System Architecture

This project is a web-based Minesweeper game built with HTML, CSS, and TypeScript. The architecture is based on a modular frontend structure, with a clear separation of concerns between the UI, game logic, and build process.

Frontend

The frontend is responsible for rendering the game and handling user interactions. It is composed of the following components:

- **HTML (src/html/index.html)**: Defines the basic structure of the web page, including the game grid, navigation bar, and UI elements for setting the mine count, displaying the timer, and selecting the AI difficulty.
- **CSS (src/css/index.css)**: Provides the styling for all the UI elements, including the game grid, tiles, and overall layout.
- **TypeScript (src/ts/)**: Contains the core logic of the game, divided into several modules:
    - *index.ts*: The main entry point of the application. It initializes the game grid, sets up event listeners for user interactions, and orchestrates the overall game flow.
    - *create_grid.ts:* Responsible for dynamically generating the HTML elements for the game grid and its row and column labels.
    - *minefield_gen.ts*: Generates the 2D array representing the minefield, ensuring that the first click is always in a safe area.
    - *reveal.ts*: Implements the core gameplay logic, including revealing cells, handling mine explosions, performing a flood-fill for empty areas, and checking for victory or game-over conditions.
    - *flagging.ts*: Manages the user's ability to flag and unflag cells.
    - *userMineCount.ts*: Handles the user's input for the number of mines to be placed on the grid.
    - *status.ts*: Manages the display of the current game status (e.g., "Playing", "Game Over", "You Win!").
    - *timer.ts*: Implements the game timer, which starts on the first click and stops when the game ends.
    - *ai_solver.ts*: Contains the logic for the AI opponent, with different difficulty levels (Easy, Medium, Hard).
    - *localTypes.ts*: Defines custom TypeScript types used throughout the project to ensure type safety.

Build Process

The project uses a modern build process to bundle and transpile the source code into a format that can be run in a web browser.

- *Webpack (webpack.config.js)*: Acts as the module bundler. It takes all the TypeScript and CSS files as input and outputs a single JavaScript file (index.js) and an HTML file (index.html) in the dist directory. It is configured with the following loaders:
    - ts-loader: Transpiles the TypeScript code into JavaScript.
    - style-loader and css-loader: Inject the CSS into the DOM.
    - Asset resource management for audio files.
- *TypeScript Compiler (tsconfig.json)*: Configures the TypeScript compiler with strict type-checking rules and other settings to ensure code quality and maintainability.

Dependencies

The project's dependencies are managed through npm and are listed in the package.json file. All dependencies are development dependencies, meaning they are only needed for the build process and not at runtime. These include:

- webpack and webpack-cli
- typescript
- ts-loader
- css-loader and style-loader
- html-webpack-plugin

Version Control

- *Git (.gitignore)*: The project is managed using the Git version control system. The .gitignore file is used to exclude files and directories that should not be committed to the repository, such as node_modules and the dist build output directory.