

## General guidelines

At Sahaj, we strive to build high-quality software that has strong aesthetics (is readable and maintainable), has extensive safety nets to safeguard quality, handles errors gracefully and works as expected, without breaking down, with varying input.

We are looking for people who can write code that has flexibility built-in, by adhering to the principles of clean coding and Object-Oriented Development, and have the ability to deal with the real-life constraints/trade-offs while designing a system.

*It is important to note that we are not looking for a GUI and we are not assessing you on the capabilities around code required to do the I/O. **The focus is on the overall design.** So, while building a solution, it would be nicer if the input to the code is provided either via unit tests or a file. Using the command line (for input) can be tedious and difficult to test, so it is best avoided.*

Following is a list of things to keep in mind, before you submit your code, to ensure that your code focuses on attributes, we are looking for -

- Is the behaviour of an object distinguished from its state and is the state encapsulated?
- Have you applied [SOLID principles](#) to your code?
- Have you applied the principles of [YAGNI](#) and [KISS](#) (additional info [here](#))?
- Have you unit tested your code or did TDD? If you have not, we strongly recommend you read about it and attempt it with your solution. Having tests is a must and we do appreciate if there is ample test coverage for the given solution, it is a definite plus if you write them.
- Have you looked at basic refactoring to improve the design of your code? [Here](#) are some guidelines for the same.
- Finally, and foremost, are the principles applied in a pragmatic way. Simplicity is the strongest of the traits of a piece of code. However, easily written code may not necessarily be simple code.

## **Problem Statement - Snakes & Ladders Simulator**

(Average time to write a solution 5-6 hours)

We want to build a simulator for the game snake and ladders (also known as chutes and ladders). The simulator can be run multiple times to find out various stats about the game. The below sections provide details regarding the rules of the game, implementation constraints, and the stats to be captured.

### **Game rules:**

- The game starts at position 0 which is outside the board and it finishes when a player reaches the last square that is 100.
- The player rolls a single 6 sided dice (die) numbered from 1 to 6 to move their token by the number of squares indicated by the die rolled.
- If, on completion of a roll, a player's token lands on the lower-numbered end of a "ladder", the player moves the token up to the ladder's higher-numbered square.
- If the player lands on the higher-numbered square of a "snake" (or chute), the token must be moved down to the snake's lower-numbered square
- If a 6 is rolled by the player, the player gets an additional roll after the token is moved. This additional roll is considered as a part of the same turn of the current player (1 turn can have multiple rolls if 6 is rolled); otherwise, play passes to the next player in turn.
- The player who is first to bring their token to the last square of the board is the winner.
- The player must roll the exact number to reach the final square. For example, if a player needs a 3 to reach square 100 and rolls a 5, the player stays at the current location.
  - If the player rolls 6 when they are stuck, they get additional rolls, but the token does not move ahead. For example, 3 is needed to win and the player rolls a 6. Then they get the additional roll. If they roll 6 again, they get another roll, and so on.

### **Implementation Constraints:**

- The number of times the simulation will be run should be configurable
- The positions of snakes & ladders should be configurable and not hardcoded on the board
- The role of snakes & ladders is set once before the simulation and will stay the same during the entire simulation.
- For the purpose of the simulation, a single-player implementation is acceptable. In case multiple players are implemented, the simulation result should still be across all the players and all the runs.
- Reading of file/console input for the input to the program is not necessary











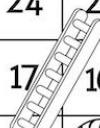

## Stats to be captured:

The following stats are to be captured across all the simulations:

- Minimum/Average/Maximum number of rolls needed to win.
- Minimum/Average/Maximum amount of climbs during the game
  - A climb is the amount of distance covered by climbing up a ladder. For example, if the token goes up a ladder from 21 to 51, the distance climbed is 30.
- Minimum/Average/Maximum amount of slides during the game
  - A slide is the amount of distance covered by sliding down a snake. For example, if the token goes down a snake from 88 to 48, the slide distance is 40.
- The biggest climb in a single turn.
- The biggest slide in a single turn.
- Longest turn. The longest turn is the highest streak of consecutive rolls due to rolling 6s.
  - Examples:
    - Roll x is 5 and roll y is 3, then the longest turn so far is 5 as it is the highest roll.
    - Rolls in turn x are [6,4] and rolls in turn y are [6,3]. The longest turn is [6,4].
    - Rolls in turn x are [6,6,6,5] and rolls in turn y are [6,6,6,6,1]. The longest turn is [6,6,6,6,1].
- Minimum/Average/Maximum unlucky rolls during the game
  - An unlucky roll is considered when any of the following is true
    - A player lands on a snake
- Minimum/Average/Maximum lucky rolls during the game
  - A lucky roll is considered when any of the following is true
    - A player lands on a ladder
    - Misses a snake by 1 or 2 steps
    - When they roll the exact number needed to win after 94 in a single roll.

## Sample Board:

Below image is a sample board that can be used for the simulations

100		98	97	96	95	94	93	92	91
81	82	83	84	85	86	87			90
80		79	78	77		75	74	73	72
61	62	63	64	65		67	68	69	70
60		59	58	57	56	55		53	52
41	42		44	45	46	47	48	49	50
	39	38		36	35	34	33	32	31
21	22	23	24	25	26		28	29	30
20	19	18	17	16	15	14	13	12	11
1	2	3	4	5	6	7	8	9	10