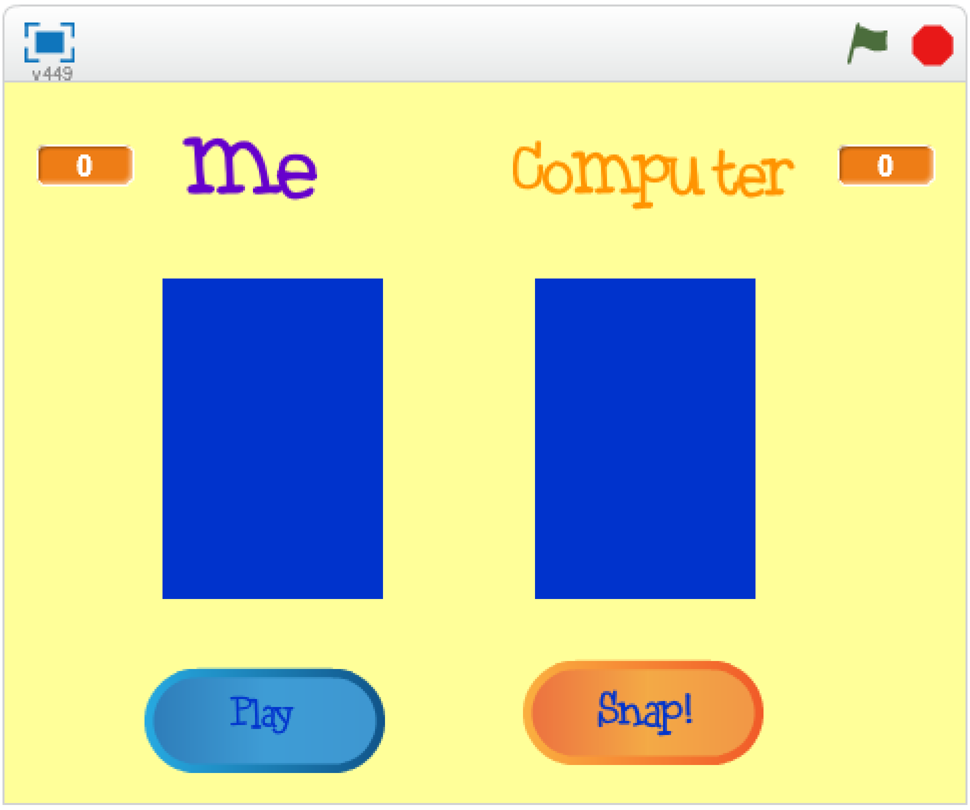
Snap!

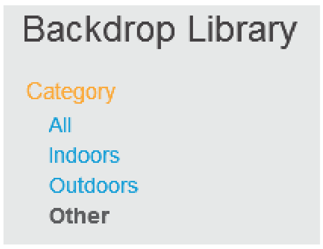
See this project online at https://scratch.mit.edu/projects/115010977/



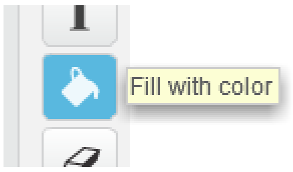
In this project we are going to code and play a card game. We will try to beat the computer at Snap! I hope you like drawing costumes…

Step 1: Set up the screen

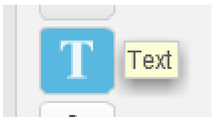
- Start a new Scratch project, and delete that poor cat.
- First we need to choose a background. It needs to be fairly plain, so you could look at the ones in the 'Other' category of the Backdrop library:



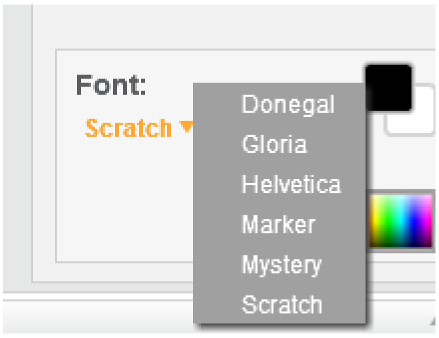
… or, you could choose to Paint a new backdrop: choose a colour you like and use the 'Fill with colour' paintpot to make the whole thing one colour (this is what I did):



- Next we need to write on the backdrop, to show which cards are yours, and which belong to the computer.
- To do this, select the Text tool:

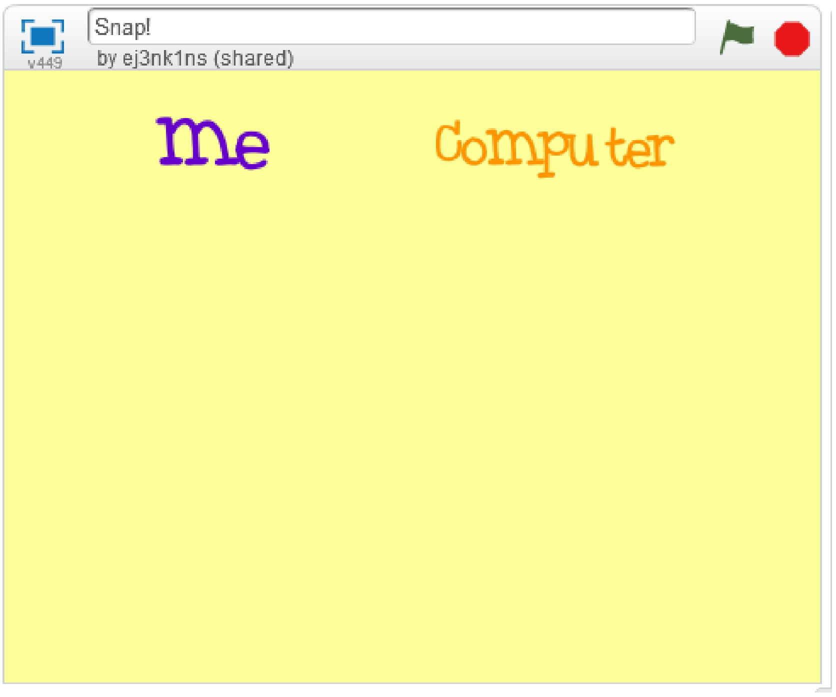


and click somewhere on your backdrop (you can always move it if it isn't in quite the right place). Type 'Me' for the first label, and have a play with the Font dropdown box near the bottom of the screen, to find a style of writing that you like:



- In the same way, add the word 'Computer' on the other side of the stage (remember you can use different colours too!).

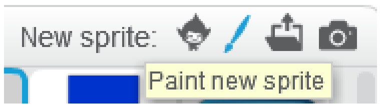
- Your stage should now look something like the one below:



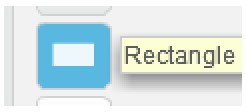
Save your project

Step 2: Design your cards

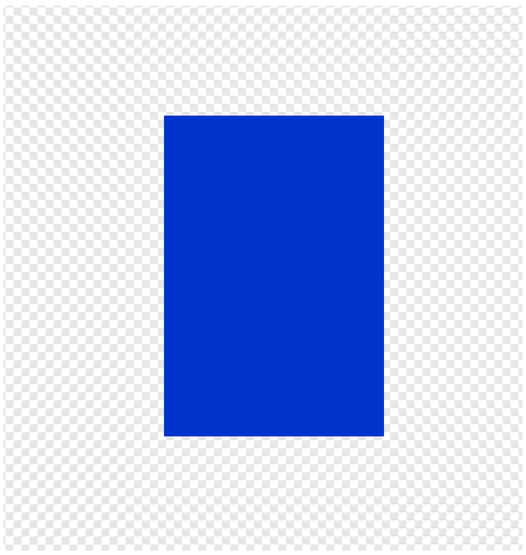
- Now we need to create a card sprite. We will design and code our deck first, and then make a complete copy of them for the computer, so that we can play the game.
- To create our card sprite, use the ‘Paint new sprite’ brush:



- Choose the Rectangle tool:

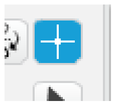


and draw a rectangle of any colour, standing on its short edge, like this:



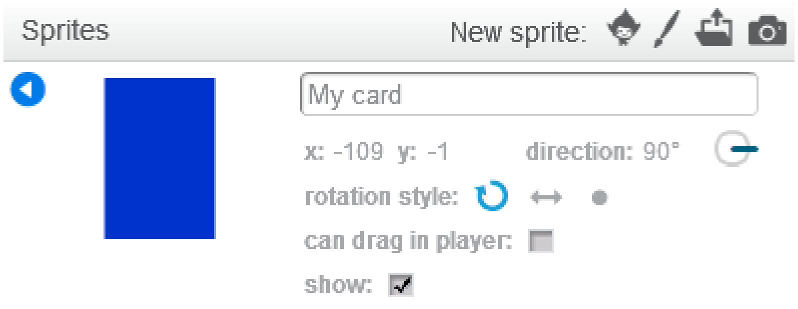
(If you are not sure how big to make it, compare yours with the picture on the front page of these instructions).

- Use the ‘Set costume centre’ crosshairs to check that the lines cross in the very middle of your card:



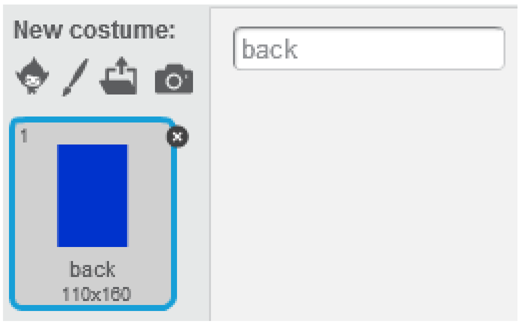
- Make sure you are happy with the size and shape of your card costume, because we will copy this one for all of the others – take your time and get it right!

- Change the name of your sprite to ‘My card’ and change the name of this first costume to ‘back’ (ie. this is what the cards look like when they are face down on the table):



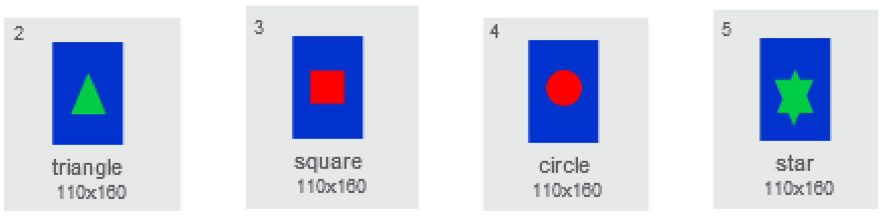
My card
x: -109 y: -1 direction: 90°
rotation style:
can drag in player:
show: ✓

and



Save your project

- Now we can design the front of our cards – be imaginative!
- I came up with four designs: triangle, square, circle and star:



- How do we draw them? Well, it is important they are all the same size and shape, so we will copy the 'back' costume each time. Right-click on this costume and choose 'duplicate'. You will get a new costume called 'back2', so change this to something more helpful (as I did above), and then draw your design on this rectangle.
- You can use vector mode or bitmap mode, whichever you find easier. Can you see that I copied the triangle shape, and turned it upside down, to make the star?
- I chose only two colours, because it makes the game a little bit harder! If you want it to be easier, you could have different colours for each shape. If you want it to be harder, you could have red circle/red square/green circle/green square, for example.

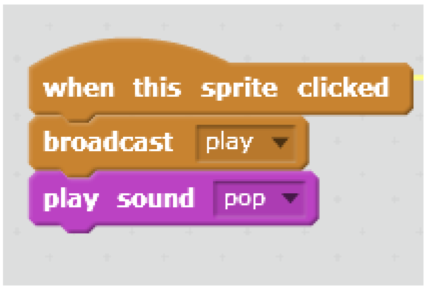
Step 3: We need a button!

- That was a lot of drawing! Time for some code.
- Click 'Choose sprite from library' and look for Button2 (in the 'Things' category):

Button2

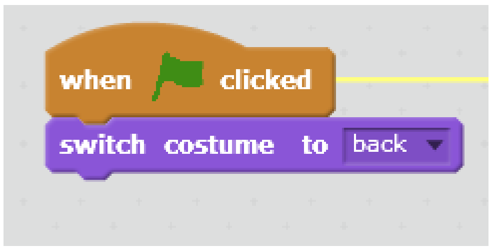
- You could choose a different button, but this one is easy to write a label on. We are going to write 'Play' on it (using the Text tool again), so it looks like this:

- You can delete its second costume, as it only needs one.
- And finally some code! We want to send a message so that our card, and the computer's card, get turned over at exactly the same time (don't worry that the computer doesn't have any cards yet), so add this code to your 'Play' button:



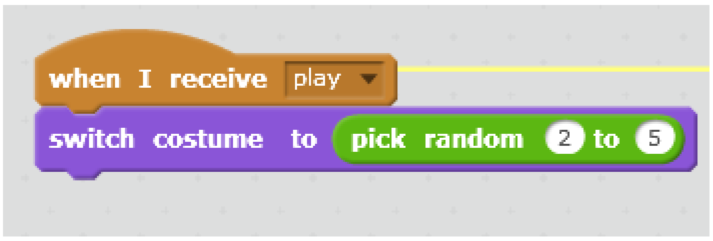
```
when this sprite clicked
broadcast play
play sound pop
```

- (If you haven't used the 'broadcast' block to send a message before, you will find it in the Events section.)
- You can of course choose a different sound instead of pop, but make it a short one, like duck, cymbal, or boing.
- Now we can add some code to our card sprite. First of all, we want to start each game with the cards face down:



```
when [flag] clicked
switch costume to back
```

- Next, we want to change our costume every time the Play button is clicked:

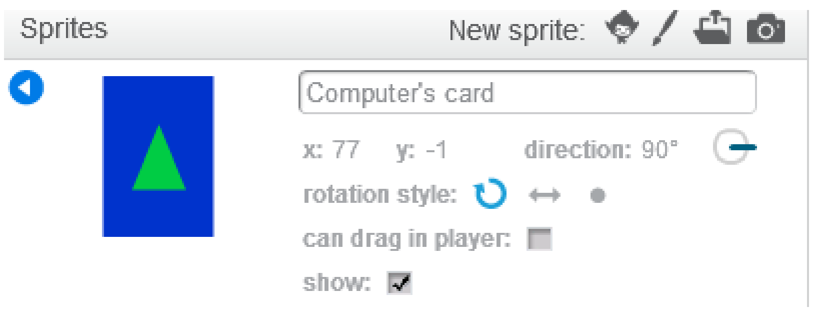


```
when I receive play
switch costume to pick random 2 to 5
```

- Notice that we can specify each costume using its name (eg. 'back'), or a number (in the 'pick random' operator). You can see the number of each costume just above it and to the left – it is very tiny! You can even change the number of a particular costume by dragging and dropping it to another position in the list.
- Why do we say 'pick random 2 to 5' and not 'pick random 1 to 5'? (If you aren't sure, change the 2 to a 1 and find out. Then change it back ☺)
- OK, Test it out! You should start with the back of the card showing, and every time you click 'Play', the design will (probably) change. Why doesn't it always change?

Save your project

- Here is a nice easy bit – we are going to copy our cards (costumes and scripts) to give the computer an identical set.
- Right-click on your ‘My card’ sprite, and choose ‘duplicate’. You will get another sprite that is an exact copy, called ‘My card2’.
- Change the name of the ‘My card2’ sprite to ‘Computer’s card’:



Computer's card

x: 77 y: -1 direction: 90°

rotation style:

can drag in player:

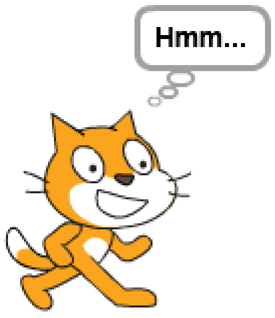
show: ✓

- Drag the ‘Computer’s card’ sprite to where you want it on the stage, and test the program again – do both cards change when you click ‘Play’?

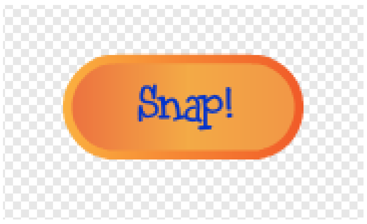
Save your project

Step 4: How do we ‘Snap!’?

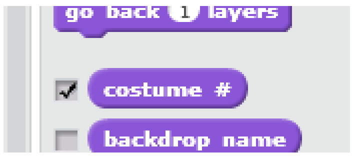
- So far so good, we have a game where two cards are turned up every time we click a button, but how do we tell the computer when it is a ‘Snap!’? And how does the computer know if we are right or not? And the computer is playing too, so how does it tell us when it spots a ‘Snap!’?
- I’m leaving the rest of this page blank, so you can think about it…



- Any ideas? If you said 'we could make a variable…' then you were spot on! Whenever computers need to remember anything, that information is called data, and it is held in variables.
- You would also be right if you said 'we need another button' so let's start there.
- Click 'Choose sprite from library' and choose Button2 again, but this time delete its first costume, and use the second one. Add the text 'Snap!' to your button:



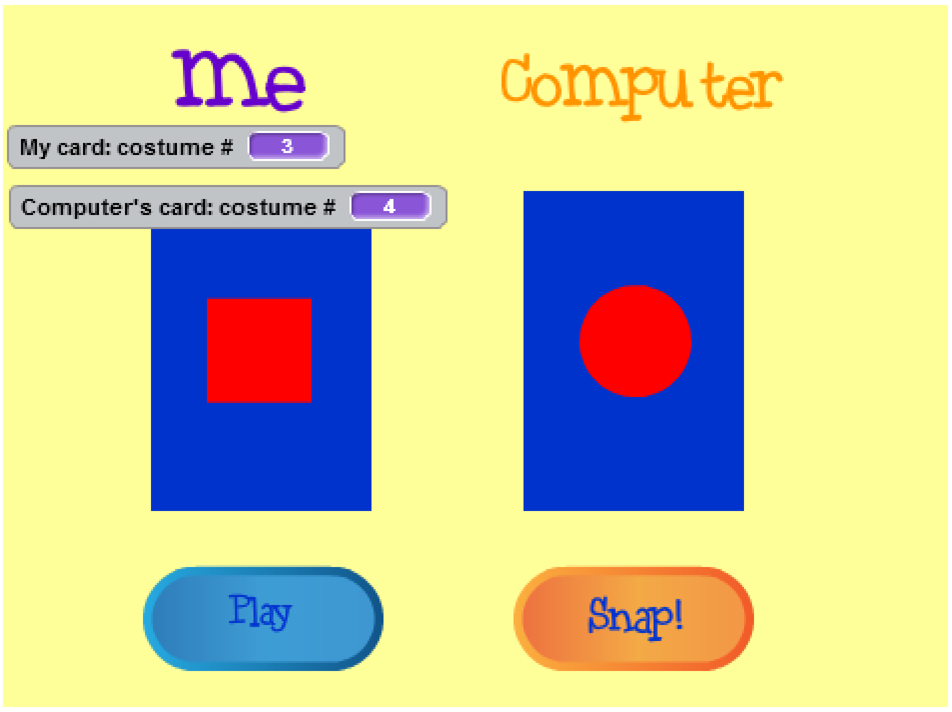
- We will click this button to tell the computer that we have spotted a 'Snap!'.
- Let's think about the code we need... When the button is clicked, we want to compare the costume on the 'My card' sprite with the costume on the 'Computer's card' sprite, and luckily enough, Scratch has a variable that we can use for this. Near the bottom of the 'Looks' section you will find a block called 'costume #':



- Tick this box to show the variable, on both 'My card' and 'Computer's card' sprites (by the way, the character '#' is called hash, and people often use it as a short way of writing the word 'number').

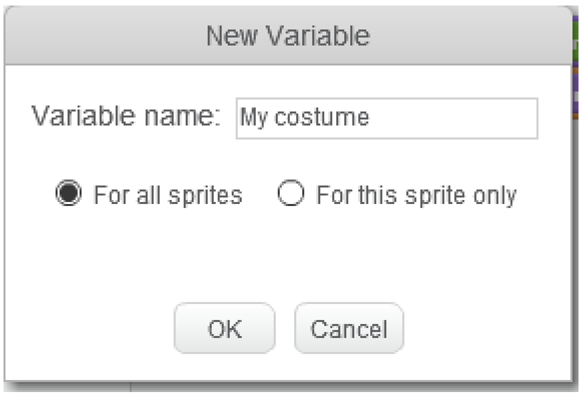
Save your project

- Your stage should now look a bit like this:



- Because we copied the costumes from ‘My card’ to ‘Computer’s card’, you can look at either of them and see that the red square is costume number 3, and the red circle is number 4 (or whatever costumes you created). Click ‘Play’ a few times and watch the numbers change, until you are happy you understand what is going on (don’t worry that the ‘costume #’ variables look untidy on the stage, we will hide them when we have finished writing the code).

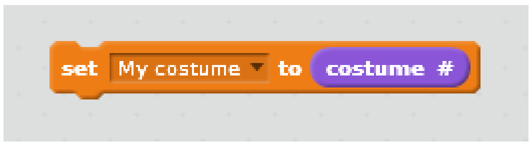
- You will realise that the ‘costume #’ variable is the sort that is set up ‘for this sprite only’ (because it has the sprite name before it, when it is displayed). We can’t compare these variables with each other on the ‘Snap!’ button, so we will need to set up our own variables, ‘for all sprites’, to hold the same costume numbers. I called mine ‘My costume’ and ‘Computer costume’:


New Variable

Variable name: My costume

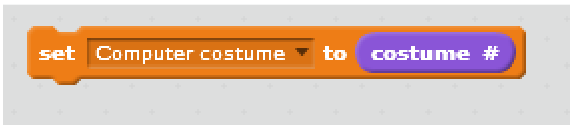
[✓ For all sprites] For this sprite only

- When you have created these two variables (on the 'Data' section, if you are not sure), keep the boxes next to them ticked for now (your stage probably looks quite untidy!).
- Add the following code to the 'when I receive play' block on the 'My card' sprite:



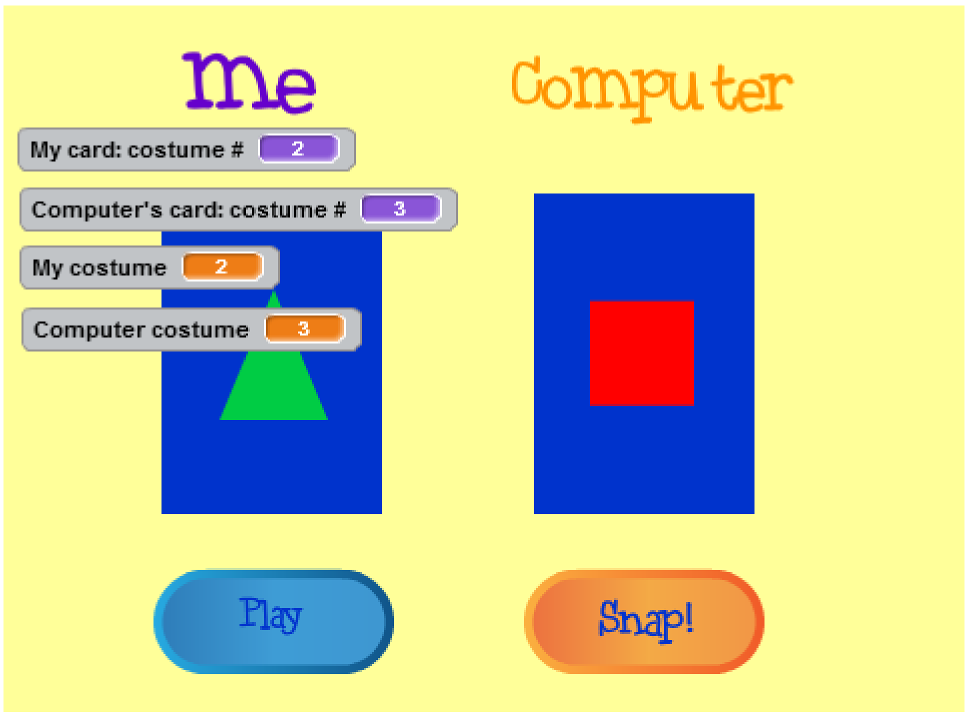
```
set My costume to costume #
```

- This should be right after the block that picks a random costume for this sprite, so the number of that costume will be stored in our 'My costume' variable.
- In the same way, add the block below to the 'Computer's card' sprite, at the end of the 'when I receive play' block:



```
set Computer costume to costume #
```

- If you have kept all the boxes next to these variables ticked, then your stage will look something like this:

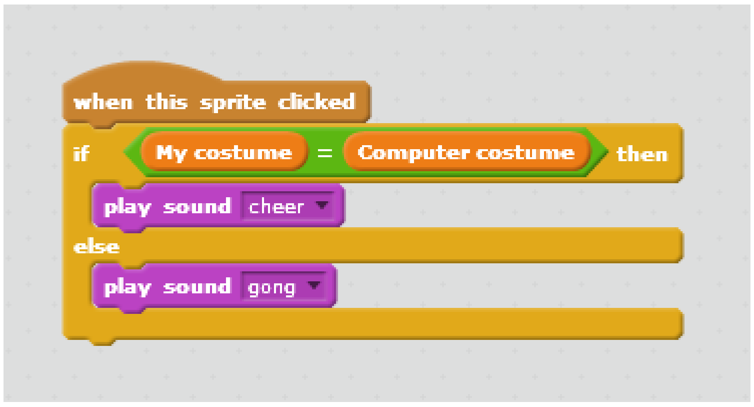


- Click ‘Play’ a few times again, and you should see that ‘My costume’ is always the same as ‘My card: costume #’, and similarly, ‘Computer costume’ is always the same as ‘Computer’s card: costume #’.
- If you are a bit confused, the reason we have done this is to change variables that are only for one sprite, into variables that all sprites can use.

Save your project

- Let’s tidy up the stage a bit – clear the tick next to the ‘costume #’ variable on both of your card sprites (we will keep the variables we created visible, for now).

- Add the code below to your 'Snap!' button sprite (you can of course choose your own sounds – a winning and a losing one):



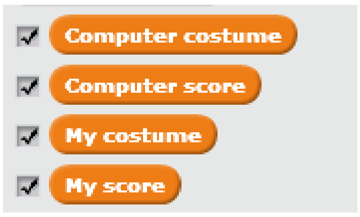
```
when this sprite clicked
if My costume = Computer costume then
  play sound cheer
else
  play sound gong
```

- Click the green flag to start the game, and keep clicking the 'Play' button to change the cards. What happens if you click the 'Snap!' button when the cards are the same? What happens if you click the 'Snap!' button when the cards are different?

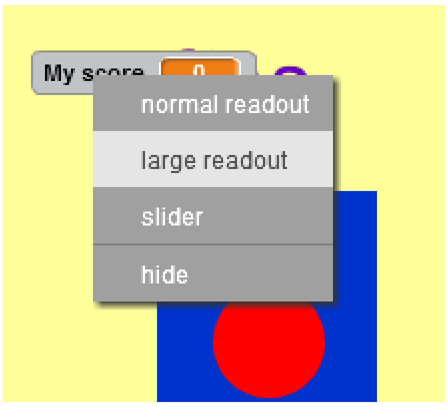
Save your project

Step 5: Keeping score

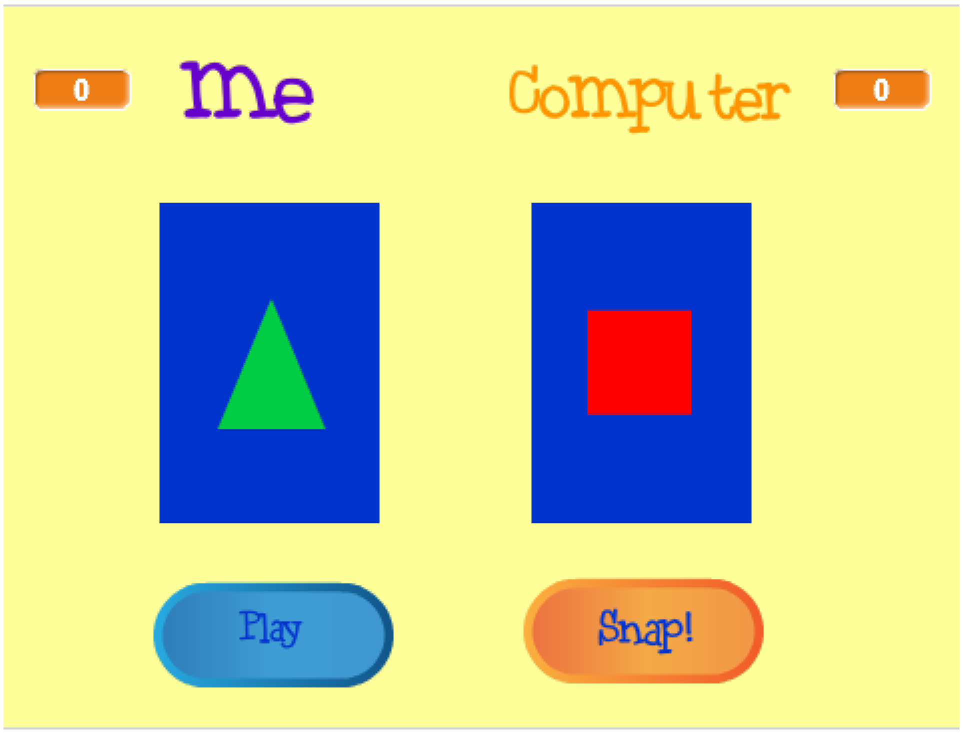
- This is great, but we need to keep score! Create two more variables, for all sprites, and call them 'My score' and 'Computer score', so now we have the follwing variables in our list:



- If you are happy how the costume variables are working, you can hide those by clearing their tickboxes now.
- I put the score variables in the top corners of the stage, and used right-click on each one to choose 'Large readout' (make sure you put them in the right corners!):

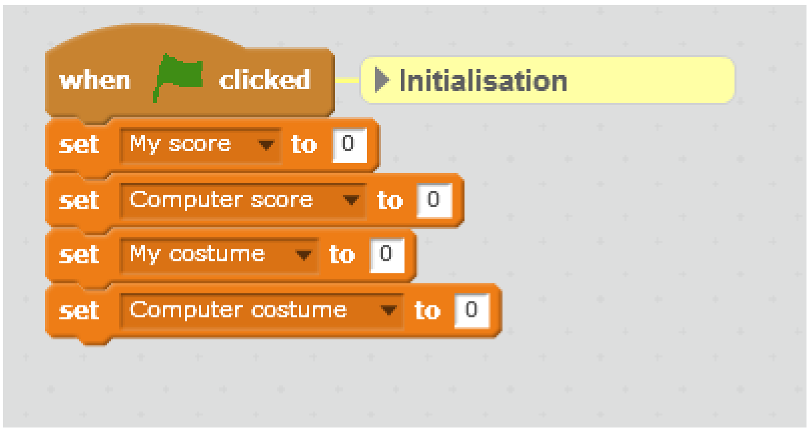


- Your stage should now look like the one below:



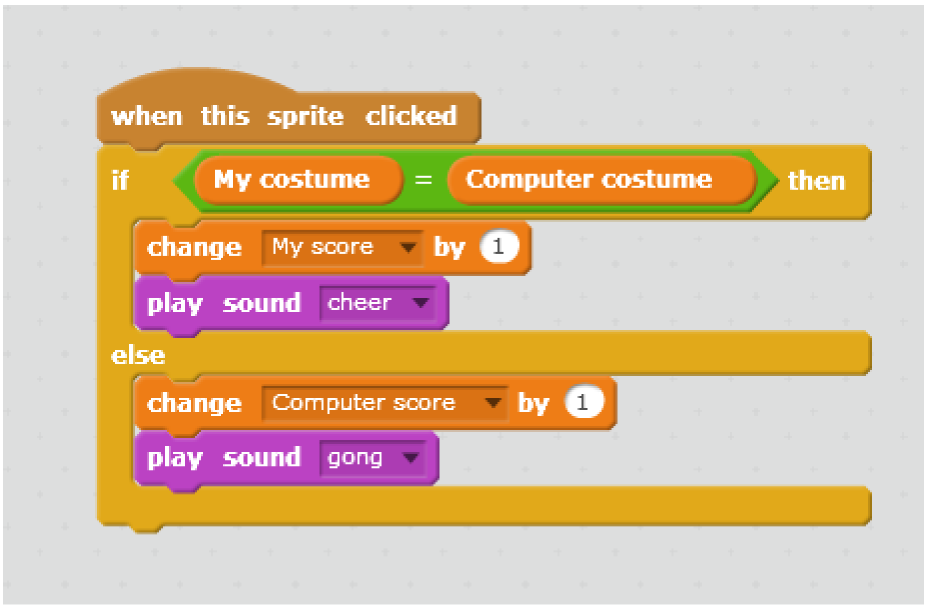
Save your project

- If you have coded projects with a score before, you probably know this next bit.
- Look at the code blocks on the 'Data' section – we will use the 'set score' block at the beginning of the project, and the 'change score' block when we score a point in the game. First of all, we set all our variables to zero at the start of our game (a good place to put this code is on the backdrop):



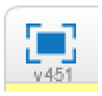
```
when [flag] clicked        Initialisation
set My score to 0
set Computer score to 0
set My costume to 0
set Computer costume to 0
```

- Next, we must decide how we will score points in the game. I decided that I would get a point if I clicked the ‘Snap!’ button when the cards were the same, but the computer would get a point if I clicked the ‘Snap!’ button when the cards were different. You could do this, or you might decide to take away one of your points for an incorrect ‘Snap!’, or not change the points at all.
- Doing it my way, the code on your ‘Snap!’ button should look like this:



```
when this sprite clicked
if <(My costume) = (Computer costume)> then
    change [My score v] by (1)
    play sound [cheer v]
else
    change [Computer score v] by (1)
    play sound [gong v]
```

- Test it out! You will probably find this easier to do if you make the stage fullscreen by clicking on the button below (top left of your stage), as then the sprites can’t be accidentally moved:

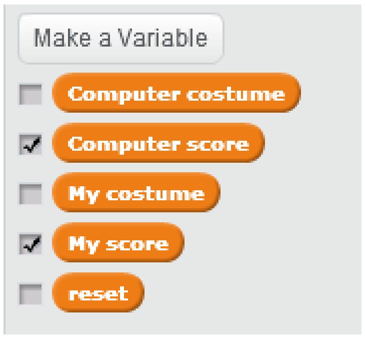


- Is everything working? Both cards should change when you click the ‘Play’ button, and the ‘Snap!’ button should play the right sound, and change the right score, when you click it. If your project doesn’t do this, go back and read that section again – you can work it out!
- Have we thought of everything…?

Save your project

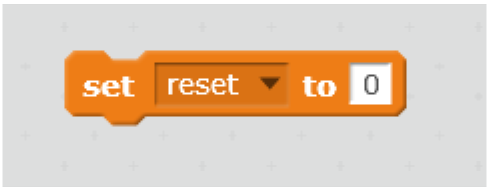
Step 6: Things we haven’t thought of yet

- There are two problems with our Snap! game at the moment; have you thought of either of them yet?
- One problem you can see quite easily – if you click the ‘Snap!’ button lots of times on the same (matching) cards, you score lots of points! This isn’t how the game is supposed to work.
- The other problem is to do with fairness – we have let the computer score points when we make a mistake, but we haven’t given it a chance to spot a Snap! before us, and take the point instead. (However computers are much faster than humans, so we might have to slow it down a bit, once we add code for this.)
- First, let’s add some code to make sure that only one point is scored for every pair of matching cards. We will need to reset the ‘Snap!’ button after a point has been added to either score, so that it only works once, and we can do that by making another variable – let’s call it ‘reset’. Your whole list of variables should now look like this:



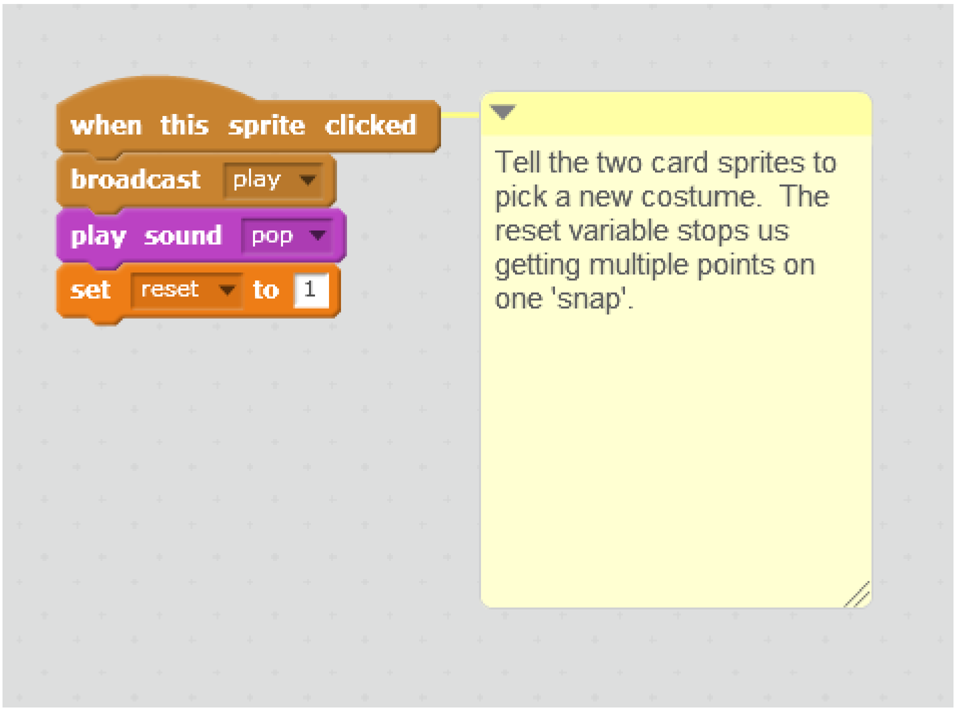
- ‘reset’ is the type of variable that coders sometimes call a ‘flag’, because it will only have two values. A flag can be flying from a flagpole, or down on the ground, but on a computer, those two values are usually ‘1’ and ‘0’. We will use ‘0’ to mean that the ‘Snap!’ button is turned off, and ‘1’ to mean that it is working.
- Tick the box to show the reset variable on the stage while you are testing this bit of code – we will hide it at the end.

- Add this block to the backdrop to initialise the reset variable at the beginning of the game, along with all the other variables:



```
set [reset] to [0]
```

- So, we start the game with the 'Snap!' button not working. This is fine, because you can only see the backs of the cards, and that does not count as a Snap!
- We want the 'Snap!' button to be made available every time the cards are changed, and the cards are changed by the 'Play' button, so that is where we add the next bit of code.
- Raise the flag (set the 'reset' variable to '1') on the 'Play' button sprite, so that the code looks like this:



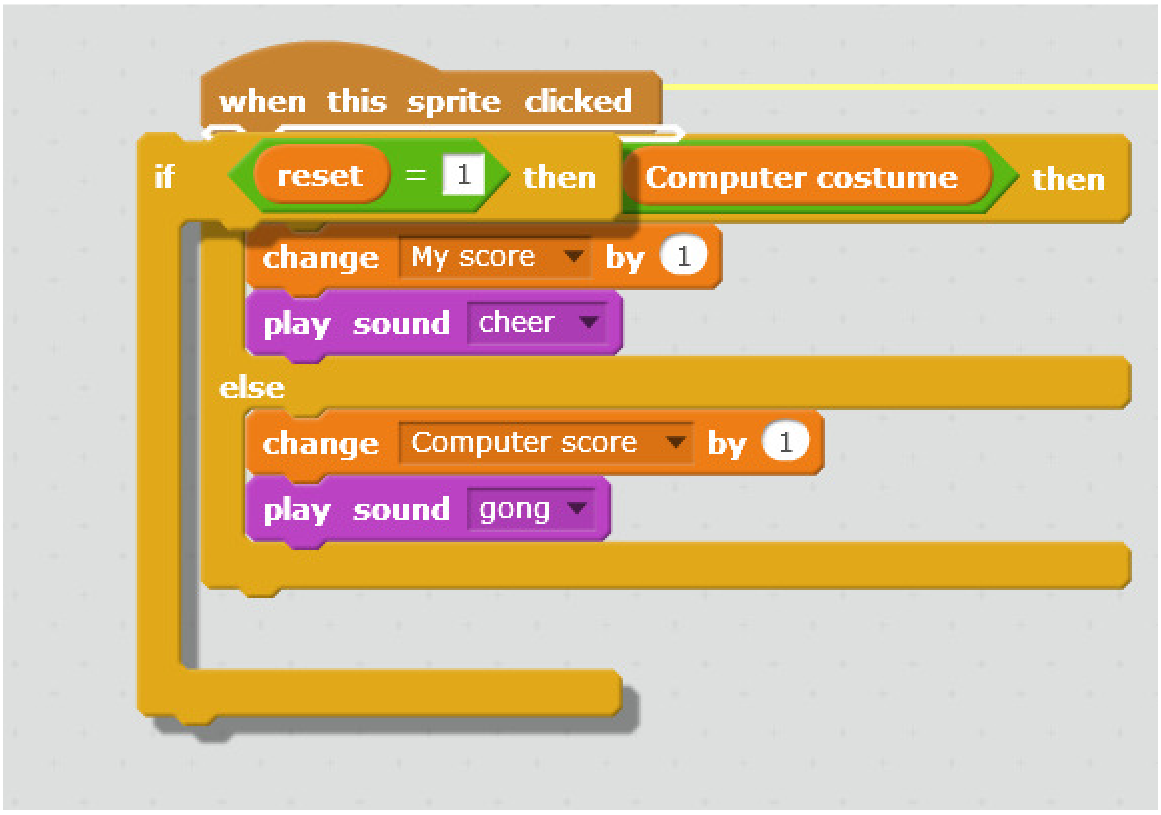
```
when this sprite clicked
broadcast [play]
play sound [pop]
set [reset] to [1]
```

Tell the two card sprites to pick a new costume. The reset variable stops us getting multiple points on one 'snap'.

Save your project

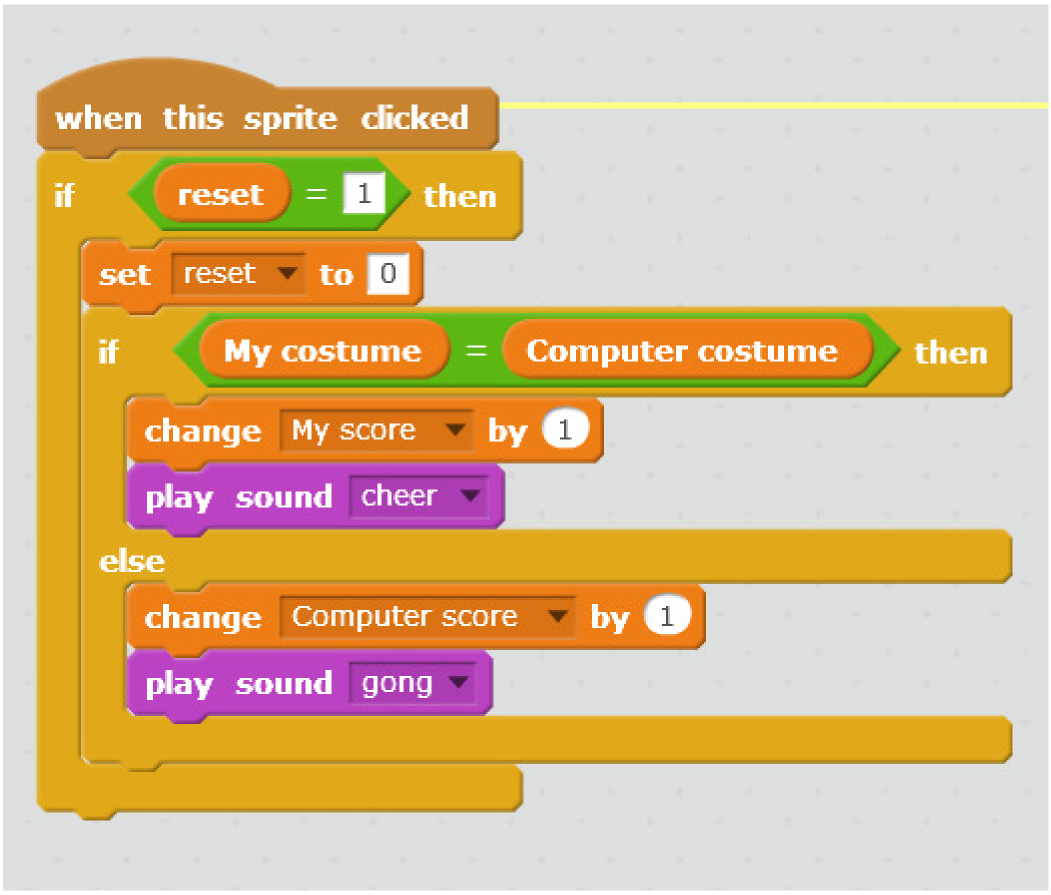
* Finally we need to add code to our ‘Snap!’ button, to only give a point if ‘reset’ is set to ‘1’. We also need to lower the flag (set ‘reset’ back to ‘0’) when we are going to give a point – this makes sure that only one point is given for each pair of matching cards.

* So, the code on your ‘Snap!’ button needs an ‘if’ block to go around all the code we already have (you can see how I added this below):



```
when this sprite clicked
if reset = 1 then
    if Computer costume then
        change My score by 1
        play sound cheer
    else
        change Computer score by 1
        play sound gong
```

* And the ‘set reset to 0’ block goes inside that ‘if’ as well. You should end up with the code below:

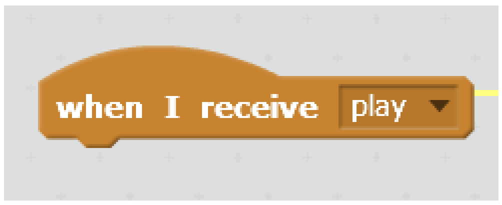


```
when this sprite clicked
if reset = 1 then
    set reset to 0
    if My costume = Computer costume then
        change My score by 1
        play sound cheer
    else
        change Computer score by 1
        play sound gong
```

- Test it out! If you can see the 'reset' variable, it should start as zero at the beginning of the game, then change to one each time you click the 'Play' button, then change back to zero each time you click 'Snap!'.
- If your code is correct, the 'Snap!' button will only work once each time the cards change – after that it will do nothing. But it should always work once, whether the cards are the same or different.
- Once you are happy that this is working, you can hide the 'reset' variable (clear the tickbox next to it).

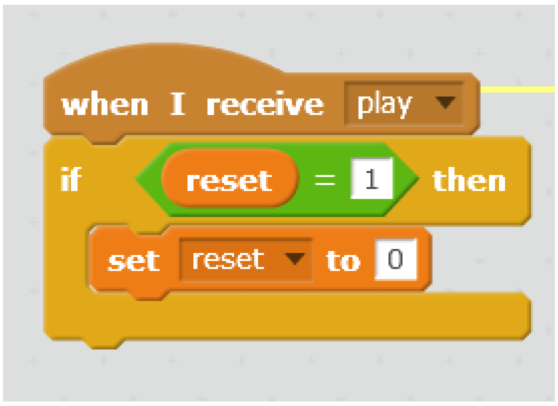
Save your project

- Now let's fix the fairness problem – we want the computer to have a chance to call Snap!, and steal the point we were hoping to get!
- Still on the 'Snap!' sprite, we are going to write some code that runs every time the cards change. We know the message 'play' is broadcast when the cards change, so we can use that to start:



```
when I receive [play ▼]
```

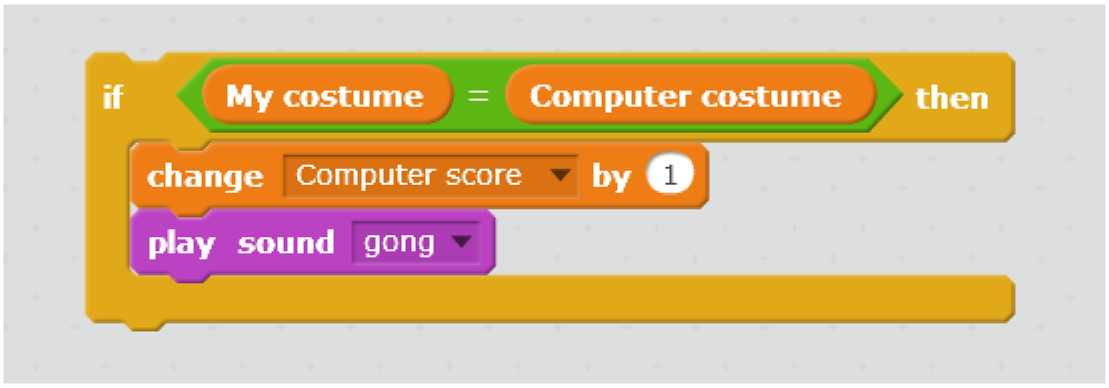
- Let's think this through. We only want the computer to get a point if we haven't already clicked the 'Snap!' button; we only want it to get one point for each pair of matching cards; and we don't want to be able to click the 'Snap!' button and claim a point, after the computer has already claimed it.
- This sounds quite like the restrictions we have already put on the 'Snap!' button, so thankfully we can reuse the 'reset' variable to do this part of the job for us – copy just this part of the code so that your new code block looks like this:



```
when I receive [play ▼]
if <(reset) = [1]> then
    set [reset ▼] to [0]
```

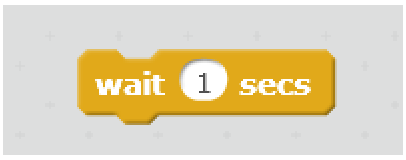
- The 'reset' variable is showing its true value here – we could have just hidden the 'Snap!' button after clicking it, in the code above, and that would have worked up to this point (but not after).

- Why don't we just copy the 'if-else' block that adds points to the scores into this code block as well – it is doing the same sort of job, isn't it? Do you know why?
- We used an 'if-else' block in the 'when this sprite clicked' code on the 'Snap!' button because there were two possible outcomes – we might be right to say Snap! because the cards were the same, or we might be wrong, because they were different. I decided that I would give the computer a point when I made a mistake. Do we need to write code for when the computer makes a mistake?
- The computer is not going to make a mistake, so there is only one outcome we need to code for – if the cards are the same the computer gets the point. [By the way, do NOT believe that computers are always right – they are programmed by people, and people can (and do) make mistakes!]
- Add the code below inside the if statement, in your 'when I receive play' block (you can copy individual blocks from 'when this sprite clicked'):



```
if <(My costume) = (Computer costume)> then
    change [Computer score] by (1)
    play sound [gong]
```

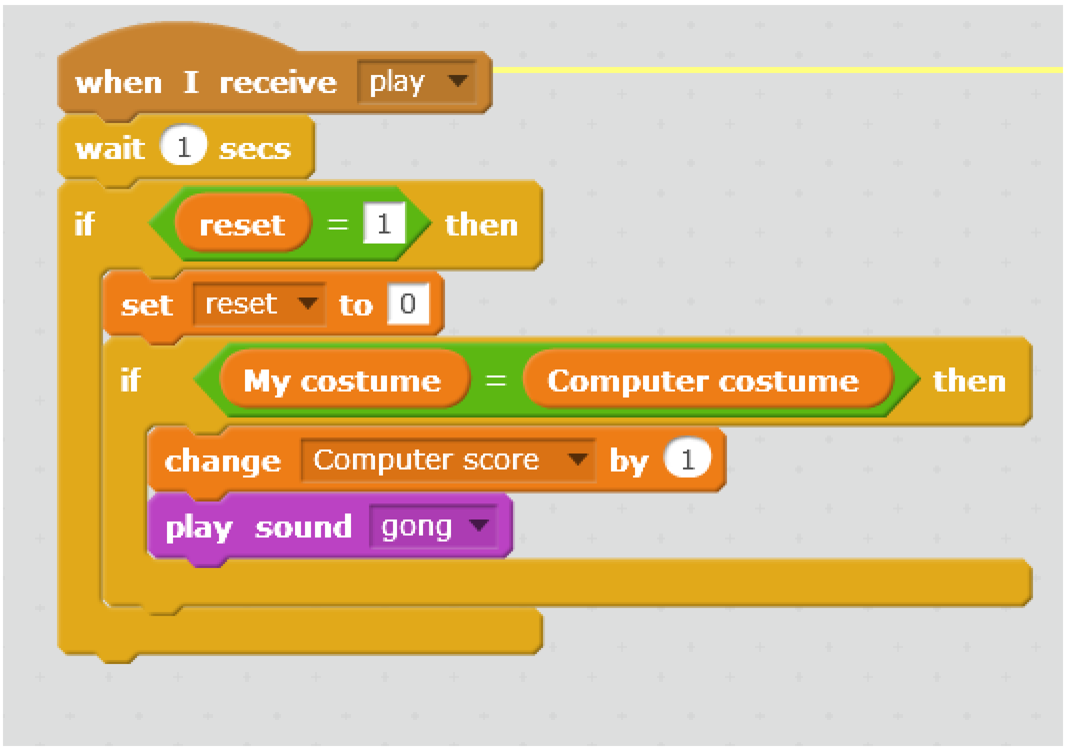
- We also need to slow the computer down a bit, so that we stand a chance to get the Snap! before it does. I used the wait block at the beginning of this code:



```
wait (1) secs
```

Save your project

- Your finished block of code should look like this:



```
when I receive [play ▼]
wait (1) secs
if <(reset) = [1]> then
    set [reset ▼] to [0]
    if <(My costume) = (Computer costume)> then
        change [Computer score ▼] by (1)
        play sound [gong ▼]
```

- Try it out! You need to test all the possible situations:

  1. Cards are the same; I click Snap! quickly
  2. Cards are different; I click Snap! (quickly)
  3. Cards are the same; I don't click Snap! (or I am too slow)
  4. Cards are different; I don't click Snap!

- And the results you should get are:

  1. I get a point
  2. Computer gets a point
  3. Computer gets a point
  4. No one gets a point.

- You might want to change the wait time to 2 seconds, or longer, if the computer always wins the point. You might want to change it to 0.8 seconds, or even shorter, if you always win the point!

Save your project

- Challenge your family and friends to play the game you have written! Can they beat the computer? Can they beat your highest score?
- And well done for getting this far!

More ideas

- You could add a timer to make the game more exciting – who can get the most points in 30 seconds?
- Or, you could choose a winning score (maybe 10 points), and the first player to reach that score is the winner (play a fanfare! change the backdrop! use some graphic effects!)
- How about adding another backdrop as a menu screen, that is displayed before the game? You could use this screen to ask the user to enter the time the game lasts, or the winning score, before they start playing.
- Some packs of cards have a very fancy design on the back, but ours is very plain. Can you design a really beautiful picture for the back of your cards? Don't try and do it twice though – once you are happy with your design on one sprite, you can drag that costume over the other sprite, and it will make a copy. You will need to drag the copied costume to the top of the costume list, and delete the old 'back' costume, so that the code still works.
- We only used four designs on our cards, but you could add many more – how about square cross, diagonal cross, five-pointed star, pentagon? Or you could try hearts, diamonds, clubs and spades; or double up: two circles, two squares, and so on. What changes do you need to make to your code to use these extra designs? (Answer: only a couple of very small changes!)

Save your project