Session 4: Duck shooting game

To investigate the rules (algorithms) for a simple shooting game

Let’s learn

Lots of computer games involve shooting. Many people worry that playing violent games makes their players more violent in the real world.

Let’s do

1. What do you think about shooting games?
2. Do you know about PEGI age restrictions? Why do you think they exist?
3. How do you feel about waiting until you’re older to play some games?
4. What would you do if you found something in a game that upset you?

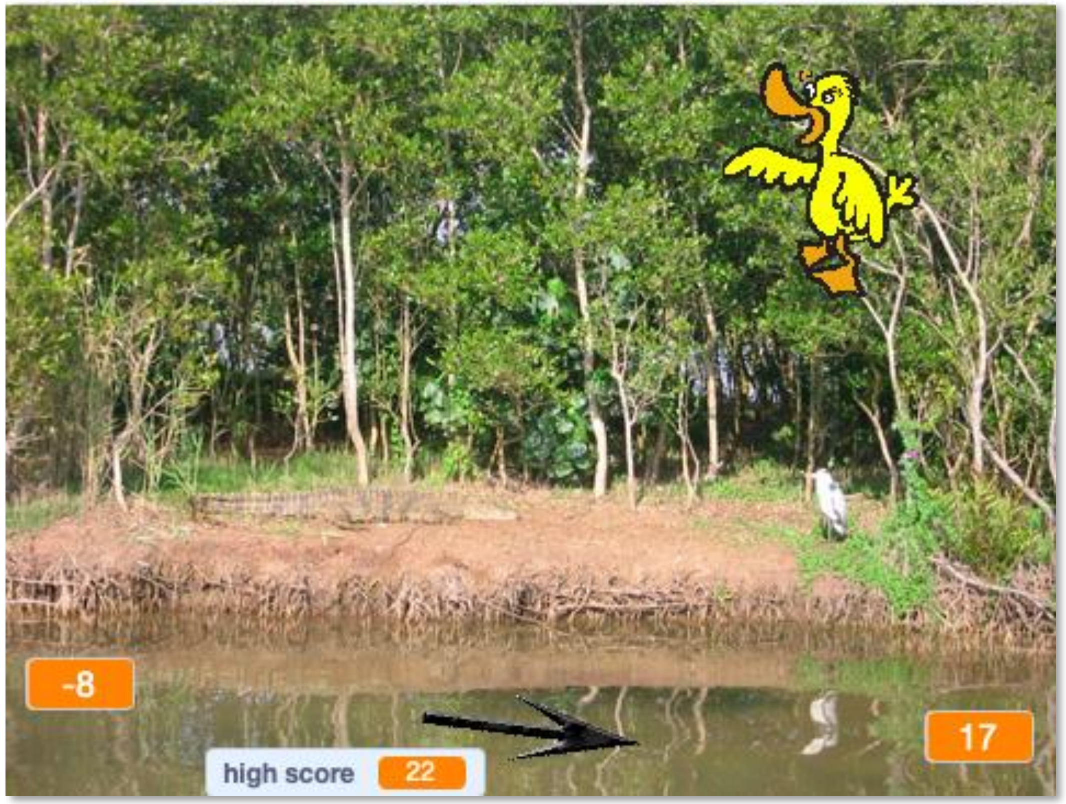
Let's try

This is called the Duck Shoot game. The arrow changes direction as you move the mouse, and pressing the space bar fires the arrow. You can use the 'R' key to reset the high score.

Let's do

1. Who would like to have a go?
2. Watch and try to work out how the game works.
3. What **algorithms** have been programmed?
4. Have you discovered more about how the game works?

Answers on next slide!



Click the image to open the game!

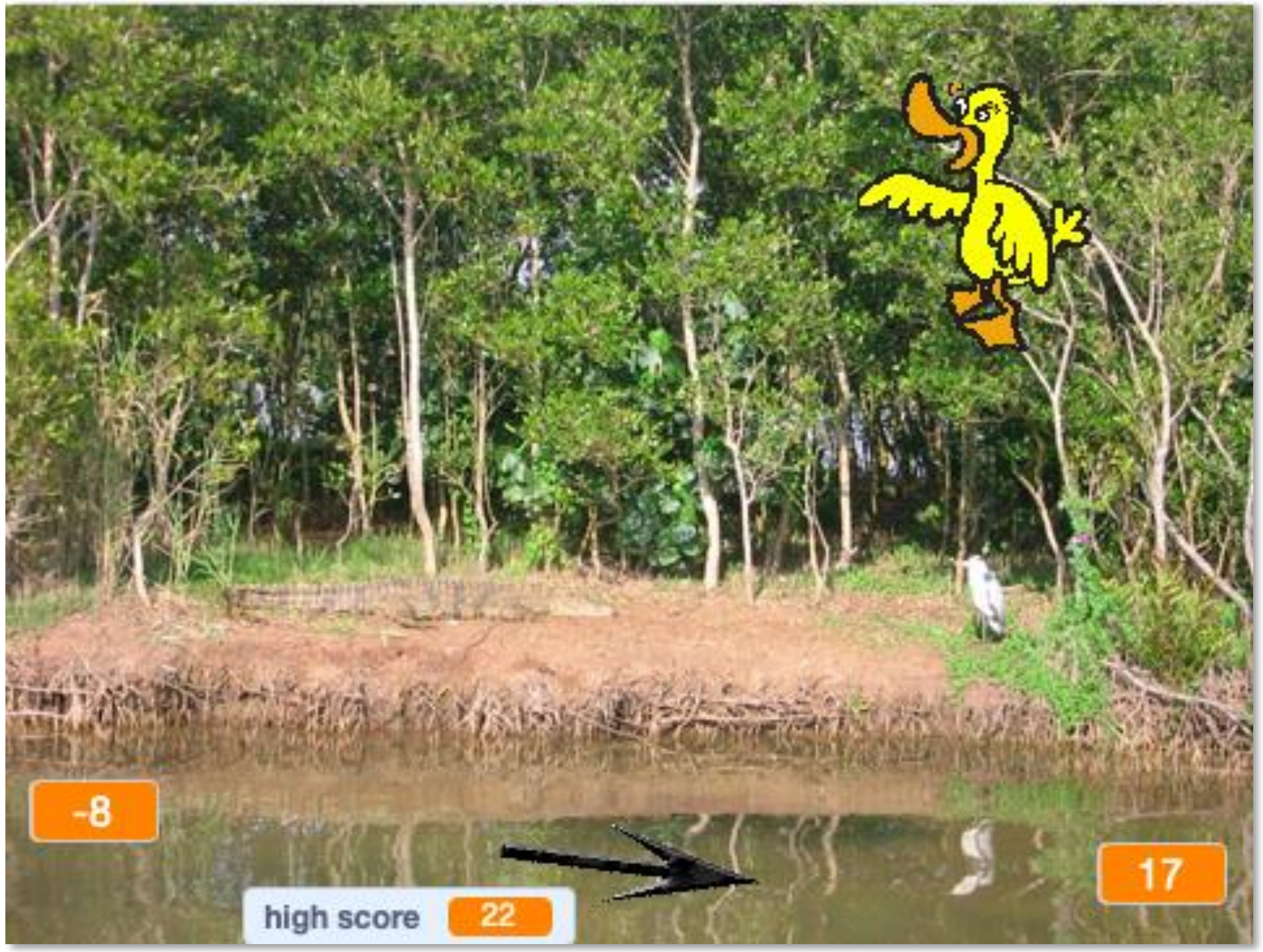
Do you remember what an **algorithm** is?
Click on this box to see the definition.

Let’s review

The arrow changes direction as you move the mouse, and pressing the space bar fires the arrow. You can use the ‘R’ key to reset the high score.

The arrow will move with the arrow, and when the space bar is pressed the arrow launches.

When the arrow touches the duck it will drop, and points will be added.



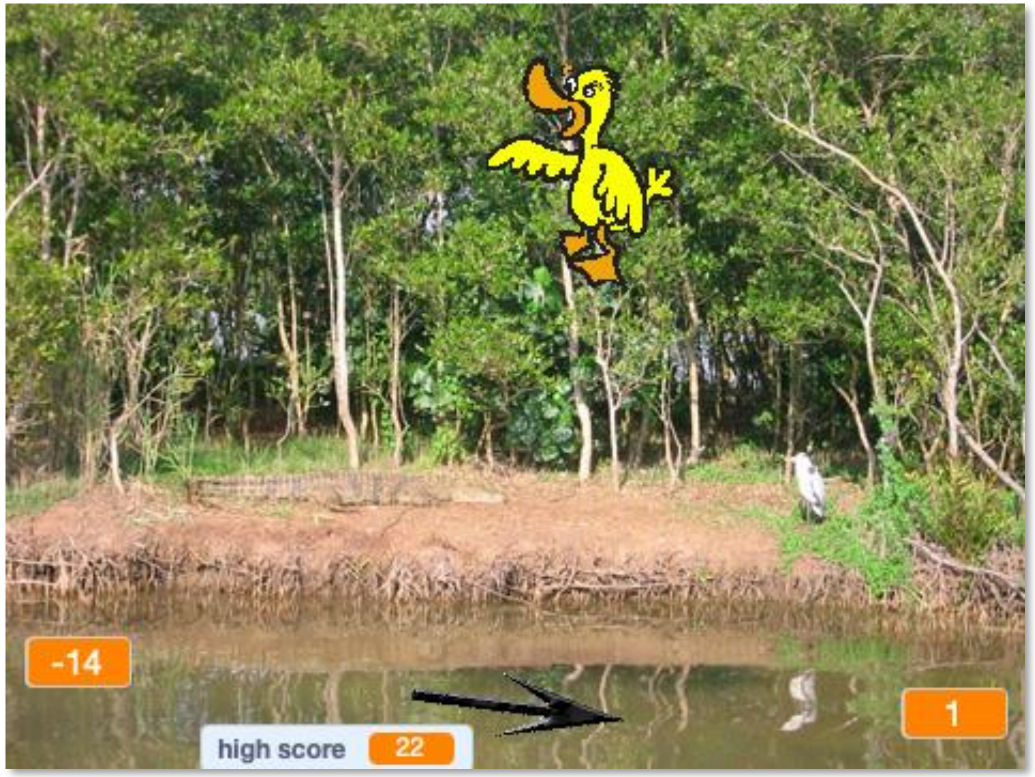
Let’s do

Play the game a few times yourselves. Once you have the hang of it, make and test predictions to work out the algorithms used in the game.

Consider the following questions:

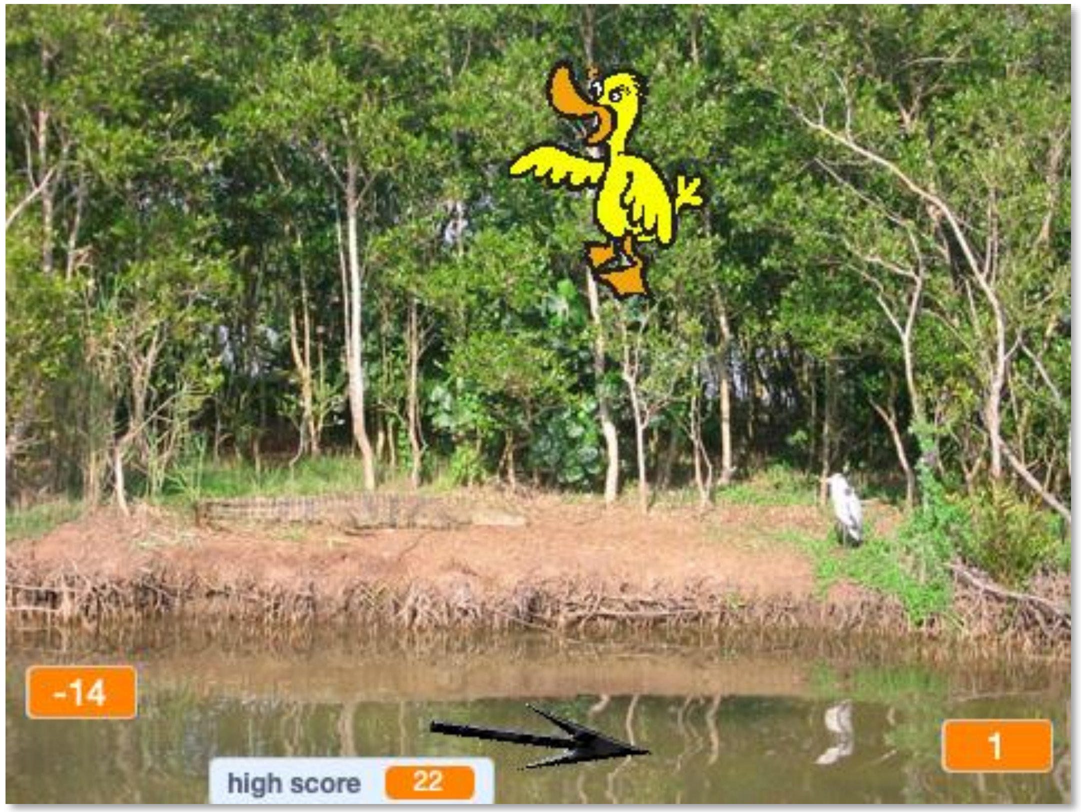
1. How does the duck move?
2. How does the arrow fly?
3. What is the scoring system?
4. What happens when the duck reaches the edge of the screen?
5. What happens when the arrow reaches the edge of the screen?
6. What happens when the duck is hit?

Answers on the next slide!



Let’s review:

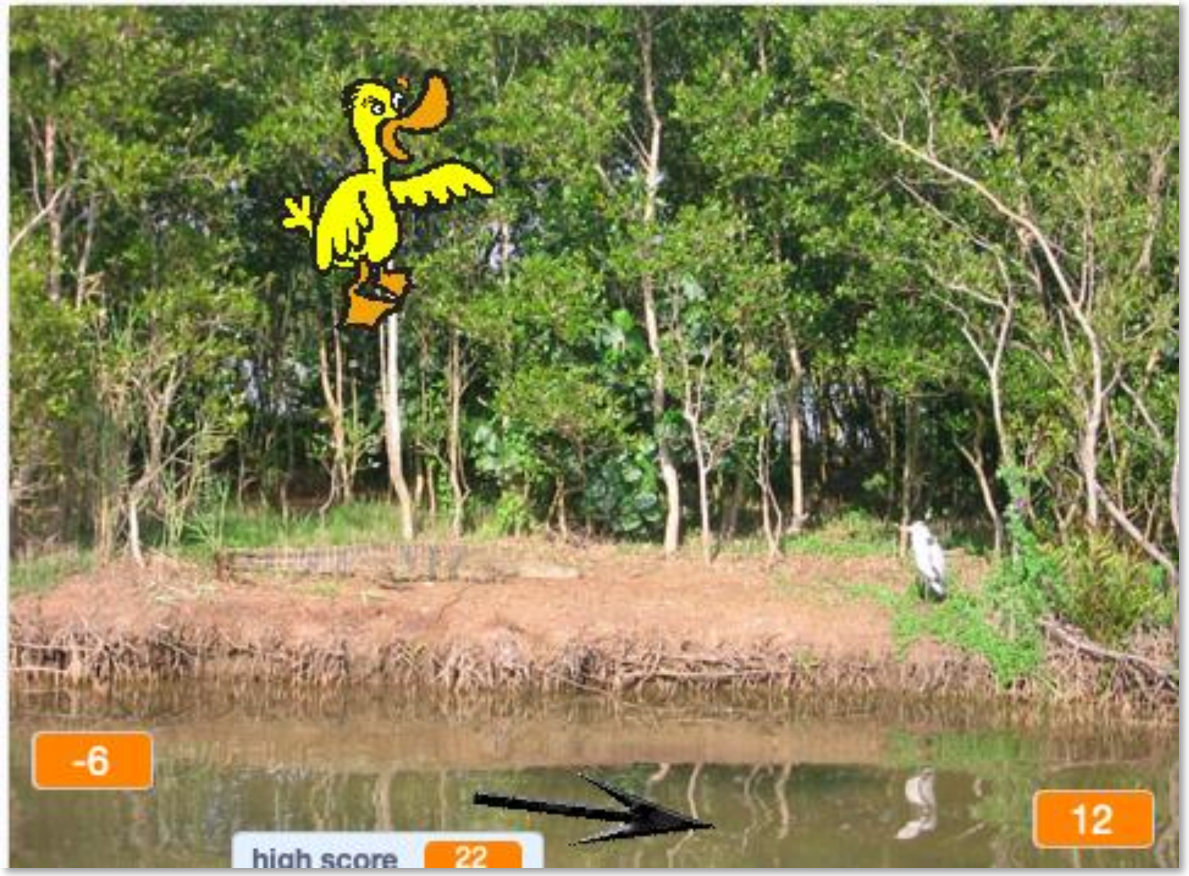
1. The duck goes from one side of the screen to the other and repeats.
2. The arrow moves in a straight line in the direction it is facing when the Space bar is pressed.
3. +10, – 10 or −2
4. – 2 points
5. – 10 points for a miss.
6. 10 points for hitting the duck.



Let's do

Work with a partner to record what you have worked out about the rules that have been implemented in the game.

Can you guess what the code for the game must be like, before you look at it?



Let's learn

Looking at the **source code**, can you see the connection between the rules that you worked out and what the code says?

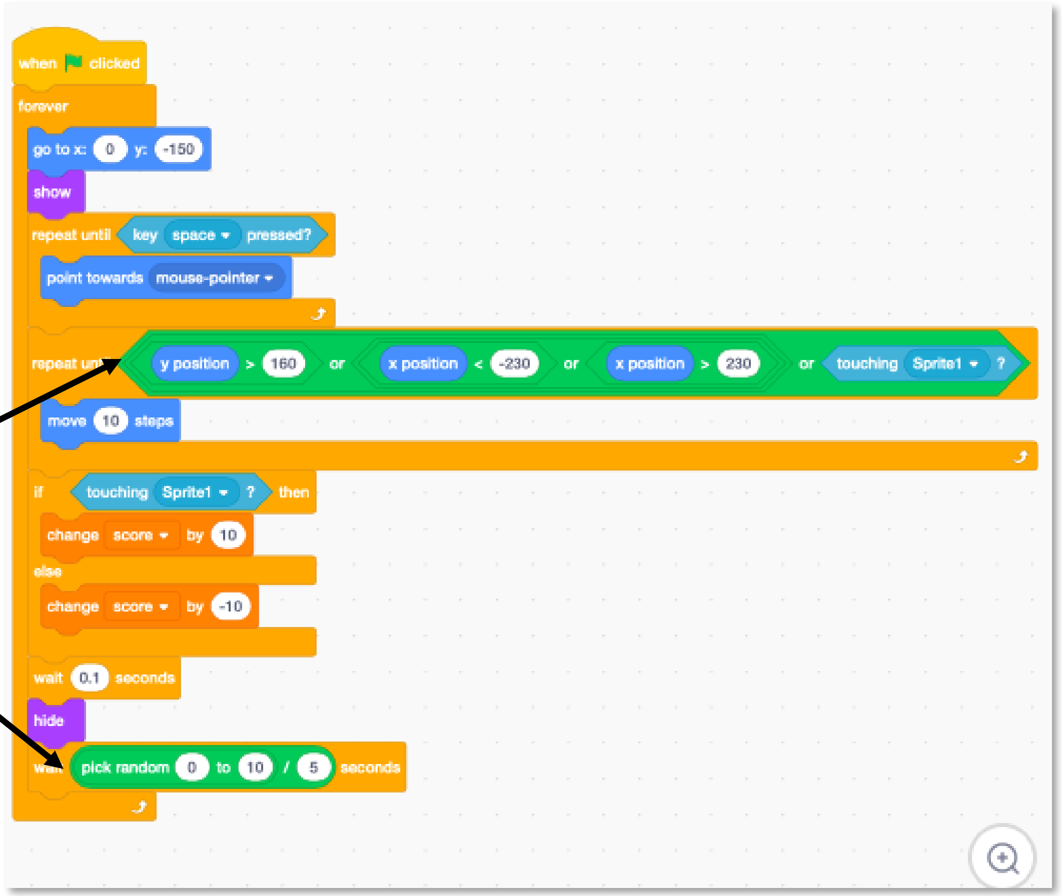
Let's do

1. Do you notice the different uses of the *pick random* blocks here?
2. What do the *pick random* blocks do?

Answers on the next slide!

Do you remember what **source code** means?

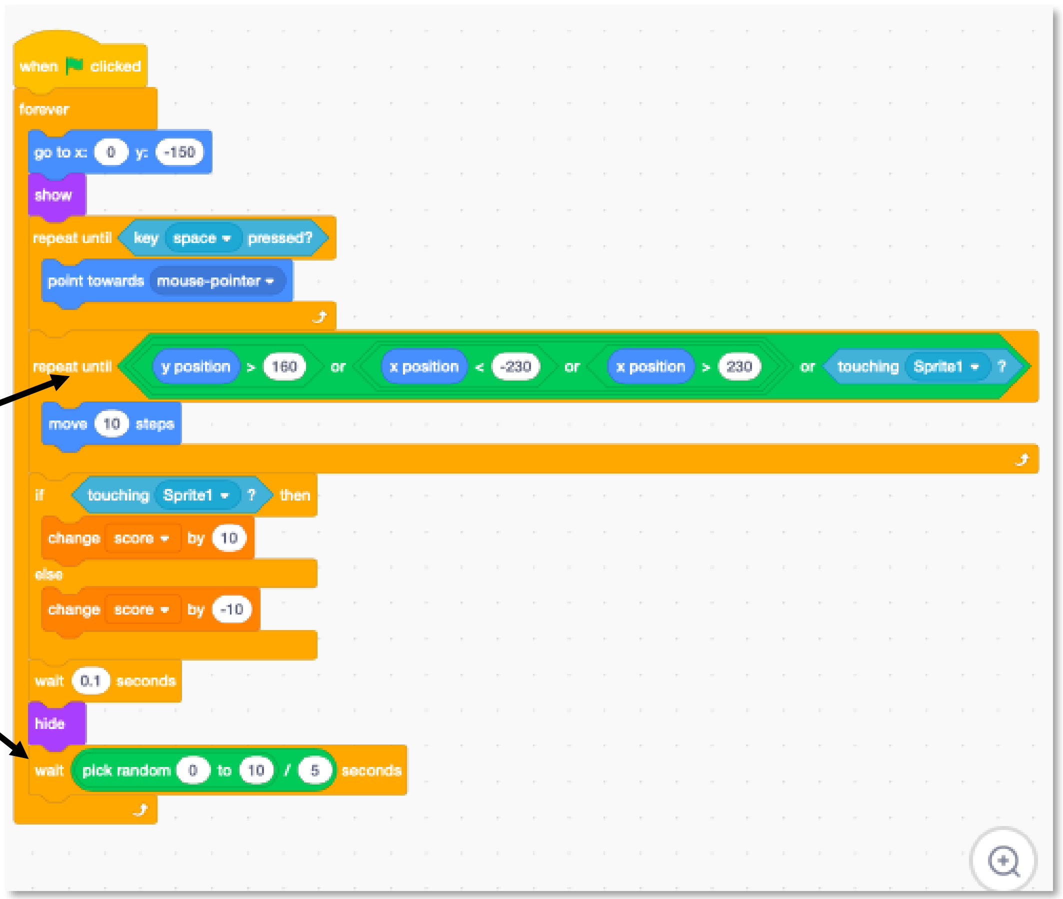
Click on this box to see the definition.



```
when 🏳 clicked
forever
    go to x: (0) y: (-150)
    show
    repeat until <key (space ▾) pressed?>
        point towards (mouse-pointer ▾)
    end
    repeat until <<<(y position) > (160)> or <(x position) < (-230)>> or <<(x position) > (230)> or <touching (Sprite1 ▾) ?>>>
        move (10) steps
    end
    if <touching (Sprite1 ▾) ?> then
        change (score ▾) by (10)
    else
        change (score ▾) by (-10)
    end
    wait (0.1) seconds
    hide
    wait ((pick random (0) to (10)) / (5)) seconds
end
```

Let's review

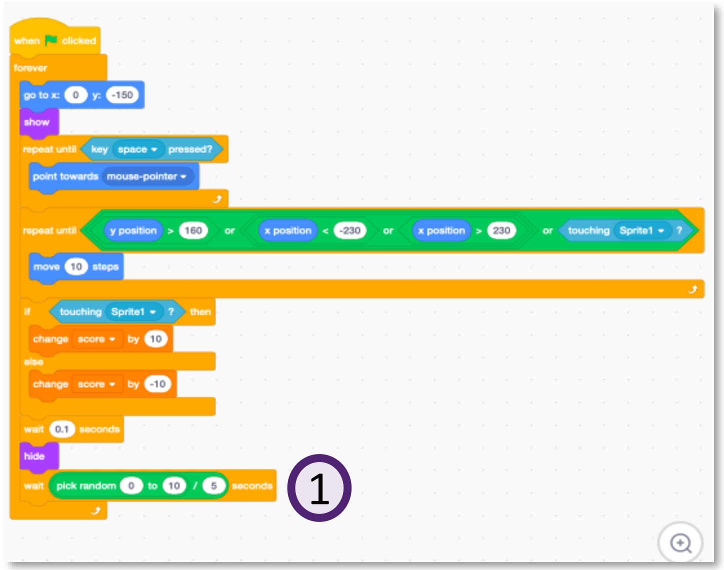
1. First, operator blocks control the location of the arrow.
2. The second operation is the amount of seconds the arrow sprite will take to reappear.



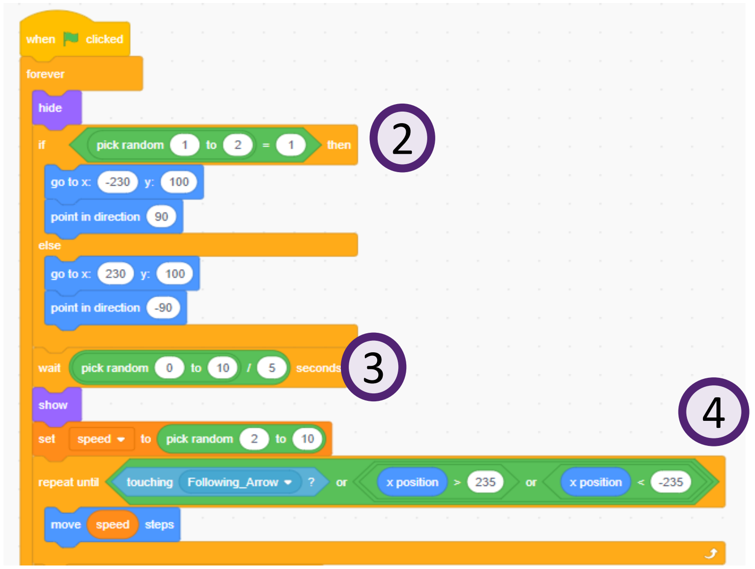
```
when ⚑ clicked
forever
    go to x: (0) y: (-150)
    show
    repeat until <key [space ▾] pressed?>
        point towards [mouse-pointer ▾]
    repeat until <<(y position) > (160)> or <<(x position) < (-230)> or <<(x position) > (230)> or <touching [Sprite1 ▾] ?>>>>
        move (10) steps
    if <touching [Sprite1 ▾] ?> then
        change [score ▾] by (10)
    else
        change [score ▾] by (-10)
    wait (0.1) seconds
    hide
    wait ((pick random (0) to (10)) / (5)) seconds
```

Let's learn

Look at the different uses of the *pick random* blocks.



```
when [flag] clicked
forever
    go to x: (0) y: (-150)
    show
    repeat until <key (space) pressed?>
        point towards (mouse-pointer)
    end
    repeat until <<<<(y position) > (160)> or <(x position) < (-230)>> or <(x position) > (230)>> or <touching (Sprite1) ?>>
        move (10) steps
    end
    if <touching (Sprite1) ?> then
        change (score) by (10)
    else
        change (score) by (-10)
    end
    wait (0.1) seconds
    hide
    wait ((pick random (0) to (10)) / (5)) seconds      1
end
```



```
when [flag] clicked
forever
    hide
    if <(pick random (1) to (2)) = (1)> then      2
        go to x: (-230) y: (100)
        point in direction (90)
    else
        go to x: (230) y: (100)
        point in direction (-90)
    end
    wait ((pick random (0) to (10)) / (5)) seconds      3
    show
    set (speed) to (pick random (2) to (10))      4
    repeat until <<touching (Following_Arrow) ?> or <<(x position) > (235)> or <(x position) < (-235)>>>
        move (speed) steps
    end
```

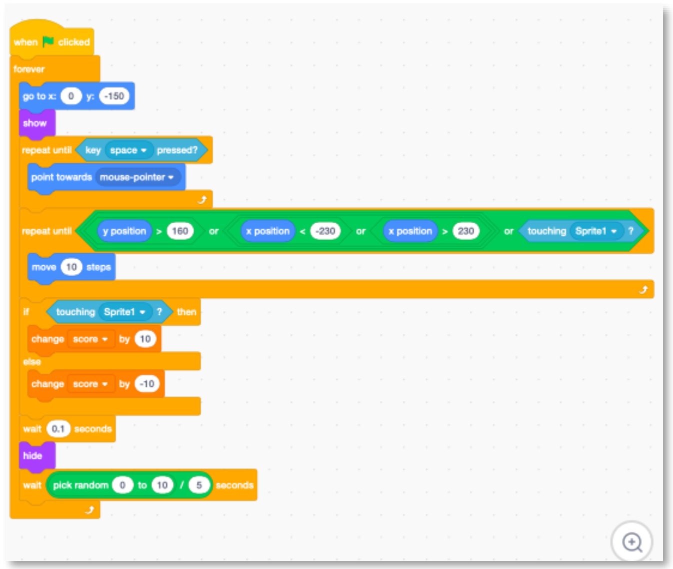
Let’s learn

Can you think of ways in which this simple game could be improved?
Let’s try to improve the game together as a class.

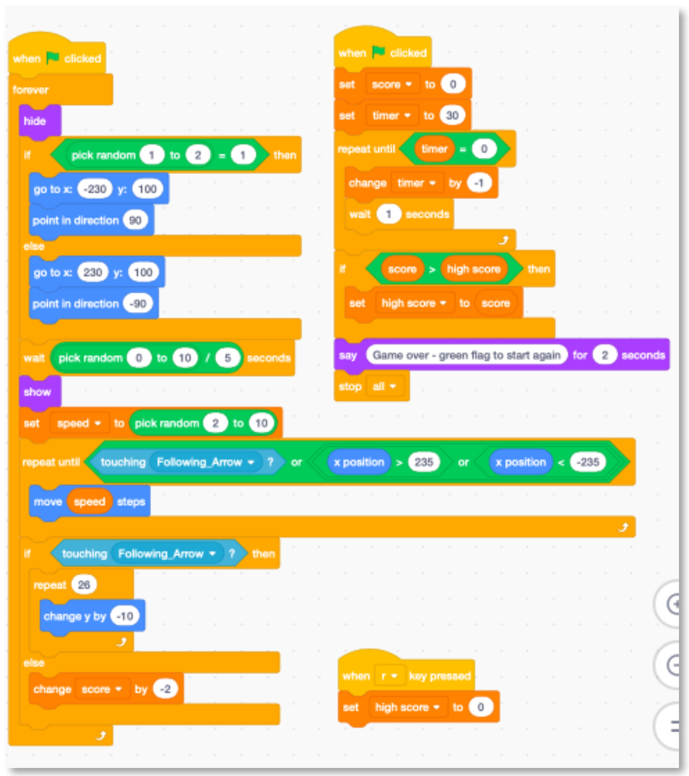
Let’s try

1. Could you change the duck **sprite** to make it more appropriate?
2. Why doesn't the arrow work correctly? How could we change that?

Answers on the next slide!



```
when [flag] clicked
forever
    go to x: (0) y: (-150)
    show
    repeat until <key (space) pressed?>
        point towards (mouse-pointer)
    end
    repeat until <<<(y position) > (160)> or <(x position) < (-230)>> or <<(x position) > (230)> or <touching (Sprite1)?>>>
        move (10) steps
    end
    if <touching (Sprite1)?> then
        change (score) by (10)
    else
        change (score) by (-10)
    end
    wait (0.1) seconds
    hide
    wait ((pick random (0) to (10)) / (5)) seconds
end
```



```
when [flag] clicked
forever
    hide
    if <(pick random (1) to (2)) = (1)> then
        go to x: (-230) y: (100)
        point in direction (90)
    else
        go to x: (230) y: (100)
        point in direction (-90)
    end
    wait ((pick random (0) to (10)) / (5)) seconds
    show
    set (speed) to (pick random (2) to (10))
    repeat until <<touching (Following_Arrow)?> or <<(x position) > (235)> or <(x position) < (-235)>>>
        move (speed) steps
    end
    if <touching (Following_Arrow)?> then
        repeat (26)
            change y by (-10)
        end
    else
        change (score) by (-2)
    end
end
```

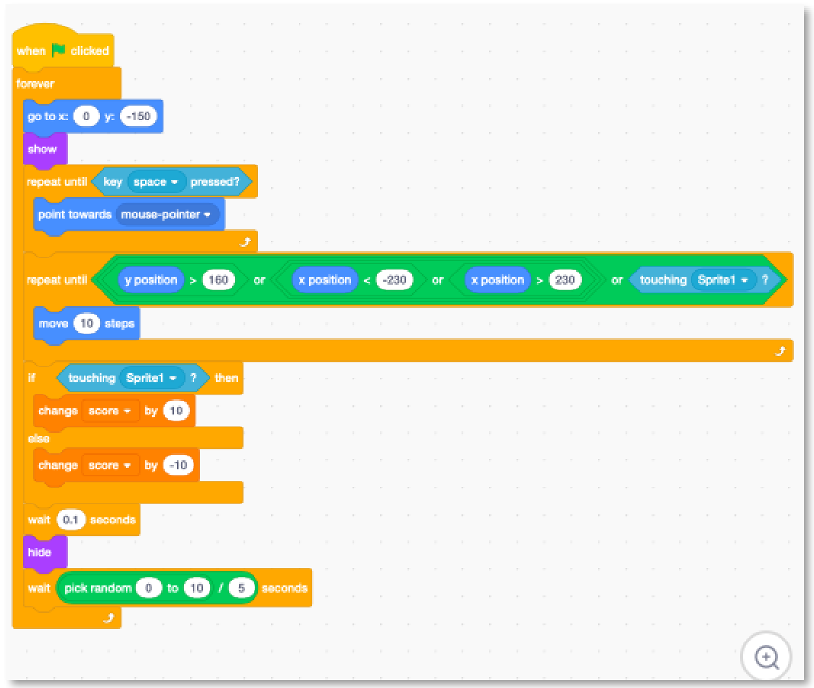
```
when [flag] clicked
set (score) to (0)
set (timer) to (30)
repeat until <(timer) = (0)>
    change (timer) by (-1)
    wait (1) seconds
end
if <(score) > (high score)> then
    set (high score) to (score)
end
say (Game over - green flag to start again) for (2) seconds
stop (all)
```

```
when (r) key pressed
set (high score) to (0)
```

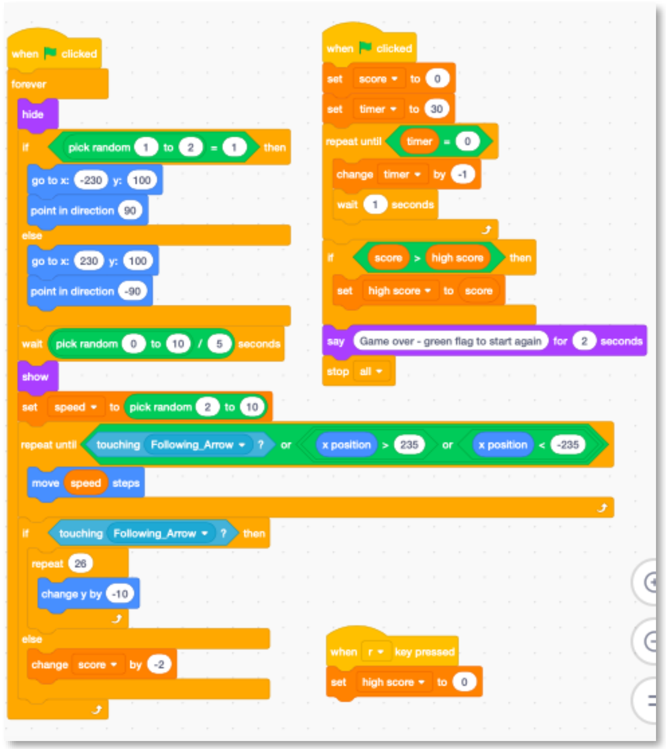
Do you remember what a **sprite** is?
Click on this box to see the definition.

Let's review

1. Go to the costumes library to change the sprite to something more appropriate, or you could paint a target sprite.
2. You could change it to point to the duck sprite. The *x* and *y* values could be edited.



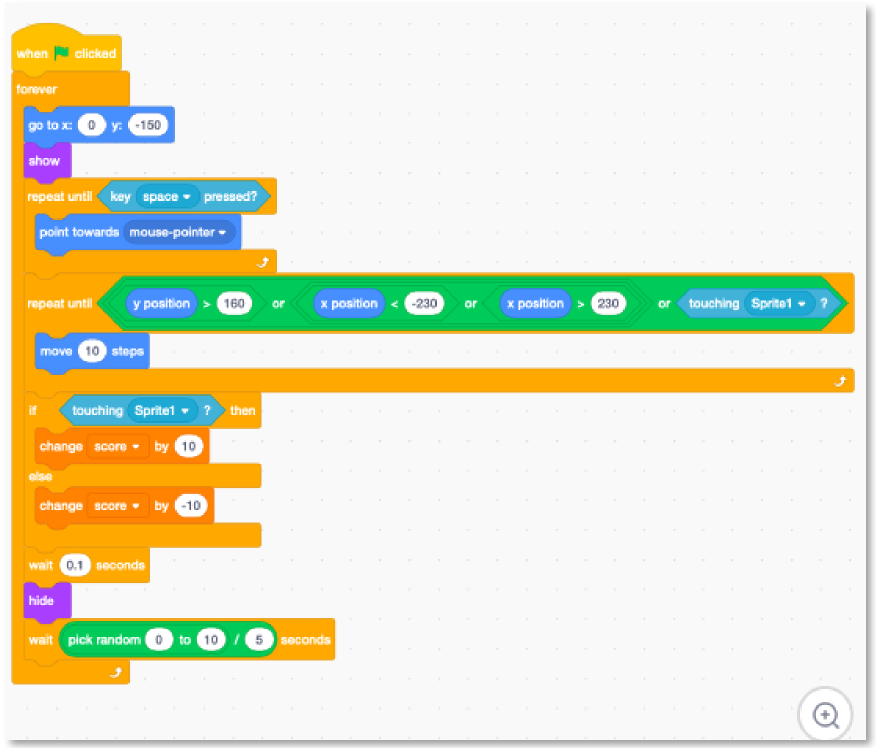
```
when [flag] clicked
forever
  go to x: 0 y: -150
  show
  repeat until <key space pressed?>
    point towards mouse-pointer
  end
  repeat until <<y position > 160> or <x position < -230> or <x position > 230> or <touching Sprite1 ?>>
    move 10 steps
  end
  if <touching Sprite1 ?> then
    change score by 10
  else
    change score by -10
  end
  wait 0.1 seconds
  hide
  wait (pick random 0 to 10) / 5 seconds
end
```



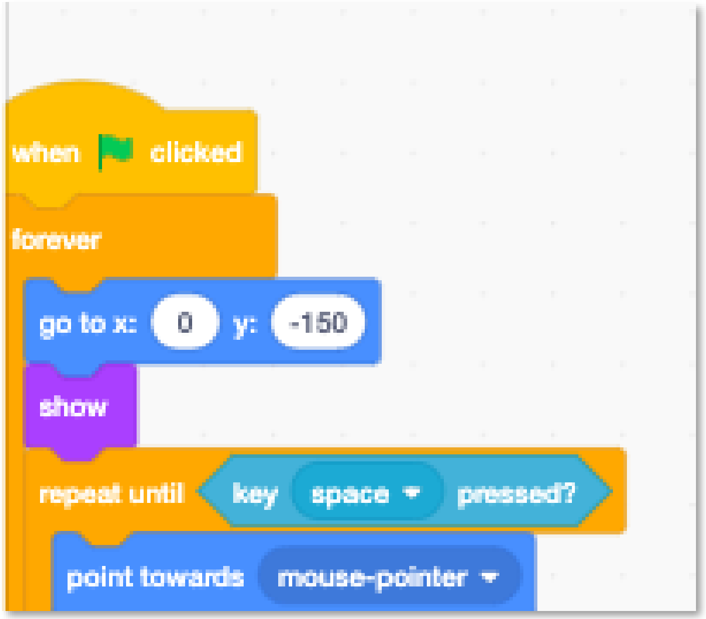
```
when [flag] clicked
forever
  hide
  if <(pick random 1 to 2) = 1> then
    go to x: -230 y: 100
    point in direction 90
  else
    go to x: 230 y: 100
    point in direction -90
  end
  wait (pick random 0 to 10) / 5 seconds
  show
  set speed to (pick random 2 to 10)
  repeat until <<touching Following_Arrow ?> or <<x position > 235> or <x position < -235>>>
    move (speed) steps
  end
  if <touching Following_Arrow ?> then
    repeat 26
      change y by -10
    end
  else
    change score by -2
  end
end

when [flag] clicked
set score to 0
set timer to 30
repeat until <(timer) = 0>
  change timer by -1
  wait 1 seconds
end
if <(score) > (high score)> then
  set high score to (score)
end
say Game over - green flag to start again for 2 seconds
stop all

when r key pressed
set high score to 0
```

Let's review

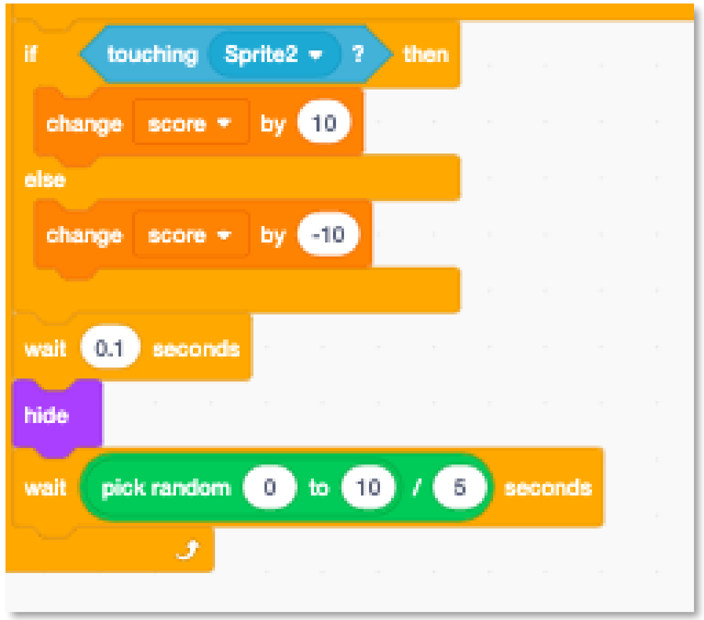
1. You will need to change the point towards to *point towards sprite*.
2. You may need to change some of the *x* and *y* values, so that they match the screenshots.



```
when 🏴 clicked
forever
    go to x: (0) y: (-150)
    show
    repeat until <key (space ▾) pressed?>
        point towards (mouse-pointer ▾)
    end
    repeat until <<<(y position) > (160)> or <(x position) < (-230)>> or <<(x position) > (230)> or <touching (Sprite1 ▾) ?>>>
        move (10) steps
    end
    if <touching (Sprite1 ▾) ?> then
        change (score ▾) by (10)
    else
        change (score ▾) by (-10)
    end
    wait (0.1) seconds
    hide
    wait ((pick random (0) to (10)) / (5)) seconds
end
```



```
when 🏴 clicked
forever
    go to x: (0) y: (-150)
    show
    repeat until <key (space ▾) pressed?>
        point towards (mouse-pointer ▾)
```



```
if <touching (Sprite2 ▾) ?> then
    change (score ▾) by (10)
else
    change (score ▾) by (-10)
end
wait (0.1) seconds
hide
wait ((pick random (0) to (10)) / (5)) seconds
end
```