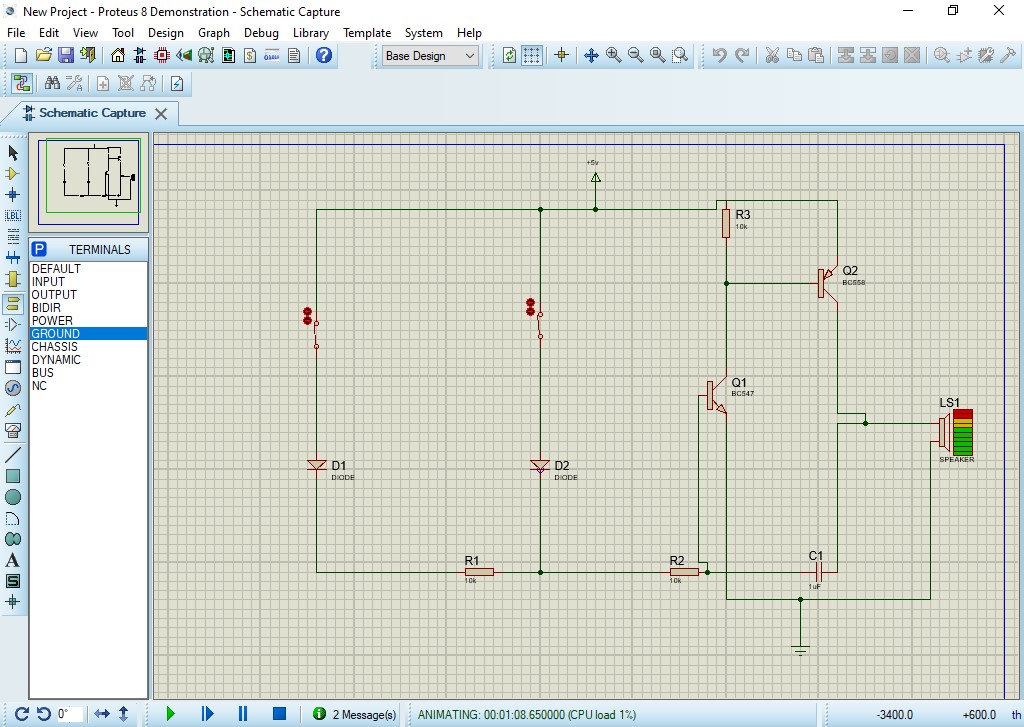
INDEX

|  |  |  |
| --- | --- | --- |
| Sr. No | Topics | Sign |
| 1 | Design and implement basics embedded circuits  1. Automatic Alarm system- Alarm should get trigger by senor  2. Timer based buzzer  3. Sensor based Counting device |  |
| 2 | Demonstrate communication between two embedded devices using UART port |  |
| 3 | Demonstrate an IoT based game which can be played between two player who are physically at a considerable distance. |  |
| 4 | Develop an IoT application for monitoring water levels in tanks and automatically start the motor to fill the tank if the level goes below the critical level. |  |
| 5 | Develop an IoT module to which measure the intensity of light and send the same to your PC/ Phone |  |
| 6 | Develop an IoT application for Motion detection. |  |

# PRACTICAL NO: 01

**Aim:** Design and implement basics embedded circuits Timer based Counting device.

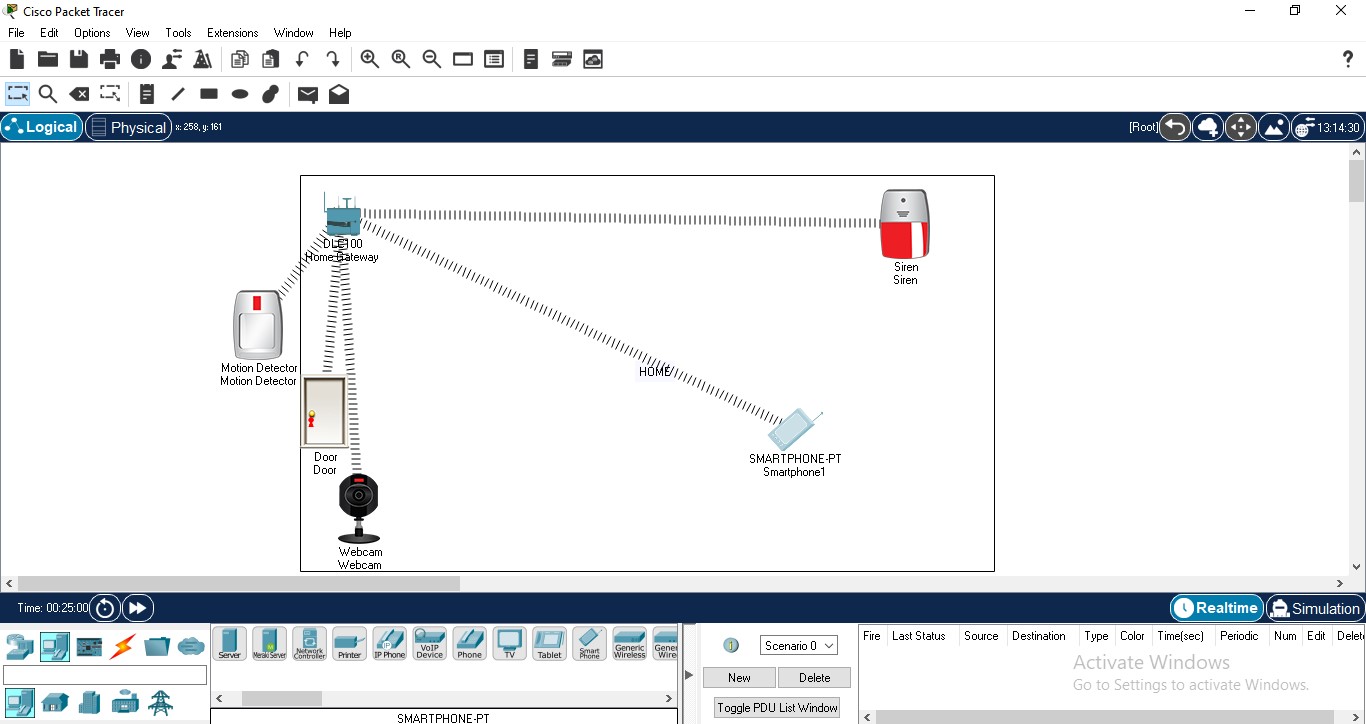
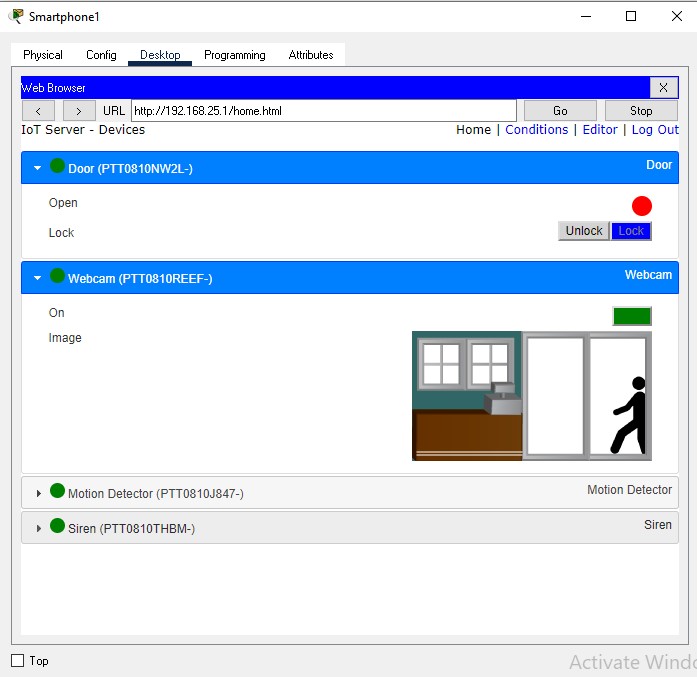
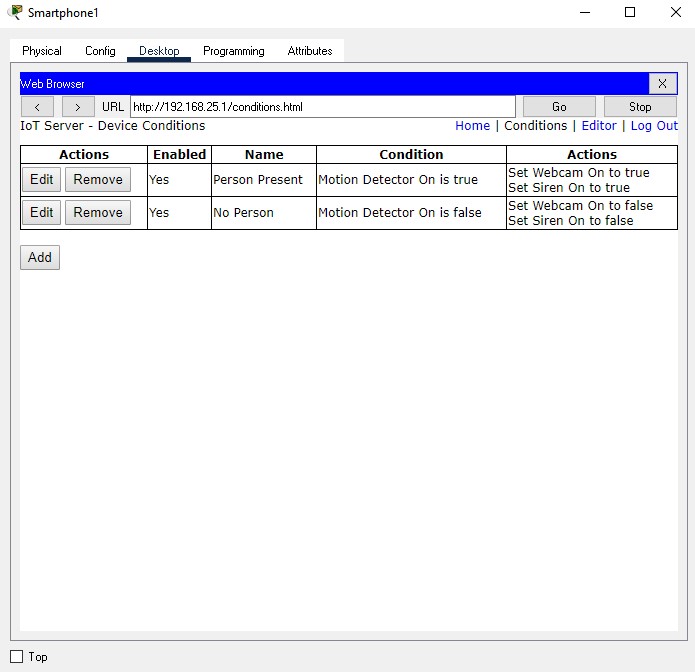
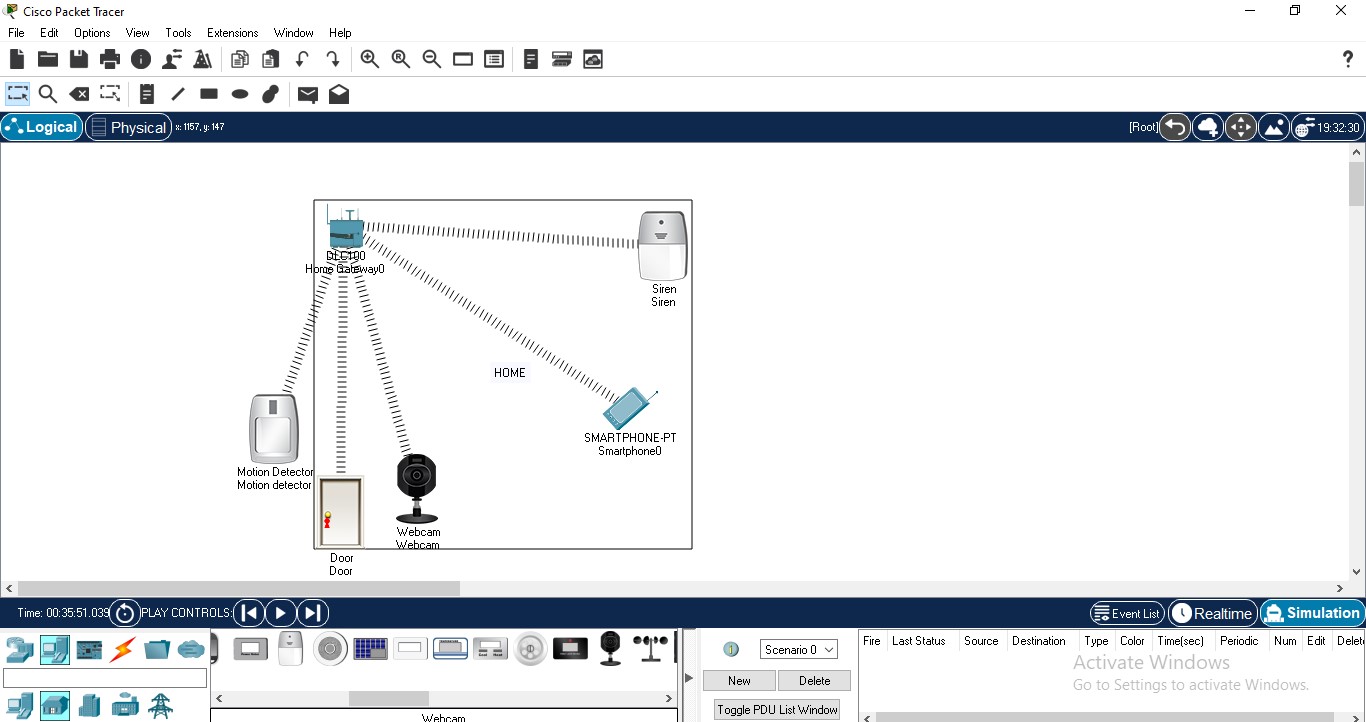
**Output:**



# PRACTICAL NO: 1

**Aim:** Automatic Alarm System –Alarm should get trigger by sensor.

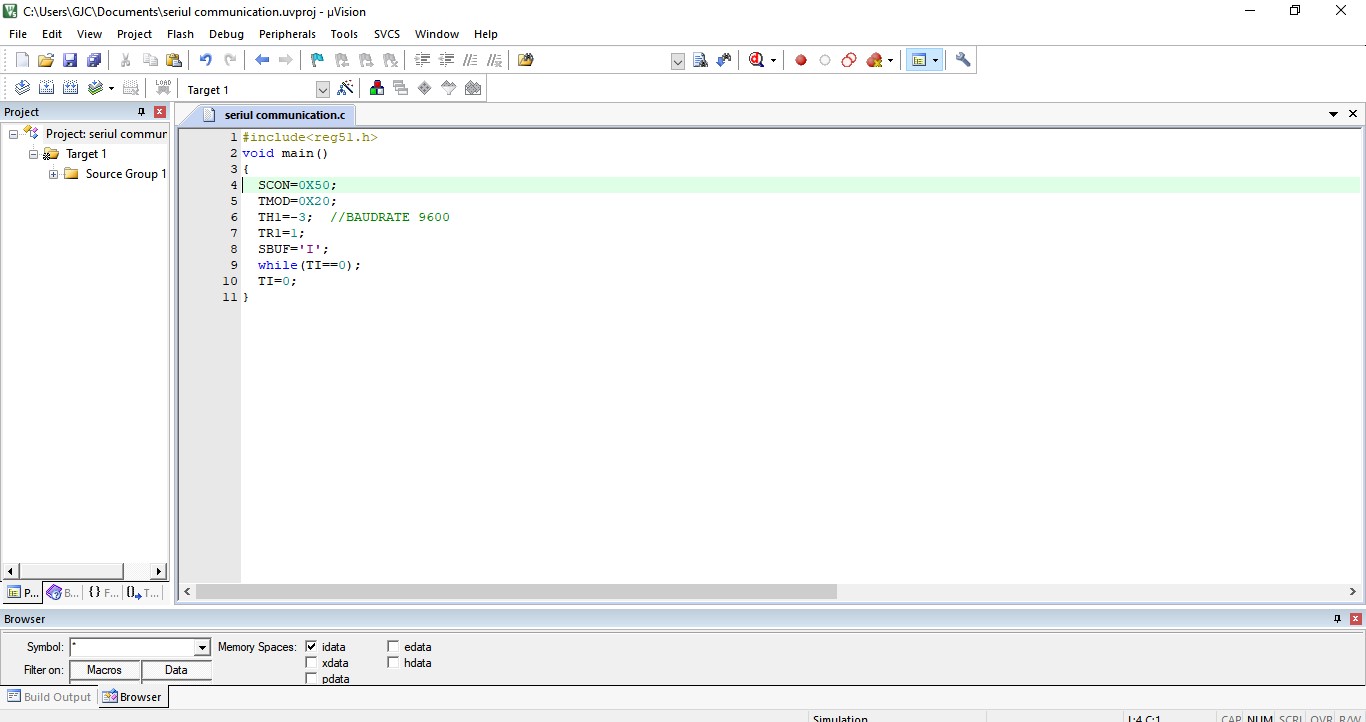
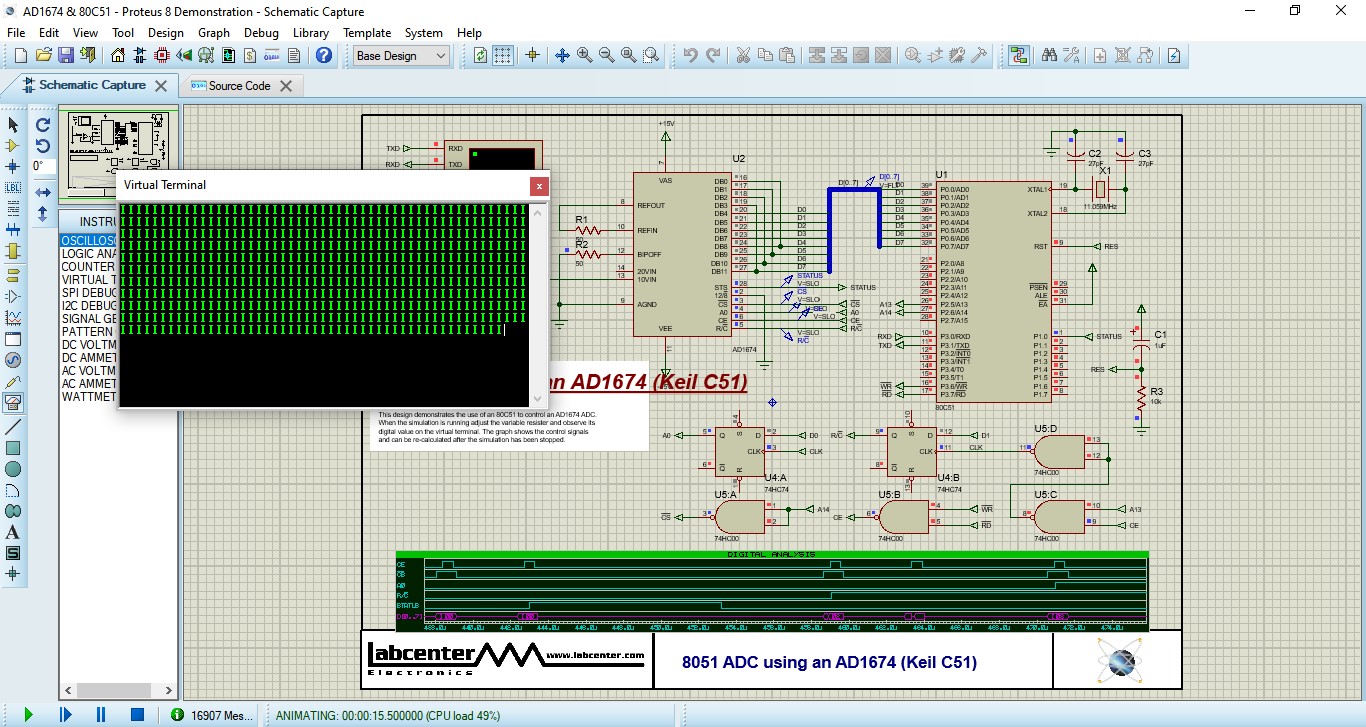
**Output:**



# PRACTICAL NO: 02

**Aim:** Demonstrate communication between two embedded devices using UART port .

**Output:**



# PRACTICAL NO: 03

**Aim:** Demonstrate an IoT based game which can be played between two player who are physically at a considerable distance.

**Code:** Server.py: import socket # for networking

import pickle # for sending/receiving objects from tic\_tac\_toe import TicTacToe

HOST = '127.0.0.1' # this address is the "local host" PORT = 12783 # port to listen on for clients s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) s.bind((HOST, PORT))

s.listen(5)

client\_socket, client\_address = s.accept() print(f"\nConnnected to {client\_address}!") player\_x = TicTacToe("X") rematch = True while rematch == True:

print(f"\n\n T I C - T A C - T O E ")

while player\_x.did\_win("X") == False and player\_x.did\_win("O") == False and player\_x.is\_draw() == False:

print(f"\n Your turn!") player\_x.draw\_grid()

player\_coord = input(f"Enter coordinate: ") player\_x.edit\_square(player\_coord)

player\_x.draw\_grid()

x\_symbol\_list = pickle.dumps(player\_x.symbol\_list) client\_socket.send(x\_symbol\_list)

if player\_x.did\_win("X") == True or player\_x.is\_draw() == True: break

print(f"\nWaiting for other player...") o\_symbol\_list = client\_socket.recv(1024) o\_symbol\_list = pickle.loads(o\_symbol\_list) player\_x.update\_symbol\_list(o\_symbol\_list) if player\_x.did\_win("X") == True: print(f"Congrats, you won!")

elif player\_x.is\_draw() == True:

print(f"It's a draw!") else:

print(f"Sorry, the client won.") host\_response = input(f"\nRematch? (Y/N): ") host\_response = host\_response.capitalize() temp\_host\_resp = host\_response

client\_response = ""

host\_response = pickle.dumps(host\_response) client\_socket.send(host\_response) if temp\_host\_resp == "N":

rematch = False else:

print(f"Waiting for client response...") client\_response = client\_socket.recv(1024) client\_response = pickle.loads(client\_response) if client\_response == "N":

print(f"\nThe client does not want a rematch.") rematch = False else:

player\_x.restart()

spacer = input(f"\nThank you for playing!\nPress enter to quit...\n") client\_socket.close()

**Client.py:** import socket # for networking import pickle # for sending/receiving objects from tic\_tac\_toe import TicTacToe

HOST = '127.0.0.1' # the server's IP address

PORT = 12783 # the port we're connecting to s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) s.connect((HOST, PORT))

print(f"\nConnected to {s.getsockname()}!") player\_o = TicTacToe("O") rematch = True while rematch == True:

print(f"\n\n T I C - T A C - T O E ") player\_o.draw\_grid() print(f"\nWaiting for other player...") x\_symbol\_list = s.recv(1024)

x\_symbol\_list = pickle.loads(x\_symbol\_list) player\_o.update\_symbol\_list(x\_symbol\_list) while player\_o.did\_win("O") == False and player\_o.did\_win("X") == False and player\_o.is\_draw() == False:

print(f"\n Your turn!") player\_o.draw\_grid()

player\_coord = input(f"Enter coordinate: ") player\_o.edit\_square(player\_coord)

player\_o.draw\_grid()

o\_symbol\_list = pickle.dumps(player\_o.symbol\_list) s.send(o\_symbol\_list) if player\_o.did\_win("O") == True or player\_o.is\_draw() == True: break

print(f"\nWaiting for other player...") x\_symbol\_list = s.recv(1024)

x\_symbol\_list = pickle.loads(x\_symbol\_list) player\_o.update\_symbol\_list(x\_symbol\_list) if player\_o.did\_win("O") == True: print(f"Congrats, you won!") elif player\_o.is\_draw() == True:

print(f"It's a draw!") else:

print(f"Sorry, the host won.") print(f"\nWaiting for host...") host\_response = s.recv(1024)

host\_response = pickle.loads(host\_response) client\_response = "N" if host\_response == "Y":

print(f"\nThe host would like a rematch!") client\_response = input("Rematch? (Y/N): ") client\_response = client\_response.capitalize() temp\_client\_resp = client\_response client\_response = pickle.dumps(client\_response) s.send(client\_response)

if temp\_client\_resp == "Y":

player\_o.restart() else: rematch = False else:

print(f"\nThe host does not want a rematch.")

rematch = False

spacer = input(f"\nThank you for playing!\nPress enter to quit...\n") s.close()

**Tic-toc.py:**

class TicTacToe(): def \_\_init\_\_(self, player\_symbol):

self.symbol\_list = [] for i in range(9):

self.symbol\_list.append(" ") self.player\_symbol = player\_symbol def restart(self): for i in range(9):

self.symbol\_list[i] = " " def draw\_grid(self):

print("\n A B C\n")

row\_one = " 1 " + self.symbol\_list[0] row\_one += " ║ " + self.symbol\_list[1] row\_one += " ║ " + self.symbol\_list[2] print(row\_one) print(" ═══╬═══╬═══") row\_two = " 2 " + self.symbol\_list[3] row\_two += " ║ " + self.symbol\_list[4] row\_two += " ║ " + self.symbol\_list[5] print(row\_two) print(" ═══╬═══╬═══") row\_three = " 3 " + self.symbol\_list[6] row\_three += " ║ " + self.symbol\_list[7] row\_three += " ║ " + self.symbol\_list[8] print(row\_three, "\n") def edit\_square(self, grid\_coord): if grid\_coord[0].isdigit():

grid\_coord = grid\_coord[1] + grid\_coord[0] col = grid\_coord[0].capitalize()

row = grid\_coord[1] grid\_index = 0 if row == "1": if col == "A": grid\_index = 0 elif col == "B": grid\_index = 1 elif col == "C": grid\_index = 2 elif row == "2": if col == "A": grid\_index = 3 elif col == "B": grid\_index = 4 elif col == "C": grid\_index = 5 elif row == "3": if col == "A": grid\_index = 6 elif col == "B": grid\_index = 7 elif col == "C": grid\_index = 8 if self.symbol\_list[grid\_index] == " ":

self.symbol\_list[grid\_index] = self.player\_symbol def update\_symbol\_list(self, new\_symbol\_list): for i in range(9):

self.symbol\_list[i] = new\_symbol\_list[i] def did\_win(self, player\_symbol): g = [] for i in range(9):

g.append(self.symbol\_list[i]) sym = player\_symbol if g[0] == sym and g[1] == sym and g[2] == sym:

return True

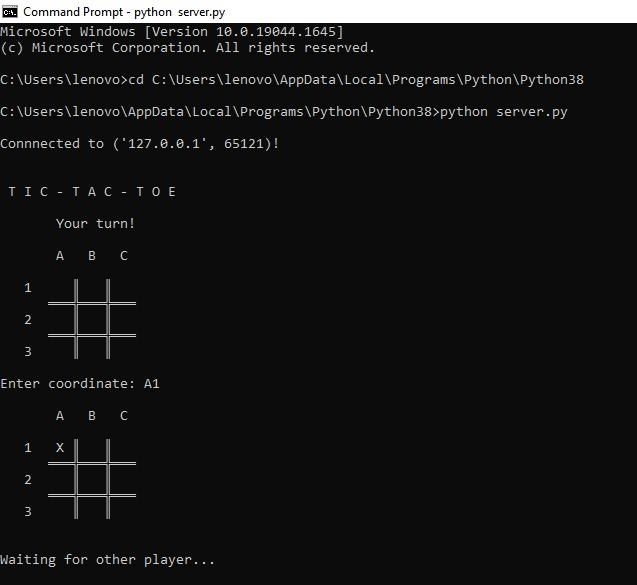
elif g[3] == sym and g[4] == sym and g[5] == sym: return True elif g[6] == sym and g[7] == sym and g[8] == sym: return True elif g[0] == sym and g[3] == sym and g[6] == sym: return True elif g[1] == sym and g[4] == sym and g[7] == sym: return True elif g[2] == sym and g[5] == sym and g[8] == sym: return True elif g[2] == sym and g[4] == sym and g[6] == sym: return True elif g[0] == sym and g[4] == sym and g[8] == sym:

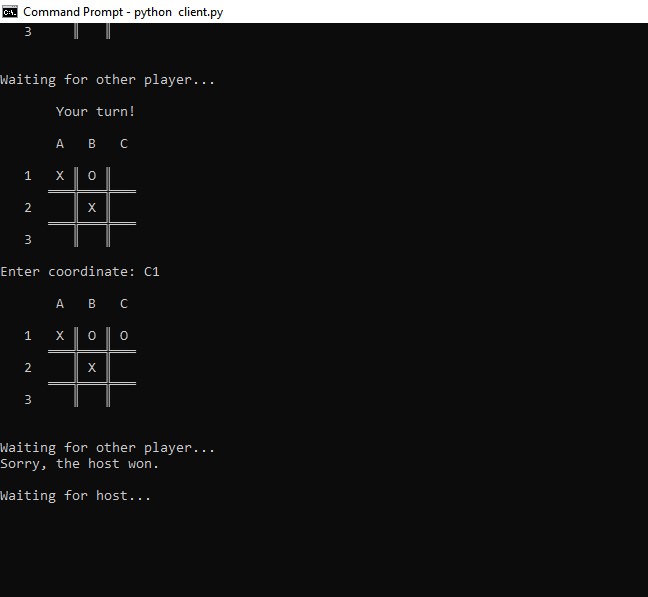
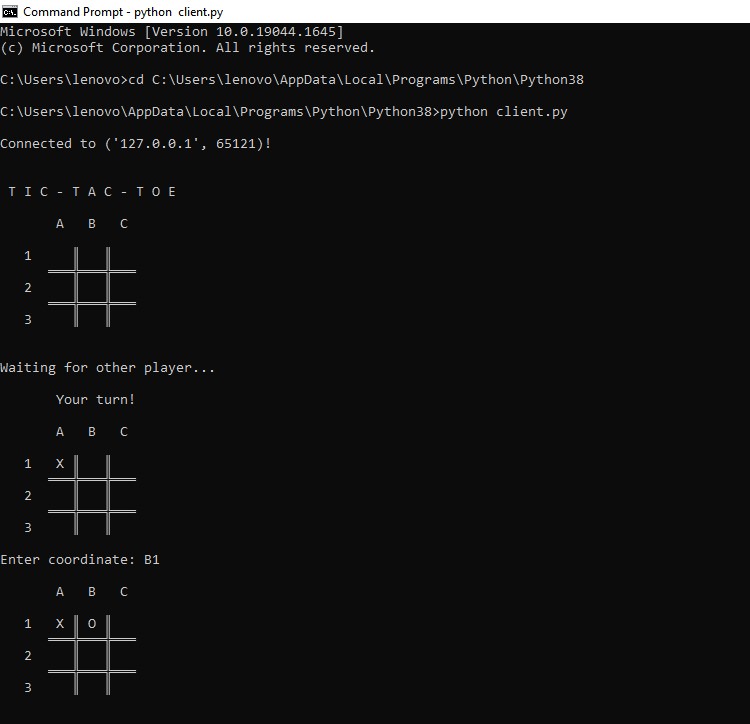
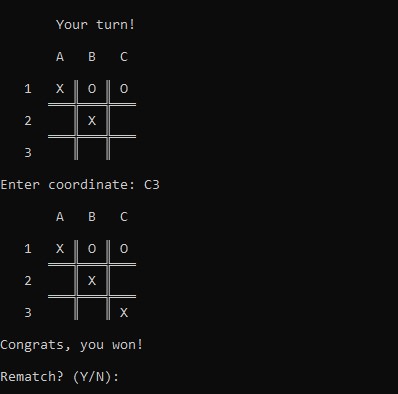
return True return False def is\_draw(self): num\_blanks = 0 for i in range(9): if self.symbol\_list[i] == " ":

num\_blanks += 1 if self.did\_win(self.player\_symbol) == False and num\_blanks == 0: return True else:

return False

**0utput:**

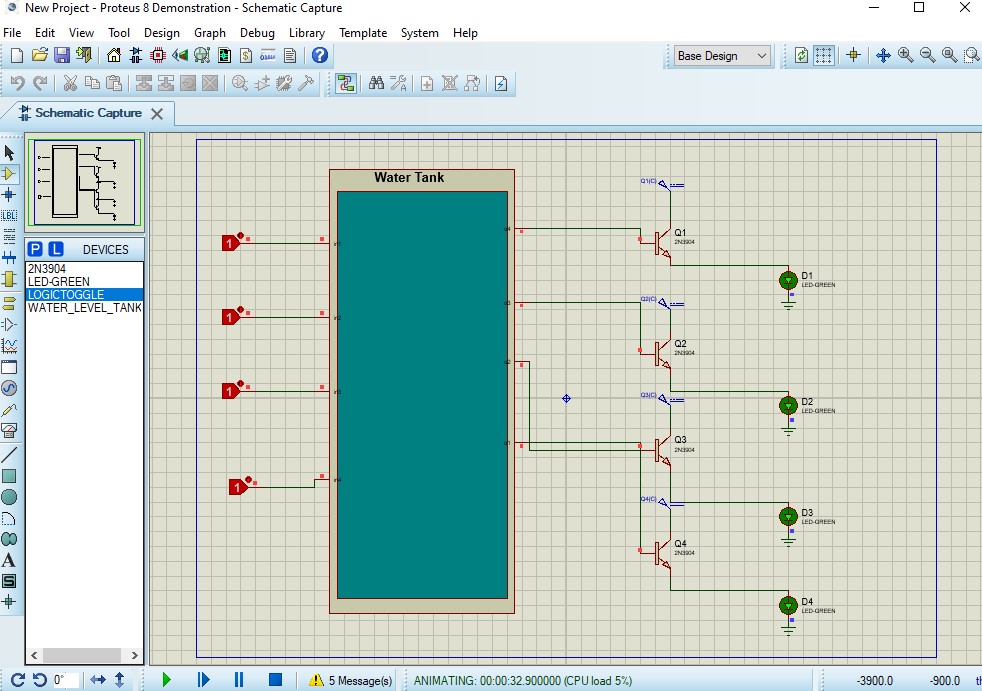




# PRACTICAL NO: 05

**Aim:** Demonstrate an IoT application for monitoring water levels in tanks and automatically start the motor to fill the tank if the level goes below the critical level.

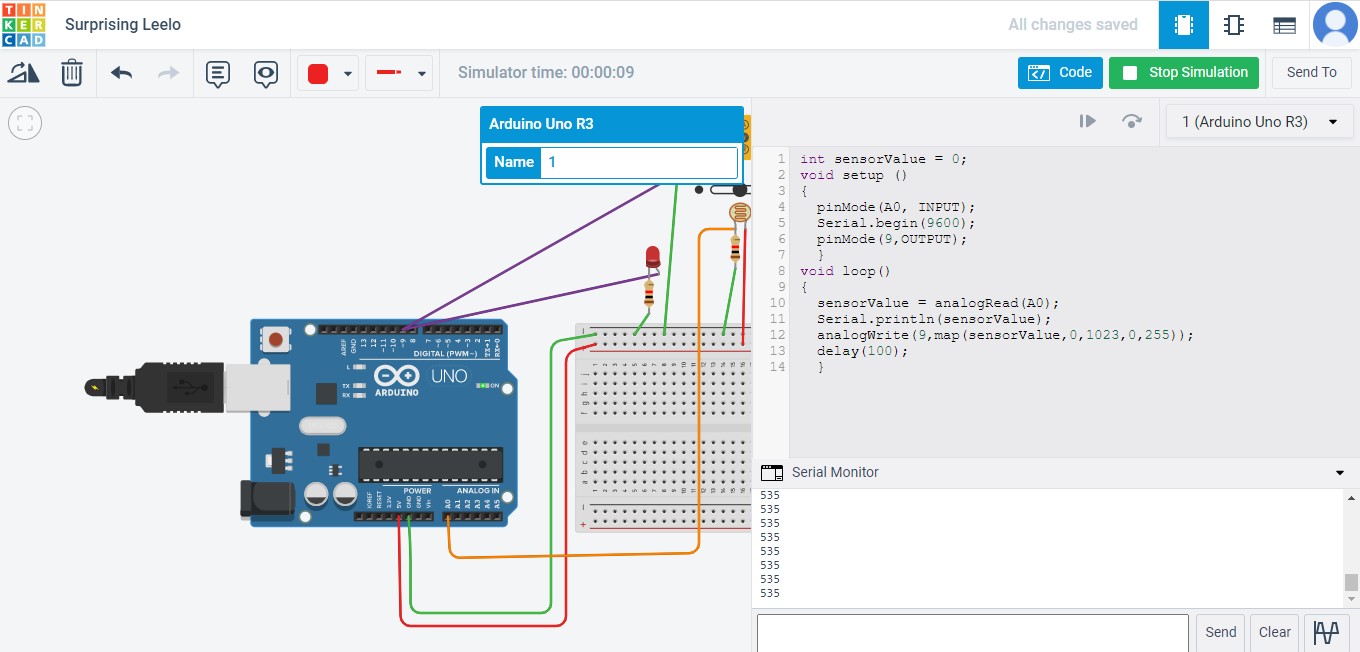
**Output:**

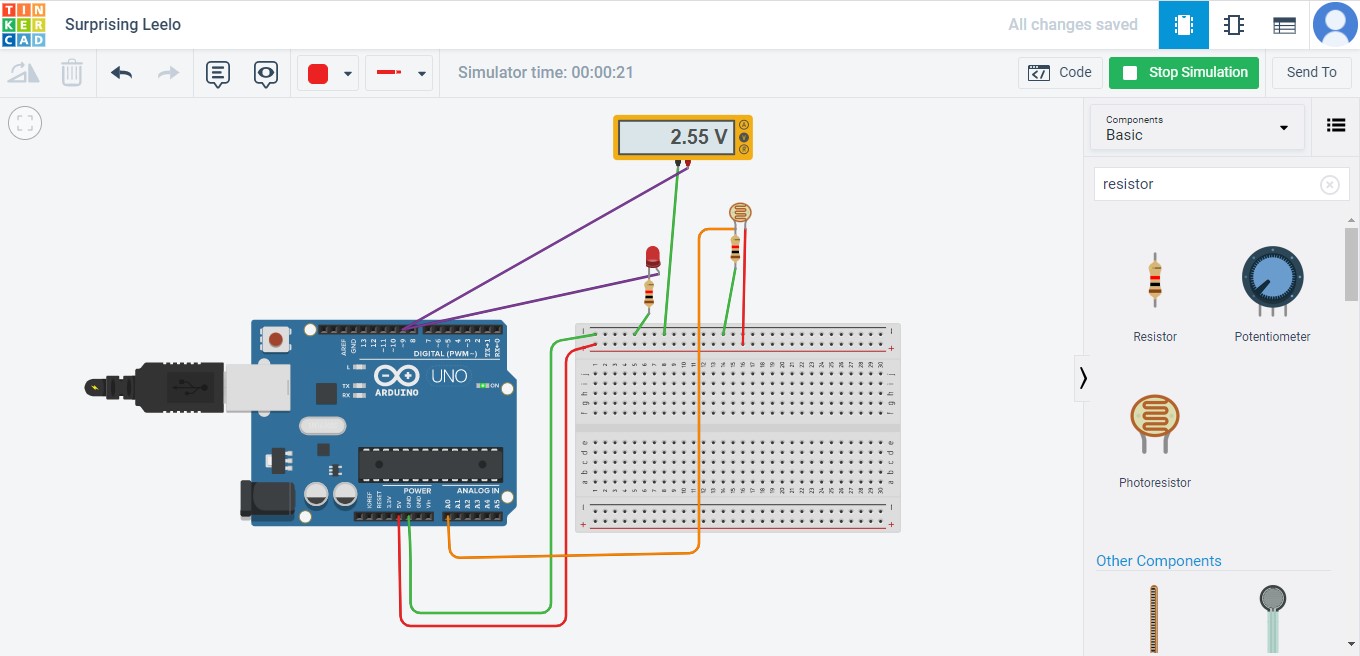


# PRACTICAL NO: 06

**Aim:** Develop an IoT module to which measure the intensity of light and send the same to your PC/Phone.

**Output:**





# PRACTICAL NO: 07

**Aim:** Develop an IoT application for Motion detection.

**Output:**

