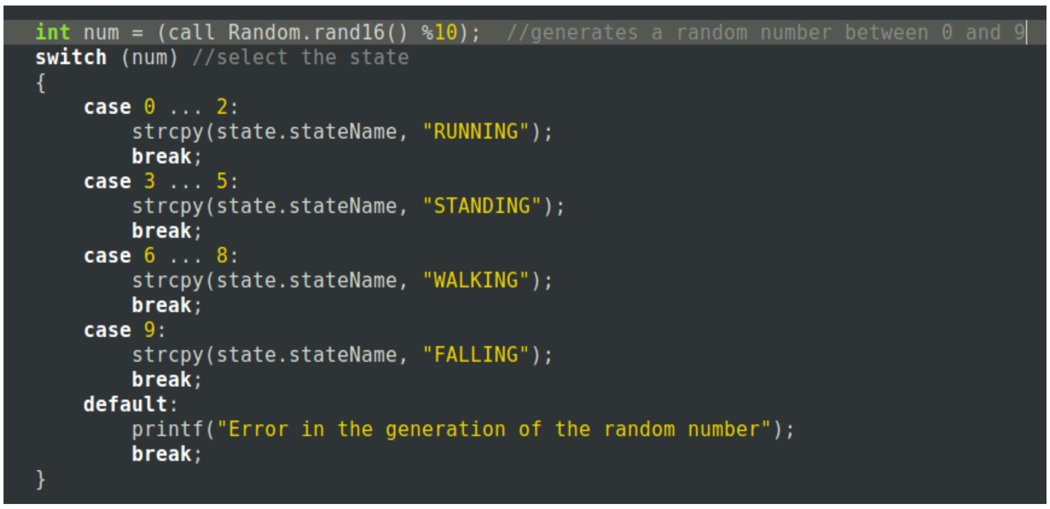
Lorenzo Vela 10593306
Giovanni Ghirardini 10806529

POLITECNICO
MILANO 1863

INTERNET OF THINGS: PROJECT SMART BRACELET REPORT

The goal of this project was to simulate two couples of smart bracelets talking to each other developing an application on TinyOS.
We've reused part of the code of the previous challenge, we list here the descriptions of the most important files.

- **FakeSensorP.nc**: Here is performed the generation of the data. We've used a random generator to determine the state and then with two other generations we generate the two coordinates.



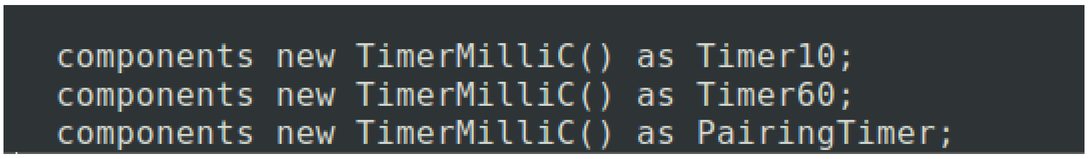
```
int num = (call Random.rand16() %10);  //generates a random number between 0 and 9
switch (num) //select the state
{
    case 0 ... 2:
        strcpy(state.stateName, "RUNNING");
        break;
    case 3 ... 5:
        strcpy(state.stateName, "STANDING");
        break;
    case 6 ... 8:
        strcpy(state.stateName, "WALKING");
        break;
    case 9:
        strcpy(state.stateName, "FALLING");
        break;
    default:
        printf("Error in the generation of the random number");
        break;
}
```

This is the switch that handles the state's selection.

- **SmartBracelet.h**: Here we've defined the two structs useful to exchange messages. The sensorState struct is for the message generated by the sensor while the sb_msg struct defines the message exchanged by the 2 motes.
We've 5 fields: **type** that indicates the sensor's phase. **Id** is the id of the message, it's intentionally never used because it was out of the scope of the project but we've implemented it anyway because of the best-practice approach. **Content** is the payload of the message, contains the key in the pairing phase and the state of the child in the operational phase. **X** and **Y** are the two coordinates.

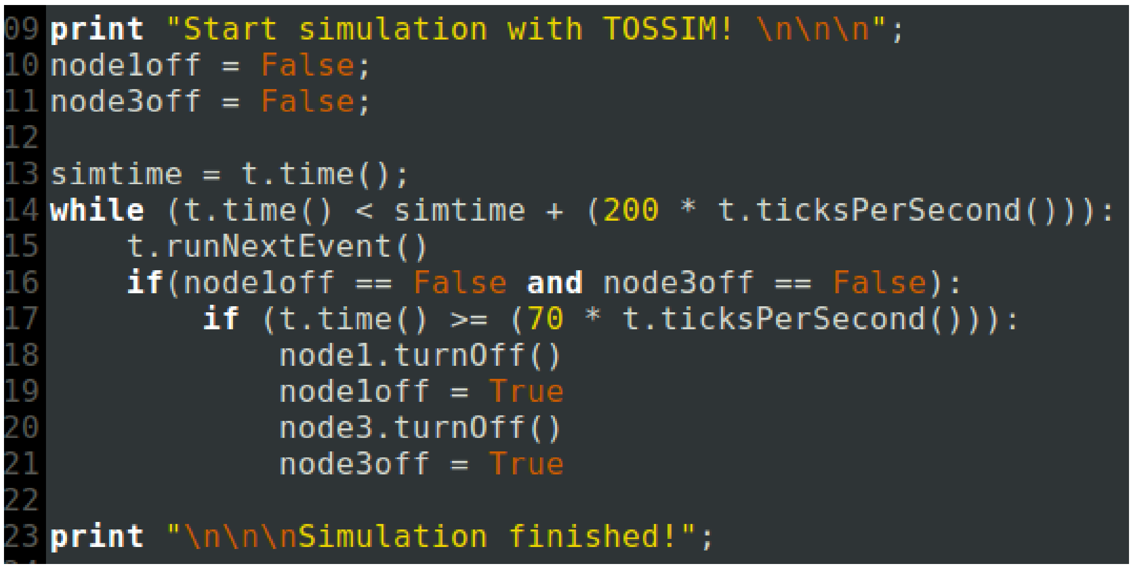
- **SmartBracelet.c**: Here is implemented all the logic of the project, the code in this file is clearly described after this section.

- **SmartBraceletAPPC.nc**: Here are listed all the components and interfaces for the system. The most relevant interfaces (leaving out the ones we've already used in this course) are the 3 different timers: one for the pairing phase, one of 10 seconds for the sensor's read and the third for stopping the simulation by turning off the parent's mote.



```
components new TimerMilliC() as Timer10;
components new TimerMilliC() as Timer60;
components new TimerMilliC() as PairingTimer;
```

- **RunSimulationScript.py**: We've just modified the last few lines of the code, by adding a control over the children nodes state (if turned on or off)

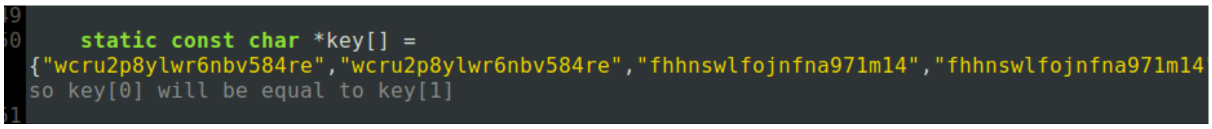


```
print "Start simulation with TOSSIM! \n\n\n";
node1off = False;
node3off = False;

simtime = t.time();
while (t.time() < simtime + (200 * t.ticksPerSecond())):
    t.runNextEvent()
    if(node1off == False and node3off == False):
        if (t.time() >= (70 * t.ticksPerSecond())):
            node1.turnOff()
            node1off = True
            node3.turnOff()
            node3off = True

print "\n\n\nSimulation finished!";
```

The implementation (file SmartBraceletC.nc)

Keys' logic

In order to guarantee the fact that 2 motes have the same key, we've just intialiazed an array of 4 char element: in the first and the second cell we've the first key (repeated twice) and in the cell number 3 and 4 the second key. This was done because the mote can retrieve its own key by simply accessing the array key[TOS_NODE_ID]. Mote 0 and mote 1 will so access 2 different elements but with the same key.



```
    static const char *key[] =
{"wcru2p8ylwr6nbv584re","wcru2p8ylwr6nbv584re","fhhnswlfojnfna971m14","fhhnswlfojnfna971m14
so key[0] will be equal to key[1]
```

The 3 phases

The value of the phase variable determines the behavior of the motes. They're all initialized with phase = 1 that is the pre-pairing phase. In this phase all the motes simply broadcast a message with their one key every time that the PairingTimer fires (periodically every 250ms). The first mote that receives the message executes the first "if" of the Receive function: it stops its own PairingTimer, it changes his phase in 2 (that means, I've found the other bracelet, now we should setup the connection) and calls the function PairingSucc() that handles the sending of the unicast message of confirmation. Now the second mote will always execute the receiving function, but not the same code of the first mote (because now the message is not in broadcast), so it will change its phase value in 2 and will stop its PairingTimer and will send a confirmation message to the other mote. When the two motes have the phase equal to two after the sending of the message, firstly they update their phase at 3 (operational) is performed the roles assignment by using the id of the motes. (0 and 2 will be the parent while 1 and 3 will be the child). Right after the role assignment every mote starts its own timer. (Children will start a periodic timer of 10 seconds while parents will start a periodic timer of 1 minute).
During the operational phase all the required information exchanges are performed.