Design Document: 2048 Game Implementation

1. Introduction

The 2048 game implementation provides the core game logic for the popular puzzle game. The objective of the game is to slide numbered tiles on a 4x4 grid to combine them and create a tile with the number 2048. This implementation focuses on game mechanics, user interaction, and the logic for tilting the grid, while integrating provided components like Tile, BoardWidget, and Side.

---

2. Overview of Components

1. User-Provided Logic

- **Model Class:** Implements the core logic of the 2048 game, including tile merging, board tilting, and score tracking.
- **Main Class:** Initializes the game and manages gameplay configurations (e.g., board size, custom starts, and random seed).
- **Board Class:** Implements the core logic for managing the tiles on the grid, including movement, merging, and board state representation

2. Provided Components

- **Tile Class:** Represents individual tiles on the board with their value, position, and merged status.
- **Game Class:** Coordinates game flow, user input, and tile generation.
- **BoardWidget Class:** Handles rendering of the game board and visual effects.
- **Side Class:** Provides utility methods to interpret board coordinates based on tilt direction.

---

3. Key Features

Game Logic

- **Tile Movement:** Implements rules for sliding and merging tiles in any direction (NORTH, SOUTH, EAST, WEST).
- **Merge Handling:** Combines tiles of equal value and doubles their value.
- **Score Calculation:** Updates the score based on the values of merged tiles.
- **Random Tile Generation:** Adds a new tile (value 2 or 4) in an empty space after every valid move.
- **Game Over Detection:** Determines if no valid moves are possible.

User Interaction

- **Keyboard Input:** Processes arrow key presses to tilt the board in the corresponding direction.
- **New Game & Quit Options:** Provides menu buttons to start a new game or quit the application.

Custom Configuration

- Supports customizable board size, random tile generation probabilities, and custom starting states for debugging.

---

4. System Design

Model Class

The Model class encapsulates the game logic:

- **Data Members:**
    - Tile[][] board: Represents the 4x4 grid of tiles.
    - int score: Tracks the player's score.
    - boolean gameOver: Flags when no valid moves remain.
- **Core Methods:**
    - addTile(Tile t): Adds a new tile to the board.
    - tilt(Side s): Handles tile movement and merging in the specified direction.
    - gameOver(): Checks for the end-game condition.
- **Board Class**

The Board class encapsulates the game grid's logic and state:

- **Data Members:**
    - Tile[][] _values: Represents the grid's tiles, with null for empty spaces.
    - Side _viewPerspective: Tracks the current board orientation.
- **Core Methods:**
    - addTile(Tile t): Adds a new tile to the board.
    - move(int x, int y, Tile t): Moves or merges a tile into a new position.
    - tile(int x, int y): Retrieves the tile at the specified coordinates.
    - clear(): Resets the board to an empty state.
    - resetMerged(): Resets the merge status of all tiles.

    - setViewingPerspective(Side s): Adjusts the board's perspective for directional tilts.

Integration with Provided Components

- **Tile Management:** Utilizes the Tile class for tile properties (value, position, merging).
- **Coordinate Transformations:** Relies on the Side class for reorienting the board during tilts.
- **Visualization:** Outputs the current board state using the toString() method for debugging.

---

5. Algorithms

1. Tile Merging and Movement

- Iterates over tiles based on the tilt direction (Side).
- Moves tiles to their furthest valid position in the specified direction.
- Merges adjacent tiles of the same value and marks them as merged for the current move.

2. Random Tile Generation

- Generates a tile with value 2 (90% probability) or 4 (10% probability).
- Finds a random empty space on the board to place the new tile.

3. Game Over Detection

- Checks for the following conditions:
    - No empty spaces on the board.
    - No adjacent tiles with equal values.

---

6. Game Flow

1. **Initialization:**
    - Creates a new board of the specified size.
    - Adds a random tile to start the game.
2. **Gameplay Loop:**
    - Waits for user input (arrow keys).
    - Tilts the board in the specified direction.
    - If the board state changes:
        - Adds a new random tile.

        - Updates the score.
    - Checks for the game-over condition.
3. **Game Over:**
    - Displays the "Game Over" message if no valid moves remain.

---

7. Challenges and Solutions

- **Challenge:** Handling multiple merges in a single move.
    - **Solution:** Used a merged flag in the Tile class to prevent double merging.
- **Challenge:** Translating board coordinates for different tilt directions.
    - **Solution:** Utilized the Side class for consistent coordinate transformations.