OS 310 PROJECT

Snake and Ladder Game using IPC

Akshad Vivek Gajbhiye 2203101
Ishita Partha Chail 2203107

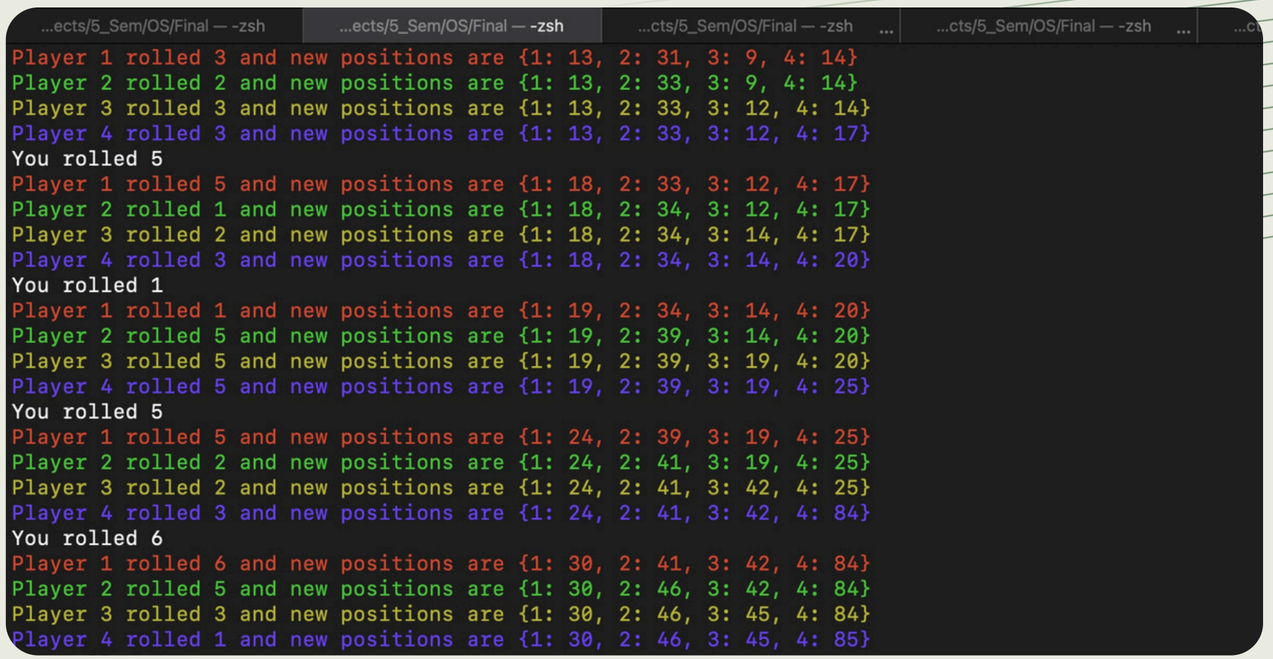
23 November, 2024

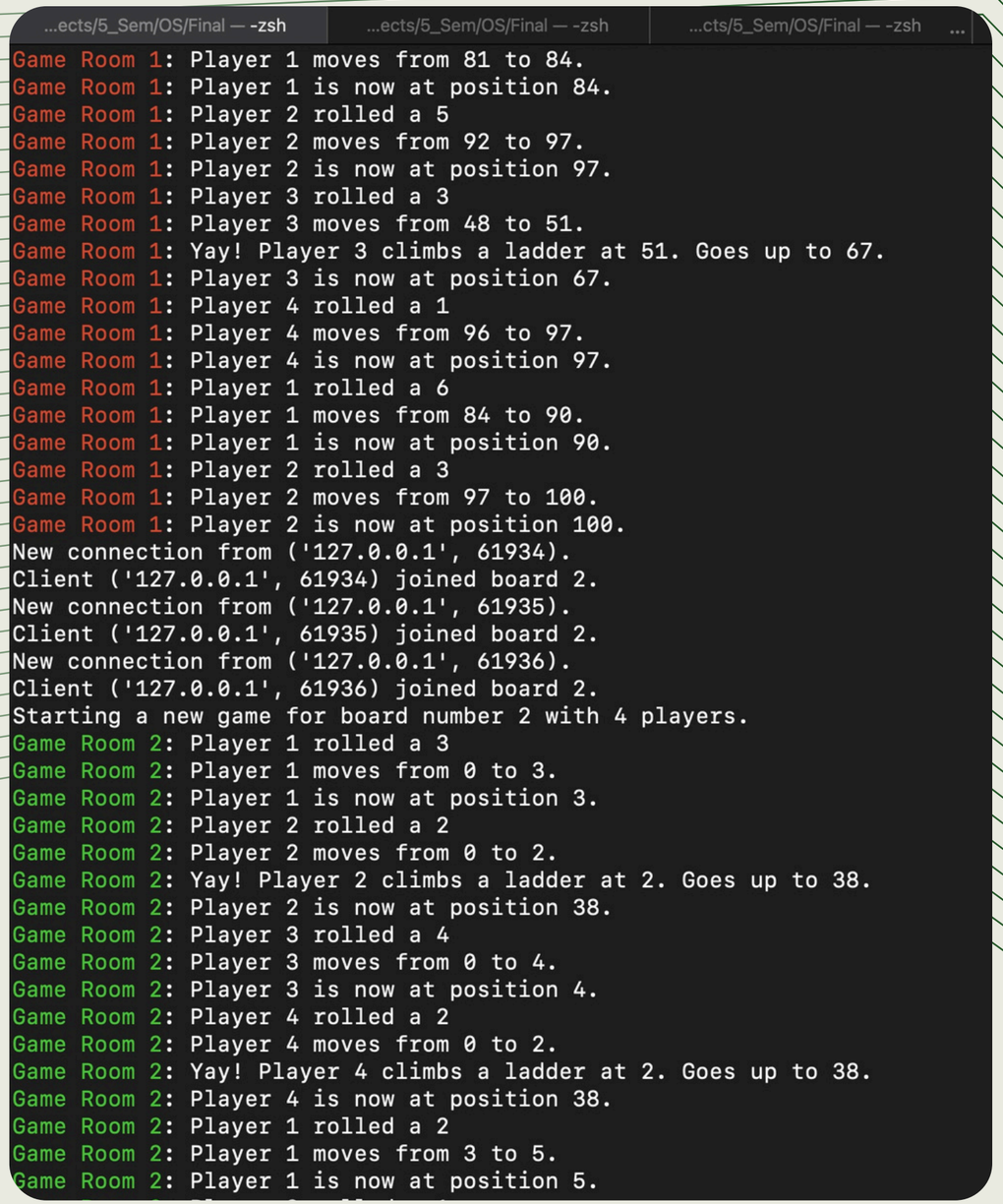
PROJECT

Snake and Ladder Game using IPC

Main Objectives:

- Demonstrate inter-process communication using sockets.
- Showcase concurrency using threads.

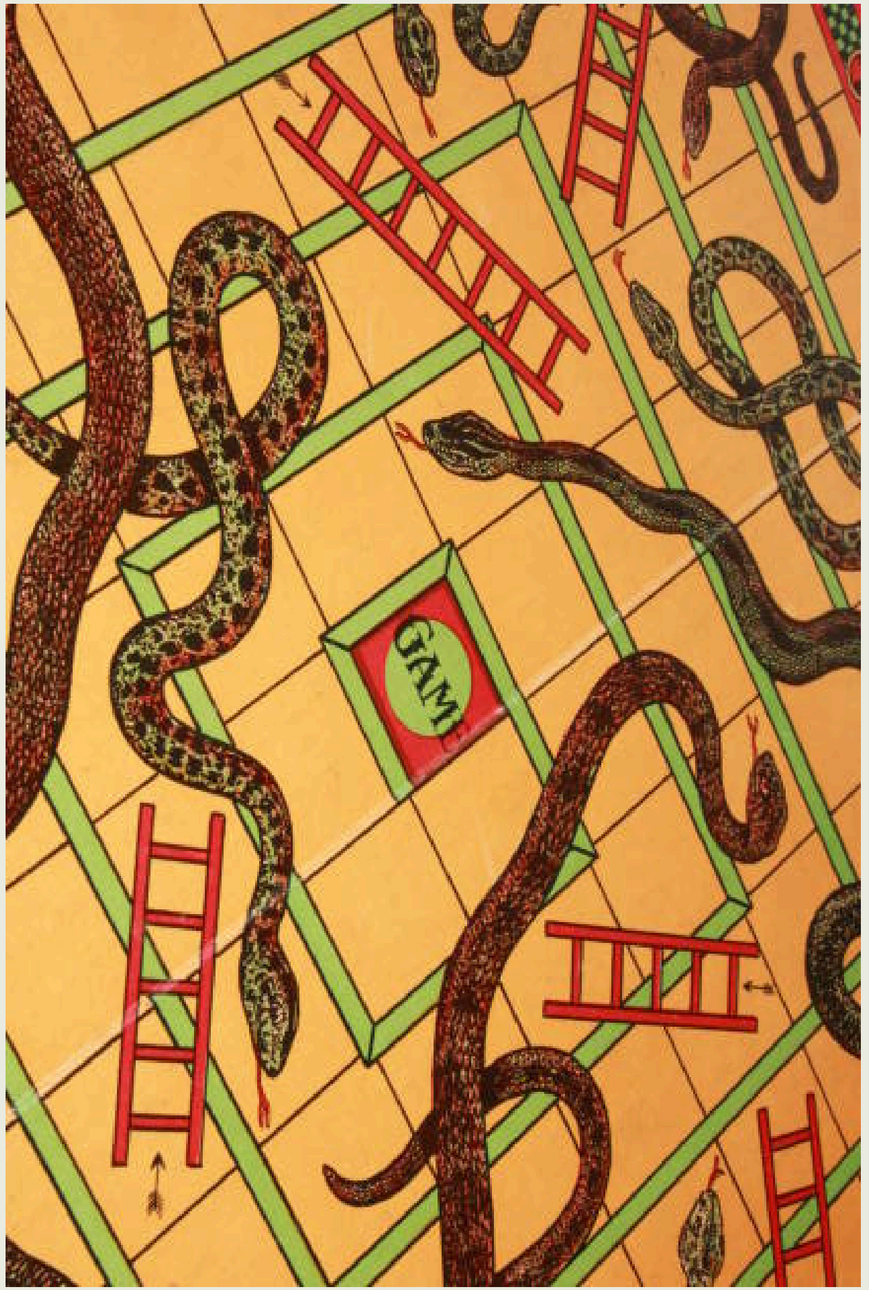




4 PLAYER SNAKES AND LADDERS GAME

Features of the Game:

- Played across different devices
- A **Server** is continuously running and can handle multiple game rooms simultaneously
- The **Players** connect to the server using the **IP address** and can join a specific game room using a **Game Code**.
- **IPC** (Inter Process Communication) is achieved using **socket programming** in Python.
- **Real-time updates** on player positions.
- Cross Device Compatibility



ARCHITECTURE

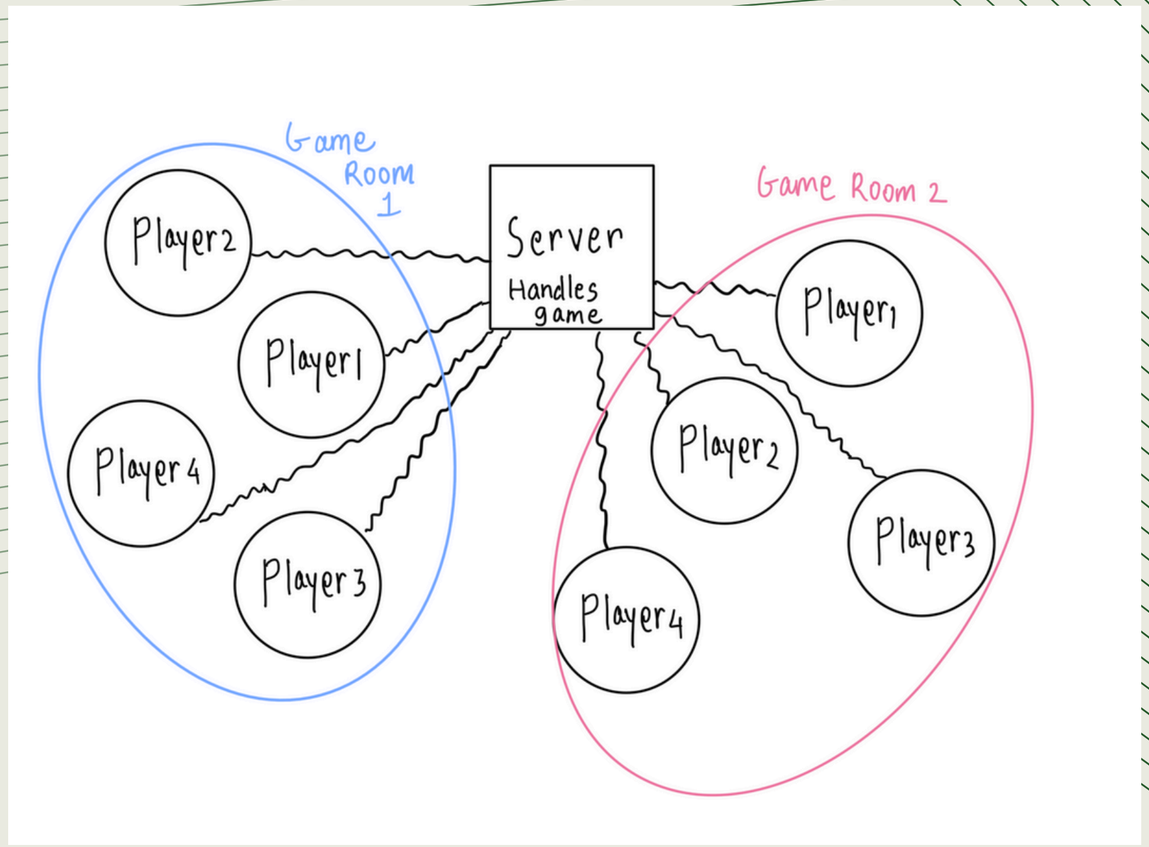
Snake and Ladder Game using IPC

Server:

- Accepts client connections.
- Groups clients into game rooms.
- Manages the game logic and broadcasts updates.

Clients:

- Connect to the server.
- Interact via dice rolls and receive updates.



HOW TO RUN THE GAME

To run your multiplayer Snake and Ladder game on different devices, follow these steps. This assumes you have the server and player code saved in separate files, Python installed, and a network connection:

Step 1 Set Up the Server:

- Open a terminal on the device that will act as the server.
- Navigate to the directory where server.py is located.
- Run the server using the command: python server.py
- You should see output like:

  Server started on 127.0.0.1:65431 and waiting for connections...

HOW TO RUN THE GAME

Step 2 Start the Players:

- On each player's device, open a terminal.
- Ensure the device can connect to the server. The server's IP address must be reachable (e.g., via LAN or a local network).
- For the local machine, use 127.0.0.1 as the IP.
- For different devices on the same network, use the server's local IP (find it using ipconfig on Windows or ifconfig on macOS/Linux).
- Replace 127.0.0.1 in the player code with the server's IP address.
- Navigate to the directory where player.py is located.
- Run the player code using the command: python player.py
- The player will see a prompt:

  Enter your board number:
- Enter a number (e.g., 1) to join a specific game room.

HOW TO RUN THE GAME

Step 3 Add More Players:

- Repeat Step 2 for additional players. Ensure all players in the same game room enter the same board number.

Step 4 Game Play:

- Once 4 players join the same board number, the server will notify all players: Game starting for board [board_number]. You are Player [X].
- Each player takes turns rolling the dice. When prompted: ROLL
- The dice will automatically roll, and the result will be displayed, e.g.: Player 1 rolled 5 and new positions are {1: 5, 2: 0, 3: 0, 4: 0}.
- The game continues until one player reaches position 100, and the server announces the winner: Player X won the game

Step 5 End Game:

- After the game ends: all players see the message END and their connection to the server is closed.
- The server output confirms the game has ended and the connections have been cleaned up.

TOOLS AND TECHNOLOGY

Programming Language: Python

Core Concepts:

- **Sockets:**

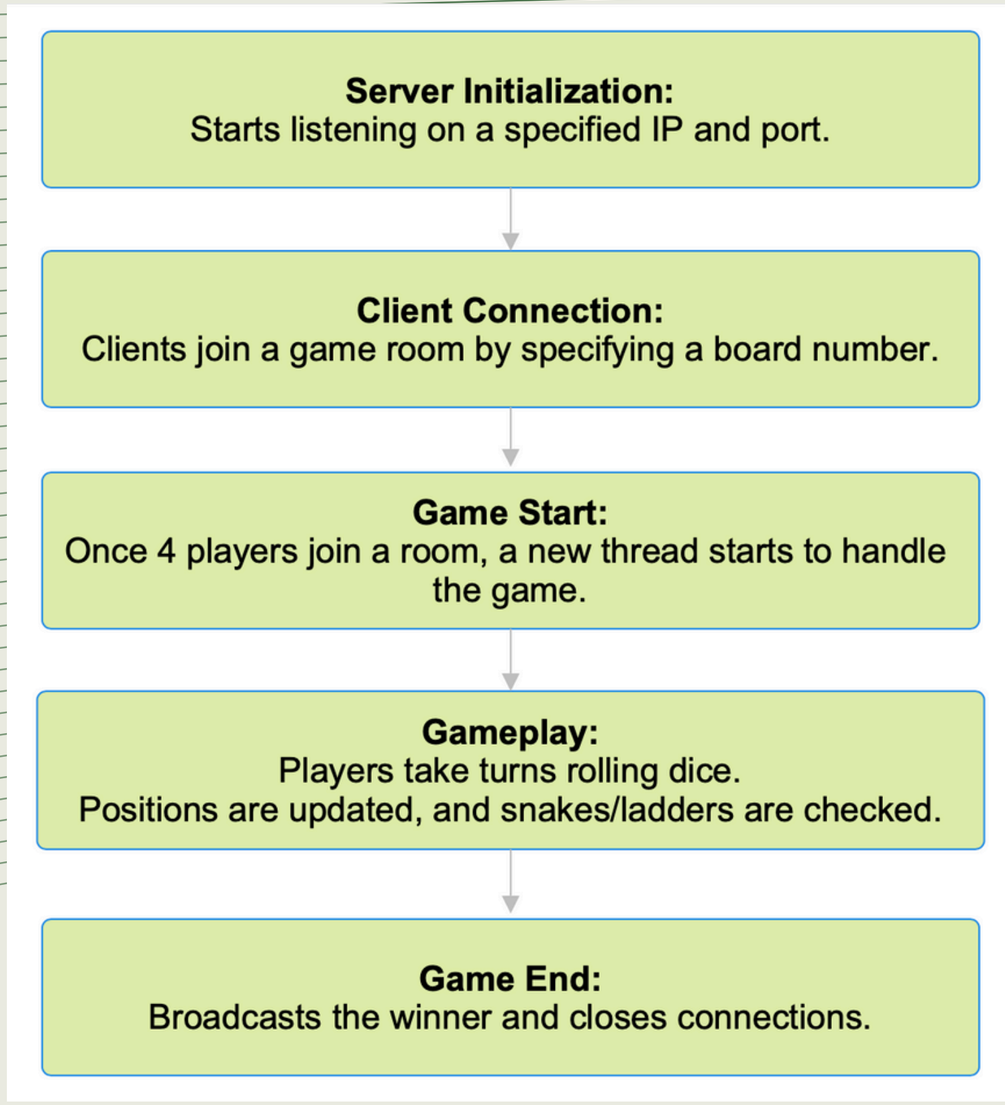
TCP sockets for reliable communication between the server and clients. Bidirectional communication for real-time updates.

- **Threads:**

Each game room is handled by a separate thread. Enables concurrent gameplay for multiple rooms.

Python Libraries:

- <socket> for networking.
- <threading> for concurrency.
- <random> for dice rolls.



WORKFLOW

CODE WALKTHROUGH

```python
class BoardGame:
    def __init__(self):
        # Initialize board, players, snakes, and ladders
        self.boardNum = 0
        self.boardCol = 0
        self.board_size = 100  # Board from 1 to 100
        self.players = {1: 0, 2: 0, 3: 0, 4: 0}  # Player positions start at 0
        self.snakes = {16: 6, 47: 26, 49: 11, 56: 53, 62: 19, 64: 60, 87: 24, 93: 73, 95: 75, 98: 78}
        self.ladders = {2: 38, 7: 14, 8: 31, 15: 26, 28: 84, 21: 42, 36: 44, 51: 67, 71: 91, 78: 98, 87: 94}

    def move_player(self, player, steps): ...
        print(f"Player {player} is now at position {new_position}.")

    def is_winner(self, player):
        """Check if the player has won."""
        return self.players[player] == self.board_size
```

Files:

- Server.py
- Player.py

Server.py

- **Class Boardgame :**
  - Manages the board, players, snakes, and ladders.
  - Key methods:
  a. <move_player> handles player movement and interactions with snakes/ladders.
  b. <is_winner> checks if a player has reached position 100
- **def handle_game(game_code):** Starts a new game for 4 players in a room. Handles dice rolls, player moves, and broadcasts updates. Ends the game when a player wins.

CODE WALKTHROUGH

Server.py

- **def handle_client(client_socket, client_address):** Accepts client connections. Assigns clients to game rooms based on board numbers. Starts a game thread when 4 players join.
- **def start_server():** Sets up the server socket. Continuously accepts and spawns threads for new client connections.

```
# Dictionary to track clients grouped by game codes
players = {}

# Lock for thread-safe updates to the `players` dictionary
lock = threading.Lock()

# Server configuration
HOST = '127.0.0.1'  # Localhost
PORT = 65431        # Port to listen on


def handle_game(game_code): ...
    print(f"Game for board {game_code} ended and connections closed.")


def handle_client(client_socket, client_address): ...
        client_socket.close()


def start_server(): ...
        server_socket.close()
```

CODE WALKTHROUGH

Player.py

- **Server Configuration:**
  SERVER_IP = '127.0.0.1'
  PORT = 65431
- **Dice Roll Simulation:**
  def rollDice():
      return random.randint(1,6)
- **Establishing a Connection:**
  client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  client_socket.connect((SERVER_IP, PORT))
- **Gets the Room Code**:
  if message=="Enter your board number: ":
      boardNum = int(input(message))
      client_socket.sendall(f"{boardNum}".encode())
- **MessageHandling :** Receives a "ROLL" prompt from the server. Simulates a dice roll using rollDice(). Sends the dice result back to the server. If the server sends "END", the game terminates, and the client exits the loop.

DEMO

https://github.com/akshad-gajbhiye/Snakes-Ladders-IPC

FUTURE IMPROVEMENTS

- **GUI:** Add a graphical interface for better user experience.
- **No. of Players:** Extend to variable number of players per room.
- **Disconnected Players:** Implement persistent game state for disconnected players.

THANK *YOU.*

ANY QUESTIONS?