Methodology

1. Define Constraints and Constants (`mapConfig.js`)

- Set up a configuration process asking the user's input for map width, map height, min/max room size, and max rooms .

2. Grid and Room Representation

- Represent the map as a 2D array, initially filled with `#`. (`initializeMap method`)
- Rooms are stored as objects with properties like `x`, `y`, `width`, and `height`.

3. Room Generation (`generateRoom method`)

- Generate random rooms:
    - Ensure room sizes are within the min/max range.
    - Randomly place rooms within the grid.
    - Check for overlap and enforce a one-tile gap between rooms.

4. Room Placement

- For each valid room: (`canPlaceRoom method`)
    - Update the 2D grid with room walls (`|`, `_`) and interior (`.`).(`generateRoom method`)
    - Number the rooms in the top-left corner starting from 1.

5. Print the Map (`displayMap method`)

- Once all rooms are placed, print the 2D array to visualize the dungeon.

Libraries

install random-js

Why i chose to work this way

This implementation might not be the most optimal in terms of utilizing as much free map area as possible (trying to generate a room on a completely random position on the map checking if it colides with any of the other rooms) and probably also wastes extra resources needed to run the programme since it has some complexity coming from the unoptimized way of finding room collision (for example if the search for available map area was starting from top left going to bottom right since also the enumeration of the room is done accordingly).
Initially i tried to figure out a way that tries to generate to most rooms possible by crawling the map from top-left to bottom-right (making projections on X axis first and then on Y), using a Class to work with rooms having some helper methods in the class such as getting bounds)