BP Project: Haim Ben Joya & Gidi Bar Ilan, BGU university, CS. April 2018:

**Introduction:**

In our third year as CS students, we took the IOT course which led us to meet in the first time the BP.js engine, which has been developed by Michael Bar Sinai. This engine let Dr. Gera Weiss and Dr. Achiya Elyasaf develop few examples such as “hot\_cold” and “tic\_tac\_toe”, which you can read about in the GitHub of BGU\_Hack.

In the project we did with Dr. Weiss in IOT course, we developed a Tic-tac-toe game in the environment suggested by IBM, which is called “Node\_red”. This project gave us the opportunity to implement the game from different aspects: We created code in a graphical way, meaning we used nodes to express our intents. For example, when we want the computer, which participate in the game as the “O” player, to put a “O” in the middle of the board, we created a node which was described as “Put O in (1,1)”. However, the catch is that we had to use JS code to get things work. So, although the Node Red tool enabled us to think more and code less, it still requires us to use our JS knowledge and to code few lines in each node.

With this experience, we came to our second project with Dr. Weiss. This time the idea was to develop our own tool which may let us in the future to develop things as we see them, even if in Node Red they are not possible. For example, one of the ideas that has been suggested is to use visual debug: When the user runs the program with the tool, in each time node is executed, it will be marked with some color. Another idea was to increase and decrease as we want the number of inputs and outputs from and to each node.

So, indeed Dr. Weiss and us have developed a tool in Java, using MX\_graph, which is actually a maven project which includes both the known and familiar BP.js (which has not directly been used in the Node Red project) and our new classes. Using Mx\_graph tools we drawed a Tic Tac Toe game with all the features we implemented in Node Red, but this time in the new tool. Now, we have gathered more experience and more things to think about while developing the game. We have been thinking all the time which nodes can be added to the interface of the user to support implementing Tic Tac Toe game with only really few lines of code. In this way, even a student of cognition will be able to develop a game or a simple or a complex program easily, without the need to read about JS and to use it.

In this paper we suggest few nodes which are possible to implement, so they let the user implement a tic tac toe game easily. We take to account that several users can play in the same time, and each player has his own game, independently from other users, meaning the server needs to include the whole information about everyone. Hope it will be both interesting and helpful for you.

Enjoy!

**Nodes:**

Notice that when we choose node to the graph, it will ask the users to fill required arguments.

Another note: A node can receive has arguments as variable from the flow (for example: if a node expects to receive I as an argument, it may receive it from a previous node)

Start game:

Start

Arguments: -

Description: This node will give Id. Id is an integer which represents a game. Each user who connects to the server play a different game, and this id will let us to classify each event to the appropriate game. So the node “Start” will create a new line in the data base which is kept in Context. From obvious reasons Id should be unique for each game, and technically it can be implemented in the following way: When a user connects to the server and N users have participated until now, Id will be N+1. From natural reasons the Id property should be kept in a smart way if we look at the synchronization aspect (we don’t want two games will get the same id): Using a lock/semaphore/more complex monitor will solve this issue for us.

Finally, “Start” node will create the line N+1 in the data base which is kept in Context using INSERT command in SQL.

Output: ID (so the other nodes will be aware on the current id)

|  |  |
| --- | --- |
| Id | Last Event |
| 1 | … |
| 2 | … |
| 3 | … |

Start game with configurations:

Start and configure

Arguments: Players names (List of String)

Description: This node is similar to “Start” node, but gives more options to the user. It reads the arguments and create the new line in the described data base in Context as follows: Id (unique integer), Players names. The idea here is to enable the users to play one against the other, and not one against the computer.

Technically, also here the command INSERT in SQL will be called with the relevant information.

Output: ID (so the other nodes will be aware on the current id)

|  |  |  |
| --- | --- | --- |
| Id | Players names | Last Event |
| 1 |  | … |
| 2 |  | … |
| 3 | Achiya, Haim | … |

Note: Players names in game #1 and game #2 are empty because those users that the game will be against the computer.

Pause game:

Pause game

This node will cause the BP system to block the events for this ID until “resume game” will be called.

Arguments: -

Resume game:

Resume game

This node will cause the BP system to unblock events for this ID, after the game paused earlier. Note: If “resume game” is used without the game paused earlier – nothing will happen.

Arguments: -

End game:

End game

Arguments: Winner (String – “X” or “O”)

This node will cause the immediate finish of the game, with the announcement (some nice text) on the winner of the game (using the String argument). Notice that this operation will cause the BP engine to stop select events, so no event will be chosen anymore.

Notice that we suggest that this operation will cause to DELETE (with SQL command) the relevant line from the database which is kept in Context. For example, if the “end game” happens in the game with id M, the line with M in the id will be deleted.

Before:

|  |  |  |
| --- | --- | --- |
| Id | Players names | Last Event |
| 1 |  | … |
| 2 |  | … |
| … |  |  |
| M-1 | Michael, GIdi |  |
| M | Gera, Jihad | … |

After:

|  |  |  |
| --- | --- | --- |
| Id | Players names | Last Event |
| 1 |  | … |
| 2 |  | … |
| … |  |  |
| M-1 | Michael, GIdi |  |

Get number of users:

Get number of users

Output: Number of ID which are shown now in the datebase which is held in the context.

Implementation: SELECT command in SQL (O(n)).

Arguments: -

Counter:

Counter

Arguments: Number of times (Integer)

The usage is probably obvious. If we want to do a operation several times, we don’t want to copy the node/the sequence of nodes. It is really not smart, and it is not elegant. With the “counter” feature one can look at the graph and realize what is going on.

We suggest that if the user put the node of Counter (and fill the number of times as argument) and connect it with a node/few nodes – they will happen the requested number of times. For this purpose, we will use regular arrow and special yellow arrow. The regular arrow will happen while the counter is smaller than Number of times. The yellow arrow will happen when the value of the counter is equal to Number of times. For example if we use Counter of 5:

The game will pause and resume 5 times, and after 5 times the number of the users will be received as output and the user can use it for its purposes.

Resume game

Pause game

Counter

Get number of users

Wait for I from group of J events and request the rest J-I events:

Wait for part of group, request the other part

Arguments: I (Number of events to wait for, an integer), List of events from which I will be waited and the rest will be requested, priority for the requested events (optional).

While implementing Tic tac toe we have faced several times the need to wait for two from three events, and then to request the third. This node will answer the general need to wait for some part of group of events, and then request the other to happen.

Example:

We want to develop a strategy which leads the computer (the “O” player) to complete a row full of “O”. For this purpose, for example we define that we should wait for O(0,0). Then, we wait for either O(0,1) or O(0,2). Imagine O(0,1) has happened – now we want, because of our strategy, to request the event: O(0,2). However, a similar scenario can happen: if O(0,1) first happen, then we wait for either O(0,0) or O(0,2). Imagine now that O(0,2) happened – now we want, because of our strategy, to request the event: O(0,0).

To avoid dealing with all this various cases we could just use “Wait for I from group of J events and request the rest J-I events” with the following parameters: I=2, J=3, and the group itself would be: O(0,0), O(0,1), O(0,2). Then in each described scenario – the requested event will be what we want.

Wait for group of events and request other group:

Wait for events from X and request the rest from Y

Arguments: A map of keys and values. Each key has at least one value. I (integer value) which define the number of keys the user ask to wait for) , priority for the requested events (optional).

The idea behind this node is that many times we want to wait for some specific events that are similar (keys), and according to the specific events that have happened, we want act accordingly and to request some other events which are similar (values). The motivation for it for us comes from Tic-Tac-Toe: There we defined a strategy which waits for 2 events of “X” in the same row/col/diagonal, and then requests to put an “O” in the last remaining cell in the same row/col/diagonal w.r.t.

For example, imagine the player put X(0,0) and X(0,1). Now we want to “block” him so the event O(0,2) will happen – as you see (0,2) is the remaining cell in the same row. For this example, we would use the node “Wait for events from X and request the rest from Y” with the following arguments: I = 2, and the map is:

{X(0,0) : O(0,0), X(0,1) : O(0,1), X(0,2) : O(0,2)}

One can think about this idea as following:

Wait for I events from “Keys”

Remove the events which happened

Request the “Values” from the remaining pairs in the map

Wait for group of events while blocking other group:

Wait for events from M while blocking group Y

Arguments: 2 lists: List M: group of events to wait for any of them, while blocking every event from List Y.

For example:

If we want to enforce turns in Tic Tac Toe, we would use the following nodes:

First node:

List M (wait for): X(0,0), X(0,1), X(0,2), X(1,0), X(1,1), X(1,2), X(2,0), X(2,1), X(2,2)

List Y (block): O(0,0), O(0,1), O(0,2), O(1,0), O(1,1), O(1,2), O(2,0), O(2,1), O(2,2)

Second node:

List M (wait for): O(0,0), O(0,1), O(0,2), O(1,0), O(1,1), O(1,2), O(2,0), O(2,1), O(2,2)

List Y (block): X(0,0), X(0,1), X(0,2), X(1,0), X(1,1), X(1,2), X(2,0), X(2,1), X(2,2)

And we will use an arrow between the nodes:

Wait for events from M while blocking group Y

Arguments:

List M (wait for): X(0,0), X(0,1), X(0,2), X(1,0), X(1,1), X(1,2), X(2,0), X(2,1), X(2,2)

List Y (block): O(0,0), O(0,1), O(0,2), O(1,0), O(1,1), O(1,2), O(2,0), O(2,1), O(2,2)

Wait for events from M while blocking group Y

Arguments:

List M (wait for): O(0,0), O(0,1), O(0,2), O(1,0), O(1,1), O(1,2), O(2,0), O(2,1), O(2,2)

List Y (block): X(0,0), X(0,1), X(0,2), X(1,0), X(1,1), X(1,2), X(2,0), X(2,1), X(2,2)

Wait for event from a group, request a corresponded event and block a corresponded event:

Wait for event from X, request from Y, and then block from Z

Arguments: A map of keys and pair with values. Each key has at least one pair of value, priority for the requested events (optional).

The idea behind this node is that many times we want to wait for some specific event , and according to the specific event that has happened, we want act accordingly and to request some other event (value), and in addition after that to block some other event (value). The motivation for it for us comes from Tic-Tac-Toe: There we wait for the event “click” and request “X” in the same position w.r.t., and according to the rules of the games to make sure no “X” or “O” will be put in this position afterward.

For example, imagine the player clicks on (0,0) we wait for that event and then we request X(0,0), and after the request happens we block both X(0,0) and O(0,0).

For example we use that dictionary:

{click(0,0): X(0,0): X(0,0),O(0,0) , click(0,1): X(0,1): X(0,1),O(0,1) , click(0,2): X(0,2):X(0,2),O(0,2) , click(1,0): X(1,0):X(1,0),O(1,0) , click(1,1): X(1,1) :X(1,1),O(1,1), click(1,2): X(1,2):X(1,2),O(1,2) , click(2,0): X(2,0): X(2,0),O(2,0) , click(2,1): X(2,1):X(2,1),O(2,1) , click(2,2): X(2,2):X(2,2),O(2,2)}

One can think about this idea as following:

Wait for 1 events from “Keys”

Request the corresponded “Value” from the map.

BP Node:

BP NODE

Arguments: “Request”: List of requested events, “Wait”: List of events we want to wait for, “Block”: List of events we want to block, “Priority”: Integer which indicates the importance we give to the event

Probably the most important node, because it enables the user to use all the properties of BP engine without coding anything.

For example:

If we want to use the following strategy for the computer (player O) to put O(1,1) in the center of the board in order to have good position for the win. Here we add priority 1 to this event so that from all the events that request to put an O on the board, which by default their priority is -1, the event that will happened will be the O(1,1), if there isn’t event with higher priority.

In order to achieve that strategy we will use this BP NODE with request O(1,1) and without any block or wait.

BP NODE

Arguments:

Request: O(1,1)

Block: -

Wait: -

Priority: 1

Creating variable node:

**Creating Variable**

Imagine we want to bring the same node each time different parameter. In the tic tac toe world you should remember we want to fill 3 in a row. However, we don’t want to create 3 nodes for completing a row of O, and 3 nodes for completing a column of O, and then for x, and then diagonal, exc… Therefore we want a node that will create a variable, for example, variable I which will be given the value “O”.

Arguments: name of variable, value of variable.

**Implementation of Tic Tac Toe with the nodes we described – see the attached file (“tic\_tac\_toe\_implemantaion.doc”)**

Note: start and creating variables are shown more than once but there is only one instance of them. The reason you see them more that once is because we didn’t have enough space to put it all in one A4 page.

We hope you have enjoyed reading our paper.

Thank you.