Topic	GAME WINDOW
Class Description	**Students will learn to create a game window using Tkinter. Students will also learn to create different boards for each player. And add dice for players to start playing the game.**
Class	**PRO C205**
Class time	**45 mins**
Goal	• Create a game window using tkinter. • Add the left board and right board for two player.bodies. • Add dice to the game

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** **10 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?	**ESR**: Hi, thanks, Yes I am excited about it!

Q&A Session	
Question	**Answer**

TEACHER-LED ACTIVITY - 10 mins	
Teacher Initiates Screen Share	
ACTIVITY • **Create a game window using Tkinter.** • **Adding left and right boxes for the two players.**	
Teacher Action	**Student Action**
In the last class we learned to create a server. And we also created a login window. And we were able to see the players who joined our server on the terminal/command prompt *Teacher clones the boilerplate code from Teacher Activity 1* *Note :- code for finishingBox() function and code to create dice is present as boilerplate . This will be used in the code later.* *Now the teacher can comment the code for finishingBox() and to create dice.* *Teacher explains the code for creating a game window to the student.*	**ESR:** **Yes!**

So we already have created the login window in last class so today we'll start by creating the game window. When do we want this game window to open? Correct, so to create the login window we will write a function called as **gameWindow() function.** And we'll use this function inside the **saveName().**	**ESR:** We want to open the game window when the player enters the name in the name window.

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

    playerName = nameEntry.get()
    nameEntry.delete(0, END)
    nameWindow.destroy()

    SERVER.send(playerName.encode())
```

```
#calling the gameWindow()
  gameWindow()
```

So here we have called the **gameWindow()** function inside the **saveName()** function.

Now we need to work on the code for this function to create the game window. What is the first step that we do? Yes. So let's do that first.	**ESR:** import/ use all the necessary variables in the function.

```
canvas2 = None
gameWindow = None
dice = None
```

So first we have declared the variables.

```
def gameWindow():
    global gameWindow
    global canvas2
    global screen_width
    global screen_height
    global dice
```

So we declared the variables.

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

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

In the **gameWindow** variable we'll create the parent window.
Then using the **.title()** method set the title of the game to the game window.
To make the window full screen we'll use the **attributes()** function and pass two values fullscreen and True to it.

```
canvas2 = Canvas( gameWindow, width = 500,height = 500)
canvas2.pack(fill = "both", expand = True)
```

Using the canvas attribute we'll create the canvas for gameWindow 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.

```
# Display image
   canvas2.create_image( 0, 0, image = bg, anchor = "nw")

   # Add Text
   canvas2.create_text( screen_width/2, screen_height/5, text = "Ludo
Ladder", font=("Chalkboard SE",100), fill="white")
```

Then using the create_image() method we'll add the background image to the window.
Using the **create_text()** method we will add the "Ludo Ladder" text on the canvas.

```
   gameWindow.resizable(True, True)
   gameWindow.mainloop()
```

The **resizable** method we'll set it to true as we do want the window to resize.
The **mainloop**() tells Python to run the **Tkinter** event loop.

Finally run the code and show the output.
Note -*Boilerplate for creating game window ends here.*

This is how the canvas will look.

This is a blank canvas as of now. And as we are making our version of the Ludo game we'll change it by just keeping a single line of boxes on both the sides of the Home and have a little race on who reaches first to the home base by rolling the dice.

To create these boxes we'll write two functions **leftBoard()** and **rightBoard().**

leftBoard() function will create the board for player one and **rightBoard()** will create board for player two.

```
leftBoxes = []
rightBoxes = []
```

First we'll declare the two variables as **leftBoxes** and **rightBoxes** and set the **empty array** as it's **value.**

```
def leftBoard():
    global gameWindow
    global leftBoxes
    global screen_height
```

Here we first define the function and inside it call the global variables.
Then we define a local variable **xPos** and set its value to 30 as this will be the initial x position of the boxes that we'll be creating .

```
xPos = 30
```

Then using the for loop we'll create 10 boxes. Our loop will start from 0 and finish at 10. In the loop we use the value from 0 to 11 this means that count the values between 0 and 11 but excluding 11.

```
for box in range(0,11):
```

We want the first box to be the player character and denote it with a red box. So to do that we'll use the if condition which will check if the box is 0 (the first box) then using the **Label()** method of Tkinter label the position with the red box and append it to the **leftBoxes array**.

```
        if(box == 0):
            boxLabel = Label(gameWindow, font=("Helvetica",30), width=2,
height=1, relief='ridge', borderwidth=0, bg="red")
```

```
            boxLabel.place(x=xPos, y=screen_height/2 - 88)
            leftBoxes.append(boxLabel)
            xPos +=50
```

Else label the box with white box and append in the **leftBoxes().**

```
          else:
            boxLabel = Label(gameWindow, font=("Helvetica",55), width=2,
height=1, relief='ridge', borderwidth=0, bg="white")
            boxLabel.place(x=xPos, y=screen_height/2- 100)
            leftBoxes.append(boxLabel)
            xPos +=85
```

Teacher codes to write the leftBoard() and call it inside the gameWindow() function.
Note:- the height and width will differ according to the screen size. Make sure to change the values for your screen size.

And how will we create the right board for another player? Yes! We'll follow the similar steps to create the right board.	**ESR:** We'll follow the same steps we followed for the **leftBoard**

```
def rightBoard():
    global gameWindow
    global rightBoxes
    global screen_height

    xPos = 988
    for box in range(0,11):
        if(box == 10):
            boxLabel = Label(gameWindow, font=("Helvetica",30), width=2,
height=1, relief='ridge', borderwidth=0, bg="yellow")
            boxLabel.place(x=xPos, y=screen_height/2-88)
            rightBoxes.append(boxLabel)
            xPos +=50
        else:
            boxLabel = Label(gameWindow, font=("Helvetica",55), width=2,
height=1, relief='ridge', borderwidth=0, bg="white")
            boxLabel.place(x=xPos, y=screen_height/2 - 100)
            rightBoxes.append(boxLabel)
            xPos +=85
```

The code will be the same for right board as left board, just change the initial X position.

Note:-The teacher can use the same code for the right board by making some changes to the code of lft board.

We have the functions but we haven't called them yet. So we'll need to call these first. Where should we call these? Yes! So we'll call these functions inside the **gameWindow()** function	**ESR:** We will call these functions inside the **gameWindow().**

```
leftBoard()
rightBoard()
```

Then run the code and to check the output.

Here we have the player boxes ready to play the game. But we don't have a destination for the players to reach to win. So let's add the finishing box to game window.

Note:- The code for finishing box is given as boilerplate.

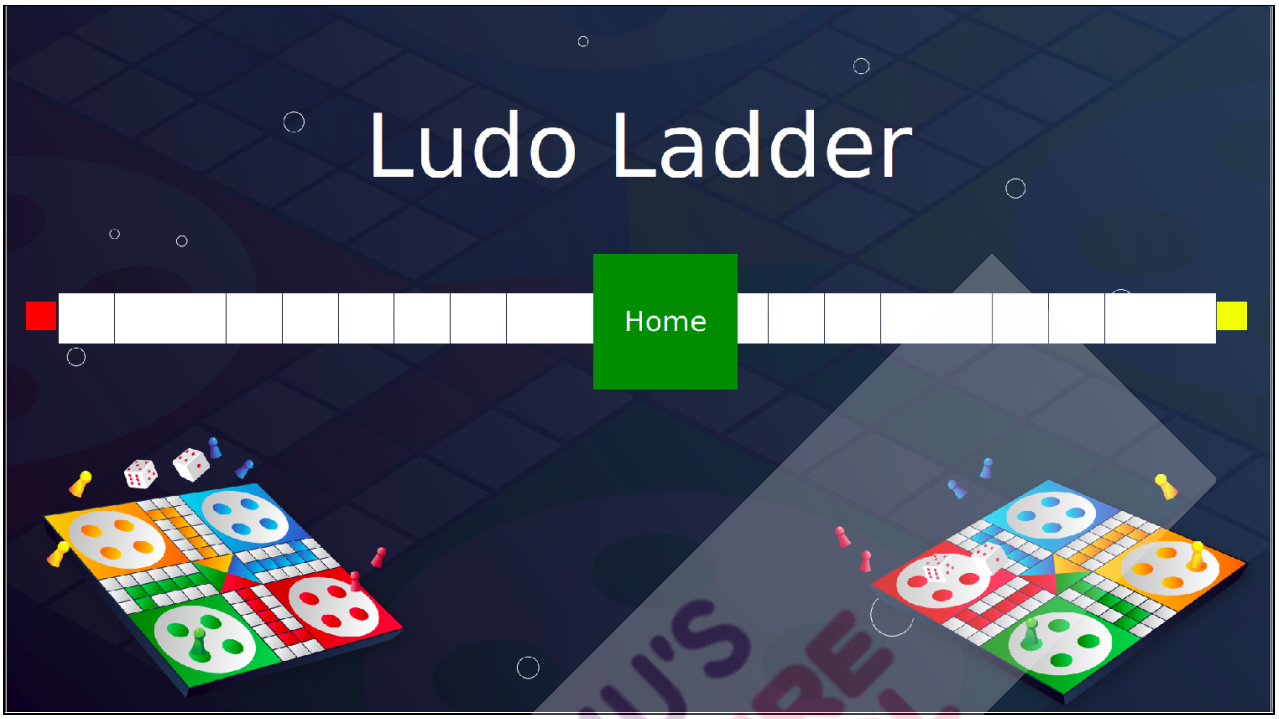
```
def finishingBox():
```

```
    global gameWindow
    global finishingBox
    global screen_width
    global screen_height

    finishingBox = Label(gameWindow, text="Home", font=("Chalkboard SE", 32), width=8, height=4, borderwidth=0, bg="green", fg="white")
    finishingBox.place(x=screen_width/2 - 68, y=screen_height/2 -160)
```

As we have done for other functions here also we'll first call all the necessary variables.
Using the **Label()** method, create the box with the Home text in between.
Using the **place()** method, place it on the screen.

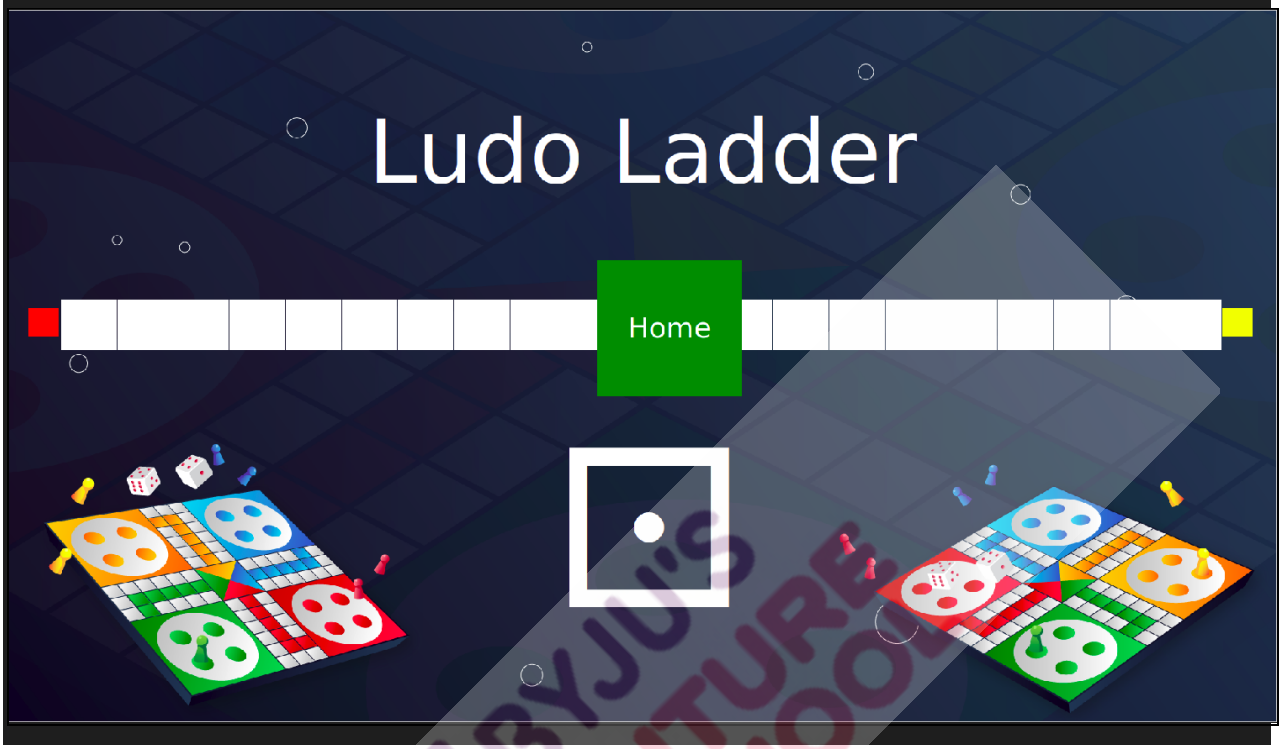
When we run the code we'll see an output like this.



Now we have the players ready but we are missing a crucial part of the ludo and that is the dice. Without the dice we can't play this game.

To create the dice we'll use the **create_text() method.** At the beginning this dice will only have the 1 as it value.

```
dice = canvas2.create_text(screen_width/2 + 10, screen_height/2 + 250, text
= "\u2680", font=("Chalkboard SE",250), fill="white")
```



Now we have the dice on the screen but we are not able to roll the dice to move any player. Also we wouldn't know which player has his/her first turn. What can we do about

ESR:

it? Awesome! Would you like to add code for the same?	We can create a button on the screen which when pressed will rotate the dice to show random numbers till 6. And we can show the button to the player who has his/her turn and not the other player. 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 • **Add a button to roll the dice.** • **Write a function to roll the dice.**	

- **Explore the working of handleClient() function to see how messages are sent and received.**

Teacher Actions	Student Action
The teacher guides the student to clone the code from Student Activity 1 So as discussed earlier we just have a static dice on the screen and we need to roll it in order for the game to proceed. So let's start by writing a function to roll the dice.	*Student clones the code from Student Activity 1* *Student starts writing the code for **rollDice()** function.*

```
def rollDice():
    global SERVER
    #create a number variable in which the list of all the ASCII characters
of the string will be stored
```

First start by defining the function.
Inside this function we will call all the required variables such as **SERVER.**

```
diceChoices=['\u2680','\u2681','\u2682','\u2683','\u2684','\u2685']
```

Declare a variable called **diceChoices.** This variable will contain the list of ASCII characters which represent the sides of the dice.

```
#configure the label
    value = random.choice(diceChoices)
```

We need to get the dice choices as random when the dice is rolled so we'll use the random function to get the random choice from the list and store it in the **value** variable.

When the dice is rolled we want both the players to see what the other player has scored. But as both the players will be playing on different screens how can we show the dice score on both the screens?

Yes! So here on the client side we'll only be doing the first part and that is the sending the score to the server.

ESR:
We can send the score of one player to the server and then the server can send the same score to the other player.

In the rollDice() function

```
    global playerType
    global rollButton
    global playerTurn
```

We'll first get all the global variables in the function that we require.

```
rollButton.destroy()
   playerTurn = False
```

Then we'll remove the roll button from the screen using **destroy() method.** As we don't want the same player to play continuously. We'll get the button back when the player turn changes. And we'll set the playerTurn to False for this player.

```
if(playerType == 'player1'):
      SERVER.send(f'{value}player2Turn'.encode())

  if(playerType == 'player2'):
      SERVER.send(f'{value}player1Turn'.encode())
```

Now to send the score to the server we'll first check the player type. If it's player1 then send the message to the server that it's player2Turn along with the dice score.

We'll write a similar condition for player2 and in the message we;ll write it's player1Turn.

Now we have the function ready. Where do we wanna call it?	*Student Completes the code for rollDice().* **ESR:** We want to call this function when the roll dice button is

So let's start by creating the roll button and call this function in it.	pressed.

```
global rollButton

   rollButton = Button(gameWindow,text="Roll Dice", fg='black',
font=("Chalkboard SE", 15), bg="grey",command=rollDice, width=20, height=5)
```

First we'll be starting with getting the required global variable which is **rollButton.** This variable will contain the button. So using **Button() method,** we'll create the button and call the **rollDice()** function in it.

```
   global playerTurn
   global playerType
   global playerName


   if(playerType == 'player1' and playerTurn):
       rollButton.place(x=screen_width / 2 - 80, y=screen_height/2  + 400)
   else:
```

```
        rollButton.pack_forget()
```

To get the roll button on screen when it's the players turn we'll write a condition which will check if the player type and player turn are of the same player then show the button else make it invisible using the **pack_forget()** method.

Earlier in the class we sent some message form the client to the server but we also need the server to send the message back to the other client as we want to show what the player1 what player2 is doing. So we'll write a function called as **handleClient()** which will do this for us.

```
def handleClient(player_socket,player_name):
    global CLIENTS
```

First we start by declaring the function which takes parameters such as **player_socket** and **player_name.**

```
 # Sending Initial message
    playerType =CLIENTS[player_name]["player_type"]
```

Next part would be to send the message to the player on whose turn it would be. Here we'll declare the variable called as **playerType.** This will contain the type of the player if its player 1 or player 2.

```
if(playerType== 'player1'):
        CLIENTS[player_name]['turn'] = True
```

```
        player_socket.send(str({'player_type' :
CLIENTS[player_name]["player_type"] , 'turn': CLIENTS[player_name]['turn'],
'player_name' : player_name }).encode())
```

If the player type is player1 then inside the **CLIENTS** dictionary we have a key called **turn.** We'll set it's value to **True.** And we'll send the message it's player1 turn and encode the message

```
 else:
        CLIENTS[player_name]['turn'] = False
        player_socket.send(str({'player_type' :
CLIENTS[player_name]["player_type"] , 'turn': CLIENTS[player_name]['turn']
}).encode())
```

In the else part we'll set the player turn to False
And send the message that it's player2 turn.

```
while True:
        try:
            message = player_socket.recv(2048)
            if(message):
                for cName in CLIENTS:
```

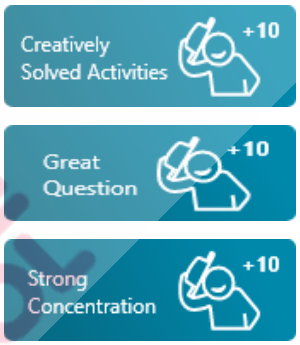
```
                cSocket = CLIENTS[cName]["player_socket"]
                cSocket.send(message)
        except:
            pass
```

We want this to happen every time so we'll write another condition which checks is the message is received in the player socket.
Then send the same message using the player socket..
We'll be using a **try except** block to send the message and pass if there are any exceptions.

Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 5 mins	
Quiz time - Click on In-Class Quiz	
Question	**Answer**
End the quiz panel	

FEEDBACK

- **Appreciate the student for their 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 Discussion	

Teacher Clicks	✖ End Class

ADDITIONAL ACTIVITIES	
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-C205-TA
Teacher Activity 2	Reference code	https://github.com/whitehatjr/Reference-code--205
Student Activity 1	Boilerplate Code	https://github.com/whitehatjr/PRO-C205-SA1