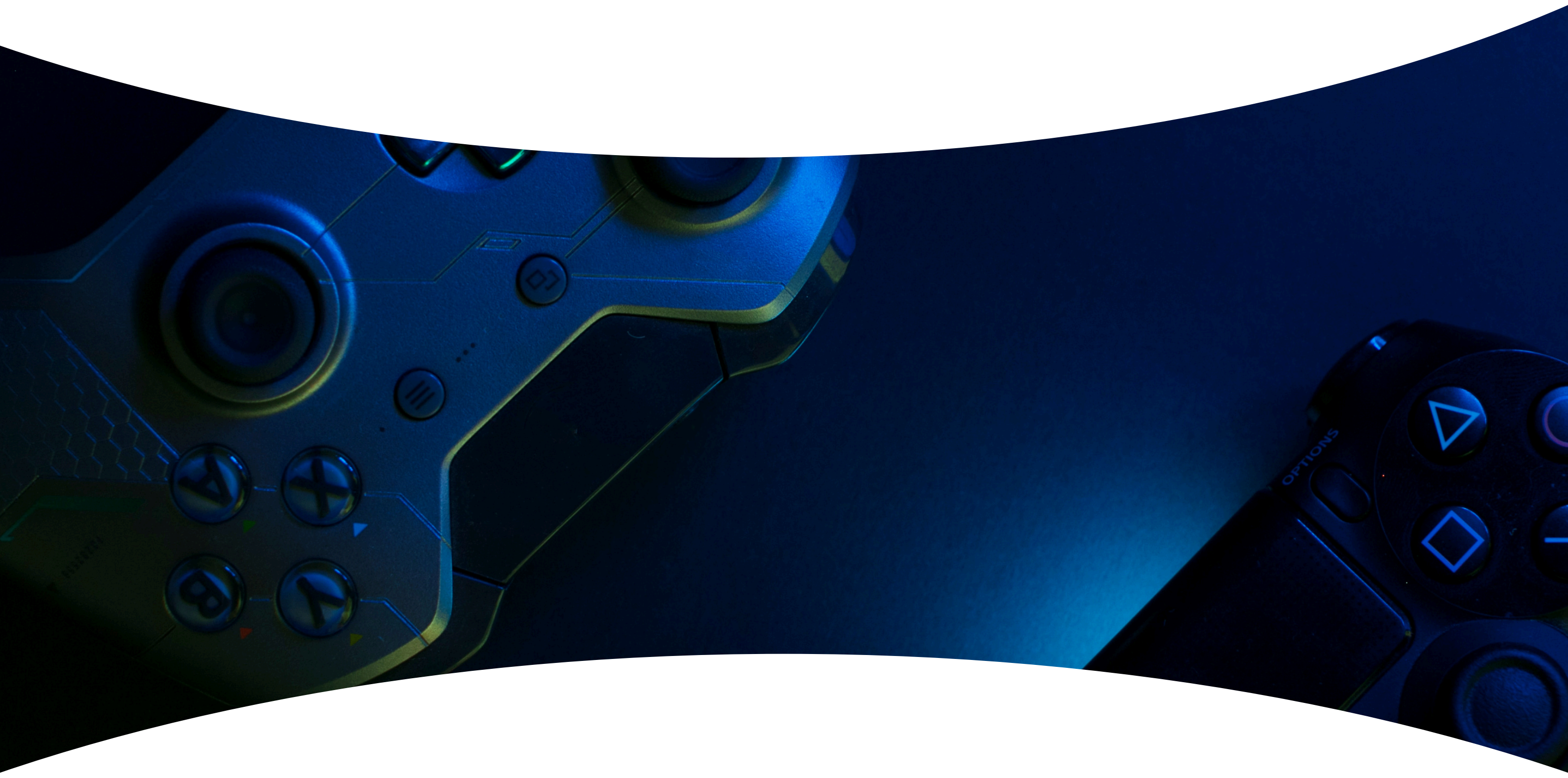
DATA STRUCTURE



MULTIPLAYER TURN-BASED GAME ENGINE

TEAM 6

- Snake and Ladder
- Ludo
- Chess Game

Project Description

Project Title:
Multiplayer Turn-Based Game Engine for **Snake & Ladder, Chess, and Ludo.**

Overview:
This project is designed to create a versatile multiplayer turn-based game engine that can host and manage multiple classic board games: Snake & Ladder, Chess, and Ludo. The engine's primary function is to handle game logic, enforce rules, manage player turns, and maintain the state of the game board. It aims to provide a seamless experience for multiple players to compete in any of these games through a unified platform.

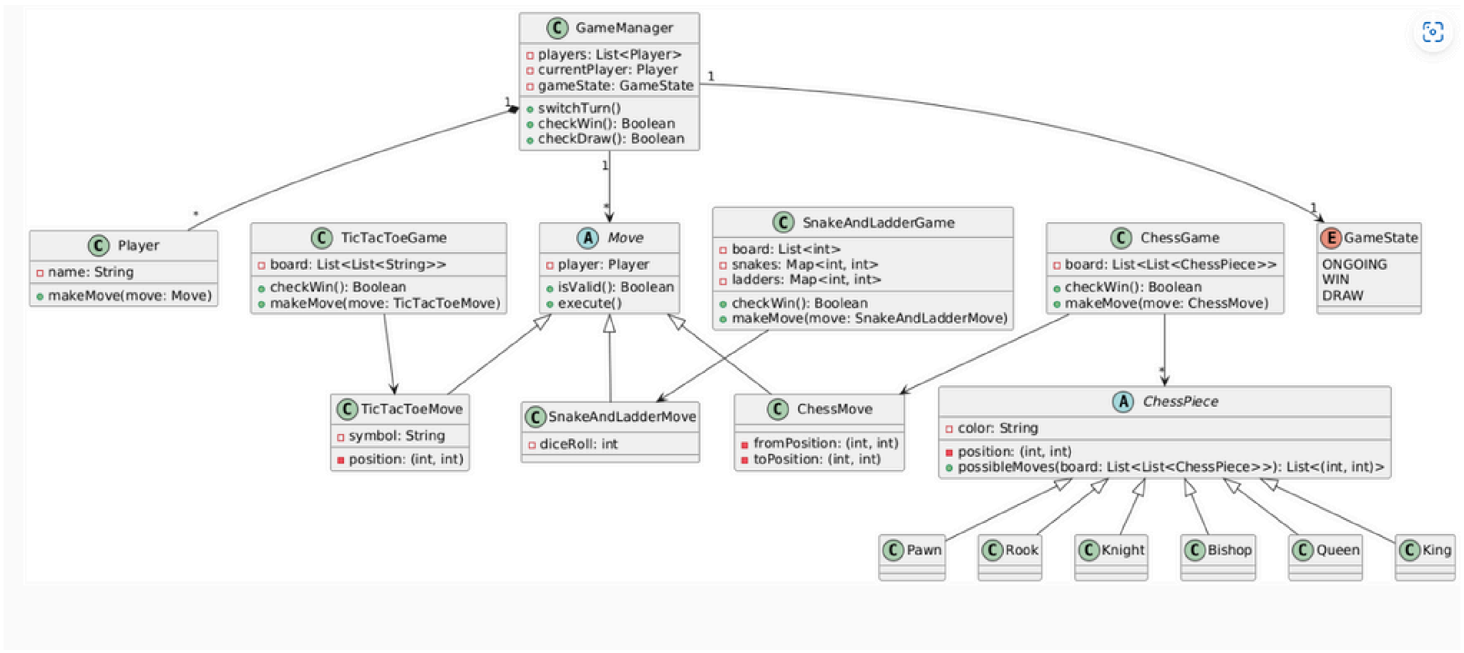
Goal of the project:

- To demonstrate mastery of fundamental and advanced data structures and algorithms by implementing three different board games with unique rules and game mechanics.
- To design a reusable, modular game engine framework that can be extended to include more games.
- To implement efficient game state management, move validation, and turn management using appropriate data structures.
- To provide an interactive user experience where players can input moves and see real-time updates of the game state.

What the Project Does:
The engine allows a group of players to connect and play turn-based board games where each player waits for their turn before making a move. It supports:

- **Game Initialization:** Setting up the board and players according to the chosen game.
- **Turn Management:** Ensuring players alternate turns correctly.
- **Move Validation:** Checking if player moves follow the game rules.
- **Game State Updates:** Applying moves to the board and updating piece positions.
- **Win/Draw Detection:** Automatically identifying when a game ends and declaring the winner or a draw.
- **User Interaction:** Accepting player input for moves and displaying updated game states.
- **Multiple Games Support:** Switching between different games without restarting the application.

WorkFLow



UML Just to Show the WorkFlow we can Edit and Improve the Classes and Function

1. Game Initialization

- User selects a game: Snake and Ladder, Chess, or Ludo.
- Load the board as a 2D array (10x10 for Snake and Ladder, 8x8 for Chess, 3x3 for Ludo).
- Register players in a queue for turn order.
- Initialize game-specific data:
  - Snake and Ladder: store snakes and ladders positions (e.g., in maps).
  - Chess: place pieces on board and track positions (e.g., hashmap).
  - Ludo: setup tokens and player-specific positions.

2. Turn Management

- Loop through players in queue order.
- On each turn:
  - Prompt the player to make a move.
  - Validate the move according to the game rules.
  - Update the board and game state.
  - Apply special actions (snakes/ladders jumps, check/checkmate, token captures).
  - Move the player to the back of the queue.

3. Game State Update

- The system checks if game-ending conditions are met:
  - Ludo: All four tokens of a player reach the goal zone.
  - Chess: Checkmate, stalemate, or player resignation.
  - Snake and Ladder: A player reaches square 100.
- If the game ends, the system announces the winner or a draw.
- If the game has not ended, the next player is prompted to move.

4. Undo/Redo (Optional)

- Players may choose to undo a move.
- The system uses a stack to save previous board states before each move.
- Undo pops the last state and restores the board and player positions.
- Redo re-applies the undone move if needed.
- (Note: Undo may be disabled or limited in games like Snake and Ladder depending on rules.)

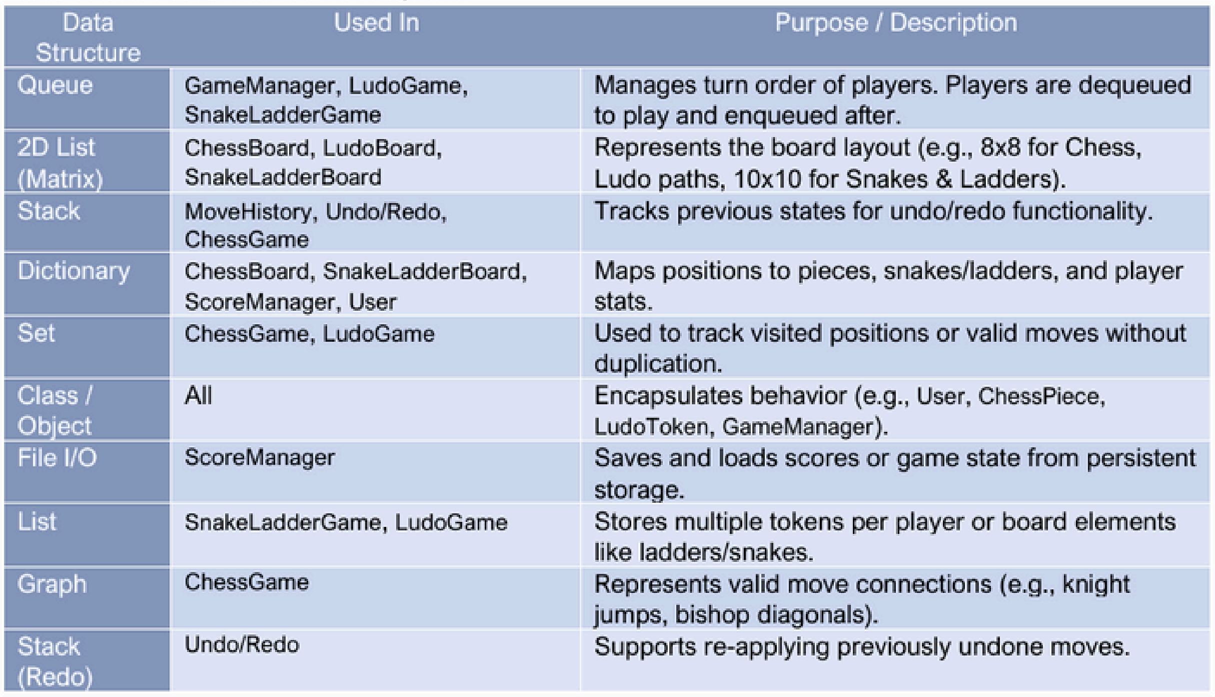
5. Game End

- After the game finishes, the system:
  - Displays the final results.
  - Optionally allows players to restart the game or exit.
  - Clears or resets data structures for a new game session.

MAIN COMPONENET

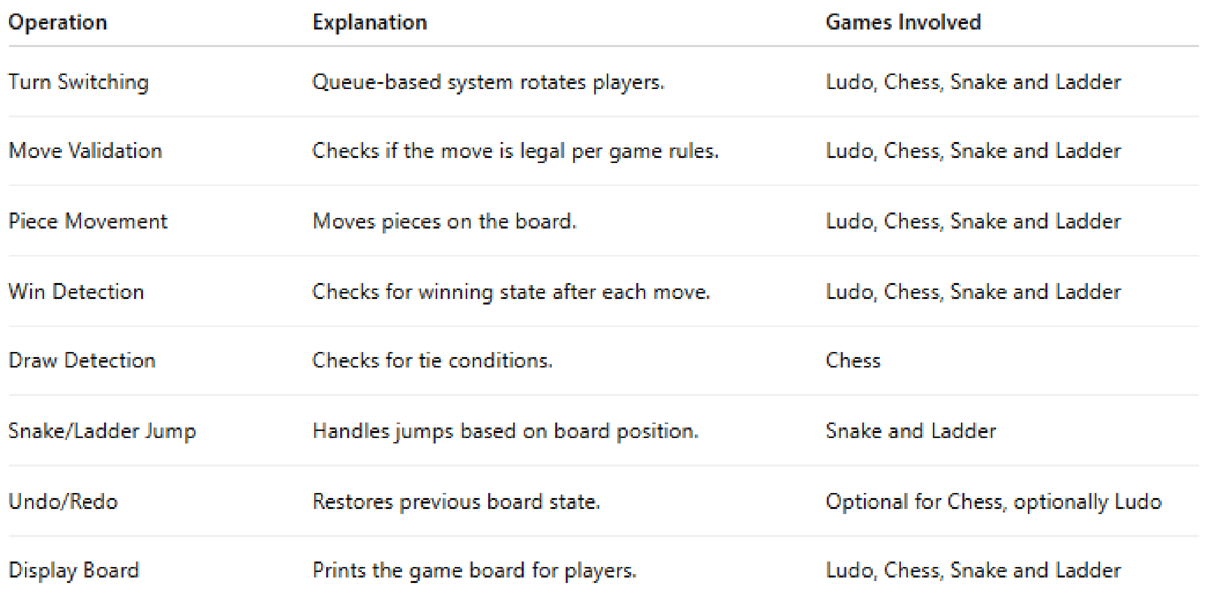
Component Name	Description	Related Game(s)	Data Structures Used
User	Represents a player with username and score tracking.	All	Dictionary / Class
Game Manager	Handles switching between games and managing player flow.	All	Queue, Class
Ludo Board	Represents the Ludo board with home bases, safe zones, and final path.	Ludo	2D List (Matrix), Dictionary
Ludo Game	Game logic, dice rolls, token movement, capturing, and turn logic.	Ludo	Queue (Turns), List, Dictionary
SnakeLadderBoard	Represents the board with snakes and ladders.	Snake and Ladder	Dictionary (snakes/ladders), List
SnakeLadderGame	Manages dice rolls, player positions, and turn-based movement.	Snake and Ladder	Queue (Turns), List
Chessboard	8x8 board with all chess pieces and their positions.	Chess	2D List (Matrix), Dictionary
Chess Piece	Abstract class for chess pieces (Pawn, Rook, etc.).	Chess	Inheritance, OOP
Chess Game	Turn-taking logic, move validation, check/checkmate.	Chess	Graph (Possible Moves), Set
Move History	Tracks past moves for undo/redo functionality.	All (especially Chess, Ludo)	Stack
GUIController	Manages UI, renders boards, collects user input.	All	Not DS-specific, integrates logic
ScoreManager	Tracks wins, losses, draws, and player stats.	All	Dictionary, File I/O
MainApp	Launch point, initializes the system and games.	All	OOP Structure

DATA STRUCTURE USED



Data Structure	Used In	Purpose / Description
Queue	GameManager, LudoGame, SnakeLadderGame	Manages turn order of players. Players are dequeued to play and enqueued after.
2D List (Matrix)	ChessBoard, LudoBoard, SnakeLadderBoard	Represents the board layout (e.g., 8x8 for Chess, Ludo paths, 10x10 for Snakes & Ladders).
Stack	MoveHistory, Undo/Redo, ChessGame	Tracks previous states for undo/redo functionality.
Dictionary	ChessBoard, SnakeLadderBoard, ScoreManager, User	Maps positions to pieces, snakes/ladders, and player stats.
Set	ChessGame, LudoGame	Used to track visited positions or valid moves without duplication.
Class / Object	All	Encapsulates behavior (e.g., User, ChessPiece, LudoToken, GameManager).
File I/O	ScoreManager	Saves and loads scores or game state from persistent storage.
List	SnakeLadderGame, LudoGame	Stores multiple tokens per player or board elements like ladders/snakes.
Graph	ChessGame	Represents valid move connections (e.g., knight jumps, bishop diagonals).
Stack (Redo)	Undo/Redo	Supports re-applying previously undone moves.

Main Operations Implemented



Operation	Explanation	Games Involved
Turn Switching	Queue-based system rotates players.	Ludo, Chess, Snake and Ladder
Move Validation	Checks if the move is legal per game rules.	Ludo, Chess, Snake and Ladder
Piece Movement	Moves pieces on the board.	Ludo, Chess, Snake and Ladder
Win Detection	Checks for winning state after each move.	Ludo, Chess, Snake and Ladder
Draw Detection	Checks for tie conditions.	Chess
Snake/Ladder Jump	Handles jumps based on board position.	Snake and Ladder
Undo/Redo	Restores previous board state.	Optional for Chess, optionally Ludo
Display Board	Prints the game board for players.	Ludo, Chess, Snake and Ladder

User Interaction: Using Tkinter & PyGame Libraries

- **Interface:** Graphical User Interface (GUI) developed in Python.
- **Game Selection:** Users select the game (Snake and Ladder, Chess, or Ludo) from a graphical menu.
- **Player Registration:** Users input their names through GUI forms to register players in order.
- **Turns:**
  – Players make moves by interacting with the board visually (e.g., clicking tokens, selecting cells).
  – Dice rolls or move commands are triggered via GUI controls (buttons, dialogs).
- **Board Display:**
  – The board is displayed graphically with pieces, tokens, and highlights for valid moves and special actions.
- **Feedback:**
  –The system shows messages and visual cues for invalid moves, captures, snakes/ladders jumps, checks, and game status updates.

Cases and Scenario:

The user launches the Python game application. A graphical menu appears, allowing the user to select one of the three games: Snake and Ladder, Chess, or Ludo. Once a game is selected, the system initializes the appropriate board, registers players, and starts the game loop where players interact through the GUI using Tkinter.

Use Case 1: Game Selection

- **A**ctor: User
- Precondition: Application is running

Steps:

1. **U**ser selects a game from the main menu.

2.System loads the selected game's board and logic.

Postcondition: Player registration screen is displayed.

Use Case 2: Player Registration

- **A**ctor: User
- Precondition: Game has been selected

Steps:

1. **G**UI prompts users to enter names.
2. Players input names via text fields.
3. System stores names and determines turn order.

Postcondition: Game is initialized with registered players.

Use Case 3: Player Turn

- **A**ctor: Registered player

Precondition: Game is active and it's the player's turn

Steps:

1. **P**layer interacts with the GUI to make a move (e.g., roll dice, select piece).
2. System checks the validity of the move.
3. If valid, the move is applied and board is updated.
4. If invalid, an error message is shown.

Postcondition: Turn ends and passes to the next player.

Use Case 4: Special Game Actions

Examples:

- Snake or ladder interaction (Snake and Ladder)
- Check or checkmate (Chess)
- Token capture or reaching home (Ludo)

System Response:

Updates the board and displays relevant messages or effects.

Use Case 5: Game End

- **A**ctor: System

Trigger: A player meets the win condition

Steps:

- **S**ystem detects winning condition (e.g., square 100, checkmate, all tokens home).
- Displays the winner and offers restart or exit options.

Postcondition: Game ends or restarts based on user choice.