# Catan Map Generator Documentation

## Overview

The **Catan Map Generator** is a web-based application designed to generate customizable maps for the board game *Catan*. It allows users to create maps with specific configurations, such as controlling the placement of tiles and numbers, ensuring compliance with game rules, and enabling features like night mode for better visual accessibility.

The project is built using **Preact**, a lightweight alternative to React, and **Vite**, a fast build tool. It leverages TypeScript for type safety and LESS for styling.

## Features

### 1. Custom Map Generation

* Users can generate maps with specific rules:
  + Control whether good numbers (6 & 8) or bad numbers (2 & 12) can touch.
  + Ensure no two tiles of the same resource type are adjacent.
  + Randomly generate maps ignoring all constraints.

### 2. Night Mode

* A toggleable night mode changes the map’s appearance for better visibility in low-light environments.

### 3. Modular Regeneration

* Regenerate only tiles or numbers without affecting the rest of the map.

### 4. Responsive Design

* The application is fully responsive, ensuring usability across devices of varying screen sizes.

## Project Structure

### 1. Core Logic

The core logic for map generation resides in the src/Logic directory. Key files include: - **GenerateMap.ts**: Manages the overall map generation process. - **GenerateTiles.ts**: Handles the placement of resource tiles. - **placeNumbers.ts**: Assigns number tokens to tiles based on user-defined constraints. - **generateMapWithBacktracking.ts**: Implements a backtracking algorithm to ensure no adjacent tiles have the same resource type. - **getNeighboursFromIndex.ts**: Provides the indices of neighboring tiles for a given tile index.

### 2. User Interface

The UI components are located in the src/GUI directory: - **Hexagons**: Components for rendering individual hex tiles (e.g., ForestHex.tsx, DesertHex.tsx). - **Containers**: Higher-level components like Map.tsx and OptionsPanel.tsx that manage the layout and user interactions.

### 3. State Management

The MapViewStateHandler.ts file manages the application’s state, including the current map configuration and night mode status.

## Key Files and Their Responsibilities

### 1. src/index.tsx

This is the entry point of the application. It initializes the App component and renders it into the DOM.

### 2. src/Logic/GenerateMap.ts

* Implements the singleton MapGenerator class.
* Provides the generateMap method, which orchestrates the generation of tiles and numbers based on user options.

### 3. src/Logic/placeNumbers.ts

* Places number tokens on tiles while adhering to constraints like:
  + Good numbers (6 & 8) not touching.
  + Bad numbers (2 & 12) not touching.
  + No two tiles having the same number.

### 4. src/GUI/Containers/Map.tsx

* Renders the map using hexagonal tiles.
* Dynamically adjusts the layout based on the map configuration.

### 5. src/GUI/Containers/OptionsPanel.tsx

* Provides a user interface for configuring map generation options.
* Includes buttons for random generation, smart generation, and modular regeneration.

## How It Works

### 1. Map Generation

The map generation process involves two main steps: 1. **Tile Placement**: - Tiles are placed using either random generation or a backtracking algorithm to ensure no adjacent tiles have the same resource type. 2. **Number Placement**: - Numbers are assigned to tiles based on user-defined constraints.

### 2. State Management

The MapViewStateHandler class maintains the application’s state, including: - The current map configuration. - Night mode status.

### 3. Rendering

The CatanMap component renders the map using hexagonal tiles. Each tile is represented by a specific component (e.g., ForestHex, DesertHex).

## Configuration Options

### 1. Tile Placement

* **Random Generate**: Ignores all constraints and places tiles randomly.
* **Same Resources Can Touch**: Allows adjacent tiles to have the same resource type.

### 2. Number Placement

* **Good Numbers Can Touch**: Allows 6 & 8 to be adjacent.
* **Bad Numbers Can Touch**: Allows 2 & 12 to be adjacent.
* **Same Numbers Can Touch**: Allows tiles with the same number to be adjacent.

### 3. Modular Regeneration

* **Regenerate Tiles**: Replaces all tiles while keeping the numbers intact.
* **Regenerate Numbers**: Reassigns numbers without changing the tiles.

## Development Setup

### 1. Prerequisites

* Node.js (v16 or higher)
* npm (v7 or higher)

### 2. Installation

1. Install dependencies:

* npm install

### 3. Running the Application

* Start the development server:
* npm start
* Open your browser and navigate to http://localhost:5173.

### 

## Testing

The project uses **Jest** for testing. Test files are located in the src/\_\_tests\_\_ directory.

### Running Tests

* Run all tests:
* npm test
* Generate a coverage report:
* npm run test:coverage

### Test Coverage

The following functionalities are covered by tests: - Tile placement logic (generateMapWithBacktracking.test.ts) - Number placement logic (placeNumbers.test.ts) - Utility functions (getNeighboursFromIndex.test.ts, getRandomHexType.test.ts)

## Styling

The project uses **LESS** for styling. Key stylesheets include: - **index.less**: Global styles. - **OptionsPanel.less**: Styles for the options panel. - **Map.less**: Styles for the map layout.