NetsBlox Lesson: Multi-Player Games
21 Pebbles

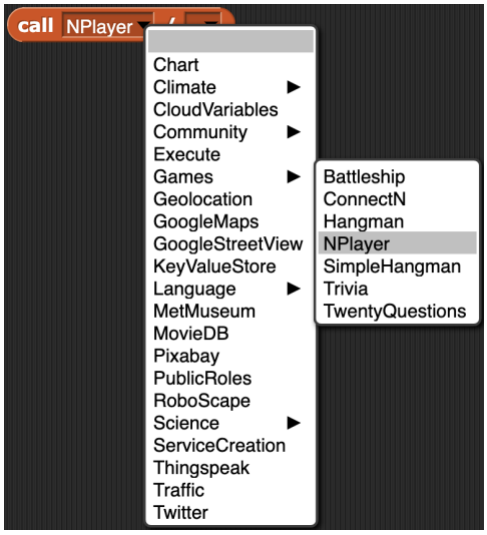
As we have seen in previous lessons, message passing makes it possible for NetsBlox projects to communicate with each other. That means that we can create distributed multi-player games where each player runs a project (or a role) on their own computer. In this lesson, we'll focus on turn-based games. Most games before the computer era were turned-based, such as chess, other board games and most card games too. It seems like a good place to start.

Turn-based Games

From a programming point of view, the biggest challenge when implementing a distributed turn-based game is maintaining whose turn it is and who comes next. The main reason for the difficulty is that all roles run on different computers and they have to agree on the state of the game. To make this easier, NetsBlox comes with a service called **`NPlayer`** that does exactly that. It has a number of RPCs and it also sends messages to roles when the game starts and when it is their turn.

We are going to start with a dummy game to see how the **`NPlayer`** game service works before shifting our focus to the real game, 21 pebbles that we are going to build.

First, let's see what RPCs the **`NPlayer`** service has. You can find it under the Games submenu of the first pull-down menu of the **`call`** block in the Network tab:

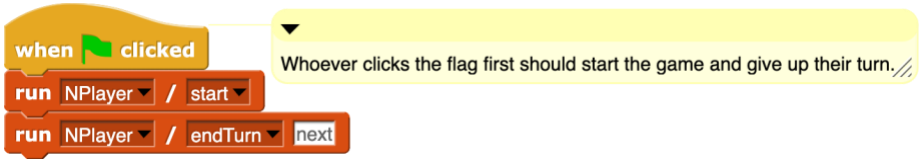


The two most important RPCs shown above are the **start** and the **endTurn** calls. The rest can be useful in some cases, but are not required to create a game. The **getActive**, **getNext** and **getPrevious** RPCs simply return the role names of the current, the next and the previous roles based on the order of turns. The **getN** RPC returns the number of roles, that is, players. We are not going to use these today.

The **start** RPC starts the game. Once it is called, the server sends a **start game** message to every role. Whichever role calls the **start** RPC has the current turn.

The **endTurn** RPC needs to be called by the role that has the current turn. It returns **true** if it succeeds which basically means that the correct role called it. If another role does, the RPC returns **false**. Once the correct role calls the **endTurn** RPC, the server sends a **start turn** message to the next role. The **endTurn** RPC has an optional input argument called **next**: the name of the role which should have the next turn. In most cases, we leave this blank and let the server decide whose turn it is. The server simply uses a fixed order as you do when you play a card game and you go around the table.

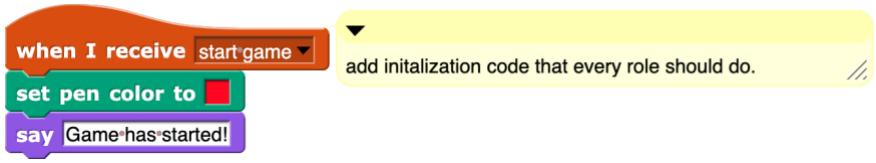
To start the game, one of the players should click the green flag, and that same role needs to call the **start** RPC and immediately give up its turn:



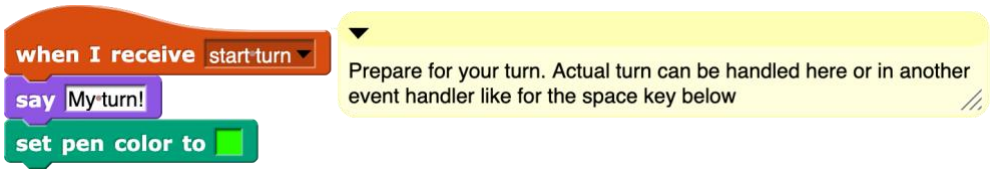
The server will send a **start game** message to every role including the one whole called the **start** RPC, followed by a **start turn** message to one of the roles, specifically the one that comes after the one who just started the game and ended its turn.

First, we need to make these two message types that the server is using. Click the "Make a message type" button under the Network tab. The first message type should be called "**start game**" and it has no data fields, so click the left arrow to delete the one it comes with. The second message type is "**start turn**" and it has no data fields either.

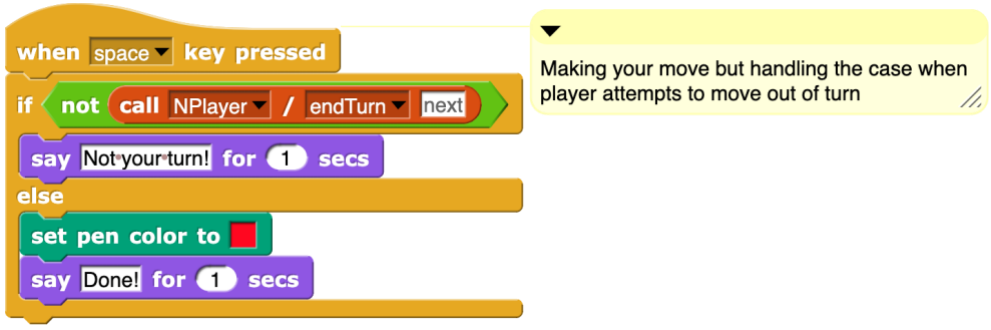
Now we are ready to implement the **when I receive start game** message handler. In this game, we will simply show whose turn it is by turning the sprite green, waiting for a space key press, turning red and passing the baton to the next player. The **when I receive start game** message handler should carry out all the initialization that every role needs at startup:



In our case, we turn red and display a message. The **`when I receive start turn`** message handler needs to prepare for the turn:

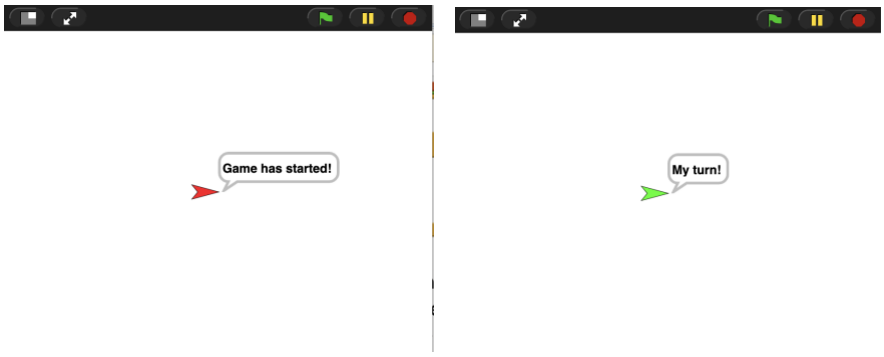


In our case, we turn green and display that it is our turn now. The only thing left is to write the script that handles the space key press:



First, we need to make sure that when the space key is pressed it is indeed our turn. This prevents playing out of turn. Fortunately, the server will let us know by returning **`false`** after calling the **`endTurn`** RPC. In our case, we simply display a warning message. Otherwise, it was our turn, so we turn red and display that we are done with the turn.

Make sure to save the role and in the Room tab, click on the role (not its name) and select "Duplicate." Do this at least twice to have three or more roles altogether. Rename the roles by clicking on their name if desired. Start NetsBlox in additional browser windows and invite yourself to the game by clicking on a role and picking "Invite User" and then selecting "myself." Position the browser windows next to each other and click the green flag in one of them. Hit the space bar in the various browser tabs and see what happens.

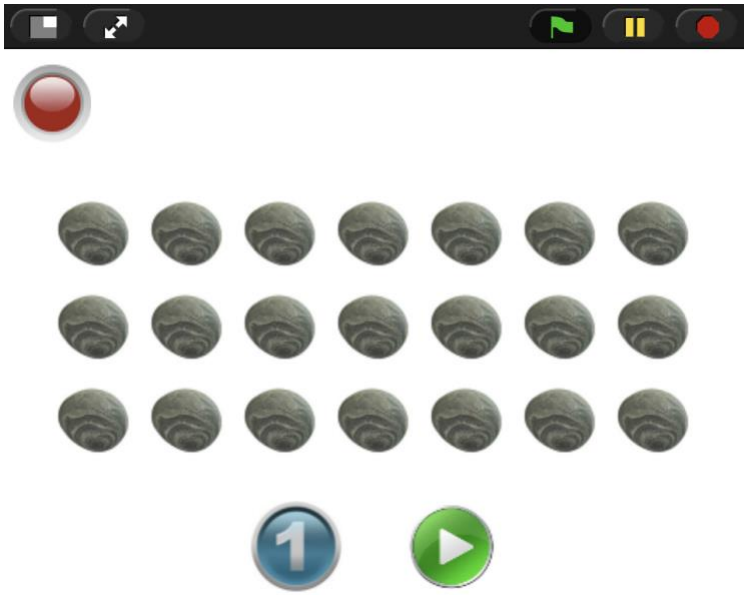


Here is our very first, but pretty simple distributed multi-player game:

https://editor.netsblox.org/?action=present&Username=ledeczi&ProjectName=DummyGame&

21 Pebbles

The game of 21 pebbles is played by two players who take turns of removing at least 1 and at most 3 pebbles from a single pile of 21 pebbles. The person who removes the last pebble loses the game. Here is how the stage looks like:

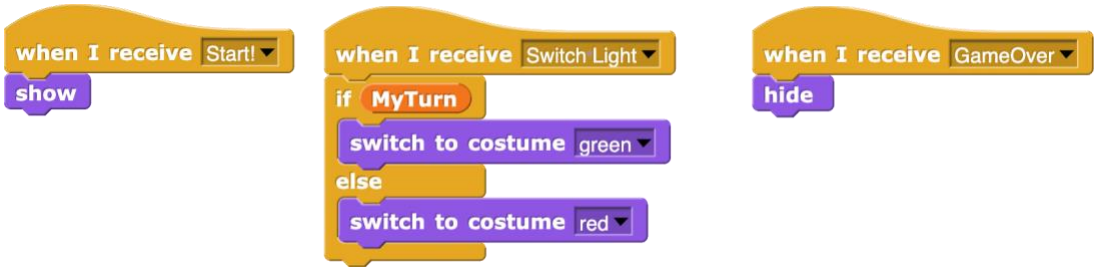


The circle sprite in the top left corner is red when it is not our turn and green when it is. The blue circle with the number helps us pick how many pebbles we want to remove. You click on it to change the number between 1, 2 or 3. The green arrow sprite is to be clicked when we want to make our move. The pebble sprite is actually not visible most of the time, we simply use stamping to show how many are left on the stage.

Since our focus is on how to make turn-based games, we provide you with a shell that has all the costumes and much of the game-specific code already implemented. But we left out the crucial pieces related to the distributed aspects of the game. Here is the shell:

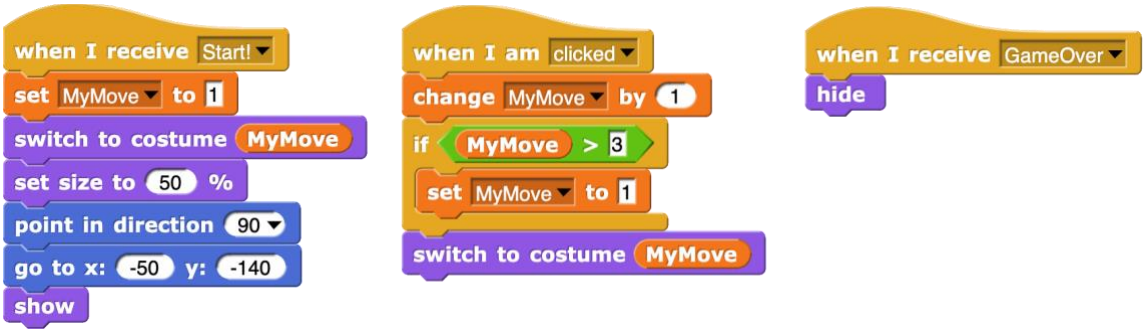
https://editor.netsblox.org/?action=present&Username=ledeczi&ProjectName=VDN%2021%20Pebbles%20Shell&

Let's look at the sprites one by one. Here is the script for the red/green light:



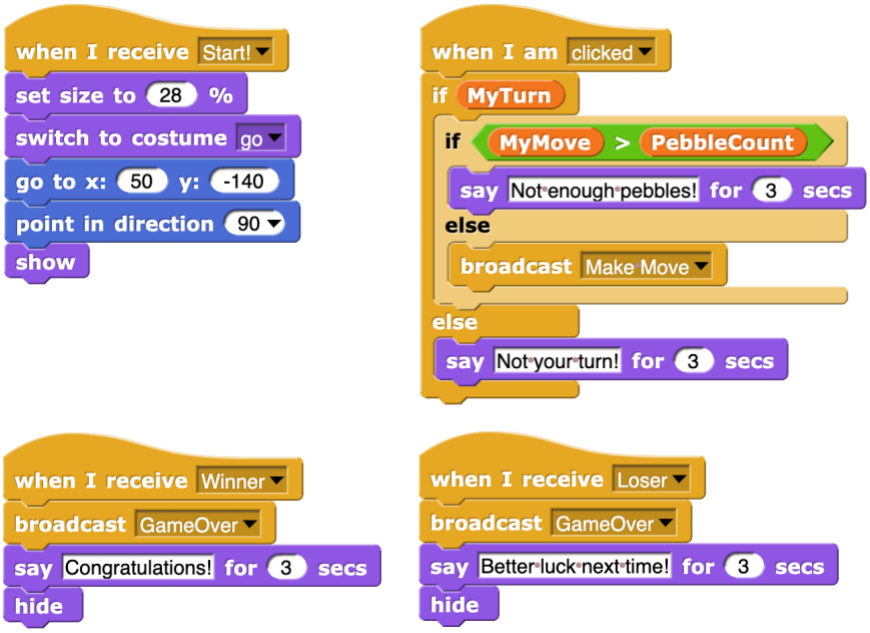
We hide when the game is not active, and show when it is. Whenever we receive an event called **`Switch Light`**, we display the correct color based on the variable visible to all sprites called **`MyTurn`**.

Here is the code for the Number sprite:



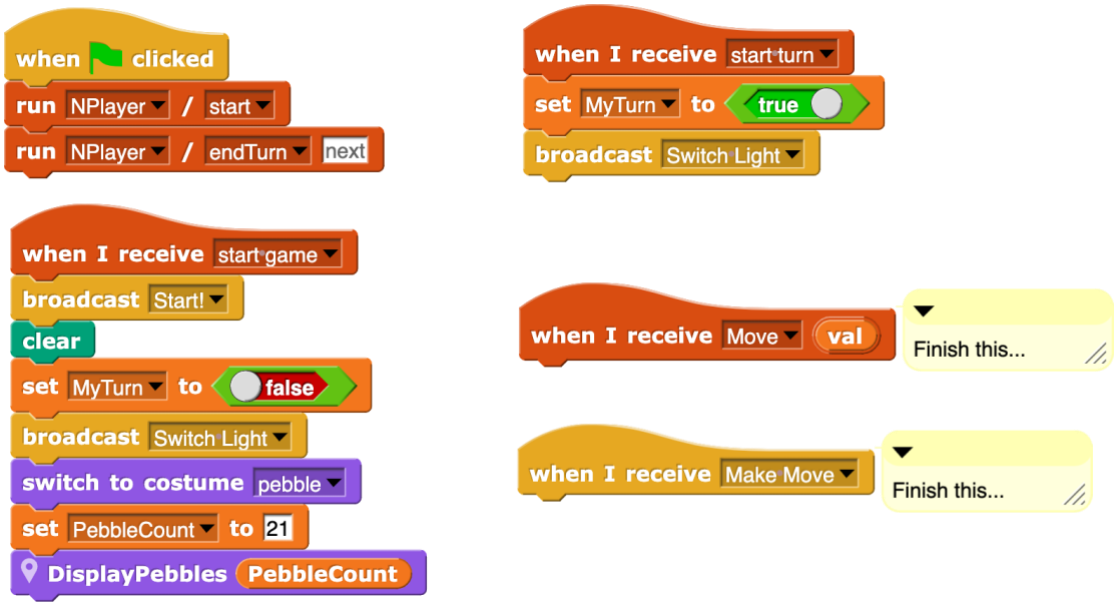
The initialization code is pretty simple. When we receive the **`Start!`** event, we set the **`MyMove`** variable to 1, shows the corresponding costume and place the sprite in its desired position. The **`when I am clicked`** event handler switches the numbers around, from 1 to 2, from 2 to 3 and from 3 back to 1.

The green arrow, that is, the Go sprite, is the one that provides messages to the user in addition to initiating our own move:



The only non-obvious script here is the **`when I am clicked`** event handler. First of all, it checks whether it is our turn and provides a warning message if it is not. If it is our turn, it checks whether we are trying to remove more pebbles than what's left in the pile. If everything is fine, it broadcasts an event, **`Make Move`** which is handled by the Pebble sprite.

It is the Pebble sprite that is the most complicated and the one that is incomplete.



The **`when green flag clicked`** code should look familiar: we start the game and end our turn immediately. The **`when I receive start game`** script does all kinds of initialization. First, it broadcasts **`Start!`** to the other sprites. It sets **`MyTurn`** to false. It tells the light to set the color to

red. It makes sure the costume is the pebble one. It sets the **PebbleCount** variable to 21. And finally, it calls the **DisplayPebbles** custom block that displays the current number of pebbles in up to three neat rows.

The **when I receive start turn** script sets **MyTurn** to **true** and asks the light to change colors.

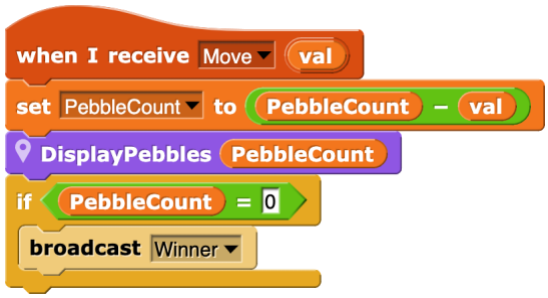
So, what's left? Making the move and handling the case when the other player has made a move. How do we do that? Well, we have a new message type called **Move**. We need this. The **NPlayer** service only deals with the turn-based nature of the game. The actual game-specific logic is still our job. And it involves sending a message with the number of pebbles we have just removed. So, the new **Move** message has one field called **val** that contains a number with the value of 1, 2 or 3.

Why don't you stop reading here and try to finish the project by yourself? The yellow **when I receive Make Move** script runs when the Go sprite was clicked by the current player. What do we need to do here?

The brownish **when I receive Move** script will run when the other user finished their turn and let us know how many pebbles they removed. What do we need to do then?

Take your time! Once you are done, turn the page!

Let's start with the code that handles the case when the other player moved because it is simpler. We have just learned that **`val`** number of pebbles have been removed. So, we subtract **`val`** from **`PebbleCount`** and refresh the screen showing only the remaining pebbles. Then we check: if there are no more pebbles left, we are the winner. So, we let the other sprites know. For example, the Go button will display a congratulatory message and then hide.



What about making our own move?

Most importantly, we send a **`Move`** message to the other player with our move. Note that if we use the destination **`others in room`** then both roles of the game can be the exact same. If we used a specific role name, we would need to modify it in corresponding script in the other role.

Then we update the **`PebbleCount`** variable and show the correct number of pebbles left. We also need change the **`MyTurn`** variable and turn the light red. Then we call the **`endTurn`** RPC. Finally, it we ended up with no pebbles, we have just lost the game. If that is the case, we notify the other sprites.

Here is the final project:

https://editor.netsblox.org/?action=present&Username=ledeczi&ProjectName=VDN%2021%20Pebbles&