UBFC
UNIVERSITÉ
BOURGOGNE FRANCHE-COMTÉ

Internet of Things (IoT)

Report of
"Leader election algorithm for modular robots"

subject « Modular robots course»

Performed by:
Student:
signature ___Bogdan Gorelkin
«___» _____________2021 y.

signature ______Sheikh Shah
Mohammad Motiur Rahman
«___» _____________2021 y.

signature _____Mona Rahbari
«___» _____________2021 y.

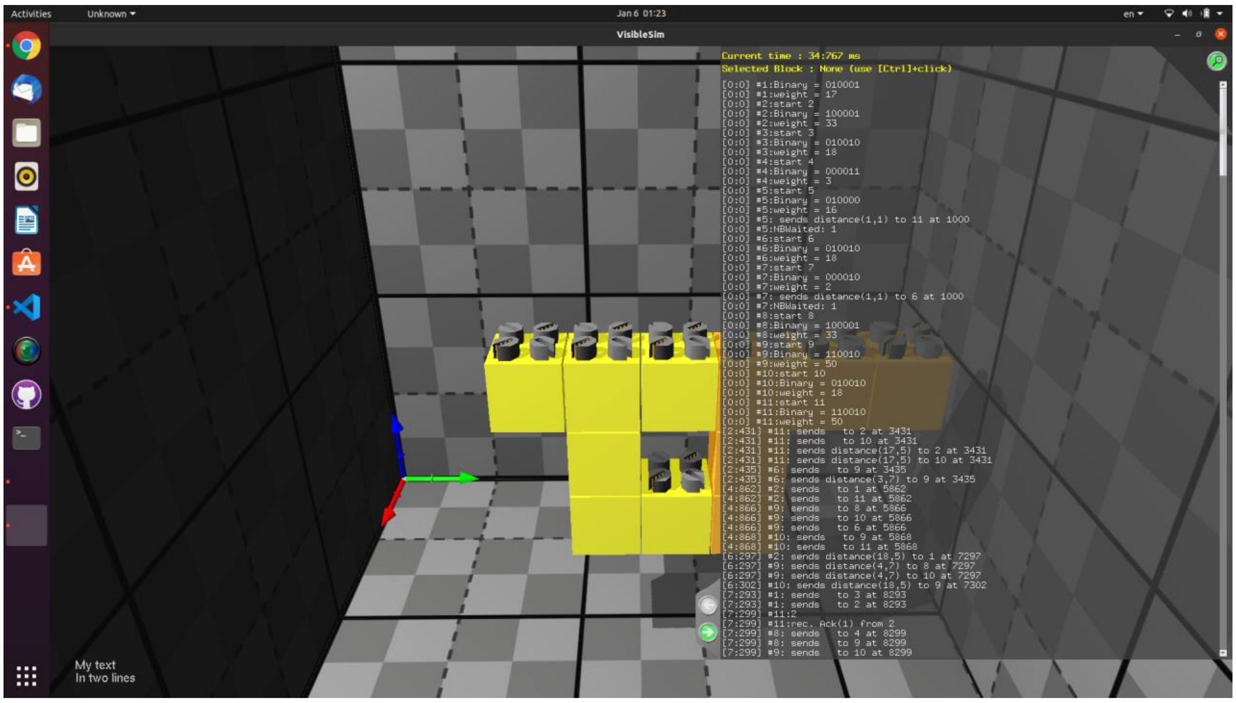
Checked:
Full Professor
Head of the Master Internet of Things
signature Abdallah Makhoul
«___»_____________2021 y.

Montbéliard 2021

1. Introduction

The algorithm is based in six phases. First, each module generates locally an integer, called weight based on the numbering of their connected interfaces. After that, spanning trees will be created using a recruitment method, then the values of the recruited modules will be calculated, the generated spanning tree will compete and the losing trees are dismantled, this step is repeated until one tree remains. And finally the leader is elected.

Computing the binary value and the weight of each block: For getting the number of interfaces we use getNbInterfaces() and generate a string by adding 1 and 0 based on the condition if connected or not respectively. After calculating the binary value, we converted the values to decimal to get the final weight value of each block. The output of this part is depicted in *Figure 1-a*.



```
Current time : 34:767 ms
Selected Block : None (use [Ctrl]+click)
[0:0] #1:Binary = 010001
[0:0] #1:weight = 17
[0:0] #2:start 2
[0:0] #2:Binary = 100001
[0:0] #2:weight = 33
[0:0] #3:start 3
[0:0] #3:Binary = 010010
[0:0] #3:weight = 18
[0:0] #4:start 4
[0:0] #4:Binary = 000011
[0:0] #4:weight = 3
[0:0] #5:start 5
[0:0] #5:Binary = 010000
[0:0] #5:weight = 16
[0:0] #5: sends distance(1,1) to 11 at 1000
[0:0] #5:NBWaited: 1
[0:0] #6:start 6
[0:0] #6:Binary = 010010
[0:0] #6:weight = 18
[0:0] #7:start 7
[0:0] #7:Binary = 000010
[0:0] #7:weight = 2
[0:0] #7: sends distance(1,1) to 6 at 1000
[0:0] #7:NBWaited: 1
[0:0] #8:start 8
[0:0] #8:Binary = 100001
[0:0] #8:weight = 33
[0:0] #9:start 9
[0:0] #9:Binary = 110010
[0:0] #9:weight = 50
[0:0] #10:start 10
[0:0] #10:Binary = 010010
[0:0] #10:weight = 18
[0:0] #11:start 11
[0:0] #11:Binary = 110010
[0:0] #11:weight = 50
[2:431] #11: sends   to 2 at 3431
[2:431] #11: sends   to 10 at 3431
[2:431] #11: sends distance(17,5) to 2 at 3431
[2:431] #11: sends distance(17,5) to 10 at 3431
[2:435] #6: sends   to 9 at 3435
[2:435] #6: sends distance(3,7) to 9 at 3435
[4:862] #2: sends   to 1 at 5862
[4:862] #2: sends   to 11 at 5862
[4:866] #9: sends   to 8 at 5866
[4:866] #9: sends   to 10 at 5866
[4:866] #9: sends   to 6 at 5866
[4:868] #10: sends   to 9 at 5868
[4:868] #10: sends   to 11 at 5868
[6:297] #2: sends distance(18,5) to 1 at 7297
[6:297] #9: sends distance(4,7) to 8 at 7297
[6:297] #9: sends distance(4,7) to 10 at 7297
[6:302] #10: sends distance(18,5) to 9 at 7302
[7:293] #1: sends   to 3 at 8293
[7:293] #1: sends   to 2 at 8293
[7:299] #11:2
[7:299] #11:rec. Ack(1) from 2
[7:299] #8: sends   to 4 at 8299
[7:299] #8: sends   to 9 at 8299
[7:299] #9: sends   to 10 at 8299
```

Figure 1-a: Calculation of binary number and weight value for each module

The weight of each block is determined by the surfaces that have connections. Since the work was done in BlinkyBlocks, the weight of each block can take values from 0 to 32. Therefore, the potential leader can be the block whose weight is strictly equal to 1,2,4,8,16 or 32. That is, the block has only one connected neighbor (leaf).

However, there are forms where each block has more than one connection. In this case, the possible leader will be selected by the lowest number of connected interfaces. That is, the block has connections to the sides that correspond to a smaller digit in a binary number:

top	000001
front	000010
left	000100
right	001000
back	010000
bottom	100000

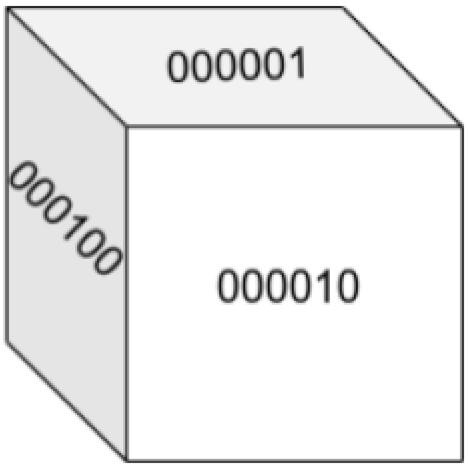


Figure1-b: Binary number of block's interfaces

Selected potential leaders begin to recruit their children. Leaders (in our shape it is blocks with B.ID: 5 and B.ID:7) send a message to child blocks and check if this block is occupied by other leaders. If the block is busy, it stops responding. Block with ID = 10 also trying to recruit block with ID = 9, but B.ID = 9 doesn't respond because he is already busy by B.ID = 6 and B.ID = 7 as well. Recruitment continues until the last block is occupied.

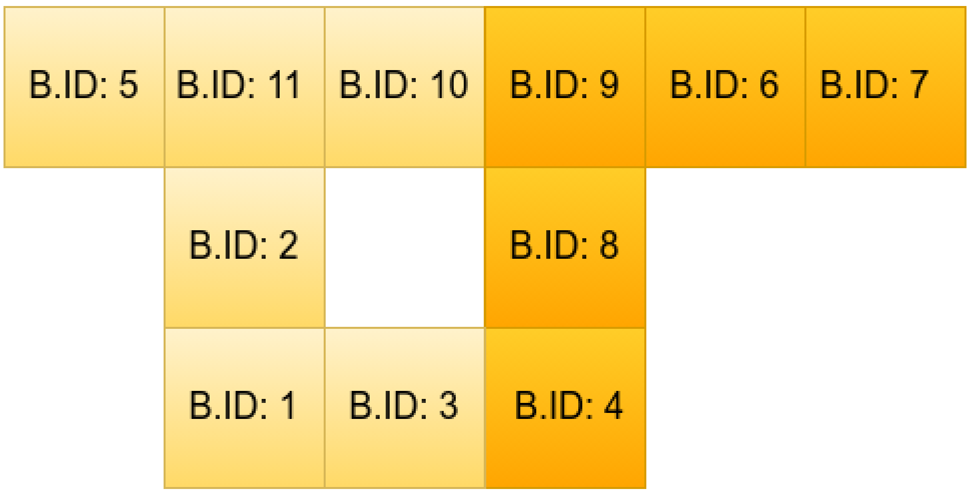


Figure 2: Recruitment blocks to get subtrees

Finally, as many supposed leaders we have, as many subtrees we can get. To finally choose the leader, we must separately work with each subtree. Since there are two proposed leaders in our form, we got two trees, which are clearly displayed in *Figure 3.*

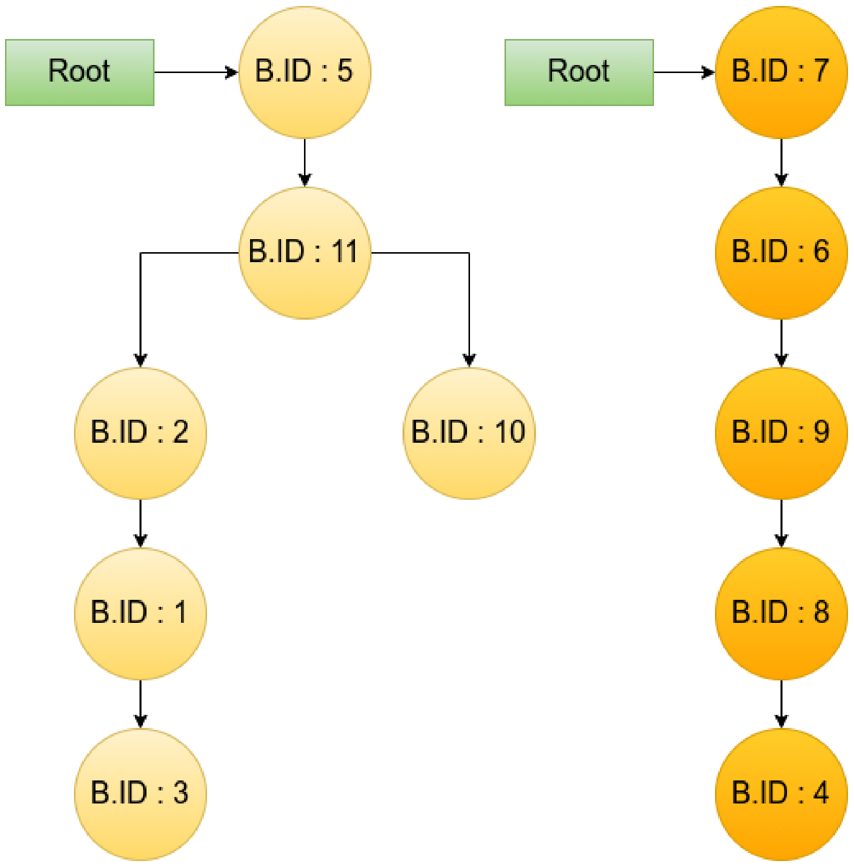


Figure 3: The resulting subtrees for further comparison

After the subtrees are determined, the competition of trees begins, this can be called evolution, where the strongest with the highest weight wins and becomes the root of the tree. The last leaf of the subtree reports its weight to the parent. The parent sums the received weight with its own and passes the information on. This continues until the seed (suggested leader) won't get the total sum of its entire tree. *Figures 4-a* and *4-b* show a detailed scheme for obtaining the weight of subtrees for the leaders B.ID = 5 and B.ID = 7, respectively.

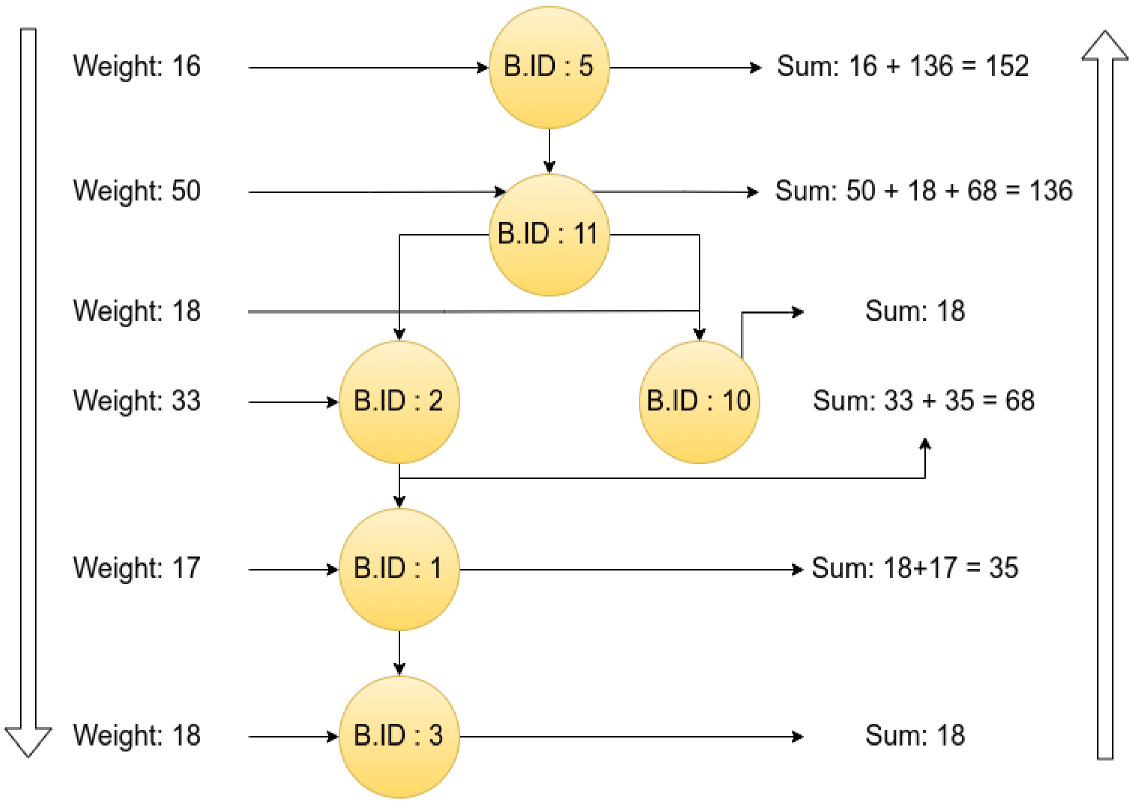


Figure 4-a: Getting the sum of the subtree for the leader B.ID = 5

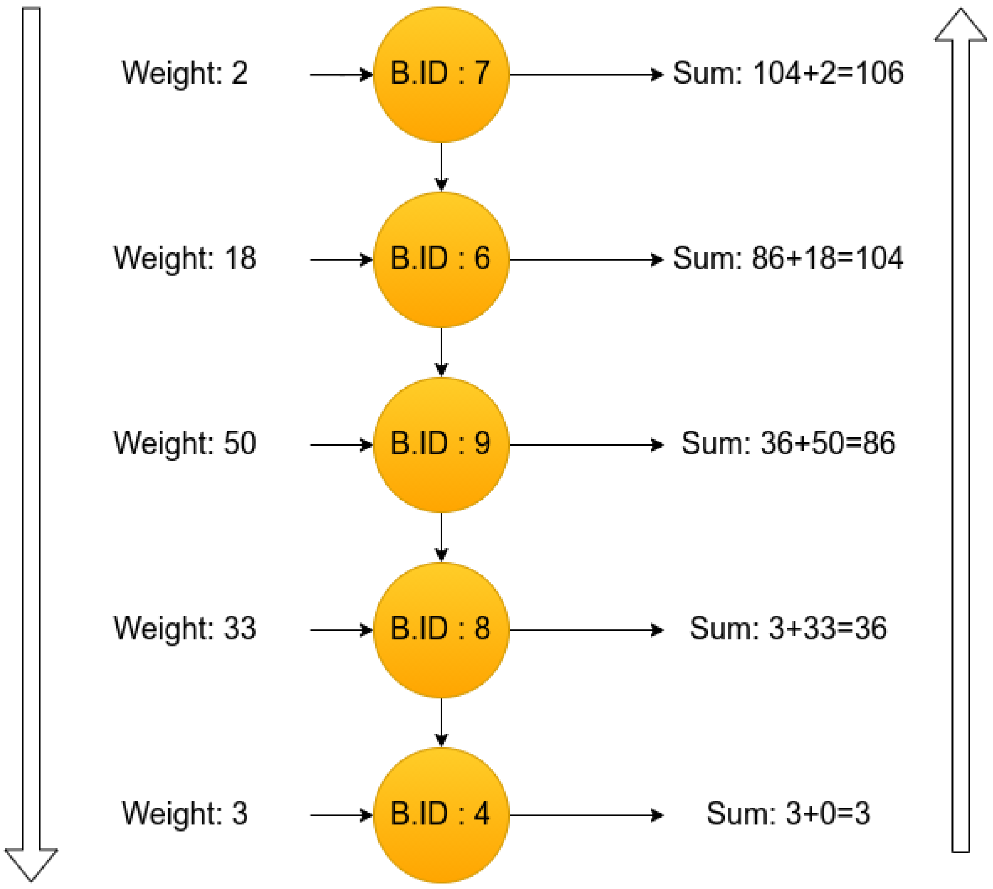
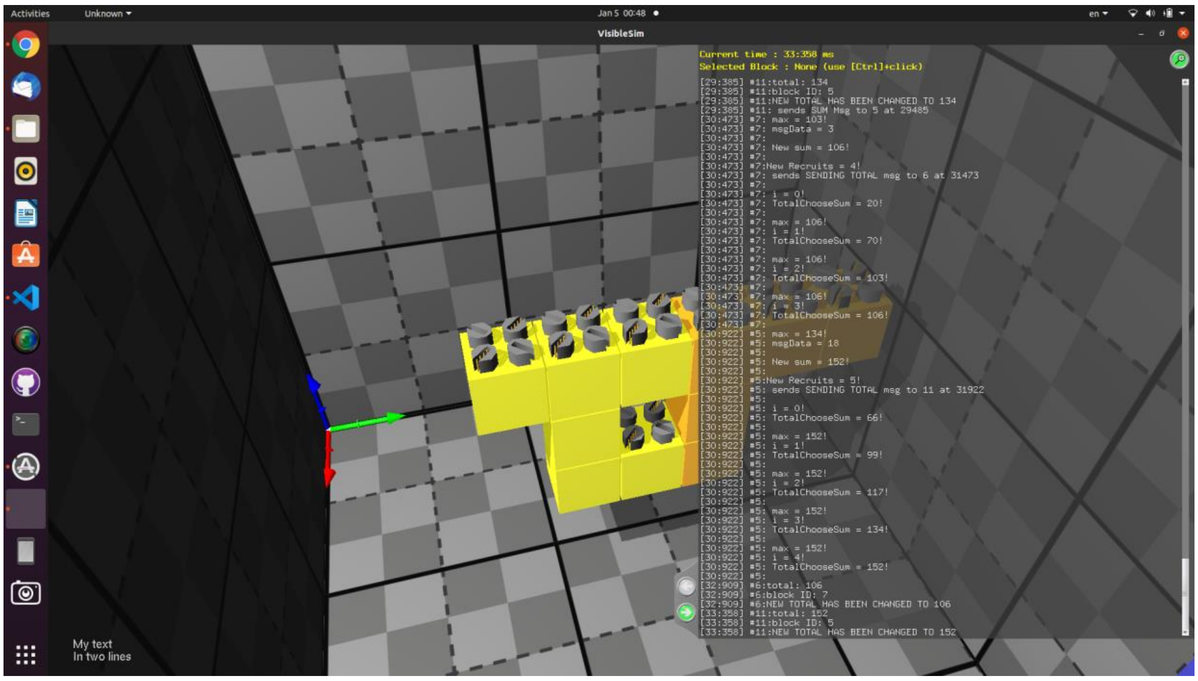


Figure 4-b: Getting the sum of the subtree for the leader B.ID = 7

When the leaders know the sum of their trees, it remains only to compare the values of all the leaders and choose the winner. At this stage, we had great difficulties that slowed down our work. The problem is that we weren't focused on the status of each block. We tried to create an array in which the values of each tree would be written, but the problem is that two arrays of the same name were created and comparison was impossible.

2. Experimental part

In the next paragraph, we provide the results of our simulation, as well as the execution time of the algorithm and the number of messages sent.



```
Current time : 33:358 ms
Selected Block : None (use [Ctrl]+click)
[29:385] #11:total: 134
[29:385] #11:block ID: 5
[29:385] #11:NEW TOTAL HAS BEEN CHANGED TO 134
[29:385] #11: sends SUM Msg to 5 at 29485
[30:473] #7: max = 103!
[30:473] #7: msgData = 3
[30:473] #7:
[30:473] #7: New sum = 106!
[30:473] #7:
[30:473] #7:New Recruits = 4!
[30:473] #7: sends SENDING TOTAL msg to 6 at 31473
[30:473] #7:
[30:473] #7: i = 0!
[30:473] #7: TotalChooseSum = 20!
[30:473] #7:
[30:473] #7: max = 106!
[30:473] #7: i = 1!
[30:473] #7: TotalChooseSum = 70!
[30:473] #7:
[30:473] #7: max = 106!
[30:473] #7: i = 2!
[30:473] #7: TotalChooseSum = 103!
[30:473] #7:
[30:473] #7: max = 106!
[30:473] #7: i = 3!
[30:473] #7: TotalChooseSum = 106!
[30:473] #7:
[30:922] #5: max = 134!
[30:922] #5: msgData = 18
[30:922] #5:
[30:922] #5: New sum = 152!
[30:922] #5:
[30:922] #5:New Recruits = 5!
[30:922] #5: sends SENDING TOTAL msg to 11 at 31922
[30:922] #5:
[30:922] #5: i = 0!
[30:922] #5: TotalChooseSum = 66!
[30:922] #5:
[30:922] #5: max = 152!
[30:922] #5: i = 1!
[30:922] #5: TotalChooseSum = 99!
[30:922] #5:
[30:922] #5: max = 152!
[30:922] #5: i = 2!
[30:922] #5: TotalChooseSum = 117!
[30:922] #5:
[30:922] #5: max = 152!
[30:922] #5: i = 3!
[30:922] #5: TotalChooseSum = 134!
[30:922] #5:
[30:922] #5: max = 152!
[30:922] #5: i = 4!
[30:922] #5: TotalChooseSum = 152!
[30:922] #5:
[32:909] #6:total: 106
[32:909] #6:block ID: 7
[32:909] #6:NEW TOTAL HAS BEEN CHANGED TO 106
[33:358] #11:total: 152
[33:358] #11:block ID: 5
[33:358] #11:NEW TOTAL HAS BEEN CHANGED TO 152
```

Figure 5: Final weight of subtrees

The process seems less complicated when everything is obvious in the configuration model. The process diagram is shown in *Figure 6-a* and experimented in *Figure 6-b.*

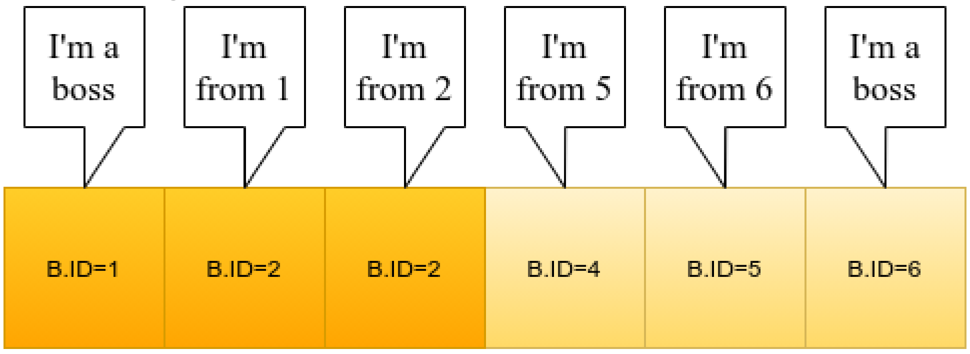
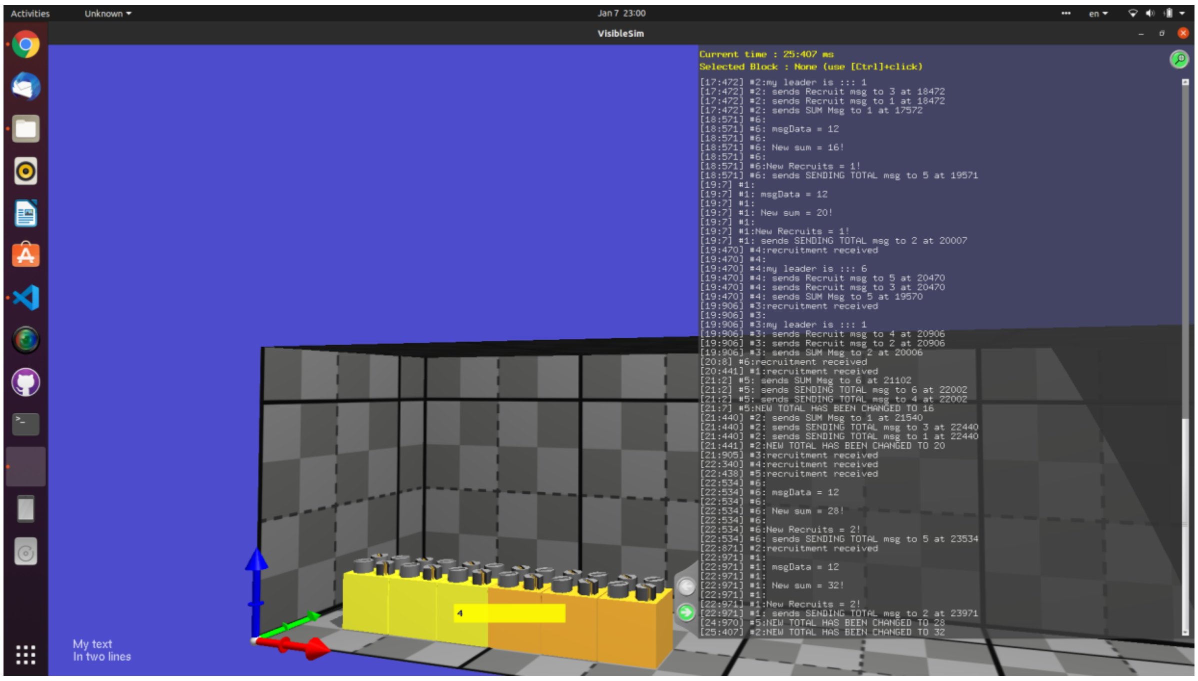


Figure 6-a: diagram for obvious model



```
Current time : 25:407 ms
Selected Block : None (use [Ctrl]+click)
[17:472] #2:my leader is ::: 1
[17:472] #2: sends Recruit msg to 3 at 18472
[17:472] #2: sends Recruit msg to 1 at 18472
[17:472] #2: sends SUM Msg to 1 at 17572
[18:571] #6:
[18:571] #6: msgData = 12
[18:571] #6:
[18:571] #6: New sum = 16!
[18:571] #6:
[18:571] #6:New Recruits = 1!
[18:571] #6: sends SENDING TOTAL msg to 5 at 19571
[19:7] #1:
[19:7] #1: msgData = 12
[19:7] #1:
[19:7] #1: New sum = 20!
[19:7] #1:
[19:7] #1:New Recruits = 1!
[19:7] #1: sends SENDING TOTAL msg to 2 at 20007
[19:470] #4:recruitment received
[19:470] #4:
[19:470] #4:my leader is ::: 6
[19:470] #4: sends Recruit msg to 5 at 20470
[19:470] #4: sends Recruit msg to 3 at 20470
[19:470] #4: sends SUM Msg to 5 at 19570
[19:906] #3:recruitment received
[19:906] #3:
[19:906] #3:my leader is ::: 1
[19:906] #3: sends Recruit msg to 4 at 20906
[19:906] #3: sends Recruit msg to 2 at 20906
[19:906] #3: sends SUM Msg to 2 at 20006
[20:8] #6:recruitment received
[20:441] #1:recruitment received
[21:2] #5: sends SUM Msg to 6 at 21102
[21:2] #5: sends SENDING TOTAL msg to 6 at 22002
[21:2] #5: sends SENDING TOTAL msg to 4 at 22002
[21:7] #5:NEW TOTAL HAS BEEN CHANGED TO 16
[21:440] #2: sends SUM Msg to 1 at 21540
[21:440] #2: sends SENDING TOTAL msg to 3 at 22440
[21:440] #2: sends SENDING TOTAL msg to 1 at 22440
[21:441] #2:NEW TOTAL HAS BEEN CHANGED TO 20
[21:905] #3:recruitment received
[22:340] #4:recruitment received
[22:438] #5:recruitment received
[22:534] #6:
[22:534] #6: msgData = 12
[22:534] #6:
[22:534] #6: New sum = 28!
[22:534] #6:
[22:534] #6:New Recruits = 2!
[22:534] #6: sends SENDING TOTAL msg to 5 at 23534
[22:871] #2:recruitment received
[22:971] #1:
[22:971] #1: msgData = 12
[22:971] #1:
[22:971] #1: New sum = 32!
[22:971] #1:
[22:971] #1:New Recruits = 2!
[22:971] #1: sends SENDING TOTAL msg to 2 at 23971
[24:970] #5:NEW TOTAL HAS BEEN CHANGED TO 28
[25:407] #2:NEW TOTAL HAS BEEN CHANGED TO 32
```

Figure 6-b: simulation of obvious model

In the course of the work, we encountered difficulties that we could not solve before meet the deadline. One of these difficulties arises when hiring blocks when there is a block in the middle between the two possible leaders. That is, this block has the same distance to the leaders.

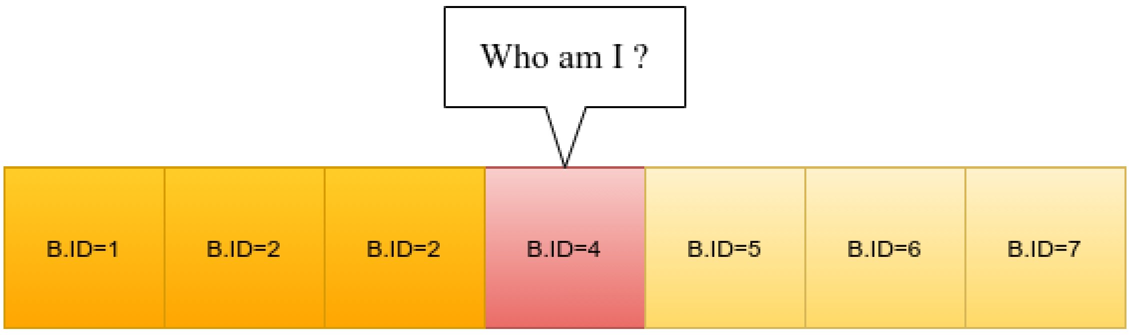
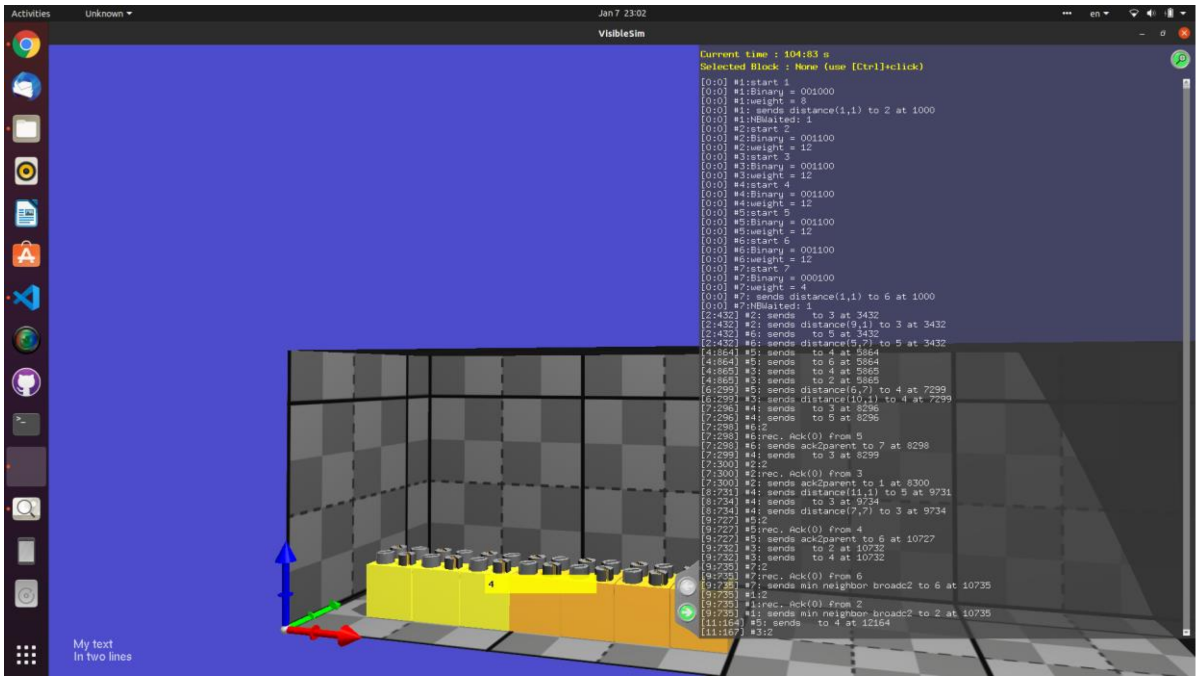


Figure 7a: model with an undefined block

This problem is shown schematically in *Figure 7-a*. Block with ID number 4 simultaneously receives messages from possible leaders and tries to answer them. Because of this, the program runs in an infinite loop and the program execution time does not stop. This can be seen when simulating this configuration in *Figure 7-b*.



```
Current time : 104:83 s
Selected Block : None (use [Ctrl]+click)
[0:0] #1:start 1
[0:0] #1:Binary = 001000
[0:0] #1:weight = 8
[0:0] #1: sends distance(1,1) to 2 at 1000
[0:0] #1:NBWaited: 1
[0:0] #2:start 2
[0:0] #2:Binary = 001100
[0:0] #2:weight = 12
[0:0] #3:start 3
[0:0] #3:Binary = 001100
[0:0] #3:weight = 12
[0:0] #4:start 4
[0:0] #4:Binary = 001100
[0:0] #4:weight = 12
[0:0] #5:start 5
[0:0] #5:Binary = 001100
[0:0] #5:weight = 12
[0:0] #6:start 6
[0:0] #6:Binary = 001100
[0:0] #6:weight = 12
[0:0] #7:start 7
[0:0] #7:Binary = 000100
[0:0] #7:weight = 4
[0:0] #7: sends distance(1,1) to 6 at 1000
[0:0] #7:NBWaited: 1
[2:432] #2: sends   to 3 at 3432
[2:432] #2: sends distance(9,1) to 3 at 3432
[2:432] #6: sends   to 5 at 3432
[2:432] #6: sends distance(5,7) to 5 at 3432
[4:864] #5: sends   to 4 at 5864
[4:864] #5: sends   to 6 at 5864
[4:865] #3: sends   to 4 at 5865
[4:865] #3: sends   to 2 at 5865
[6:299] #5: sends distance(6,7) to 4 at 7299
[6:299] #3: sends distance(10,1) to 4 at 7299
[7:296] #4: sends   to 3 at 8296
[7:296] #4: sends   to 5 at 8296
[7:298] #6:2
[7:298] #6:rec. Ack(0) from 5
[7:298] #6: sends ack2parent to 7 at 8298
[7:299] #4: sends   to 3 at 8299
[7:300] #2:2
[7:300] #2:rec. Ack(0) from 3
[7:300] #2: sends ack2parent to 1 at 8300
[8:731] #4: sends distance(11,1) to 5 at 9731
[8:734] #4: sends   to 3 at 9734
[8:734] #4: sends distance(7,7) to 3 at 9734
[9:727] #5:2
[9:727] #5:rec. Ack(0) from 4
[9:727] #5: sends ack2parent to 6 at 10727
[9:732] #3: sends   to 2 at 10732
[9:732] #3: sends   to 4 at 10732
[9:735] #7:2
[9:735] #7:rec. Ack(0) from 6
[9:735] #7: sends min neighbor broadc2 to 6 at 10735
[9:735] #1:2
[9:735] #1:rec. Ack(0) from 2
[9:735] #1: sends min neighbor broadc2 to 2 at 10735
[11:164] #5: sends   to 4 at 12164
[11:167] #3:2
```

Figure 7-b: Simulating a model with an undefined block

A simulation of the program was also performed for 199 blocks, this is demonstrated schematically in *Figure 8-a* and experimentally in *Figure 8-b*.

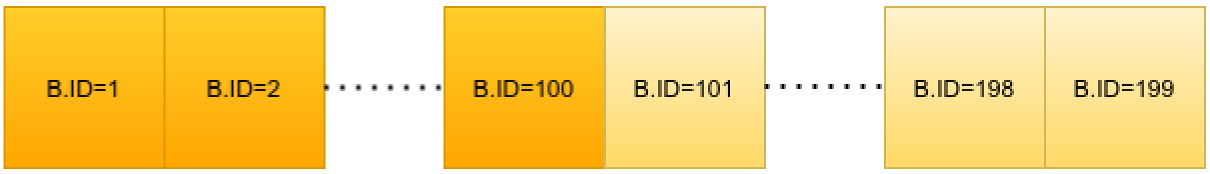
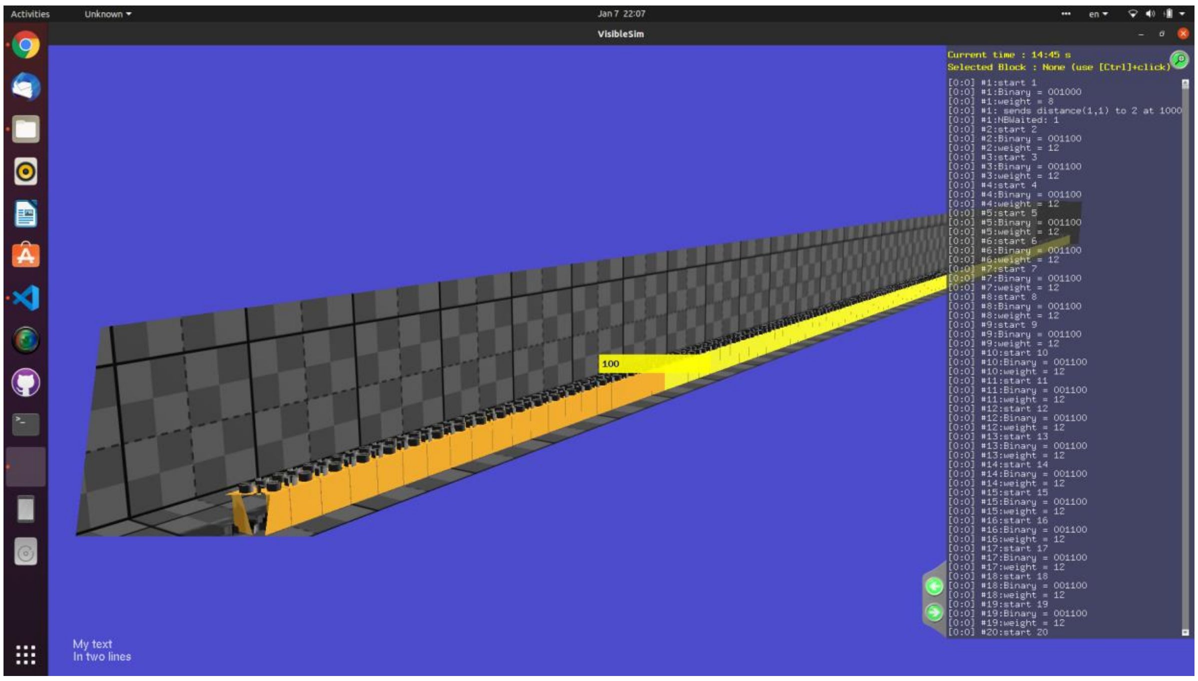


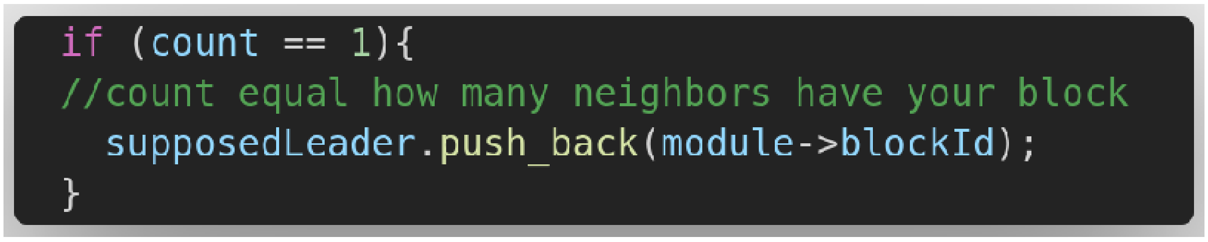
Figure 8-a: Configuration model diagram for 199 blocks



```
Current time : 14:45 s
Selected Block : None (use [Ctrl]+click)
[0:0] #1:start 1
[0:0] #1:Binary = 001000
[0:0] #1:weight = 8
[0:0] #1: sends distance(1,1) to 2 at 1000
[0:0] #1:NBWaited: 1
[0:0] #2:start 2
[0:0] #2:Binary = 001100
[0:0] #2:weight = 12
[0:0] #3:start 3
[0:0] #3:Binary = 001100
[0:0] #3:weight = 12
[0:0] #4:start 4
[0:0] #4:Binary = 001100
[0:0] #4:weight = 12
[0:0] #5:start 5
[0:0] #5:Binary = 001100
[0:0] #5:weight = 12
[0:0] #6:start 6
[0:0] #6:Binary = 001100
[0:0] #6:weight = 12
[0:0] #7:start 7
[0:0] #7:Binary = 001100
[0:0] #7:weight = 12
[0:0] #8:start 8
[0:0] #8:Binary = 001100
[0:0] #8:weight = 12
[0:0] #9:start 9
[0:0] #9:Binary = 001100
[0:0] #9:weight = 12
[0:0] #10:start 10
[0:0] #10:Binary = 001100
[0:0] #10:weight = 12
[0:0] #11:start 11
[0:0] #11:Binary = 001100
[0:0] #11:weight = 12
[0:0] #12:start 12
[0:0] #12:Binary = 001100
[0:0] #12:weight = 12
[0:0] #13:start 13
[0:0] #13:Binary = 001100
[0:0] #13:weight = 12
[0:0] #14:start 14
[0:0] #14:Binary = 001100
[0:0] #14:weight = 12
[0:0] #15:start 15
[0:0] #15:Binary = 001100
[0:0] #15:weight = 12
[0:0] #16:start 16
[0:0] #16:Binary = 001100
[0:0] #16:weight = 12
[0:0] #17:start 17
[0:0] #17:Binary = 001100
[0:0] #17:weight = 12
[0:0] #18:start 18
[0:0] #18:Binary = 001100
[0:0] #18:weight = 12
[0:0] #19:start 19
[0:0] #19:Binary = 001100
[0:0] #19:weight = 12
[0:0] #20:start 20
```

Figure 8b: Experiment on 199 blocks

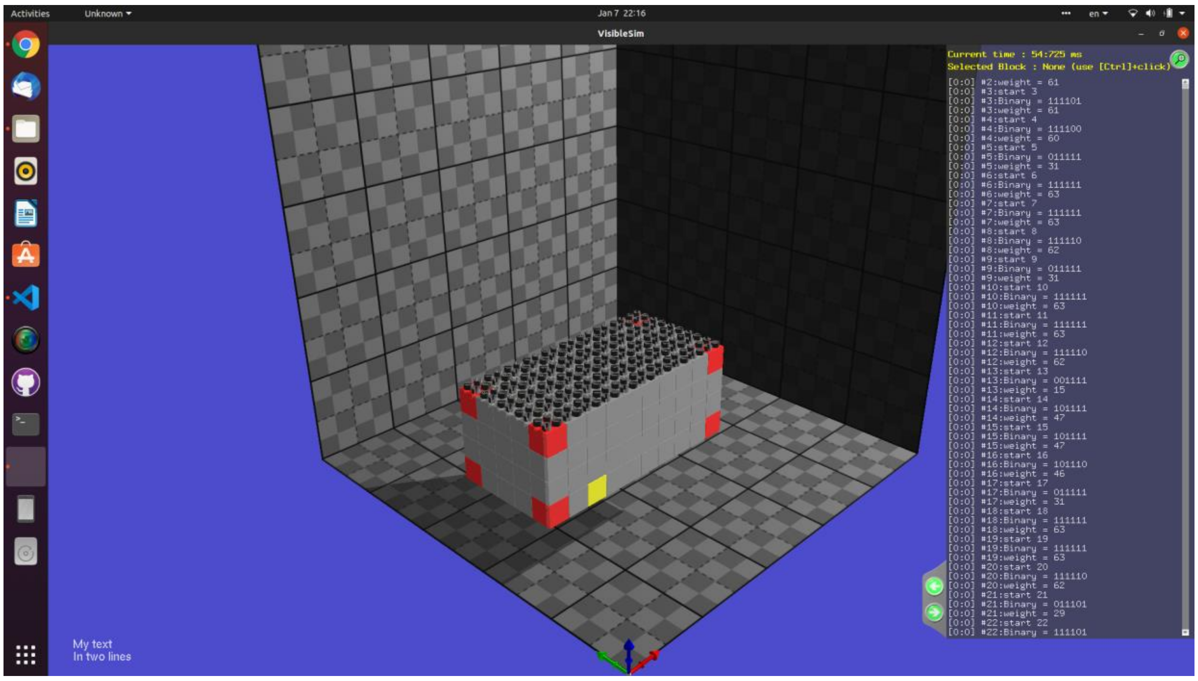
When the configuration model contains blocks with only one neighbor (that is, the weight of which is exactly 1, 2, 4, 8, 16, 32), there are no problems. But when there is no such block, the choice of the leader becomes more difficult. Our code is static, to improve it we need to make the part shown in *Figure 9* dynamic.



```
if (count == 1){
//count equal how many neighbors have your block
  supposedLeader.push_back(module->blockId);
}
```

Figure 9: Weak point in our work

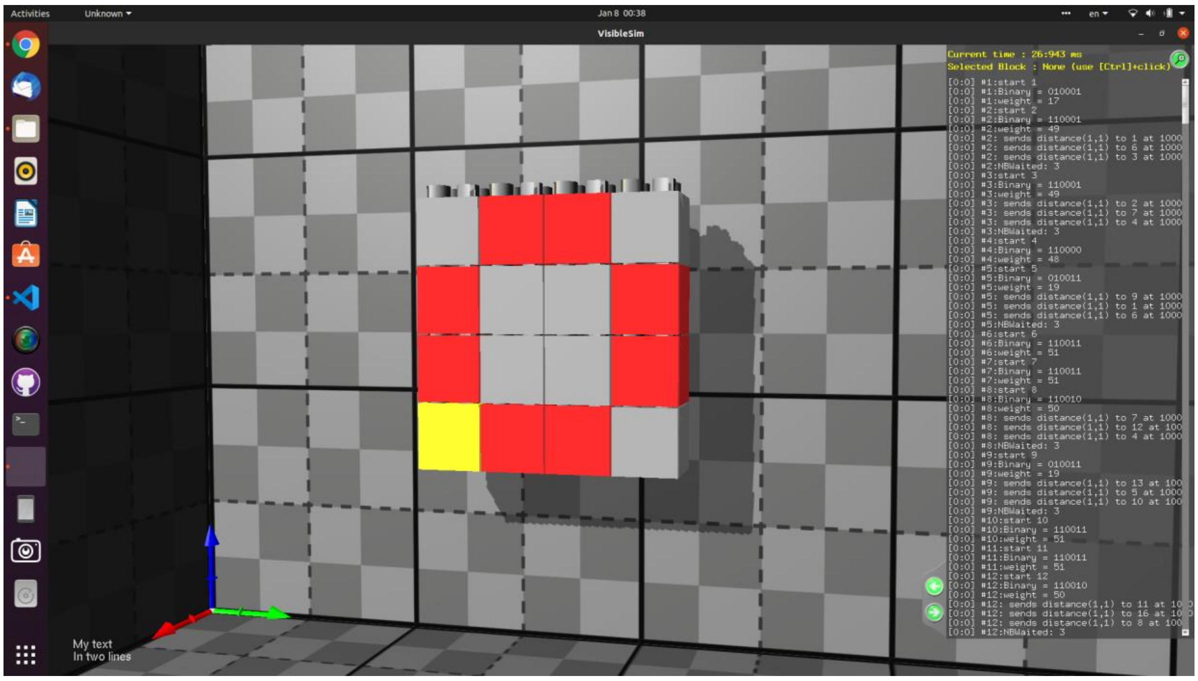
An experiment for a volumetric cube is shown in *Figure 10*. This cube consists of small cubes (modular robots). For the program to start working, the "count" must be equal to 3, this is the number of neighbors for possible leaders.



```
Current time : 54:725 ms
Selected Block : None (use [Ctrl]+click)
[0:0] #2:weight = 61
[0:0] #3:start 3
[0:0] #3:Binary = 111101
[0:0] #3:weight = 61
[0:0] #4:start 4
[0:0] #4:Binary = 111100
[0:0] #4:weight = 60
[0:0] #5:start 5
[0:0] #5:Binary = 011111
[0:0] #5:weight = 31
[0:0] #6:start 6
[0:0] #6:Binary = 111111
[0:0] #6:weight = 63
[0:0] #7:start 7
[0:0] #7:Binary = 111111
[0:0] #7:weight = 63
[0:0] #8:start 8
[0:0] #8:Binary = 111110
[0:0] #8:weight = 62
[0:0] #9:start 9
[0:0] #9:Binary = 011111
[0:0] #9:weight = 31
[0:0] #10:start 10
[0:0] #10:Binary = 111111
[0:0] #10:weight = 63
[0:0] #11:start 11
[0:0] #11:Binary = 111111
[0:0] #11:weight = 63
[0:0] #12:start 12
[0:0] #12:Binary = 111110
[0:0] #12:weight = 62
[0:0] #13:start 13
[0:0] #13:Binary = 001111
[0:0] #13:weight = 15
[0:0] #14:start 14
[0:0] #14:Binary = 101111
[0:0] #14:weight = 47
[0:0] #15:start 15
[0:0] #15:Binary = 101111
[0:0] #15:weight = 47
[0:0] #16:start 16
[0:0] #16:Binary = 101110
[0:0] #16:weight = 46
[0:0] #17:start 17
[0:0] #17:Binary = 011111
[0:0] #17:weight = 31
[0:0] #18:start 18
[0:0] #18:Binary = 111111
[0:0] #18:weight = 63
[0:0] #19:start 19
[0:0] #19:Binary = 111111
[0:0] #19:weight = 63
[0:0] #20:start 20
[0:0] #20:Binary = 111110
[0:0] #20:weight = 62
[0:0] #21:start 21
[0:0] #21:Binary = 011101
[0:0] #21:weight = 29
[0:0] #22:start 22
[0:0] #22:Binary = 111101
```

Figure 10: Volumetric cube model

However, when the program is executed, the same problem arises as in *Figure 7-a*, but on a large scale. To analyze this problem, let's turn to *Figure 11* in more detail.



```
Current time : 26:943 ms
Selected Block : None (use [Ctrl]+click)
[0:0] #1:start 1
[0:0] #1:Binary = 010001
[0:0] #1:weight = 17
[0:0] #2:start 2
[0:0] #2:Binary = 110001
[0:0] #2:weight = 49
[0:0] #2: sends distance(1,1) to 1 at 1000
[0:0] #2: sends distance(1,1) to 6 at 1000
[0:0] #2: sends distance(1,1) to 3 at 1000
[0:0] #2:NBWaited: 3
[0:0] #3:start 3
[0:0] #3:Binary = 110001
[0:0] #3:weight = 49
[0:0] #3: sends distance(1,1) to 2 at 1000
[0:0] #3: sends distance(1,1) to 7 at 1000
[0:0] #3: sends distance(1,1) to 4 at 1000
[0:0] #3:NBWaited: 3
[0:0] #4:start 4
[0:0] #4:Binary = 110000
[0:0] #4:weight = 48
[0:0] #5:start 5
[0:0] #5:Binary = 010011
[0:0] #5:weight = 19
[0:0] #5: sends distance(1,1) to 9 at 1000
[0:0] #5: sends distance(1,1) to 1 at 1000
[0:0] #5: sends distance(1,1) to 6 at 1000
[0:0] #5:NBWaited: 3
[0:0] #6:start 6
[0:0] #6:Binary = 110011
[0:0] #6:weight = 51
[0:0] #7:start 7
[0:0] #7:Binary = 110011
[0:0] #7:weight = 51
[0:0] #8:start 8
[0:0] #8:Binary = 110010
[0:0] #8:weight = 50
[0:0] #8: sends distance(1,1) to 7 at 1000
[0:0] #8: sends distance(1,1) to 12 at 100
[0:0] #8: sends distance(1,1) to 4 at 1000
[0:0] #8:NBWaited: 3
[0:0] #9:start 9
[0:0] #9:Binary = 010011
[0:0] #9:weight = 19
[0:0] #9: sends distance(1,1) to 13 at 100
[0:0] #9: sends distance(1,1) to 5 at 1000
[0:0] #9: sends distance(1,1) to 10 at 100
[0:0] #9:NBWaited: 3
[0:0] #10:start 10
[0:0] #10:Binary = 110011
[0:0] #10:weight = 51
[0:0] #11:start 11
[0:0] #11:Binary = 110011
[0:0] #11:weight = 51
[0:0] #12:start 12
[0:0] #12:Binary = 110010
[0:0] #12:weight = 50
[0:0] #12: sends distance(1,1) to 11 at 10
[0:0] #12: sends distance(1,1) to 16 at 10
[0:0] #12: sends distance(1,1) to 8 at 100
[0:0] #12:NBWaited: 3
```

Figure 11: Analysis of the reasons for ambiguity

According to the proposed algorithm, recruiting starts with the blocks that have the least number of neighbors (and also the least weight). That is, from the blocks that are in the corners of the rectangle. The first recruitment step is successful, then blocks that are colored red are trying to recruit gray blocks from the center. Recruiting fails, because the gray box does not know which request needs to be answered. This situation is considered separately in schematic *Figure 12*. Upon detailed examination, it becomes clear that this situation is no different from *Figure 7-a*

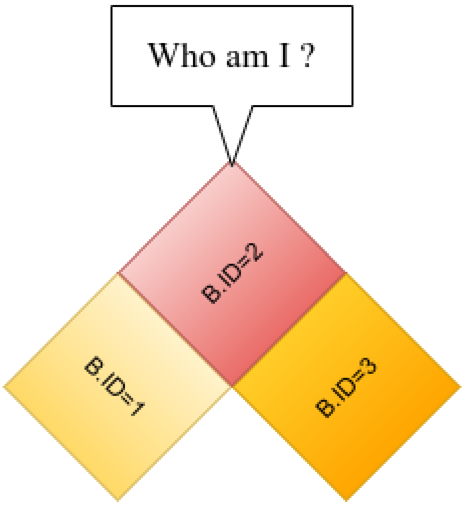
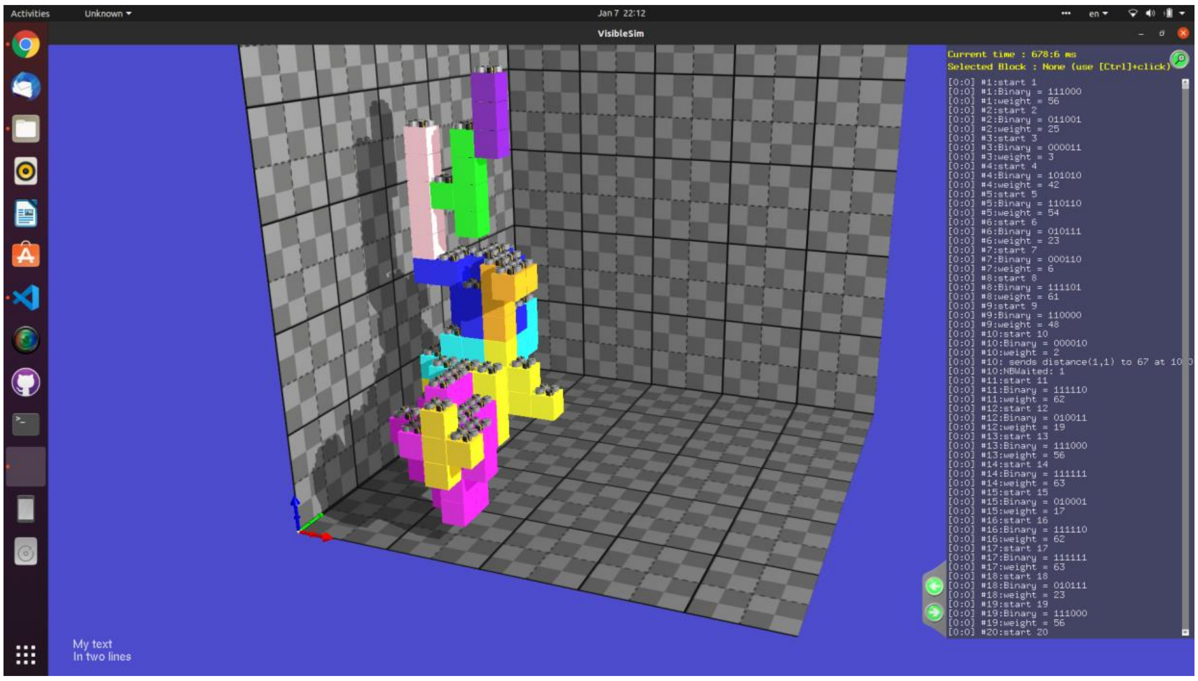


Figure 12: Detailed examination of issue

Also, this experiment was carried out for a random model, the results of which are shown in *Figure 13*.



```
Current time : 678:6 ms
Selected Block : None (use [Ctrl]+click)
[0:0] #1:start 1
[0:0] #1:Binary = 111000
[0:0] #1:weight = 56
[0:0] #2:start 2
[0:0] #2:Binary = 011001
[0:0] #2:weight = 25
[0:0] #3:start 3
[0:0] #3:Binary = 000011
[0:0] #3:weight = 3
[0:0] #4:start 4
[0:0] #4:Binary = 101010
[0:0] #4:weight = 42
[0:0] #5:start 5
[0:0] #5:Binary = 110110
[0:0] #5:weight = 54
[0:0] #6:start 6
[0:0] #6:Binary = 010111
[0:0] #6:weight = 23
[0:0] #7:start 7
[0:0] #7:Binary = 000110
[0:0] #7:weight = 6
[0:0] #8:start 8
[0:0] #8:Binary = 111101
[0:0] #8:weight = 61
[0:0] #9:start 9
[0:0] #9:Binary = 110000
[0:0] #9:weight = 48
[0:0] #10:start 10
[0:0] #10:Binary = 000010
[0:0] #10:weight = 2
[0:0] #10: sends distance(1,1) to 67 at 10
[0:0] #10:NBWaited: 1
[0:0] #11:start 11
[0:0] #11:Binary = 111110
[0:0] #11:weight = 62
[0:0] #12:start 12
[0:0] #12:Binary = 010011
[0:0] #12:weight = 19
[0:0] #13:start 13
[0:0] #13:Binary = 111000
[0:0] #13:weight = 56
[0:0] #14:start 14
[0:0] #14:Binary = 111111
[0:0] #14:weight = 63
[0:0] #15:start 15
[0:0] #15:Binary = 010001
[0:0] #15:weight = 17
[0:0] #16:start 16
[0:0] #16:Binary = 111110
[0:0] #16:weight = 62
[0:0] #17:start 17
[0:0] #17:Binary = 111111
[0:0] #17:weight = 63
[0:0] #18:start 18
[0:0] #18:Binary = 010111
[0:0] #18:weight = 23
[0:0] #19:start 19
[0:0] #19:Binary = 111000
[0:0] #19:weight = 56
[0:0] #20:start 20
```

Figure 13: Experimenting with a random model

3. Conclusions:

This algorithm is needed not only to play the evolution of digital trees, but also has a very useful and important application. in decentralized systems where the task is to select a leader, this algorithm can be used on an equal footing with the ABC-Center, k-BFS SumSweep, etc.

This work is quite voluminous. The operation of the algorithm is clear and transparent, but when implementing the algorithm, a large number of difficulties arise that must be solved.