LLD of Snake and Ladder game | Amazon SDE2 system design interview question, Java implementation

Requirement Classification:

- How many dice ?

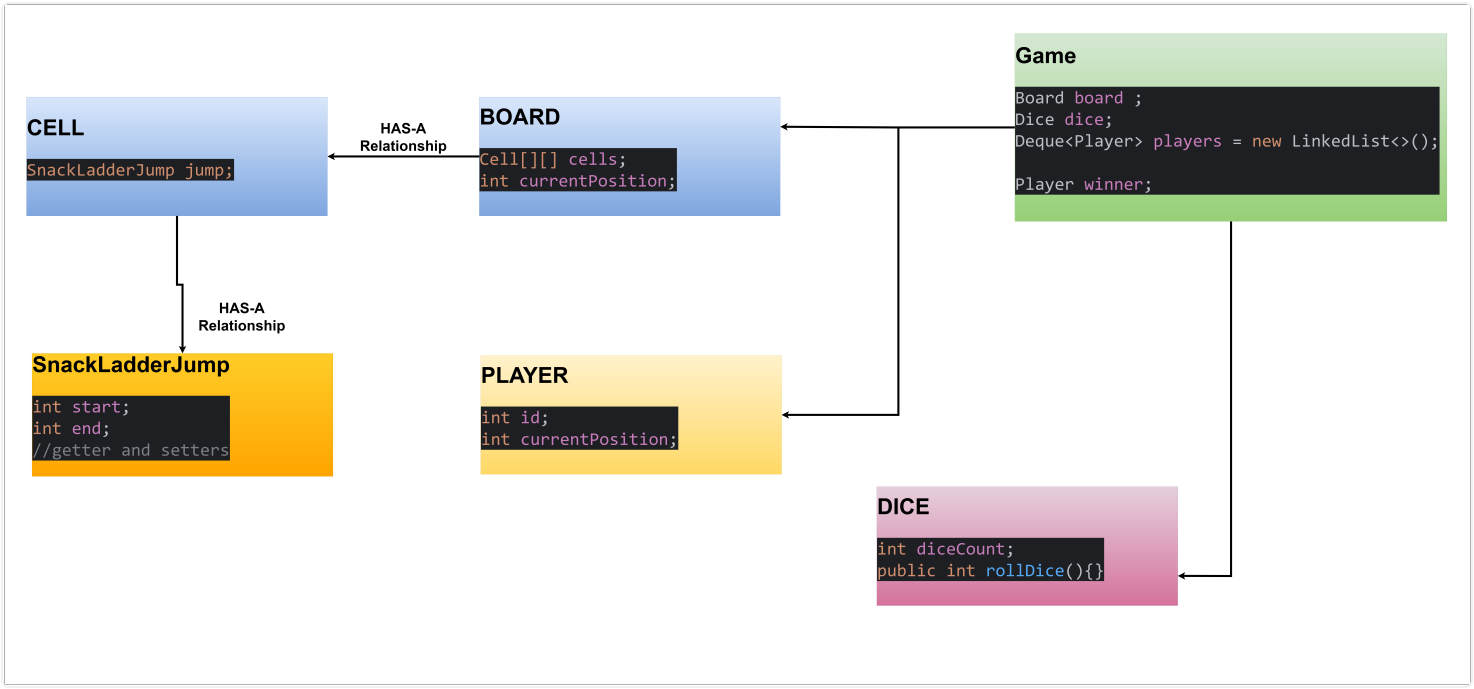
  1, but should be scalable
- How many snacks and ladder

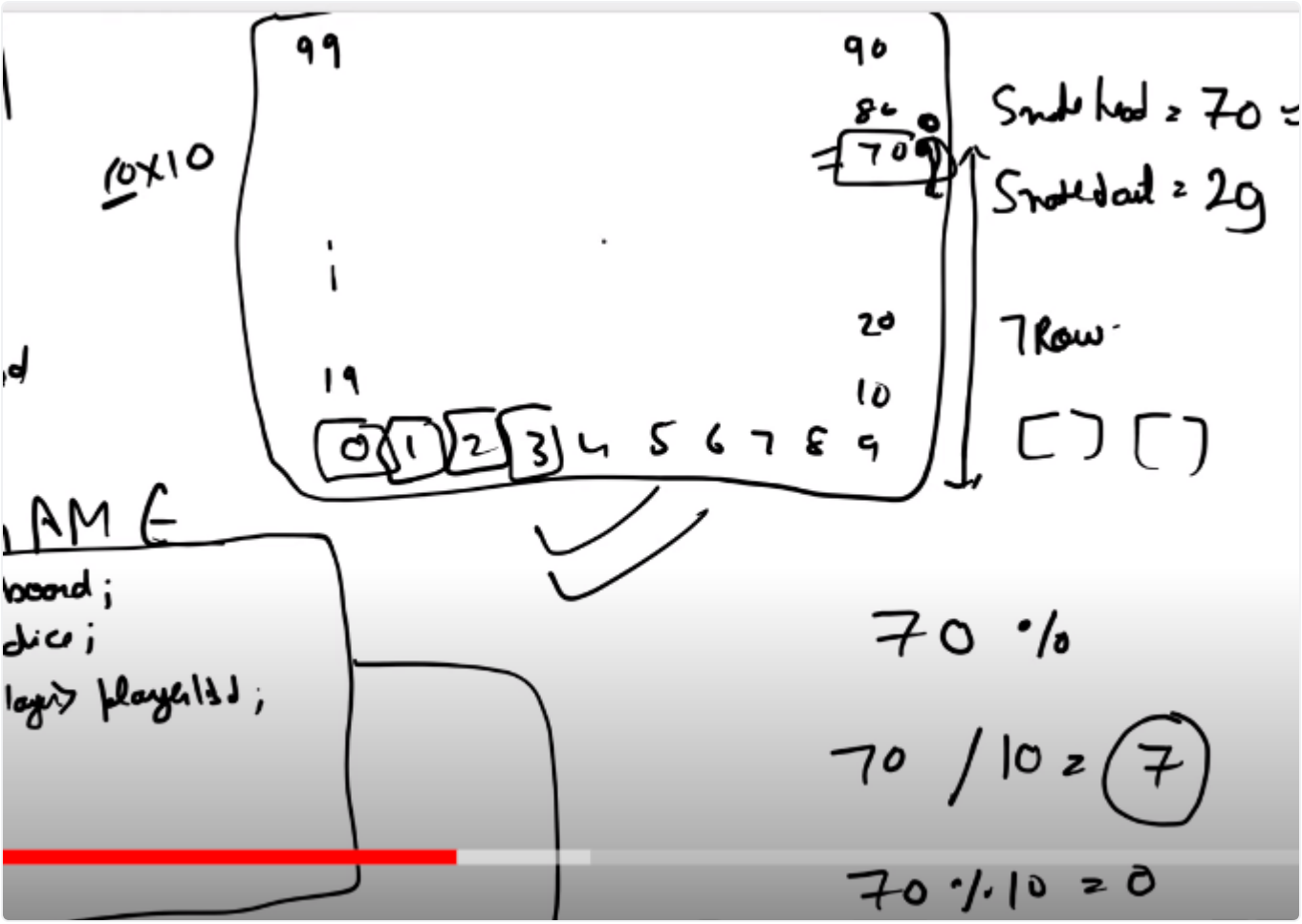
  Setup time we should be able to do dynamically.
- Winning Conditions ?

Any one wins game finished.

Objects:

- DICE
- SNAKE, LADDER
- BLOOD
- PLAYERS
- CELLS


Game
Board board ;
Dice dice;
Deque<Player> players = new LinkedList<>();
Player winner;
CELL
SnackLadderJump jump;
HAS-A Relationship
BOARD
Cell[][] cells;
int currentPosition;
HAS-A Relationship
SnackLadderJump
int start;
int end;
//getter and setters
PLAYER
int id;
int currentPosition;
DICE
int diceCount;
public int rollDice(){}


99
90
80
70
10X10
20
19
10
0 1 2 3 4 5 6 7 8 9
70 %
70 / 10 = 7
70 %. 10 = 0

Code:

```
package System_Design.LLD.Snack_and_ladder;

public class SnackLadderJump {
    int start;
    int end;


    //getter and setters
    public int getStart() {
        return start;
    }

    public void setStart(int start) {
        this.start = start;
    }

    public int getEnd() {
        return end;
    }

    public void setEnd(int end) {
        this.end = end;
    }
}
```

```
package System_Design.LLD.Snack_and_ladder;

public class Cell {
    SnackLadderJump jump;

    //getter and setter

    public SnackLadderJump getJump() {
        return jump;
    }

    public void setJump(SnackLadderJump jump) {
        this.jump = jump;
    }
}
```

```
package System_Design.LLD.Snack_and_ladder;

import java.util.concurrent.ThreadLocalRandom;
```

```
public class Board {
    Cell[][] cells;

    Board(int boardSize, int numOfSnakes, int numOfLadders){
        initializeCells(boardSize);
        addSnakesLadder(cells,numOfLadders,numOfSnakes);
    }

    private void addSnakesLadder(Cell[][] cells, int numOfLadders, int numOfSnakes) {

        while (numOfLadders>0){
            int ladderStart =
ThreadLocalRandom.current().nextInt(1,cells.length*cells.length-1);
            int ladderEnd =
ThreadLocalRandom.current().nextInt(1,cells.length*cells.length-1);

            if(ladderStart>=ladderEnd){
                continue;
            }

            SnackLadderJump ladderJump = new SnackLadderJump();
            ladderJump.start=ladderStart;
            ladderJump.end=ladderEnd;

            Cell cell = getCell(ladderStart);
            cell.jump = ladderJump;

            numOfLadders--;
        }

        while (numOfSnakes>0){
            int snakeHead =
ThreadLocalRandom.current().nextInt(1,cells.length*cells.length-1);
            int snakeTail =
ThreadLocalRandom.current().nextInt(1,cells.length*cells.length-1);

            if(snakeTail>=snakeHead){
                continue;
            }

            SnackLadderJump snakeJump = new SnackLadderJump();
            snakeJump.start=snakeHead;
            snakeJump.end=snakeTail;

            Cell cell = getCell(snakeHead);
            cell.jump = snakeJump;

            numOfSnakes--;
        }
```

```
    }

     Cell getCell(int playerPosition) {
        int boardRow = playerPosition/cells.length;
        int boardColumn = (playerPosition%cells.length);
        return cells[boardRow][boardColumn];
    }

    private void initializeCells(int boardSize) {
        cells = new Cell[boardSize][boardSize];
        for(int i=0;i<boardSize;i++){
            for(int j=0;j<boardSize;j++){
                Cell cell = new Cell();
                cells[i][j] = cell;
            }
        }
    }


}
```

```
package System_Design.LLD.Snack_and_ladder;

import java.util.concurrent.ThreadLocalRandom;

public class Dice {
    int diceCount;
    int min=1;
    int max=6;
    public Dice(int diceCount){
        this.diceCount=diceCount;
    }
    public int rollDice(){
        int totalSum  =0;
        int diceUsed=0;


        while (diceUsed<diceCount){
            totalSum+= ThreadLocalRandom.current().nextInt(min,max+1);
            diceUsed++;
        }
        return totalSum;
    }
}
```

```
package System_Design.LLD.Snack_and_ladder;

public class Player {
    String id;
    int currentPosition;

    public Player(String id, int currentPosition) {
        this.currentPosition=currentPosition;
        this.id=id;
    }
}
```

```
package System_Design.LLD.Snack_and_ladder;

import java.util.Deque;
import java.util.LinkedList;

public class Game {
    Board board ;
    Dice dice;
    Deque<Player> players = new LinkedList<>();

    Player winner;

    public Game(){
        initializeGame();
    }

    private void initializeGame() {
        board = new Board(10,5,4);
        dice = new Dice(1);
        winner=null;
        addPlayers();

    }

    private void addPlayers() {
        Player p1 = new Player("p1",0);
        Player p2 = new Player("p2",0);

        players.add(p1);
        players.add(p2);
    }


    public void startGame(){
        while(winner==null){
            //check whose turn now;
```

```
            Player playerTurn = findPlayerTurn();
            System.out.println("Player turn is: "+playerTurn.id + " current Position is
"+playerTurn.currentPosition);

            //roll the dice;
            int diceNumbers = dice.rollDice();

            //get the new position
            int playerNewPosition = playerTurn.currentPosition +diceNumbers;

            playerNewPosition=jumpCheck(playerNewPosition);

            playerTurn.currentPosition=playerNewPosition;

            System.out.println("Player turn is: "+playerTurn.id + " new Position is
"+playerNewPosition);


            //check for wining conditions
            if(playerNewPosition >=board.cells.length*board.cells.length-1){
                winner=playerTurn;
            }
        }
        System.out.println("Winner is: "+ winner.id);
    }

    private int jumpCheck(int playerNewPosition) {
        if(playerNewPosition>board.cells.length*board.cells.length-1){
            return playerNewPosition;
        }

        Cell cell = board.getCell(playerNewPosition);
        if(cell.jump!=null && cell.jump.start == playerNewPosition){
            String jumpBy = (cell.jump.start<cell.jump.end ? "ladder":"snake");
            System.out.println("Jump done by: "+jumpBy);
            return cell.jump.end;
        }

        return playerNewPosition;
    }

    private Player findPlayerTurn() {
        Player p = players.removeFirst();
        players.addLast(p);
        return p;
    }

}
```

```
package System_Design.LLD.Snack_and_ladder;

public class Main {
    public static void main(String[] args){
        Game game = new Game();
        game.startGame();
    }
}
output:
Player turn is: p1 current Position is 0
Player turn is: p1 new Position is 6
Player turn is: p2 current Position is 0
Player turn is: p2 new Position is 5
Player turn is: p1 current Position is 6
Player turn is: p1 new Position is 9
Player turn is: p2 current Position is 5
Player turn is: p2 new Position is 10
Player turn is: p1 current Position is 9
Player turn is: p1 new Position is 10
Player turn is: p2 current Position is 10
Jump done by: ladder
Player turn is: p2 new Position is 81
Player turn is: p1 current Position is 10
Player turn is: p1 new Position is 14
Player turn is: p2 current Position is 81
Player turn is: p2 new Position is 84
Player turn is: p1 current Position is 14
Player turn is: p1 new Position is 20
Player turn is: p2 current Position is 84
Player turn is: p2 new Position is 86
Player turn is: p1 current Position is 20
Player turn is: p1 new Position is 26
Player turn is: p2 current Position is 86
Player turn is: p2 new Position is 88
Player turn is: p1 current Position is 26
Player turn is: p1 new Position is 28
Player turn is: p2 current Position is 88
Player turn is: p2 new Position is 91
Player turn is: p1 current Position is 28
Player turn is: p1 new Position is 33
Player turn is: p2 current Position is 91
Player turn is: p2 new Position is 93
Player turn is: p1 current Position is 33
Jump done by: snake
Player turn is: p1 new Position is 13
Player turn is: p2 current Position is 93
Player turn is: p2 new Position is 94
Player turn is: p1 current Position is 13
Player turn is: p1 new Position is 17
```

```
Player turn is: p2 current Position is 94
Player turn is: p2 new Position is 99
Winner is: p2

Process finished with exit code 0
```