Cloud Pad

Our project was indeed a learning experience. It brought forward what we didn't knew about LAN, wifi, bluetooth communication. When we started we had following aims in our mind -

1. Make a multiplayer game and implement socket programming on Ubuntu
2. Realtime chatting over Bluetooth.
3. Playing a multiplayer game with pandaboard and a computer via wifi.
4. Remote Desltop Sharing of Pandaboard screen over Desktop.
5. Develop a GUI that will contain all the applications we make i.e, Multiplayer Game, Singleplayer Game, Music Blayer and Bluetooth App.

We did all work in python which is an interpreter language. You could learn about python from -

http://en.wikibooks.org/wiki/Think_Python

Also we used pygame for our game which could be learnt from

http://thepythongamebook.com/en:start

How to install Ubuntu on Pandaboard can be found at

http://pandaboard.org/

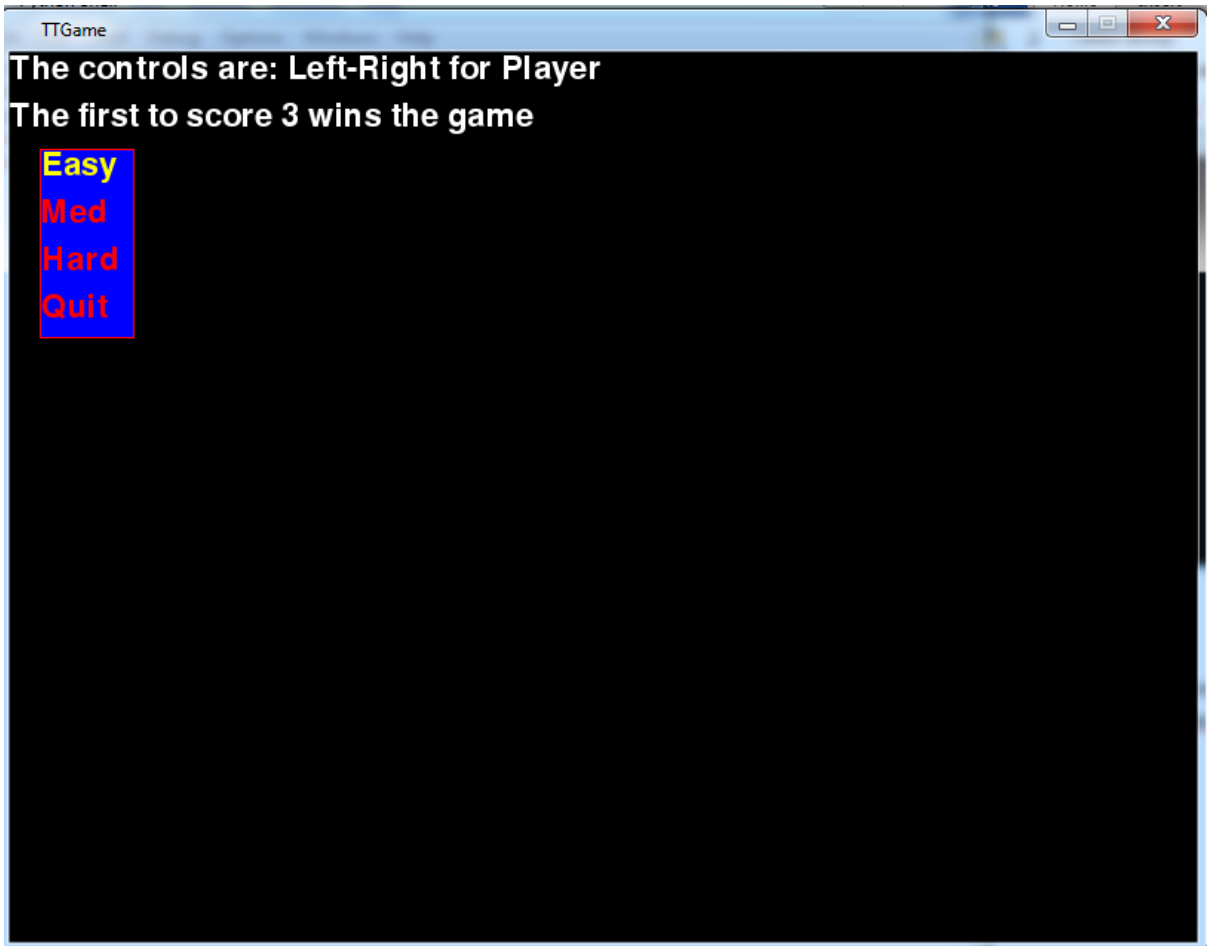
Game Designing

Our first target was to make the game on which we would implement LAN communication and cloud computing. We had to decide not only the game but also the language. After mutual consensus, we agreed upon Python. A simple 'TT-Game' was chosen to keep things simple. It was probably the simplest game on which we could try LAN gaming and also implement perspective view. Perspective view means that for each of the player, he would be player 1 on his machine and the view of the game would be compatible to him.

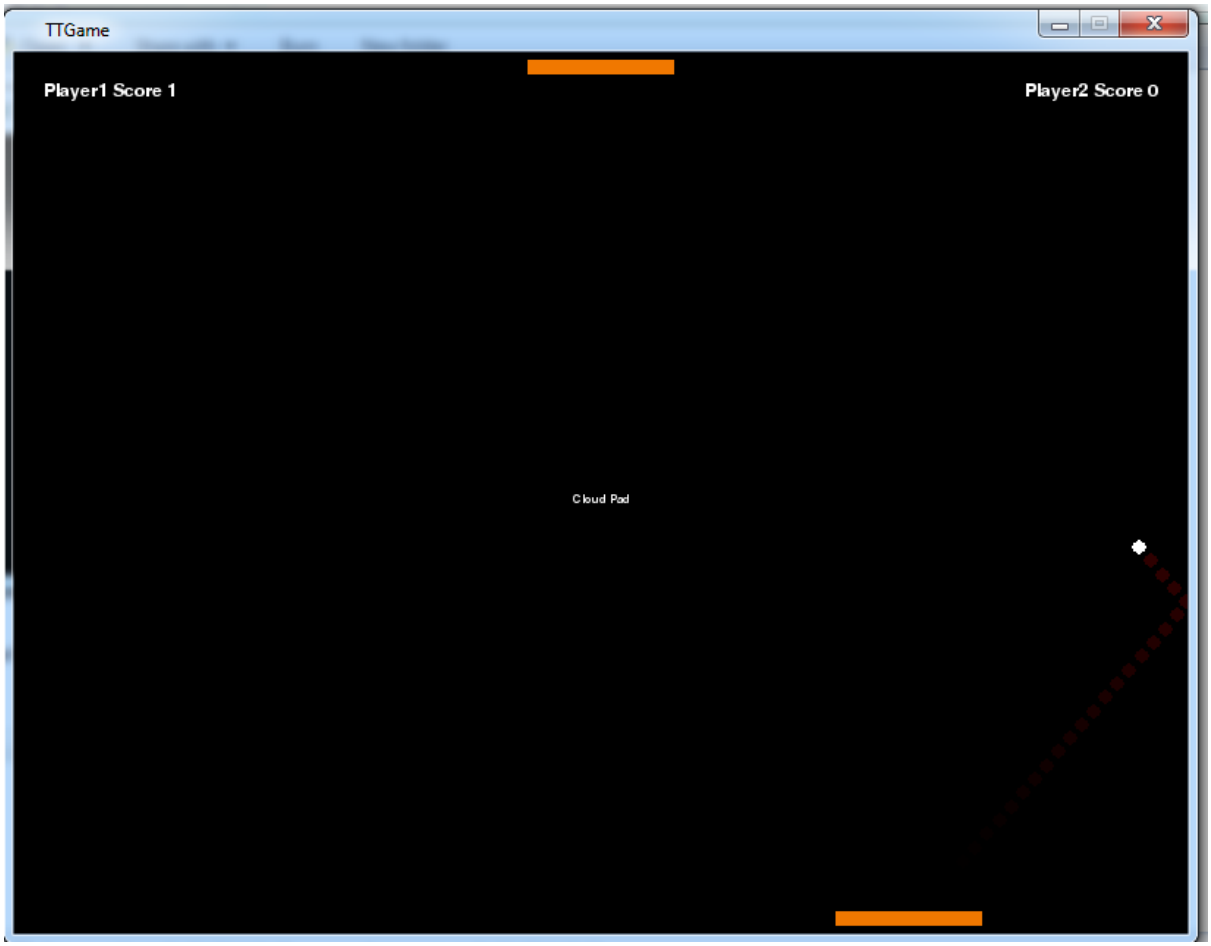
In TT-Game there are two paddles, one at the top of the screen, the other at the bottom and a ball travels between them. The target of the player is to prevent the ball reaching its screen end. The first to score 3 would win.

The game was made using pygame module of python. For the very first version of game we made one player game and the game would open with SimMen – menu class module which would direct us to the main screen of game.

The logic behind motion was to blit frames again and again with positions of ball and paddles shifted a bit in each frame.



Menu



Main Screen of TT-Game

In the one player version of Game the logic of AI was that the paddle would follow the ball's x-coordinate. The extent of following depended upon the level which you would choose in the menu screen.

Then we shifted to two player version. We just removed menu from this version. For menu we would have to send mouse-clicks via LAN, this would make the game much difficult to design.

We used two classes in the program, one for Paddle and the other for Ball. If a ball goes outside the screen, a new ball is appended at the centre of screen.

Text Editor

The text editor was written in python using pygame module. Through this editor you will be able to write on one machine and see the character-wise display on the other. We took each character from one user and stored it in a string and sent the whole string on the other

system and then this string is printed. While printing the string we had to take care of enter key, space bar, etc. For enter we increased the column pixels so that it appeared as if we were printing on next line.

AUDIO/VIDEO PLAYER

Our Audio/Video Player was made in python using wxpython. It consisted of a window which contains buttons for load, play, pause and stop. You can load any file of format .mid, .mp3, .wav, .au, .avi, .mpg.

GUI

GUI was written in python using wxpython module. Our GUI consisted of three icons. First icon was audio/video player in which you can load a song from computer and play it. Second icon was of text editor which when clicked gave way to a small window which asked if you wished to become server or client. If client you would have to enter IP of server. Then you would be directed to text-editor window where client would write anything and it would be displayed character by character on both machines. The third icon was of two player TT game. When the icon was clicked it also gave the same small window which asked if you wish to become server or client. Then the game is directed to the TT-Game screen where you could play TT-Game. What we linked via icons were actually separate program files. Linking was done just by simply importing these program files.

Socket Programming

1. Introduction

We decided to use Python as our programming language and pygame library of Python as we had some experience with. So, recalling it was not a big issue and we were comfortable with it. Morever, the biggest advantage of Python over Java or C is that it makes the codes much shorter and syntax is also user friendly as compared to C or Java. But whenever we had some problem with some syntax of python, we referred to free ebooks like 'Byte of Python' and 'Think Python'. We also used satndard documentation of pygame to refer to standard documentation of pygame:www.pygame.org/docs/.

Advice-Always use spaces for indentation, not both. This is because tabs are interpreted differently in different OS. For some OS, 1 tab may be equivalent to 4 spaces and for other it may be 8 spaces and python is very strict about indentation.

If you want to talk to a particular person, say, manager of an office request the receptionist to give u his/her no.,he/she will give u the extension code, say, 2435(if he/she agrees for the same). This extension is analogous to a socket of a server. A server has many communication ends. These ends are called sockets. To connect to a server, we have to connect to a socket. This socket can be considered to be a means of communication between a server and a client.

Getting nothing useful while trying to read big books on networking or socket library, I suddenly found a lifesaver short document on building

There are 2 types of types of sockets, namely:-

1. TCP-Transmission Control Protocol

2. UDP-User Datagram Protocfol

1. TCP: TCP is an Internet Protocol that manages the transmission of data such that it ensures the transmission of data to the receiver. It does so by set of conformation messages between server and client. But it is not suitable for applications where large data has to be transmitted like in games, video streaming, etc.

2. UDP: It is an Internet Protocol that does not ensure the transmission of data to the receiver. One might think it to be useless. But it is very useful in the applications where TCP fails i.e. games, video streaming, etc. It provides fast way of data transmission due to lack of confirmation messages to ensure data receival at the client end.

2. Game Building

So, we chose to use UDP Protocol for our game. According to our point of view, the best way for a novice to get a beginner's idea to sockets and their programming is to read the documentation in the below link(a 22 page document):-

http://www.ibm.com/developerworks/linux/tutorials/l-sock2/l-sock2-pdf.pdf

After reading this you can built simple server and client for chatting on terminal. In the beginning we needed this much only to allow data transfer in our game.

BEWARE!!!Always use timeout for sockets, otherwise, once the program gets struck, it may lead to blocking of current port being used.

Advice-Always keep track of the changes you made to the latest working game as socket programming may sometimes give strange results for beginners. Keeping track will allow you to guess which step made the program give strange results.

We did not develop the networking part of our game all at once. We developed our game in following steps:-

- First, we sent the keynames of keys we pressed while playing game on client to the server and just printed them on it(server).
  This required a simple server program which just received messages from a client and printed it on screen and the game we built above using pygame library was modified to act as a client by modifying the function which took input from keyboard such that not only it took input from keyboard but also sent the input taken immedietly to the server.

  Trap....Never forget to close the sockets after use, otherwise, you wouldn't be able to use them. They are not blocked permanently but may be until you restart your system(we did'nt researched this topic)

  First problem we faced was that we couldn't send data and found that we were using wrong destination. So, we globalized a variable where we stored the address of destination socket as a tuple((ip,port))

- Then we sent keynames of pressed keys from client to the server which translated them and used them to display what the client was playing i.e. instead of playing game on server through server keyboard, we played the game on server through client keyboard. The server used the received data instead of keyboard input to run the game.
  Trap....Never forget to set timeout for all sockets involved in receiving data because if for some reason the data fails to be received, then the socket would hang waiting for data receival and may lead to forced shut down of game which in turn will lead to blocking of all sockets remained open
- Next, we upgraded our game to online multiplayer game. We considered base paddle as representation of Player1 and top paddle as that of Player2. Player1 played on client and player 2 on server. From client, we sent the coordinates of base paddle from client to the server and the coordinates of top paddle from server to client. We did this in the final Draw() function where the final image of game screen was created and displayed. Both server and client received the keyboard input for the players they represented.

  In this we faced the greatest problems.....

Problem 1: Data Loss

The greatest disadvantage of using UDP protocol is data loss. This led to lagging of game and sometimes discrete movement of paddles. So we planned to send a data multiple times, like everytime we sent the coordinates of padles, we sent them 4-5 times. This would decrease the probability of data loss tremendously but will lead to totally insignificant delay. Increasing the number of times from 4 to >7 will lead to significant lag in between giving keyboard inputs and diplaying the action on screen.

Problem 2: Game stops when no data(of position of another player) present in the inbuilt buffer of received data leading to lagging

This problem was solved by playing with the timeout of socket receiving the inputs. In the beginning, we set the timeout of socket in both, server and client, to large value so that in beginning of the game the socket gave sufficient time for user to respond(start playing). But, whenever the socket was to receive data we decreased its timeout to about 0.01 sec and then again increased it to the old large value. It was like telling the socket to receive the data if avaliable in buffer, otherwise move forward and not wait for data receival.

We also put every statement that received data from another player in try-except block with pass statement in except block. This saved the game from ending the game whenever data was not present(socket raises exception whenever timeout occurs which closes the program) and us from another headache(I wouldn't use the word nightmare as we worked on nights and slept the day long)

Problem 3: Heavy transfer of data and sometimes leading to overflowing of buffer

This problem initially seemed to be solved by simply creating a list which stored the received inputs. We thought that lists in python can be extended practically infinitely and can store large data. But we were wrong!!!.....:(. So, we sought out a way to reduce the bulkiness of sent data tremendously.

We sent data in such a way that when the

- Finally, we made a multiplayer in which one player played on the server and other on the client. They both played against each other in realtime.
- But the data we sent to each other was very bulky as we sent each key pressed on both server and client to each other
- Then we used a technique. In this, we sent data only when the direction of paddle changed. Eg. when player on server was moving left and changed its direction to right, we send data to the client indicating the change in direction of server player. Similar rule applies to the client side. And our MULTIPLAYER REALTIME GAME was ready for online gaming!!!!