Distributed Operating System: Lab Assignment 1

Ayush Sharma, Neha Yadav

March 9, 2017

1 Program Design

Classes:

- **Bird** - Generate the trajectory details, coordinates where bird will land and time it takes to land.
- **Message** - Each instance of the class contains the message id, type,parameters required to invoke message of particular type.
- **NetworkMap** - It's constructor initializes the Network topology and registers the object on NameServer. A node can invoke *getNetworkNeighbours* method remotely to get it's network neighbors.
- **PhysicalMap**- It's constructor initializes the Physical topology of nodes (where the nodes are present physically) and registers the object on NameServer. In the physical map a pig is represented by an integer ≥ 0, a stone/wall by -1 and an empty space by -2. A pig can invoke following methods on this class remotely:
  - *getPhysicalAddress()* - It returns an integer denoting the physical address of a node in network.
  - *isStone()* - It returns a boolean if there is a stone at a given physical address.
  - *isEmptySpace()* - It return a boolean if there is an empty space at given physical address.
  - *getIPaddress()* - It returns the network IP for a node at given physical address.
  - *getNearestPigIP()* - It returns the network IP of pig closest to the bird launching coordinates i.e. the leftmost pig.
- **Pigs** - This class contains methods required for communication in a network and each instance registers itself with Name Server and it's methods are exposed so that they can be accessed remotely. The pigs can pass the following the following kinds of messages:

- *BIRD_APPROACHING* - This the message the pigs send to other pigs to communicate that a bird is approaching.
- *TAKE_SHELTER*- This message is used by an affected pig to alert its neighbours. The intended target is the physical neighbours of the affected pig.
- *ACKNOWLEDGEMENT* - If a pig can move to a safe space. It can send its caller this message to alert him that he can move and save himself too.
- *I_AM_SAFE*- This message is sent by an originally affected pig to his neighbours.

A detailed description of all the methods of Pig.py file is given at then end of this document.

Constants.py - This file contains all the constants and configurations. One can change the values here to test the program at different configuration. **Utils.py** - The file which contains functions which any class can access and run. sliding

2 Model assumptions

- The pigs are laid out along a linear grid like [P_1, W, P_2, _, _, W, _, P_3] where W is the wall and _ is the empty space. The indexing is from 0.
- The pigs are connected in a ring topology. The indexing is from 0.
- If a bird hits a pig, the pig can affect its immediate neighbours.
  Example: [, _, P_1, P_2, P_3, _, _].
  In this case, if the bird hits P_2, it can topple and affect P_1 and P_3.
- If a bird hits a Wall W, it can fall and affect its two immediate neighbours to either side.
  Example: [, _, P_1, P_2, W P_3, P_4, _, _].
  In this case, if the bird hits the wall, it can affect P_1, P_2, P_3, P_4.
- The pigs can try to evade the hit, by communicating through an infinite chain.
  Example: [_, _, W, P_1, P_2, P_3, P_4, _, _]
  If the bird hits W or P_1, the pigs will communicate the message through to P_4. Then, P_4 can move one place to right and the previous pigs can each shift one place to right and save themselves.

3 How it works

:

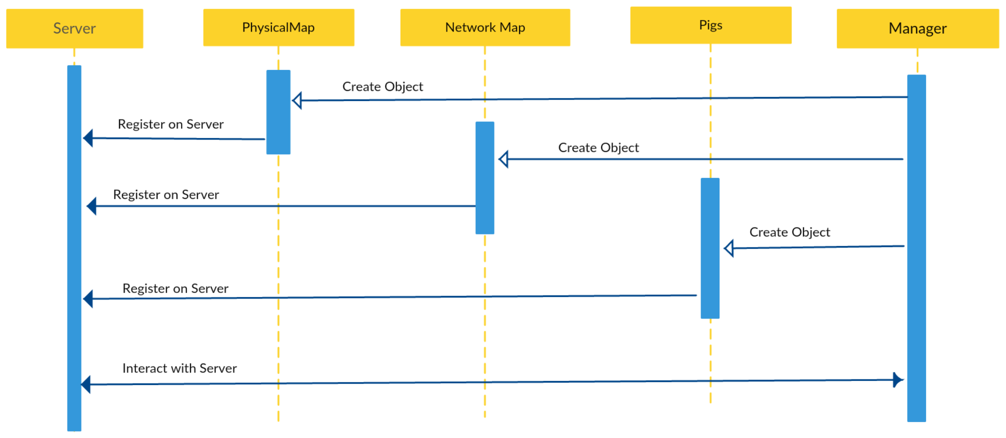


Figure 1: Sequence diagram for Registering on server.

The manager creates the Physical Map object, the network map and the pigs objects. These objects register themselves on the server running by default on localhost and port 9090.

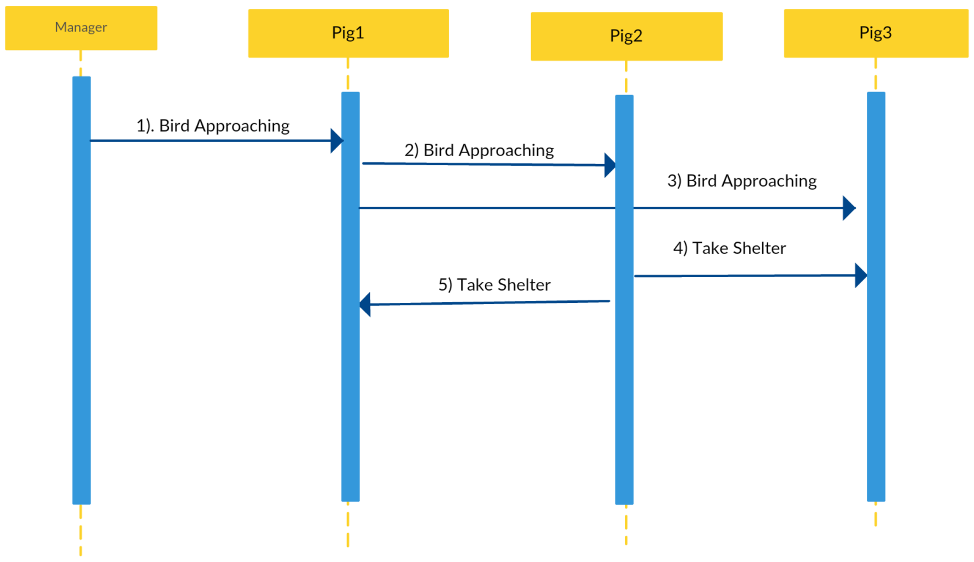
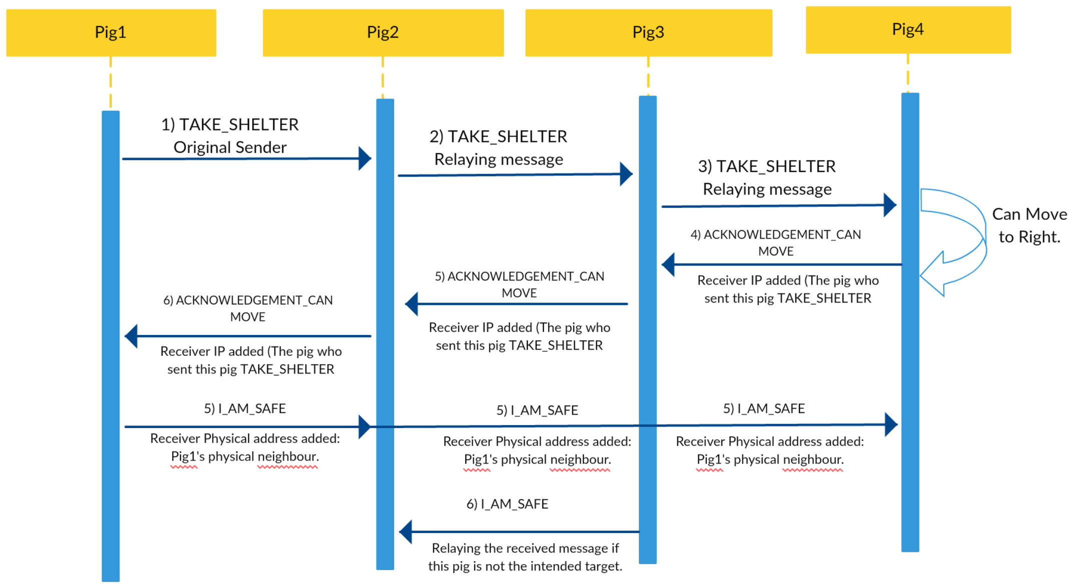


Figure 2: Sequence diagram for Bird Approaching message passing.

In the above diagram we assume that P_1, P_2 and P_3 are physical neighbors in that order and P_2 is the one getting hit. As soon as P_2 realizes that he is getting hit, he does not relay the BIRD_APPROACHING message rather, he initiates TAKE_SHELTER MESSAGE.



Pig1
Pig2
Pig3
Pig4
1) TAKE_SHELTER Original Sender
2) TAKE_SHELTER Relaying message
3) TAKE_SHELTER Relaying message
Can Move to Right.
4) ACKNOWLEDGEMENT_CAN MOVE
Receiver IP added (The pig who sent this pig TAKE_SHELTER
5) ACKNOWLEDGEMENT_CAN MOVE
Receiver IP added (The pig who sent this pig TAKE_SHELTER
6) ACKNOWLEDGEMENT_CAN MOVE
Receiver IP added (The pig who sent this pig TAKE_SHELTER
5) I_AM_SAFE
Receiver Physical address added: Pig1's physical neighbour.
5) I_AM_SAFE
Receiver Physical address added: Pig1's physical neighbour.
5) I_AM_SAFE
Receiver Physical address added: Pig1's physical neighbour.
6) I_AM_SAFE
Relaying the received message if this pig is not the intended target.

Figure 3: Sequence diagram for TAKE_SHELTER, ACKNOWLEDGEMENT and I_AM_SAFE message propagation.

In this case, we assume that $[P_1,\ P_2,\ P_3,\ P_4,\ \_,\ \_]$ is the map. They are network neighbours in the same cyclic order as well. The bird hits the P_1. Now if all the pigs can move to right then it can save themselves. The final I_AM_SAFE message is meant only for the physical neighbours of P_1 (original sender of TAKE_SHELTER). The others just relay the message until it reaches the rightful pig.

4 Possible improvements and extensions

- We can use 2 or 3 dimensional grid to position pigs and wall(stone) in the network.
- We can experiment with taking unstructured topology.
- Affect of wall falling or pig toppling can be extended with some modifications to the existing code but it seems unreasonable to do so for more than what we have decided now.

5 Steps to run

In Manager.py and Server.py :
for Random testcase:
set mode = constants.RuntimeConstants.MODE_RANDOM
to run in test mode:
set mode = constants.RuntimeConstants = MODE_TESTING

If you want to execute the code in random mode the variable should be set to constants.RuntimeConstants = MODE_RANDOM Manager.py or server.py If you want to run a custom test case make sure to change NUM_PIGS to what you have taken in Constants.MapConstants class. In case of random mode you do not need to execute the following steps **Steps to run in testing mode**

- *PhysicalMap.py* There are various versions of the method _testing_createMap1, _testing_createMap2 and so on. These methods contains different layouts for the physical map.
- Modify the init method of class PhysicalMap and replace _testing_createMap with any version of the method defined in the class.
- *Bird.py* This class also contains various methods, _testing_makeBird1, _testing_makeBird2 etc
- Modify the init method of class Bird.py and replace _testing_makeBird with any version of the defined methods in the class

Note if you have added _testing_createMap2 in the init method of class PhysicalMap then you should only add _testing_makeBird2 in init method of BirdClass. There is one two one mapping between _testing_createMap2 and _testing_makeBird2

- Start the Name Server on one terminal.
- Open another terminal and go inside the assignment folder and execute Server.py. This will start the server.

- Open one more terminal and then execute Manager.py This will get the trajectory details of the bird and then find a pig in network that is nearest to bird landing position. It will then send bird approaching method remotely for this pig which on receiving the message will pass the message to it's neighbors and so on until every node in the network have received the message.

6 Test Cases and Result

For writing here, I have assumed there is sufficient time given to pigs to evade.
You can actually modify bird attack time in testcases in code.
W represents a wall and _ represents an empty space.

- **Testcase 1:** - [P_1, _, P_0, P_2, P_3, P_4, _, _, W, -2].
  Bird hits P_0.
  Result: All pigs are saved.
- **Testcase 2:** - [P_1, W, P_0, W, P_2, P_3, P_4, _, _, W, -2].
  Bird hits P_0.
  Result: All pigs are saved except P_0.
- **Testcase 3:** - [P_1, W, P_0, P_2, P_3, P_4, _, _, W, -2].
  Bird hits Wall at position 1.
  Result: All pigs are saved except P_1.
- **Testcase 4:** - [P_1, P_0, W, P_2, P_3, P_4, _, _, W, -2].
  Bird hits Wall at position 2 .
  Result: All pigs are saved except P_0, P_1 (They die due to wall toppling).
- **Testcase 5:** - [_,P_5, P_1, P_0, W, P_2, P_3, P_4, _, _, W, -2].
  Bird hits Wall at position 4 .
  Result: All pigs are saved. They shift one place to left (P_5, P_1 and P_0) and one place to right (P_2, P_3, P_4.)

6.1 Few Output Results:

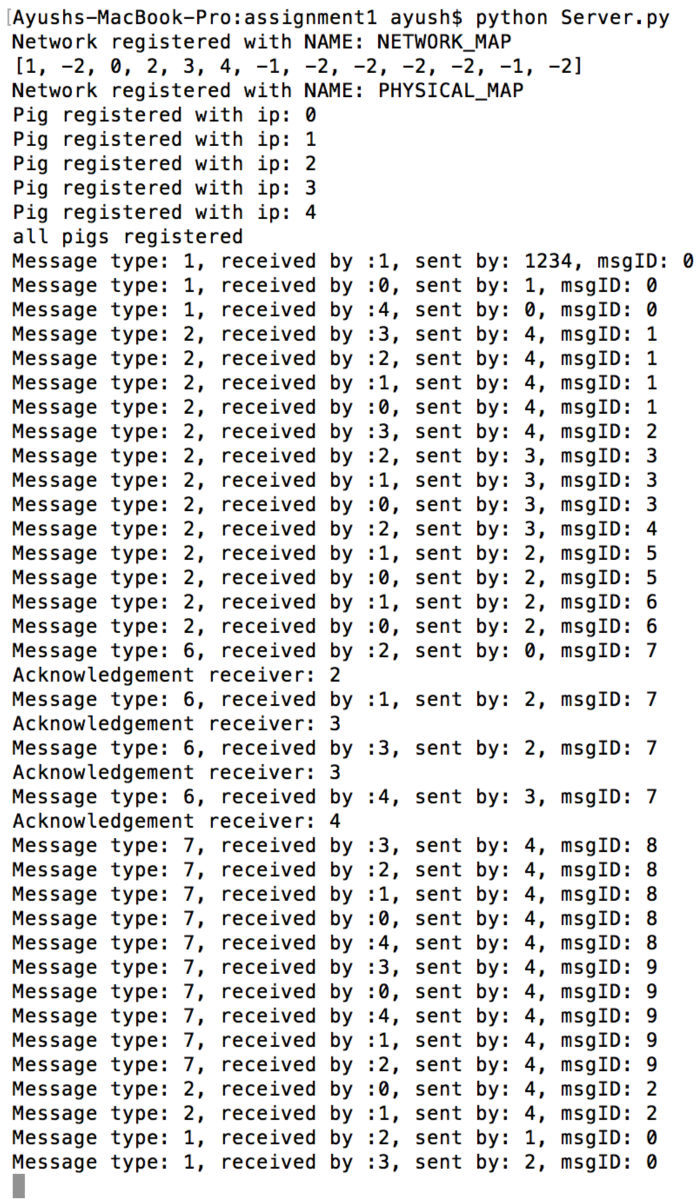
Some interpretations:

- Pigs are represented by integers ≥ 0
- Stones/walls are represented by -1
- Empty spaces are represented by -2

Pig Status interpretation:

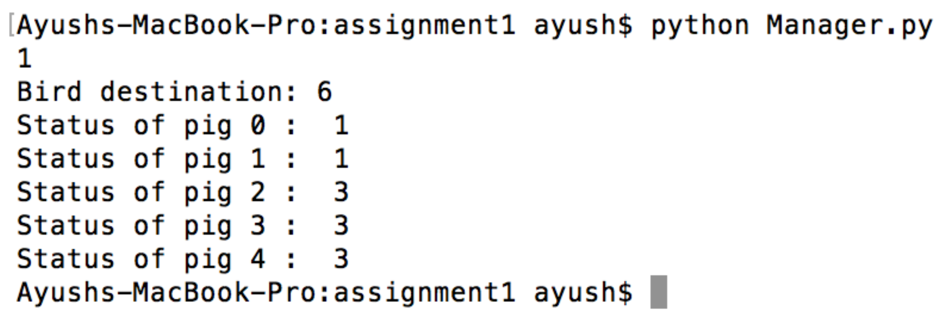
- 1 - Alive
- 2- Dead
- 3 - Evaded the attack and still alive

Test case output 1:



```
[Ayushs-MacBook-Pro:assignment1 ayush$ python Server.py
Network registered with NAME: NETWORK_MAP
[1, -2, 0, 2, 3, 4, -1, -2, -2, -2, -2, -1, -2]
Network registered with NAME: PHYSICAL_MAP
Pig registered with ip: 0
Pig registered with ip: 1
Pig registered with ip: 2
Pig registered with ip: 3
Pig registered with ip: 4
all pigs registered
Message type: 1, received by :1, sent by: 1234, msgID: 0
Message type: 1, received by :0, sent by: 1, msgID: 0
Message type: 1, received by :4, sent by: 0, msgID: 0
Message type: 2, received by :3, sent by: 4, msgID: 1
Message type: 2, received by :2, sent by: 4, msgID: 1
Message type: 2, received by :1, sent by: 4, msgID: 1
Message type: 2, received by :0, sent by: 4, msgID: 1
Message type: 2, received by :3, sent by: 4, msgID: 2
Message type: 2, received by :2, sent by: 3, msgID: 3
Message type: 2, received by :1, sent by: 3, msgID: 3
Message type: 2, received by :0, sent by: 3, msgID: 3
Message type: 2, received by :2, sent by: 3, msgID: 4
Message type: 2, received by :1, sent by: 2, msgID: 5
Message type: 2, received by :0, sent by: 2, msgID: 5
Message type: 2, received by :1, sent by: 2, msgID: 6
Message type: 2, received by :0, sent by: 2, msgID: 6
Message type: 6, received by :2, sent by: 0, msgID: 7
Acknowledgement receiver: 2
Message type: 6, received by :1, sent by: 2, msgID: 7
Acknowledgement receiver: 3
Message type: 6, received by :3, sent by: 2, msgID: 7
Acknowledgement receiver: 3
Message type: 6, received by :4, sent by: 3, msgID: 7
Acknowledgement receiver: 4
Message type: 7, received by :3, sent by: 4, msgID: 8
Message type: 7, received by :2, sent by: 4, msgID: 8
Message type: 7, received by :1, sent by: 4, msgID: 8
Message type: 7, received by :0, sent by: 4, msgID: 8
Message type: 7, received by :4, sent by: 4, msgID: 8
Message type: 7, received by :3, sent by: 4, msgID: 9
Message type: 7, received by :0, sent by: 4, msgID: 9
Message type: 7, received by :4, sent by: 4, msgID: 9
Message type: 7, received by :1, sent by: 4, msgID: 9
Message type: 7, received by :2, sent by: 4, msgID: 9
Message type: 2, received by :0, sent by: 4, msgID: 2
Message type: 2, received by :1, sent by: 4, msgID: 2
Message type: 1, received by :2, sent by: 1, msgID: 0
Message type: 1, received by :3, sent by: 2, msgID: 0
```

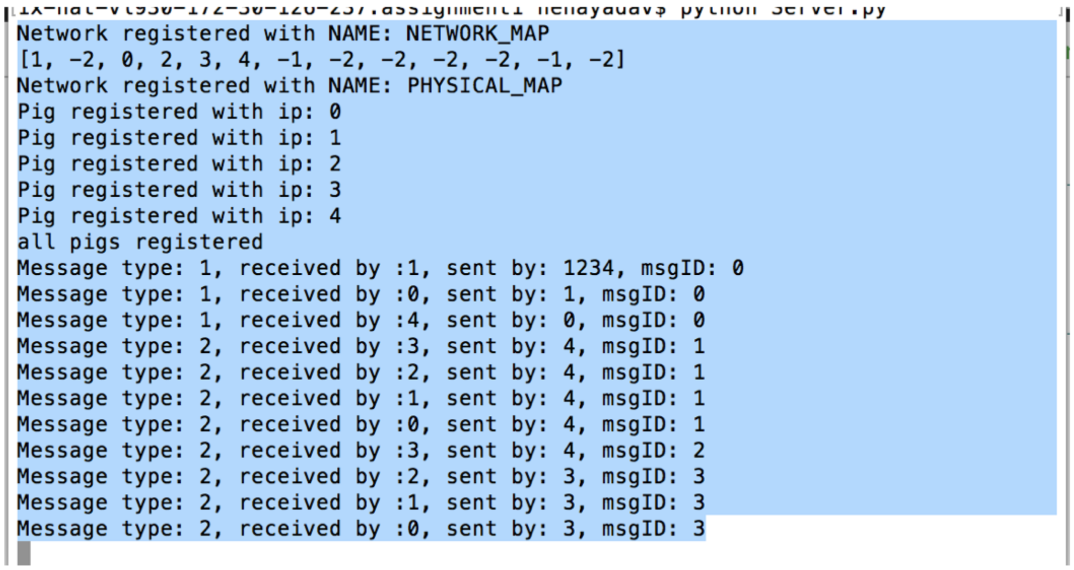
Figure 4: Output at server terminal.



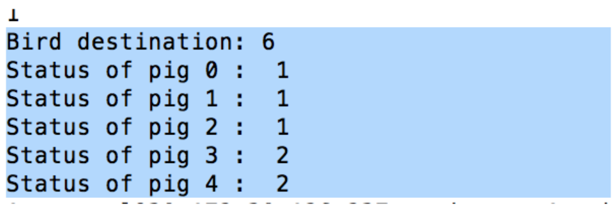
```
[Ayushs-MacBook-Pro:assignment1 ayush$ python Manager.py
1
Bird destination: 6
Status of pig 0 :  1
Status of pig 1 :  1
Status of pig 2 :  3
Status of pig 3 :  3
Status of pig 4 :  3
Ayushs-MacBook-Pro:assignment1 ayush$
```

Figure 5: Output at manager terminal.

Test case output 2:



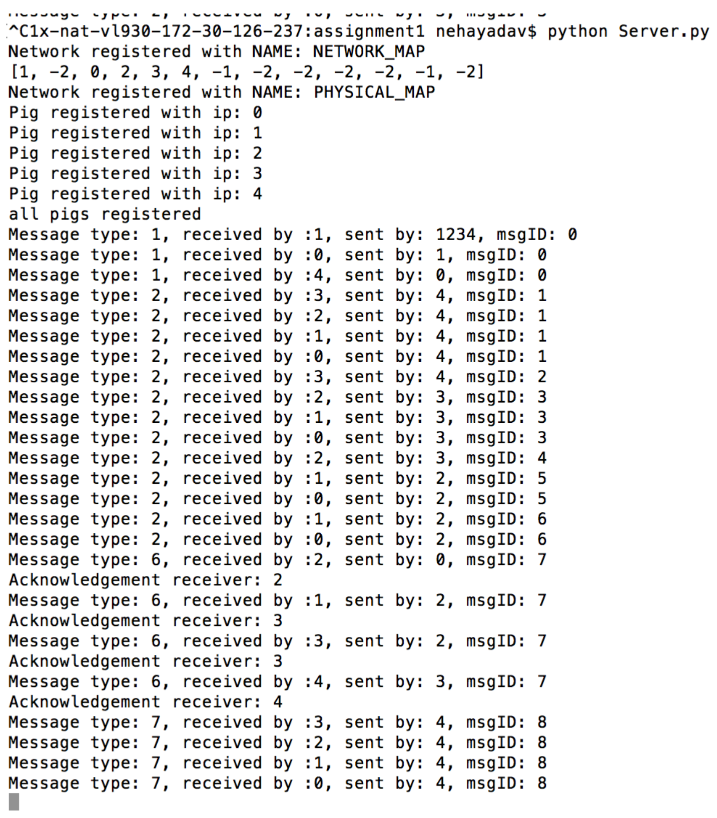
```
ix-nat-vl950-172-30-120-237:assignment1 nehayadav$ python Server.py
Network registered with NAME: NETWORK_MAP
[1, -2, 0, 2, 3, 4, -1, -2, -2, -2, -2, -1, -2]
Network registered with NAME: PHYSICAL_MAP
Pig registered with ip: 0
Pig registered with ip: 1
Pig registered with ip: 2
Pig registered with ip: 3
Pig registered with ip: 4
all pigs registered
Message type: 1, received by :1, sent by: 1234, msgID: 0
Message type: 1, received by :0, sent by: 1, msgID: 0
Message type: 1, received by :4, sent by: 0, msgID: 0
Message type: 2, received by :3, sent by: 4, msgID: 1
Message type: 2, received by :2, sent by: 4, msgID: 1
Message type: 2, received by :1, sent by: 4, msgID: 1
Message type: 2, received by :0, sent by: 4, msgID: 1
Message type: 2, received by :3, sent by: 4, msgID: 2
Message type: 2, received by :2, sent by: 3, msgID: 3
Message type: 2, received by :1, sent by: 3, msgID: 3
Message type: 2, received by :0, sent by: 3, msgID: 3
```



```
1
Bird destination: 6
Status of pig 0 :  1
Status of pig 1 :  1
Status of pig 2 :  1
Status of pig 3 :  2
Status of pig 4 :  2
```

Figure 6: Output

Test case output 3:



```
^C1x-nat-vl930-172-30-126-237:assignment1 nehayadav$ python Server.py
Network registered with NAME: NETWORK_MAP
[1, -2, 0, 2, 3, 4, -1, -2, -2, -2, -2, -1, -2]
Network registered with NAME: PHYSICAL_MAP
Pig registered with ip: 0
Pig registered with ip: 1
Pig registered with ip: 2
Pig registered with ip: 3
Pig registered with ip: 4
all pigs registered
Message type: 1, received by :1, sent by: 1234, msgID: 0
Message type: 1, received by :0, sent by: 1, msgID: 0
Message type: 1, received by :4, sent by: 0, msgID: 0
Message type: 2, received by :3, sent by: 4, msgID: 1
Message type: 2, received by :2, sent by: 4, msgID: 1
Message type: 2, received by :1, sent by: 4, msgID: 1
Message type: 2, received by :0, sent by: 4, msgID: 1
Message type: 2, received by :3, sent by: 4, msgID: 2
Message type: 2, received by :2, sent by: 3, msgID: 3
Message type: 2, received by :1, sent by: 3, msgID: 3
Message type: 2, received by :0, sent by: 3, msgID: 3
Message type: 2, received by :2, sent by: 3, msgID: 4
Message type: 2, received by :1, sent by: 2, msgID: 5
Message type: 2, received by :0, sent by: 2, msgID: 5
Message type: 2, received by :1, sent by: 2, msgID: 6
Message type: 2, received by :0, sent by: 2, msgID: 6
Message type: 6, received by :2, sent by: 0, msgID: 7
Acknowledgement receiver: 2
Message type: 6, received by :1, sent by: 2, msgID: 7
Acknowledgement receiver: 3
Message type: 6, received by :3, sent by: 2, msgID: 7
Acknowledgement receiver: 3
Message type: 6, received by :4, sent by: 3, msgID: 7
Acknowledgement receiver: 4
Message type: 7, received by :3, sent by: 4, msgID: 8
Message type: 7, received by :2, sent by: 4, msgID: 8
Message type: 7, received by :1, sent by: 4, msgID: 8
Message type: 7, received by :0, sent by: 4, msgID: 8
```

Figure 7: Output

```
class Pig:
    def __init__(self, ip, daemon, nameServer):
        """
        self.ip = pig's ip
        self.messageCache = initial message chache
        self.status = The pig's initial status
        self.alertCaller = the IP that alerted you
        self.orginalSender = Flag if this pig was the original sender of TAKE_SHELTER
        :param ip: Integer
        :param daemon: daemon to register
        :param nameServer: nameserver to register
        """

    def pushMessage(self, message):
        """
        The public function which sends the message according to its type to various handlers
        :param message: message packet
        :return:
        """

    def handleIAmSafe(self, message):
        """
        This method handles the I_AM_SAFE message.
        I_AM_SAFE message is sent by an original alerting node to its physical neighbour to mark
          itself safe now.
        An I_AM_SAFE message has following params:
        0: Message Type,
        1: Message ID
        2: Message sender
        3: Receiver's physical address
        :param message: the message packet
        :return:
        """

    def sendIAmSafe(self, nbr, message):
        """
        This method sends the I_AM_SAFE message.
        I_AM_SAFE message is sent by an original alerting node to its physical neighbour to mark i
          tself safe now.
        :param nbr: The IP neighbour
        :param message: message packet
        :return:
        """

    def _handleBirdApproachingMessage(self, message):
        """
        This function handles the BIRD_APPROACHING message.
        THE BIRD_APPROACHING message has following parameters:
        0: Message Type,
        1: Message ID
        2: Message sender
        3: Attack Target coordinate
        4: The bird attack time.
        :param message:
        :return:
        """
```

```
def _sendBirdApproachingMessage(self, nbr, message):
    """
    This function relays the BIRD_APPROACHING message to it network neighbours.
    :param nbr: the neighbour's IP
    :param message: message packet
    :return:
    """

def _handleTakeShelterMessage(self, message):
    """
    This function handles the TAKE_SHELTER message.
    A TAKE_SHELTER message has following params:
    0: Message Type,
    1: Message ID
    2: Message sender
    3: Target Physical address
    4: Sender Type : {Original, Relay}
    :param message: message packet
    :return:
    """

def _sendTakeShelterMessage(self, nbr, message):
    """
    This functions sends the take shelter message to its neighbours.
    :param nbr: neighbour IP
    :param message: message packet
    :return:
    """

def checkStatus(self):
    """
    Returns status
    Status can be three types:
    1) ALIVE
    2) DEAD
    3) EVADED - basically it means alive but could have been affected
    :return: status
    """

def _handleAcknowledgement(self, message):
    """
    This function handles the acknowledgement message.
    An ACKNOWLEDGEMENT message contains the following parameters:
    0: Message Type,
    1: Message ID
    2: Message sender
    3: Acknowledgement type.
    4: Acknowledgement reciever's IP address
    :param message: message packet
    :return:
    """

def _sendAcknowledgement(self, nbr, message):
    """
    This function sends acknowledgement to its neighbours.
    :param nbr: neighbour IP
    :param message: message packet
    :return:
    """

def _sendAlert(self):
    """
    Pig is in threat. So, it sends a Take Shelter message to its immediate PHYSICAL neighbours.
    If the pig can move then it forwards it to no one.
    The message TAKE_SHELTER is forwarded only when the pig can't move.
    :return:
    """

def _isGettingAffected(self, targetHit):
    """
    This function checks if pig is getting affected.
    It sees if the bird is attacking him or if the bird is attacking a wall and he is beside the wall.
    :param targetHit: the coordinate where bird is falling
    :return: boolean
    """

def _canMove(self):
    """
    This function checks to see, if there is an empty space to move.
    :return: boolean
    """

def _getPhysicalAddress(self):
    """
    This function returns the physical address of the pig.
    :return: physical address (integer)
    """

def _getNetworkNeighbours(self):
    """
    This function returns a list of neighbours of the pig
    :return: a list of neighbours
    """

def _registerOnServer(self, daemon, nameServer):
    """
    This function registers the pig with the server
    :param daemon: daemon process
    :param nameServer: server name
    :return:
    """
```