Programming Assignment 1: Angry Birds: Part Deux (Smarter Pigs Edition)

DESIGN DOCUMENT

Submitted by: - **Rahul Raj and Olenka Dey**

---

1. Introduction

The main aim of this assignment was to understand the working and implementation of P2P architecture. For this assignment we have constructed an unstructured P2P architecture like Gnutella and we have used Remote Procedure Calls (RPCs) for communicating between different peers. Following is the file structure which we have implemented.

--"config" - This is a basic text file which contains all the configurable parameters of the game such as Number of pigs/peers, Pigs logical Neighbors, Speed of Birds etc.

--"run_me.sh" – Top most Module. Used to Start all the peers.

|---"main.py" – Calls the readConfigFile module and checks for the return value.
|---"readConfigFile.py" – It parses the config file and calls related functions.
|---"peer.py" – Peer code which will get copied to the different folders.
|---"initializePeers.py" – Creates the grid for pigs and stones and initialized the peers depending on their logical neighbors.
|---"createPeers.py" – This will create n different copies of peers.py at n different folders which will act as n different peers.
|---"physicalNeighbors.py" – This will return the physical Neighbors of a peer if enquired.

Make sure that all these files exist in the same directory. We will discuss in-depth implementation of the peers in the implementation section.

2. How to Run

Running the game involves 2 steps:

1. config File – Look into this file to have an idea of what all parameters are configurable and you can change any of these parameters in order to test the system.

2. running run_me.sh in the terminal. This will start all the peers and no need for the user to open in n different terminals, as all the messages will be printed in the single terminal for the user convenience. Use the following command to run it.

./run_me.sh

This command should work, if it doesn't work you have some permissions issue, use **chmod 775 run_me.sh** to give the desired permission to this shell script. Please look into the Output/Results document for the different results.

3. Design and Implementation

Design:

We choose Python as the programming language for this lab. We have used simpleXMLRPC for the peers' implementation. We have written a shell script which starts all the peers together. Following are few of our design considerations.

- Parameters like number of peers/pigs, peers logical neighbors, number of stones are configurable from the config file.
- We have considered 2D plane for the peers. So each peer and stone will be assigned a random coordinate (x,y) on that 2D plane. The coordinate/location of the bird can be configured from the config file. The grid will be of the size n*n where n specifies the number of pigs.

- Each peer has a separate client and server thread running. Whenever a new request come to the server it goes into the wait queue if the server is already busy. All the peers have different port numbers.
- One of the Design consideration is that we have considered Unstructured P2P Architecture and the peers can talk in only one direction so it is unidirectional. We could have easily done it bidirectional but the flooding of messages would have been an issue in that case.
- If the peers don't have any logical neighbors, it will send the message to the first server which initiated the flooding.
- Once the game starts (using ./run_me.sh) the target coordinate i.e. the birds landing coordinate will be flooded to all the connected peers with the appropriate hop delay as specified in the config file.
- If the bird lands on the pig and the pig is able to take shelter, then the bird dies and the **score is 0**.
- If the bird lands on a pig and the target pig as well as physical neighboring pig is not able to take shelter, then both the pig dies and the **score of 2** is recorded.
- If the bird lands on a stack of stone and the stone topples and the neighboring pig is not able to take shelter, then the neighboring pig dies and the **score of 1** is awarded to the bird.
- The game will ask for use input once, if the bird dies initially and will exit if the bird lands outside the grid.
- If Hop count is shown as -1, indicates that the peer is sending status to the others.

Implementation:

In this part we will give an overview of different files and what each file intend to do.

1. **run_me.sh** – To start the game run this file. Make sure it has the required permissions to run. If not change the permissions by doing **chmod 775 run_me.sh** and then try to execute "**./run_me.sh**" in the terminal. This file will start all the peers at a go and we don't have to start each peer individually. To check the running peers, you can do **ps -eaf | grep python** on the terminal.
2. **config** file – Contains the parameters which are configurable and the user can alter any of the parameters before or after each run. If you want to change number of peers or you want to change the bird speed, you have to make changes to this file. This is the only file which user should change and no other files should be touched.
3. **main.py** – Main.py initiates setting up the architecture for the program by invoking calls to read the configuration (config) file and to restart the game based on the users input.
4. **readConfigFile.py –** This file basically parses the config file line by line and call relevant functions to create n different peers. It calculates the target Coordinate and initialize different peers based on that.
5. **physicalNeighbor.py –** This module calculates the physical neighbor relative to a peer and sends It back to the readConfigFile module. It also calculates the nearest pig to the bird launch, which basically is the first peer to start in the network.
6. **CreatePeers.py –** This file will create n different peers at n different directory locations. Those n different directories will be named as **pig[n]** here n stands for the peer number. It does all the placement of the stones, bird and the pigs on the grid and is responsible for printing it on to the console.
7. **InitializePeers.py -** This file will initialize all the peers with their placement coordinate on the grid. It gives signal to the nearest pig to the bird to start the message about bird's approaching.
8. **Peer.py-** This file is the peer/pig. It creates the server and client for each peer using SimpleXMLRPC and threading and is also responsible for establishing connections between two logical neighbors in the pig's network. It contains the functions that a peer remotely calls about the bird's approaching, status and getting location coordinates and so on.

NOTE: We have implemented the given interfaces and components with different names in the peer implementation. Following are the relation between the interface specified and what we have implemented.

- **bird_approaching(…)** -> This function is implemented with the name of "**set_target(…)**" in our peer code.
- **take_shelter(…)** -> This implementation is a part of "**check_coordinate(..)**" in our peer code.
- **status(…)**, **status_all(…)** -> "**send_status(…)**" is used in our code to send/flood status to the peer.
- "**was_hit(…)**" -> This is also a part of "**check_coordinate(…)**" and it determines whether the pig is being hit or evades.

4. **Tradeoff and Limitations**

- If hop delay > some threshold, then the program might not work. Recommended values of Hop Delay is 1 to 100 and these values are in milli-seconds.
- We are not clearing all the different folders(peers) once our program ends. One of the main reason for not doing so was the synchronization issue, as user would have given some hop delay in the config file and the peers would be running rather passing messages based on those delays, so if we would have deleted the folders/peers before all the peers finish execution it would have led to run time error.
- One of the tradeoff is that we are not flooding the score to all the peers instead we are flooding the status message, we though that flooding the status message was more informative to the other peers instead of score. Since we are not flooding the score we are not keeping track of cumulative score instead we are keeping track of scores in each round.
- One of the future work can be that we can extend the unidirectional peers to bidirectional, which will allow us to back propagate.
- One of the other assumptions is that the level of ripple is only till our own neighbors.