Design Snake and Ladder Game

- Problem Statement
- Overview
- Requirement Classification
- Components
  1. Player
  2. Board
  3. Dice
  4. Jump - Snake and Ladder
  5. Cell
  6. Game
- Class Diagram
- Implementation

Resources

- 11. LLD of Snake and Ladder game (Hindi) | SDE system design interview question, Java implementation
- Code Repo → src/main/java/com/conceptcoding/interviewquestions/snakeNladder · main · shrayansh jain / LLD-LowLevelDesign · GitLab

Problem Statement

Design a detailed low-level-design of Snake and Ladder Game including all object-oriented-design patterns and principles discussed earlier.

Overview

A Snake and Ladder Game is a 2-player game(can be extended to have more players by incorporating different winning startergies) where each player takes turns in moving their placement on the grid typically of size 10x10(i.e. 100 cells numbered from [0-99]) by rolling the dice starting from position one(i.e. cell 0).

This game consists of 3 main components:

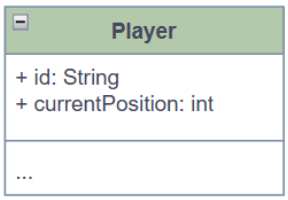
- Dice → Yields a random number between [1-6] which tells the player to move “x” number of cells in forward direction.
- Ladders → Helps the player jump to a higher position.
- Snakes → Steps down the player to a lower position.

Requirement Classification

- Snake and Ladder Game Board grid Size? → Fixed 10x10 i.e. 100 cells
- How many dice? → 1, but it should be scalable. A dice roll function should yield a random number between [1-6].
- Number of Snakes & Ladders → We should be able to dynamically define it.
- Number of Players? → 2 players but configurable.
- Winning Strategy → 2 Player Game, if anyone finishes first(reaches the last position i.e. cell 100 on the grid), the person is the winner and the game is over.
- Game → Acts as a central controller of all the above components.

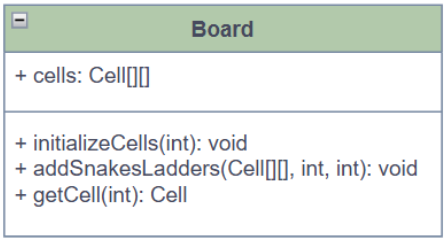
Components

1. `Player`



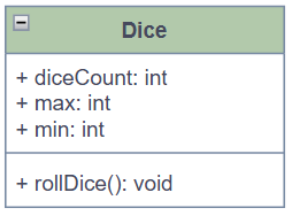
- The `Player` component encapsulates information about the person who choses to play the game.
- Holds variables to save the Name and track the movement during the game.

2. `Board`



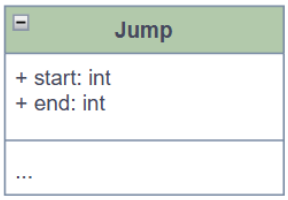
- The `Board` class is composed of cells, a 2D matrix.
- Has methods to initialise, fetch and hold game boards logic.

3. `Dice`



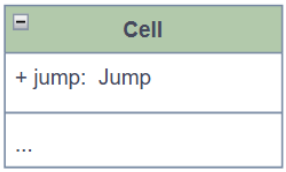
- A `Dice` class holds the max and min value of an inclusive range of numbers that a dice roll can yeild.
- This class is used to simulate the dice roll action using random number generation logic.

4. `Jump` - **Snake and Ladder**



- The `Jump` class represents the board cell behaviour and holds value of its cell positions.
- A few specific board cells is composed of `Jump` object that simulate Snake and Ladder behaviour when the player acquires that cell upon a dice roll.
- Snake behaviour is implemented using a higher cell position value as start and lower cell position value as end indicating teh player to move backward.
- Ladder value is implemented using using a lower cell position value as start and higher cell position value as end indicating the player to move forward.

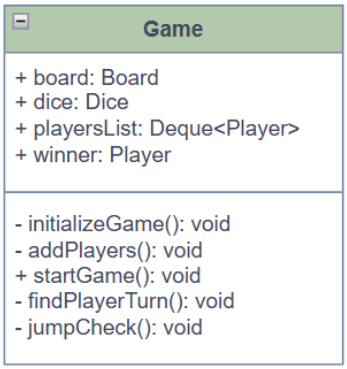
5. `Cell`



- A `Cell` component represents what a game `Board` class is made up of.
- Each `Cell` is composed of a `Jump` object.

- If the `Jump` object is not null, it implies that, this specific cell has a Snake's start or Ladder's start position from which the player either moves backward(if a snake) or moves forward(if a ladder).
- If `Jump` object is null, it means the board cell is not associated with snake or ladder, then the player occupies the new cell position.

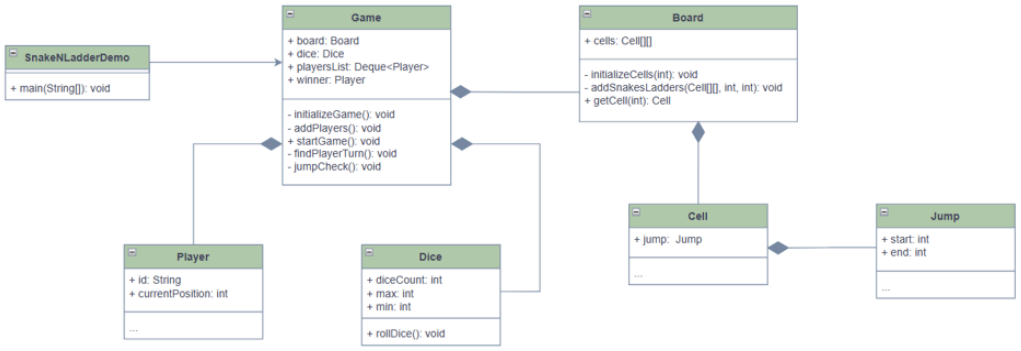
6. `Game`



- The `Game` class acts a central controller that is composed of all necessary components like `Board`, `Dice`, `Players`.
- It is responsible for orchestrating the entire Snake and Ladder Game by defining behaviours for various actions that play by the rules established.
- It initialises the `Game` with `Board`, `Dice` and `Players` - the mandatory entities needed to start a game.
- Starts the game, choses player, alternates the turns, rolls dice and moves the current player forward and checks if its new position is associated with a `Jump` behaviour and it is moved further ahead/backward mimicking the Snake/Ladder's behaviour.
- Declares the `Player` who reaches the last board cell position as winner and ends the `Game`.

Class Diagram

We combine all the components discussed above and build a final Class Diagram with all the neccessary class relationships that works as a blueprint to implement an executable solution.



Implementation

Refer to Code Repo for executable code → src/main/java/com/conceptcoding/interviewquestions/snakeNladder · main · shrayansh jain / LLD-LowLevelDesign · GitLab