Rock Paper Scissors Teams

Massively multi-player rock paper scissors!

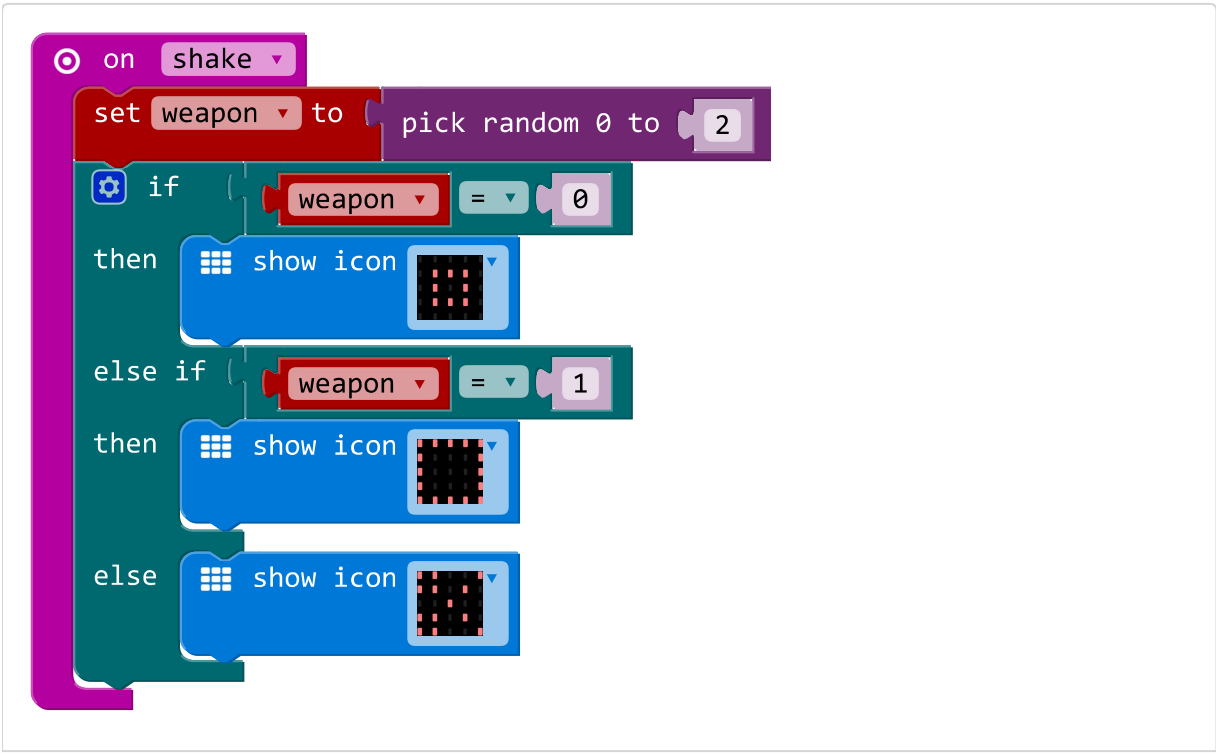
Playing rock paper scissors is usually a two player game… but it works with much more players too! When playing with more than two players, it becomes a team game: all players shake at the same time, then the number of **rock**, **paper**, **scissors** is tallied between all the players. The team with the most players wins the game.

Starting from the **basic version of the RPS game** (/projects/rock-paper-scissors) , we are going to change the code so that the micro:bit counts and displays the number of players in the same team. Using the **radio** communication, the micro:bit will send its status and receive the status of other boards.

Let's get started!

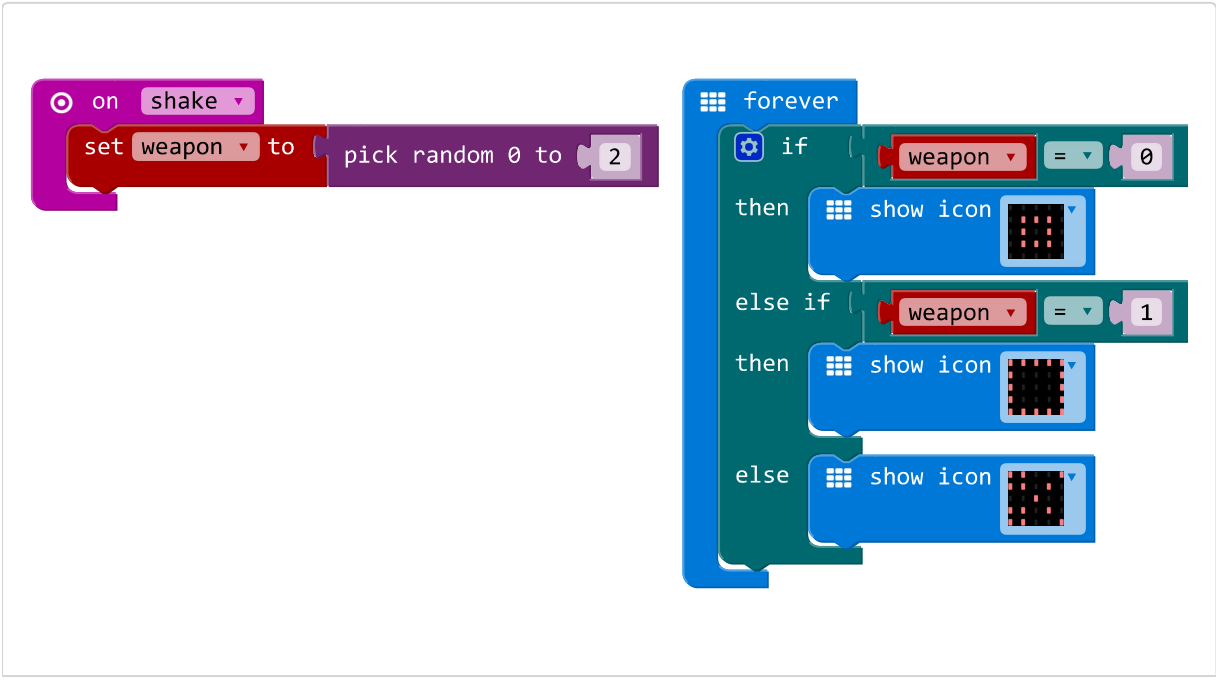
Starting blocks

Let's start form a . The basic version picks the weapon in a on shake event and display an icon accordingly. Take a peek at the code below to refresh your memory on that game.



Step 1: Refactoring the rendering

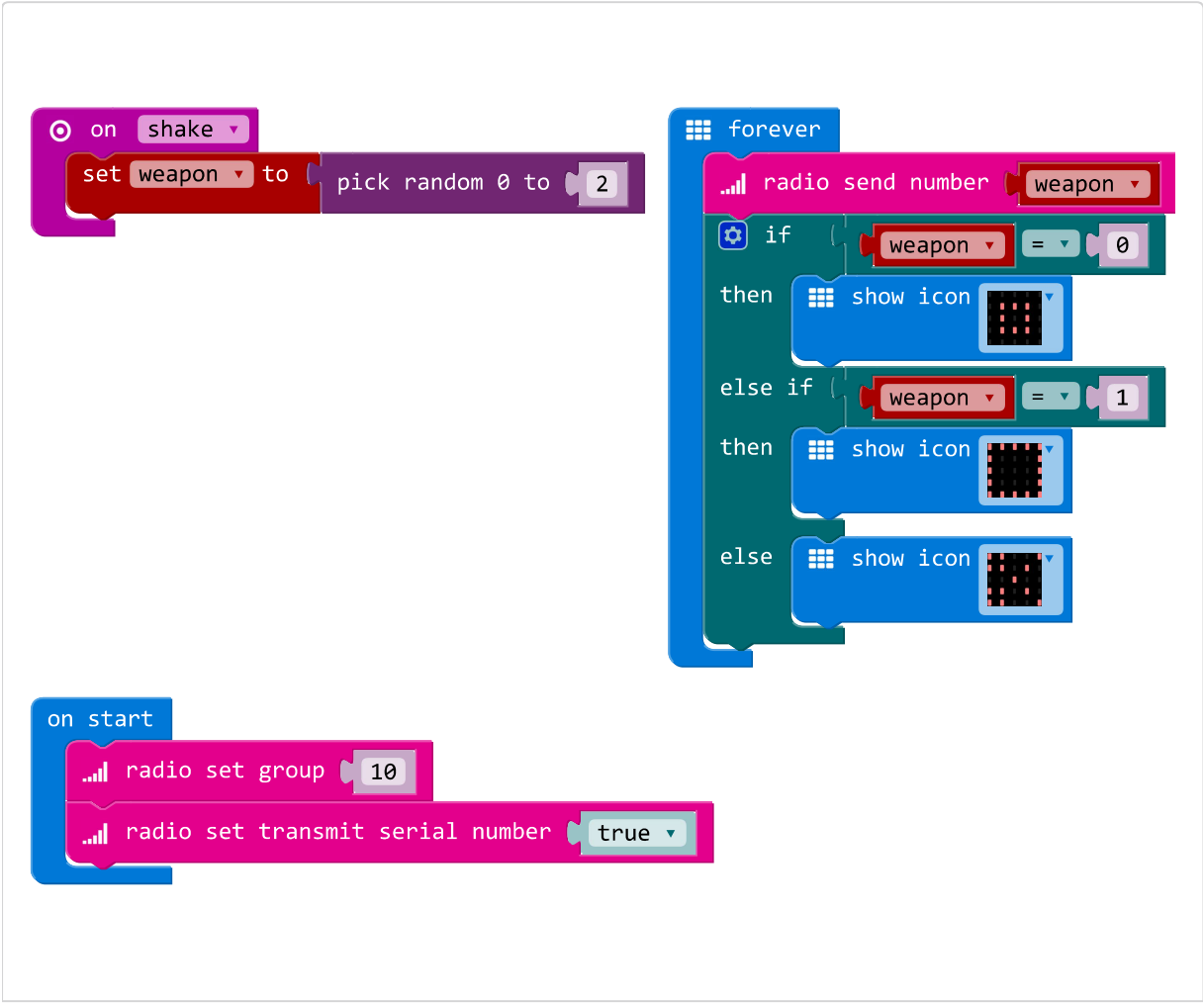
Refactoring is a funny word used in coding which pretty much means `reorganizing`. In this case, we are going to move the code that displays the rock/paper/scissor icon into its own forever loop.



Step 2: send status via radio

We send the value of `weapon` via radio to other micro:bit in the forever. Since radio packet may or may not arrive, it's a good idea to keep sending them.

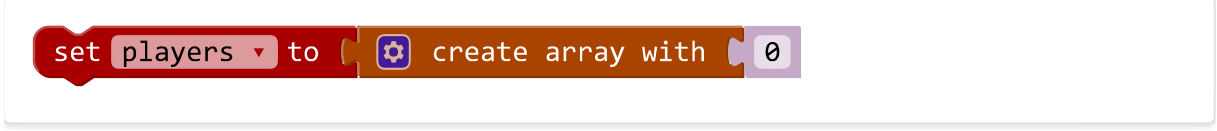
We also set the radio group and send the device serial number (a number that uniquely identifies a micro:bit) as we will need that later.



Step 3: the team roster

So all players are constantly broadcasting which face they picked to other players. Let's add the code that receives those status and counts them.

We are going to add an **Array** variable that contains all the players in the same team as the current player. That array, named `players`, is like your team roster: it contains the list of micro:bit serial numbers that have the same weapon as you.

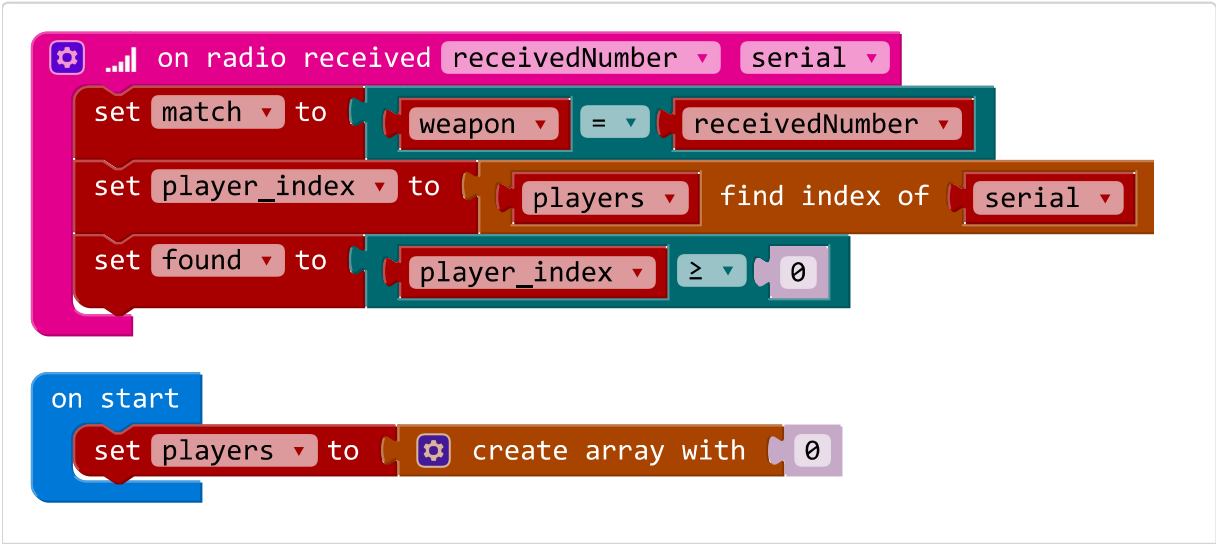


Step 4: Receiving a message (part 1)

In an the on radio received event, we receive the status from another micro:bit. Click on the *gearwheel* to add the `serial` parameter as we will need it to identify who sent that packet.

We compute three values from the data received:

- `match` , a boolean value indicating whether the weapon of the other micro:bit matches our current weapon
- `player_index` , the position in the array of the other board's serial number. It will be -1 if it is not in the array
- `found` , a boolean value indicating whether the micro:bit serial number is part of the `players` array

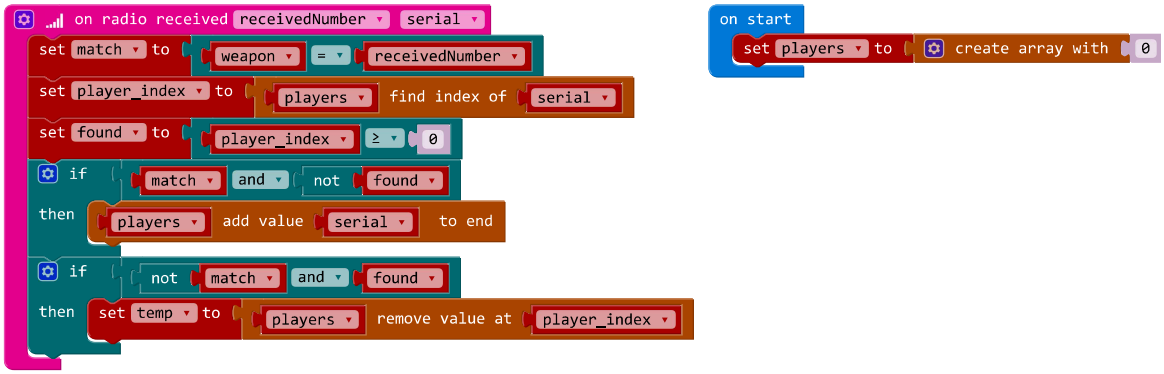


Step 5: Receiving a message (part 2)

There are two cases that we need to handle when looking at `match` and `found` :

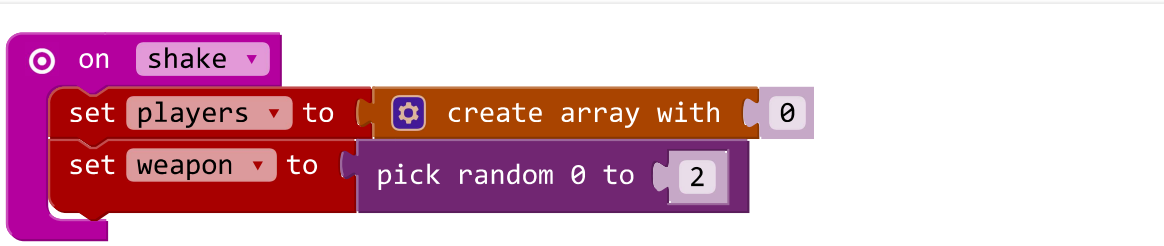
- **if** we have a `match` **and** the player is `not found` in the list, **then** we **add** it to `players`
- **if** we don't have a `match` **and** the player is `found` in the list, **then** we **remove** it from `players`

We turn the two rules above into two `if` statement where the serial number is added or removed.



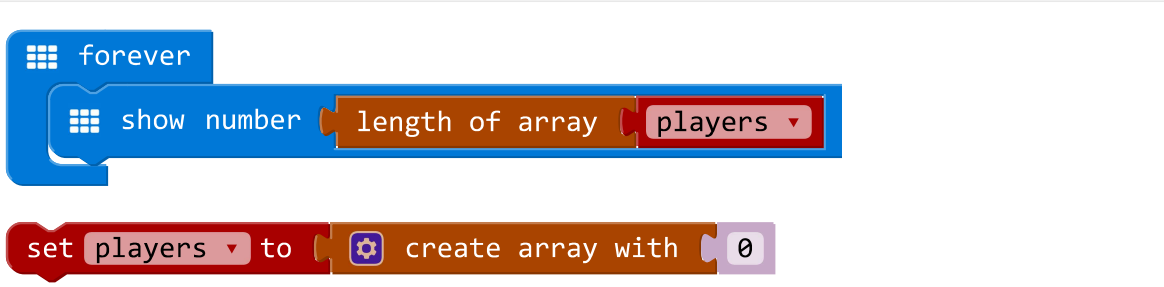
Step 6: reseting the team

What if another player would leave the game? He would stop broadcasting its status and it would stay in our list of players. To avoid this problem, we reset the `players` array each time we shake:



Step 7: showing team score

The team score is the number of players in that team... which boils down to the `length` of the `players` array. We add a show number block in the forever loop to display it.



The final code

Now it’s time to glue all together the pieces of our program. Go carefully through all the steps and assemble the various features. Eventually, it should look like the following program. Download it and plays with your friend**ssss**!