Topic	LAN GAME	
Class Description	**The student will learn to create a multiplayer game using LAN. The student will be able to create the first step of the game by creating a login window using Tkinter.**	
Class	**PRO C204**	
Class time	**45 mins**	
Goal	• Learn to create a LAN server. • Create a GUI using tkinter to accept the name of the player. • Learn to use sockets to connect multiple clients with the server.	

Resources Required	• Teacher Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	  ○ Smartphone
	• Student Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen	

Class structure	**Warm-Up** **Teacher-led Activity 1** **Student-led Activity 1** **Wrap-Up**	**10 mins** **15 mins** **20 mins** **5 mins**

WARM-UP SESSION - 10 mins	
Teacher Action	**Student Action**
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? Until now, we have played multiplayer games which use the databases. Today we'll create a LAN game which won't	**ESR**: Hi, thanks, Yes I am excited about it!

require any database and can be played with many friends. Isn't it exciting? Let's begin by quickly revising the previous class by answering a few questions.	**ESR:** Yes!
Q&A Session	
Question	**Answer**
How to add the label on the button? A. Using text() method B. Using label() method C. Using font() method D. Using add() method	A
Which method do you use to set the place of the elements on the screen? A. put() method B. place() method C. set() method D. add() method	B
TEACHER-LED ACTIVITY - 15 mins	
Teacher Initiates Screen Share	
ACTIVITY ● **Create a server and set up connections using sockets.** ● **Create a name window to get player names using Tkinter.**	
Teacher Action	**Student Action**
Great! Now that we have revised all the concepts, let's begin with today's class. Today, like I mentioned, we'll create a LAN game called LUDO, which won't require any database and can be played with many friends Let's get started, then.	**ESR:** **Yes!**

Teacher clones the boilerplate code from Teacher Activity 1

Note: Code to create a server is already provided in the boilerplate code. Explain the code to the Student.

Python has a very famous and widely used library for creating sockets. It is known as ***socket***. By adding this import line here, we have imported the sockets in the **server.py** file.

```
import socket
```

Sockets allow **you** to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralized data.

```
from  threading import Thread
```

Now to run the functions parallelly, which means that the client should receive the messages and write back to the server both at the same time, for this we'll be using threading. So first we'll import **threading** from **Thread.**

```
SERVER = None
PORT = None
IP_ADDRESS = None
```

Next we've declared some global variables such as **SERVER, PORT, IP_ADDRESS** and set their value to **None.** We are setting them to **None** as currently they won't hold any value, and we will be using them later to store values. The **None keyword** is used to define a null value, or no value at all. **None** is not the same as **0**, **False**, or an **empty string**.

```
CLIENTS = {}
```

Then we'll declare the empty **CLIENTS** dictionary, that will contain the information of the clients that have connected to the server.

We have all the necessary things imported to the **server.py** file.
Now, we just need to create the server.

```
# Boilerplate Code
def setup():
    print("\n")
    print("\t\t\t\t\t\t*** LUDO LADDER ***")


    global SERVER
    global PORT
    global IP_ADDRESS

    IP_ADDRESS = '127.0.0.1'
    PORT = 5000
    SERVER = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    SERVER.bind((IP_ADDRESS, PORT))

                                                SERVER.listen(10)


    print("\t\t\t\tSERVER IS WAITING FOR INCOMING CONNECTIONS...")
    print("\n")
```

To create the server, we created a function called **setup().**

Inside the function, we are first printing the name of the game.

Then we call all the global variables that we declared earlier which are **SERVER, PORT, IP_ADDRESS.**

We define the IP Address and the Port we are using. We are using **127.0.0.1** as the IP address and **5000** as the port. However, this port can be any number. Just make sure that it is not anything lower than ***1,024***, since those are reserved ports.

Next, we connect our server with the IP Address and the Port that we are using, and then our servers will be ready to listen for any incoming requests from the clients.
Our ***server*** is the ***socket*** that we created earlier with the ***socket.socket()*** function, and we are using the ***bind()*** function that takes a ***tuple*** with ***ip_address*** and ***port*** in it.

Once our server is binded, we can start listening on this server socket with the ***listen()*** function. Here, we have specified that we want only **10** connections.

Then we are printing a text which says that the "**SERVER IS WAITING FOR INCOMING CONNECTIONS.**"

Then finally we'll call the **setup()** function.

When we run the server, we should see this output.

```
                IP MESSENGER

    SERVER IS WAITING FOR INCOMMING CONNECTIONS...
```

Now we have the server ready, but have connected the client with the server yet.

So we won't be able to connect the server with the client.

We'll now be writing this code inside the **client.py** file to connect the server with the client.

```
#Boilerplate code
import socket
from tkinter import *
from  threading import Thread
from PIL import ImageTk, Image

screen_width = None
screen_height = None

SERVER = None
```

```
PORT = None
IP_ADDRESS = None


canvas1 = None

playerName = None
nameEntry = None
nameWindow = None
```

Inside the **client.py** we first imported all the necessary libraries such as **socket, tkinter, threading** and **PIL**.

We'll create **sockets** using the **socket**.
Using **Tkinter** we'll create a UI to accept player names and the game window.
Using **threading** we'll call multiple functions at once.
Using **PIL** we'll be able to add the images to the game.
Pillow is a **Python Imaging Library** (PIL), which adds support for opening, manipulating, and saving images

Then we declared a few variables such as **screen_width, screen_height, SERVER, PORT, IP_ADDRESS, canvas1, playerName ,nameEntry and nameWindow** and set their value to **None**. These are the global variables which we will be using in multiple functions as we want the same values to be used everywhere.

After this, we'll write a function which will set up a socket and connect the server with the client using the IP address and Port.

```
def setup():
    global SERVER
    global PORT
    global IP_ADDRESS

    PORT  = 5000
    IP_ADDRESS = '127.0.0.1'

    SERVER = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    SERVER.connect((IP_ADDRESS, PORT))
```

This will be the same as we did earlier to set up the server.

We'll create the **setup()** function. Inside the function, we'll use the global variables **SERVER, PORT, and IP_ADDRESS.**

Then we assign the Port and IP address that we are using.

Using the **connect()** function, we make a connection between the server and the client.

Note:- The boilerplate code ends here.

Now, when we have a connection with the server, we want to know the name of the person who is connecting to the server. As he/she would be the player of our game. To do so, we'll write a function called **askPlayerName()** and this function will be called in this **setup()** function.	
So what do we require to take the name of the player ?	**ESR:** We will require a form with the place to enter the name of the player and a save button to save the name.
Awesome, so we'll just design a UI which will allow us to take the name of the player.	

```
def askPlayerName():
    global playerName
    global nameEntry
    global nameWindow
    global canvas1
    global screen_width
    global screen_height
```

In this **askPlayerName()** function, we declare the global **playerName ,nameEntry. nameWindow, canvas1, screen_width, screen_height** variables that we require.

```
nameWindow  = Tk()
nameWindow.title("Ludo Ladder")
nameWindow.attributes('-fullscreen',True)
```

In the **nameWindow** variable, we'll create the parent window.

Then we'll set the name of the window using **title().** We'll set the name to "**Ludo Ladder**". To make the window full screen, we'll use the **attributes()** function and pass two values **fullscreen** and **True** to it.

```
screen_width = nameWindow.winfo_screenwidth()
screen_height = nameWindow.winfo_screenheight()
```

Then, using the **winfo_screenwidth() and winfo_screenheight()** function, we'll determine the height and width of the screen.

```
bg = ImageTk.PhotoImage(file = "./assets/background.png")
```

Using the **ImageTk.PhotoImage** attribute of **PIL** library. We will load the background image.

```
canvas1 = Canvas( nameWindow, width = 500,height = 500)
canvas1.pack(fill = "both", expand = True)
```

Using the canvas attribute, we'll create the canvas for **nameWindow** with **width = 500**, **height = 500**.
Using the **pack()** method, we'll make the window full screen. In this method, we'll pass **fill="both"** and **expand= True**.

Note:- The parameters used for width and height can change according to the screen size, so adjust it according to your screen size.

```
canvas1.create_image( 0, 0, image = bg, anchor = "nw")
```

Then, using the **create_image()** method, we'll add the background image to the window.

```
canvas1.create_text( screen_width/2, screen_height/5, text = "Enter Name",
font=("Chalkboard SE",100), fill="white")
```

Using the **create_text()** method, we will add the "**Enter Name**" text on the canvas.

```
nameEntry = Entry(nameWindow, width=15, justify='center', font=('Chalkboard
SE', 50), bd=5, bg='white')
nameEntry.place(x = screen_width/2 - 220, y=screen_height/4 + 100)
```

Using the **Entry()** method, we'll create the text input box on the canvas. And using **place()** we will give it the x and y positions.

Note: The parameters used for width and height can change according to the screen size, so adjust it according to your screen size.

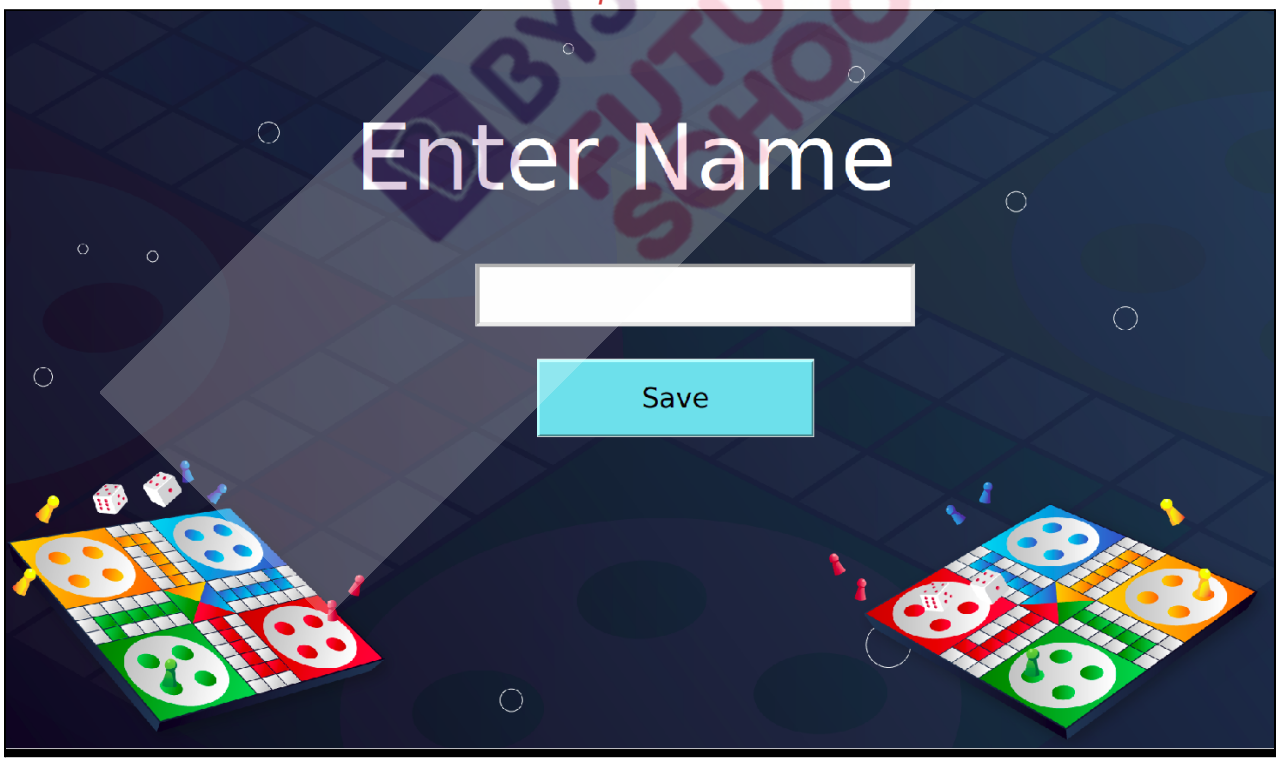
```
button = Button(nameWindow, text="Save", font=("Chalkboard SE",
30),width=15, command=saveName, height=2, bg="#80deea", bd=3)
   button.place(x = screen_width/2 - 130, y=screen_height/2 - 30)
```

Finally, we'll create the button using the **Button** attribute. And using the **place()** function we give it an x and y position on the canvas.

When this button is pressed, we'll call a function called **saveName().**

*The teacher codes to create the **askPlayerName()** and creates the Tkinter UI.*

The teacher runs the code to show the output to the student.



Till here we have created the server and added the GUI to accept the username from the user.

Can you tell me what the **saveName()** function will do? The **saveName** function will get the player name from the form. Then it will clear the entry box and close the window. Finally, it will encode the name of the person and send it to the server Doesn't that sound exciting, would you like to create this function.	**ESR:** Varied! **ESR:** Yes!

Teacher Stops Screen Share

STUDENT-LED ACTIVITY - 20 mins

- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start Screen Share.**
- **The teacher gets into Full screen.**

ACTIVITY

- **Write the saveName() function to encode the player name and send it to the server.**
- **Write the acceptConnections() function to accept connections and add players details to CLIENTS dict.**

Teacher Actions	Student Action
The teacher guides the student to clone the code from Student Activity 1. So, as discussed earlier, let's write the **saveName()**.	*The student clones the code from Student Activity 1.*

```
def saveName():
    global SERVER
    global playerName
    global nameWindow
    global nameEntry
```

First, start by defining the function. In the client file.

Inside this function call all the required variables such as **SERVER, playerName, nameWindow** and **nameEntry.**

```
playerName = nameEntry.get()
nameEntry.delete(0, END)
```

Using the **get()** method, get the player name from the global variable **nameEntry**. Then using the **delete()** method delete the data from the global variable **nameEntry**.

```
nameWindow.destroy()
```

Then close the window using the **destroy()** method.

```
# Sending Message to Server
SERVER.send(playerName.encode())
```

Finally, send the player name to the server by encoding it.

*The student codes to create the **saveName()** function*

Now we have a function to get the name from the client and send it to the server, but our server is still not accepting any connections.

So to make the server accept connections, we will write another function called **acceptConnections()** in the **server.py** file.

Can you tell me what this function will do?

ESR:

- This function will accept the connection.
- Receive the player name and decode it.
- Add the client details in the Clients dictionary that we created earlier.

Awesome. Let's write code for this.

```
def acceptConnections():
```

```
    global CLIENTS
    global SERVER
```

First, define the **acceptConnection()** function.

Write global variables **SERVER** and **CLIENTS.**

```
while True:
        player_socket, addr = SERVER.accept()
```

To keep the server running continuously and accept connections, use the **while** loop.

```
player_name = player_socket.recv(1024).decode().strip()
```

Get the player name into the **player_name** variable by receiving the data and then decoding it.

```
        if(len(CLIENTS.keys()) == 0):
            CLIENTS[player_name] = {'player_type' : 'player1'}
        else:
            CLIENTS[player_name] = {'player_type' : 'player2'}
```

As we have a multiplayer game, we need to know who is player one and who is player two, so we can check that by the data in the **CLIENTS** dictionary.

If the length of the **CLIENTS.keys()** is **0** then the player to join will be player 1 else he/she will be player 2.

Accordingly, create a key called **player_type** in the **CLIENTS** dictionary and add the player type.

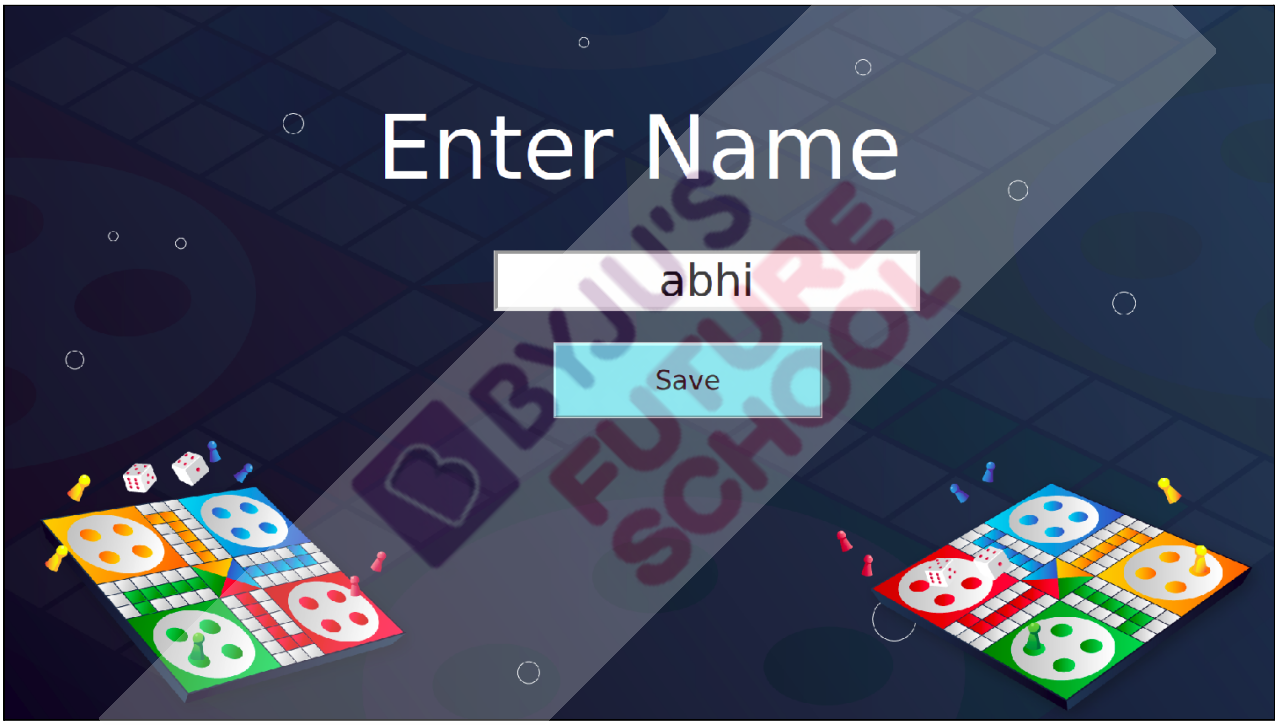
```
        CLIENTS[player_name]["player_socket"] = player_socket
        CLIENTS[player_name]["address"] = addr
        CLIENTS[player_name]["player_name"] = player_name
        CLIENTS[player_name]["turn"] = False
```

Similarly, create keys for other details such as **player_socket, address, player_name** and **turn.** And add the respective values.

```
print(f"Connection established with {player_name} : {addr}")
```

Finally, show a message that the connection has been established.

Call this function inside the **setup()** function of **server.py**	*The student codes to create the **acceptConnections()** function* *The student runs the code to check the output.*

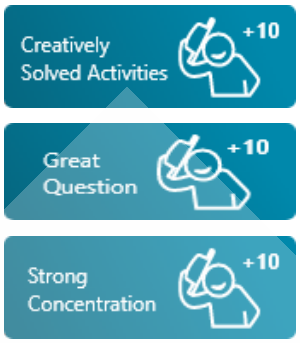



```
                                *** LUDO LADDER ***
                        SERVER IS WAITING FOR INCOMMING CONNECTIONS...

Connection established with abhi : ('127.0.0.1', 33728)
```

Till here, we were able to connect the client with the server and display which client has connected to the server.

Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 5 mins	
Quiz time - Click on In-Class Quiz	
Question	**Answer**
Why do we use sockets? A. To connect clients to the server B. To connect the server to clients C. To exchange the information between server and client D. All of the above.	**D**
What does the bind() function do? A. bind() keeps the server ready for connections B. bind() connects the server with the IP address and the Port C. bind() creates a connection between 2 clients D. bind() connects two servers.	**B**
`SERVER.listen(10)` What does the 10 indicate in this listen() function. A. 10 indicates the number of connections to listen to at a time. B. 10 indicates the number of connections that can connect to the server C. 10 indicates the bits of data that the server will listen to. D. 10 indicates the number of closed ports on the server.	**B**
End the quiz panel	
FEEDBACK • **Appreciate student's efforts in the class.** • **Ask the student to make notes for the reflection journal along with the code they wrote in today's class.**	
Teacher Action	**Student Action**

You get hats-off for your excellent work! In the next class, we'll write functions to send the message from client to client.	*Make sure you have given at least 2 hats-off during the class for:* 
Project Overview	
Teacher Clicks 	
ADDITIONAL ACTIVITIES	
Encourage the student to write reflection notes in their reflection journal using Markdown. Use these as guiding questions: ● What happened today? ○ Describe what happened. ○ The code I wrote. ● How did I feel after the class? ● What have I learned about programming and developing games? ● What aspects of the class helped me? What did I find difficult?	*The student uses the Markdown editor to write her/his reflections in the reflection journal.*

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity 1	Boilerplate code	https://github.com/whitehatjr/PRO-C204-TA1-boilerplate
Teacher Activity 2	Reference code	https://github.com/whitehatjr/PRO-C204-reference-code
Student Activity 1	Boilerplate Code	https://github.com/whitehatjr/PRO-C204-SA1-Boilerplate
Visual-Aid	Visual-Aid	
In-Class Quiz	In-Class Quiz	https://drive.google.com/file/d/1A7uJFOuaJ2jENyrj5AcI_EMwqoF8jW_r/view?usp=sharing