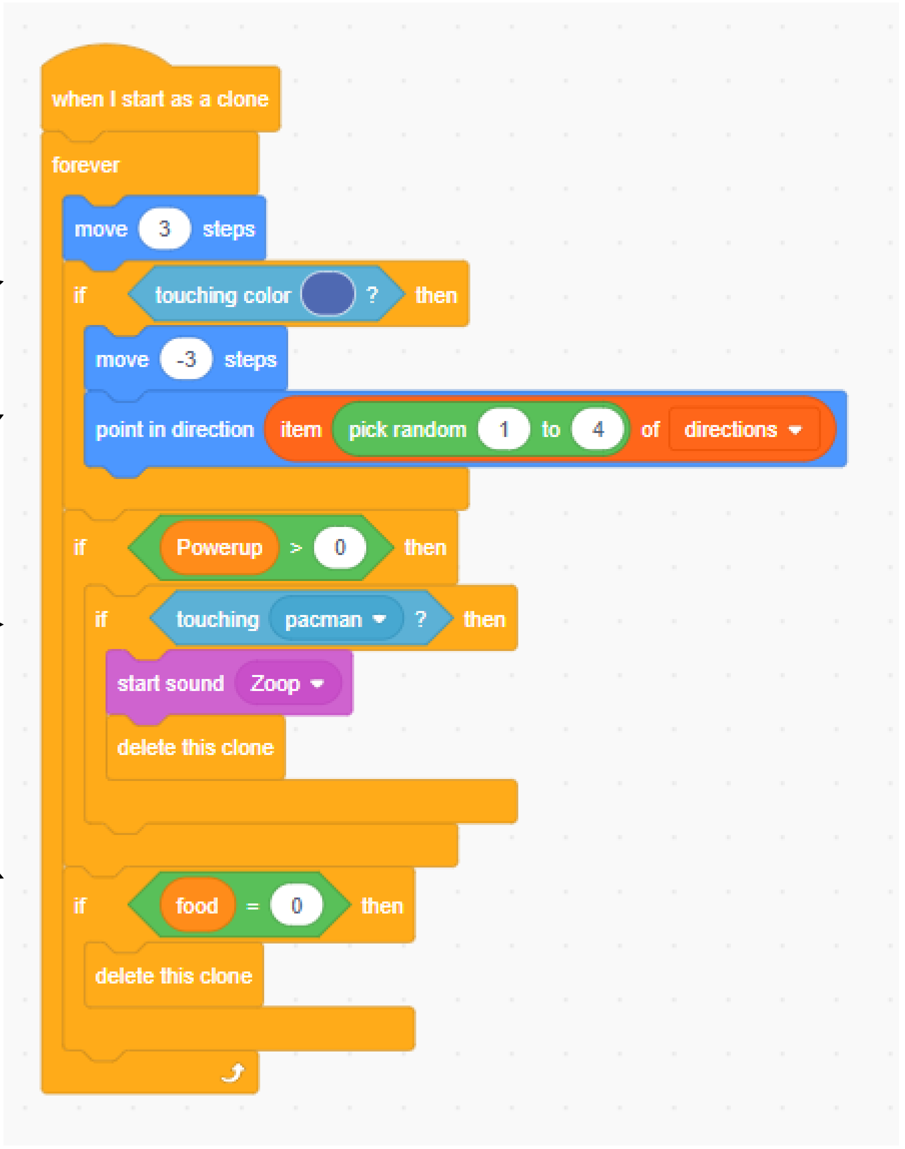
We'll use a forever loop to make the ghosts will move. They'll move in one direction until they hit a maze wall. We use sensing to check if they touch the maze wall blue colour.

If they hit a wall they move back and choose a new direction at random from a list of the available directions. We need a list called **direction** which has the values {0, +90, +180 and –0} to select a direction from.

Usually if a ghost catches PacMan, PacMan loses a life. If PacMan touches a ghost, and only if PacMan has the Powerup, we delete the clone and the ghost disappears

When PacMan has eaten all the food for a level, all the ghosts disappear.

Our ghosts aren't very intelligent. Maybe you can think of ways to make them smarter?



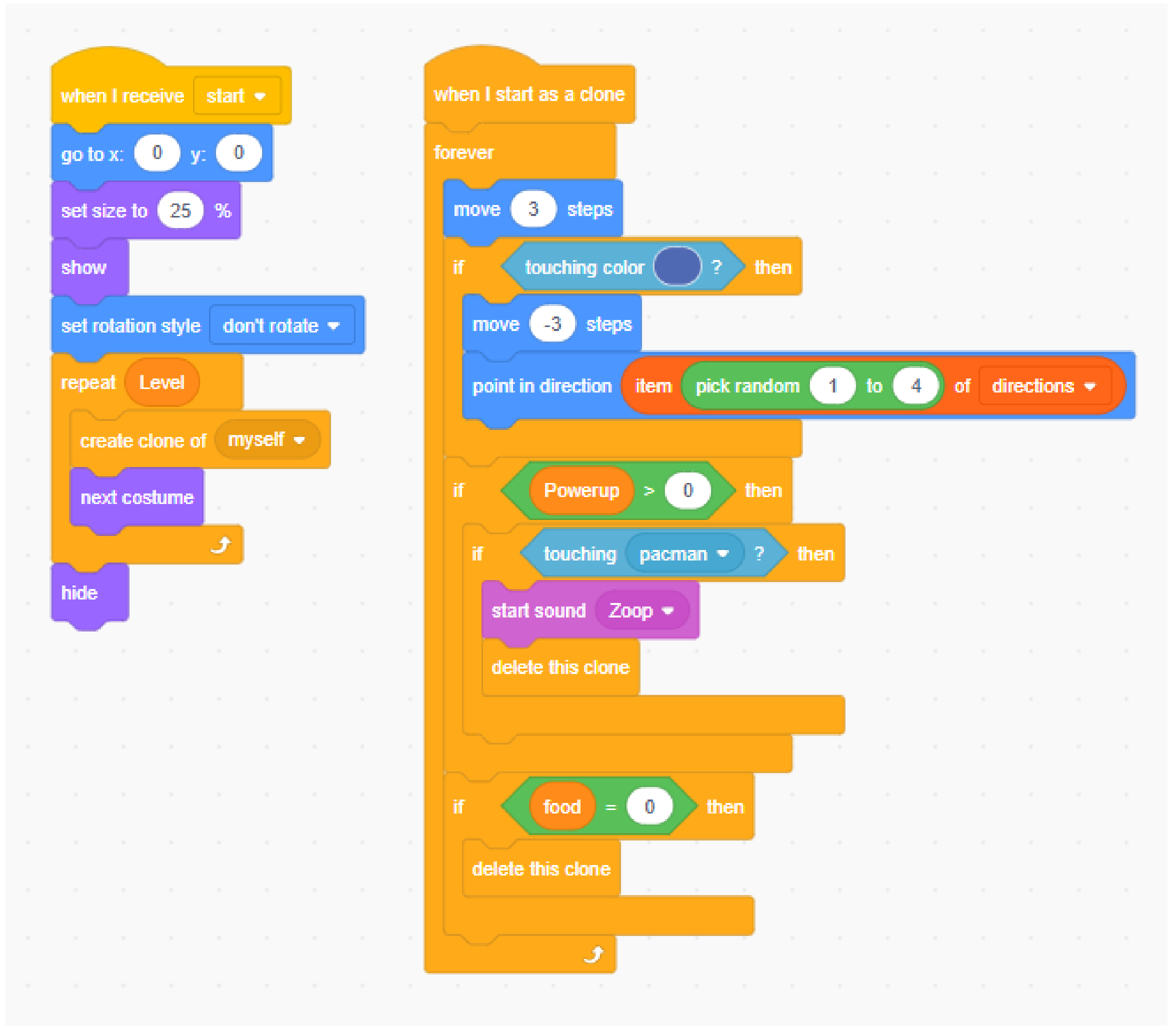
```
when I start as a clone
forever
    move (3) steps
    if <touching color (●) ?> then
        move (-3) steps
        point in direction (item (pick random (1) to (4)) of [directions ▾])
    if <(Powerup) > (0)> then
        if <touching [pacman ▾] ?> then
            start sound [Zoop ▾]
            delete this clone
    if <(food) = (0)> then
        delete this clone
```

Catford Coding Club

Scratch PacMan

Coding sprites: Ghosts

The complete code for my ghost sprites looks like this.



```
when I receive [start ▾]
go to x: (0) y: (0)
set size to (25) %
show
set rotation style [don't rotate ▾]
repeat (Level)
    create clone of [myself ▾]
    next costume
end
hide
```

```
when I start as a clone
forever
    move (3) steps
    if <touching color (●) ?> then
        move (-3) steps
        point in direction (item (pick random (1) to (4)) of [directions ▾])
    end
    if <(Powerup) > (0)> then
        if <touching [pacman ▾] ?> then
            start sound [Zoop ▾]
            delete this clone
        end
    end
    if <(food) = (0)> then
        delete this clone
    end
end
```

Scratch PacMan

Game Features

Scratch PacMan

Game Features

Most if the game should now be complete and working. We can add some more features to make it even better.

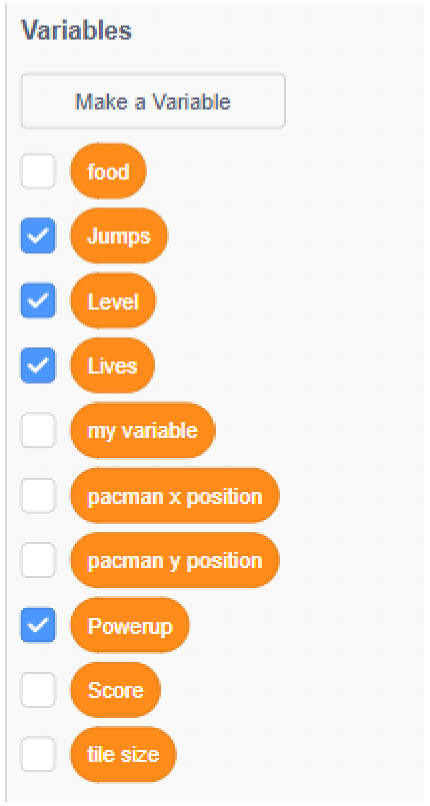
- I created a **Level** variable to keep track of which level I'm in. This variable is also used to control how many ghosts a re cloned in each level.
- I created a **Lives** variable to keep track of PacMan's lives. PacMan loses a life if he touches a ghost and doesn't have a powerup. When PacMan runs out of lives, we broadcast **game over** and the game ends.

Other ideas might include a timer for each level. You'd need a new variable for this that counts seconds until the level is completed. Can you think of any others?

Scratch PacMan

Game Features

You can use the variables control to show the **Lives**, **Level**, **Jump** and **Powerup** variables on the screen using the check box in the variable list.



Catford Coding Club

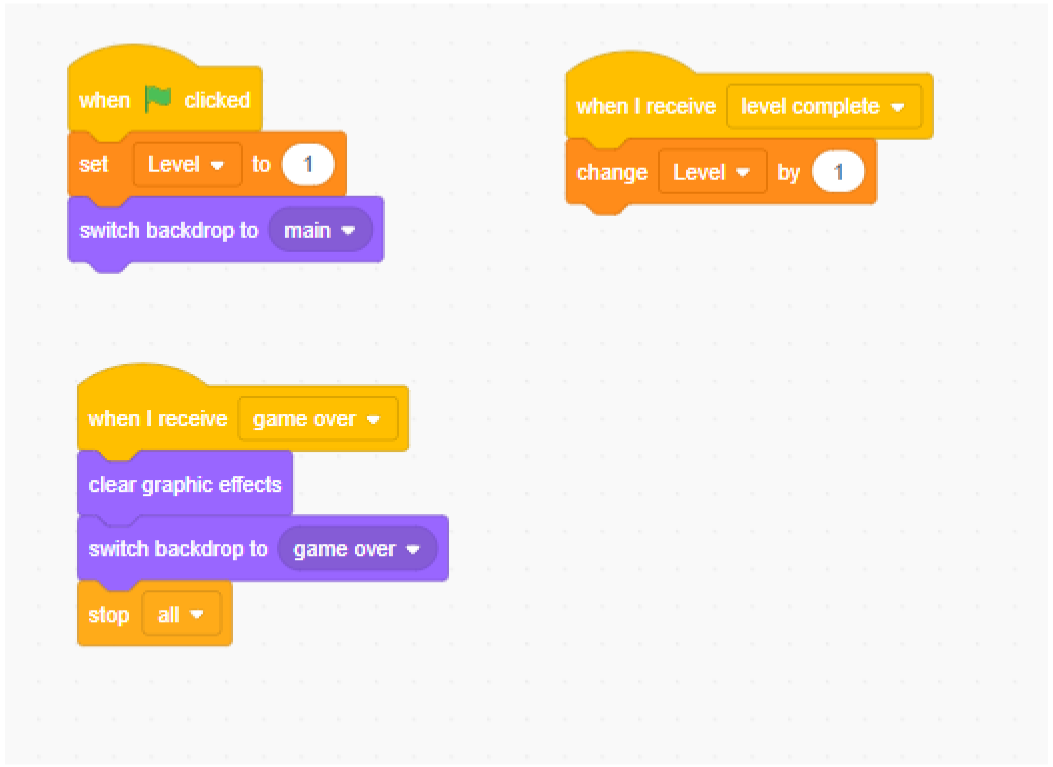
Scratch PacMan

Game Features

Some of the game feature variables are set in the code for the sprites that we have created. Some are set up in code for the backdrop.

Here I set the **Level** variable to 1 at the start of the game and control which backdrop appears when the game starts and ends.

I also add 1 to the **Level** variable each time PacMan clears a level.



```
when ⚑ clicked
set Level ▾ to 1
switch backdrop to main ▾

when I receive level complete ▾
change Level ▾ by 1

when I receive game over ▾
clear graphic effects
switch backdrop to game over ▾
stop all ▾
```

Scratch PacMan

Catford Coding Club

Testing and feedback

Once you have finished, share your game with another Scratch coder and ask them to test it.

When you are testing someone else's game, look for ideas to improve the game:

- How do the sprites look?
- Does the maze generate properly?
- Is the game playable?
- What features could you add?

When you have feedback, think about whether you agree with it and use it to make your game even better!

Catford Coding Club

Scratch PacMan

Well done!

Hopefully you now have a working game that you can share with friends.

There’s always ways to make your game more interesting. You can add new features, experiment with the maze generator, add sounds and effects…

There are also different and maybe better ways to code the game. These instructions give you a start but you can think of better ways to set up a maze or code your sprites.

Have fun!