SKILL DEVELOPMENT
Project Report

DLithe Consultancy Services Pvt. Ltd.

DLithe™
Be Agile..Be in Demand

Project Report Assessment

Student Name: Shilpa Mali

Reg. no: 2JR23CS092

Assignment: Java

Organization: DLithe Consultancy Services Pvt. Ltd.

Supervisor's Name: Archana SM

Submitted to

Signature of Training Supervisor

Date:

Signature of Students

Date:

DLithe™
Be Agile..Be in Demand

TABLE OF CONTENTS

INTRODUCTION

The 2048 game is a popular single-player sliding puzzle game that combines logic, strategy, and a bit of luck. Originally developed by Gabriele Cirulli in 2014, the game challenges players to slide numbered tiles on a 4×4 grid to combine them and create a tile with the number 2048. The simplicity of its design and addictive gameplay made it go viral within days of its release, and it has since inspired many variations and implementations.

In this project, the 2048 game has been implemented using Java. The program handles the core logic such as tile movement, merging rules, random tile generation, and game-over detection. The user interacts with the game using keyboard inputs (W/A/S/D for directions), and after every valid move, a new tile is randomly generated. The goal is to continue combining tiles until a tile with the value 2048 appears — at which point the player wins, though they can keep playing beyond that. This project helped enhance understanding of array manipulation, control flow, and basic game development logic in Java.

This project not only provides a fun and interactive game experience but also demonstrates important programming concepts such as event handling, dynamic updates to the game state, and efficient algorithms for merging and sliding tiles. Implementing the 2048 game deepens one's understanding of data structures like arrays and matrices, as well as control structures that manage game logic and user inputs. Additionally, this project serves as a great stepping stone for building more complex games and applications in the future.

DLithe™
Be Agile..Be in Demand

USE CASE REQUIREMENTS

Use Case:

- The player launches the 2048 game application.
- The game initializes a 4×4 grid with two tiles, each with the number 2.
- The player uses keyboard inputs (W for up, A for left, S for down, D for right) to slide the tiles in the chosen direction.
- After each valid move, tiles with the same number that collide merge into one tile with their sum.
- A new tile (2 or sometimes 4) spawns at a random empty position after each move.
- The game checks if the player has created a tile with the number 2048 (winning condition) or if no valid moves remain (game over).
- The player can restart the game anytime.

Functional Requirements:

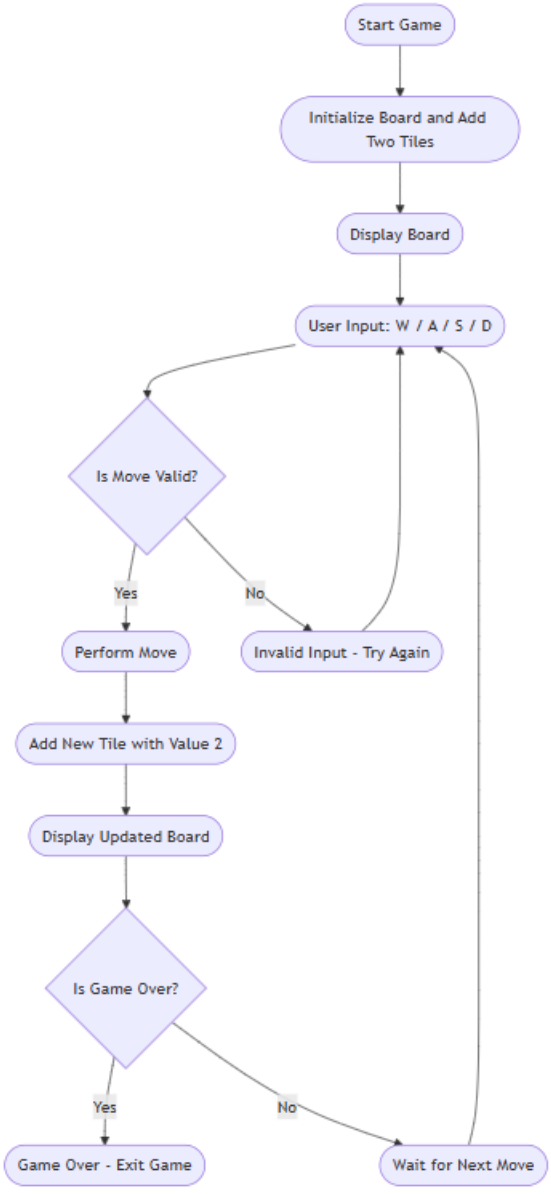
1. The game must initialize with a 4×4 grid and two tiles showing the number 2.
2. The game must accept user inputs for moving tiles up, down, left, and right.
3. Tiles with the same value merge when moved into each other, doubling their value.
4. After every valid move, a new tile (usually 2, sometimes 4) appears in a random empty cell.
5. The game should detect when no more moves are possible and declare game over.
6. The game should detect when the player reaches the tile 2048 and declare victory.
7. The game interface should update visually after every move to reflect the new state.

DLithe™
Be Agile..Be in Demand

MIND MAP

DLithe™
Be Agile..Be in Demand

FLOW CHART



DLithe™
Be Agile..Be in Demand

SOURCE CODE

Game2048.java > Game2048 > rotateOnce(int[][])

```java
import java.util.*;

public class Game2048 {
    private static final int SIZE = 4;
    private int[][] board = new int[SIZE][SIZE];
    private Random random = new Random();
    private boolean moved;

    Run | Debug
    public static void main(String[] args) {
        Game2048 game = new Game2048();
        game.start();
    }

    public void start() {
        addRandomTile();
        addRandomTile();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            printBoard();
            System.out.print(s:"Enter move (W/A/S/D): ");
            String input = scanner.nextLine().toUpperCase();

            switch (input) {
                case "W":
                    moved = moveUp();
                    break;
                case "A":
                    moved = moveLeft();
                    break;
                case "S":
                    moved = moveDown();
                    break;
                case "D":
                    moved = moveRight();
                    break;
```

```
                default:
                    System.out.println(x:"Invalid input! Use W, A, S or D.");
                    continue;
            }

            if (moved) {
                addRandomTile();
                if (isGameOver()) {
                    printBoard();
                    System.out.println(x:"Game Over! No more moves.");
                    break;
                }
            } else {
                System.out.println(x:"Move not valid. Try a different direction.");
            }
        }
        scanner.close();
    }

    private void addRandomTile() {
        List<int[]> empty = new ArrayList<>();
        for (int i = 0; i < SIZE; i++)
            for (int j = 0; j < SIZE; j++)
                if (board[i][j] == 0)
                    empty.add(new int[] { i, j });

        if (!empty.isEmpty()) {
            int[] pos = empty.get(random.nextInt(empty.size()));
            board[pos[0]][pos[1]] = random.nextInt(bound:10) < 9 ? 2 : 4; // 90% 2, 10% 4
        }
    }

    private void printBoard() {
        System.out.println(x:"----------------------------");
```

```
        for (int[] row : board) {
            for (int cell : row) {
                if (cell == 0)
                    System.out.print(s:"|    .");
                else
                    System.out.printf(format:"|%5d", cell);
            }
            System.out.println(x:"|");
        }
        System.out.println(x:"-----------------------------");
    }

    // Move and merge functions for all 4 directions
    private boolean moveLeft() {
        boolean moved = false;
        for (int i = 0; i < SIZE; i++) {
            int[] row = board[i];
            int[] compressed = compress(row);
            int[] merged = merge(compressed);
            if (!Arrays.equals(row, merged)) {
                board[i] = merged;
                moved = true;
            }
        }
        return moved;
    }

    private boolean moveRight() {
        rotateBoard(degree:180);
        boolean moved = moveLeft();
        rotateBoard(degree:180);
        return moved;
    }
```

```
private boolean moveUp() {
    rotateBoard(degree:270);
    boolean moved = moveLeft();
    rotateBoard(degree:90);
    return moved;
}

private boolean moveDown() {
    rotateBoard(degree:90);
    boolean moved = moveLeft();
    rotateBoard(degree:270);
    return moved;
}

// Helper: compress non-zero numbers to left
private int[] compress(int[] row) {
    int[] newRow = new int[SIZE];
    int index = 0;
    for (int num : row) {
        if (num != 0)
            newRow[index++] = num;
    }
    return newRow;
}

// Helper: merge adjacent tiles
private int[] merge(int[] row) {
    for (int i = 0; i < SIZE - 1; i++) {
        if (row[i] != 0 && row[i] == row[i + 1]) {
            row[i] *= 2;
            row[i + 1] = 0;
            i++; // skip next
        }
    }
```

```
// Rotate board clockwise by degrees (90, 180, 270)
private void rotateBoard(int degree) {
    for (int i = 0; i < degree / 90; i++) {
        board = rotateOnce(board);
    }
}

private int[][] rotateOnce(int[][] mat) {
    int[][] rotated = new int[SIZE][SIZE];
    for (int i = 0; i < SIZE; i++)
        for (int j = 0; j < SIZE; j++)
            rotated[j][SIZE - 1 - i] = mat[i][j];
    return rotated;
}

private boolean isGameOver() {
    for (int i = 0; i < SIZE; i++)
        for (int j = 0; j < SIZE; j++)
            if (board[i][j] == 0)
                return false;

    for (int i = 0; i < SIZE; i++)
        for (int j = 0; j < SIZE - 1; j++)
            if (board[i][j] == board[i][j + 1])
                return false;

    for (int j = 0; j < SIZE; j++)
        for (int i = 0; i < SIZE - 1; i++)
            if (board[i][j] == board[i + 1][j])
                return false;

    return true;
}
```

DLithe™
Be Agile..Be in Demand

OUTPUT

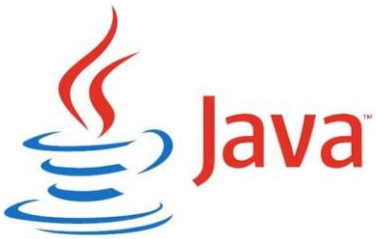
```
PS C:\Users\hp\OneDrive\Desktop\Game2048.java>  & 'C:\Program Files\Eclipse Adoptium\jdk-21.0.7.6-hotspot\bin\java.exe' '-XX:
howCodeDetailsInExceptionMessages' '-cp' 'C:\Users\hp\AppData\Roaming\Code\User\workspaceStorage\07c9881f71a069b4441528ec09f6
0b\redhat.java\jdt_ws\Game2048.java_ec5ce786\bin' 'Game2048'
-----------------------------
|    .|    .|    2|    4|
|    .|    .|    .|    .|
|    .|    .|    .|    .|
|    .|    .|    .|    .|
-----------------------------
Enter move (W/A/S/D): A
-----------------------------
|    2|    4|    2|    .|
|    .|    .|    .|    .|
|    .|    .|    .|    .|
|    .|    .|    .|    .|
-----------------------------
Enter move (W/A/S/D): W
Move not valid. Try a different direction.
```

```
-----------------------------
|    2|    4|    2|    .|
|    .|    .|    .|    .|
|    .|    .|    .|    .|
|    .|    .|    .|    .|
-----------------------------
Enter move (W/A/S/D): S
-----------------------------
|    .|    .|    .|    .|
|    .|    .|    .|    .|
|    .|    .|    .|    2|
|    2|    4|    2|    .|
-----------------------------
Enter move (W/A/S/D): D
-----------------------------
|    .|    .|    .|    .|
|    .|    .|    2|    .|
|    .|    .|    .|    2|
|    .|    2|    4|    2|
-----------------------------
```

```
-----------------------------
|    .|    2|    2|    4|
|    .|    .|    4|    .|
|    .|    2|    .|    .|
|    .|    .|    .|    .|
-----------------------------
Enter move (W/A/S/D): A
-----------------------------
|    4|    4|    .|    2|
|    4|    .|    .|    .|
|    2|    .|    .|    .|
|    .|    .|    .|    .|
-----------------------------
Enter move (W/A/S/D): S
-----------------------------
|    .|    .|    .|    .|
|    .|    .|    .|    2|
|    8|    .|    .|    .|
|    2|    4|    .|    2|
-----------------------------
```

DLithe™
Be Agile..Be in Demand

TECHNOLOGIES USED



CHALLENGES

1. **Implementing Correct Tile Movement and Merging:**Ensuring that tiles slide properly in the chosen direction and merge only once per move can be tricky. The logic must prevent multiple merges on the same tile within a single move.
2. **Handling Board Rotation or Directional**:Moves:Managing moves in all four directions (up, down, left, right) efficiently without duplicating code requires rotating the board or writing separate functions. Keeping track of tile positions during these transformations is challenging.
3. **Random Tile Generation After Moves:**Correctly spawning new tiles (usually '2' or '4') only in empty cells, and ensuring randomness, requires careful checking of board state.
4. **Detecting Win and Game Over Conditions:**Accurately identifying when the player has created the 2048 tile (win) or when no further moves are possible (game over) involves scanning the board thoroughly.
5. **User Input Validation:Accepting only valid move** inputs and handling incorrect inputs gracefully in a console-based game enhances user experience but needs careful coding.
6. **Optimizing Performance and Responsiveness:**Although 2048 is not resource-intensive, ensuring that the game updates quickly and responds promptly to user input improves playability.

APPLICATIONS

1. **Brain Training and Mental Agility:**The 2048 game improves critical thinking and decision-making skills. Players need to plan ahead, strategize moves, and manage space, which sharpens mental focus and agility.
2. **Demonstration of Game Development Concepts:**2048 serves as a great example for beginners to understand fundamental game development principles like matrix manipulation, user input handling, random number generation, and state management.
3. **Algorithm and Programming Practice**:Developing the 2048 game allows students and programmers to practice algorithms like tile merging, grid traversal, and game loop logic. It's a perfect project for understanding 2D arrays, loops, and control flow.
4. **Entertainment and Time-Pass:**At its core, 2048 is a fun and addictive puzzle game that provides an engaging way to relax and challenge oneself during free time.
5. **Foundation for Advanced Game Projects**:The logic and structure of 2048 can be extended or modified to create more complex puzzle or strategy games, making it a good stepping stone for more advanced development.

CONCLUSION

The 2048 game project was a great learning experience that helped in understanding key programming concepts such as arrays, loops, conditional statements, and user input handling. By implementing this game, we explored the logic behind tile movements, merging strategies, and how to manage game states efficiently.

This project not only strengthened our logical thinking but also gave us a hands-on introduction to game development. With some creative thinking and problem-solving, even a simple puzzle game like 2048 can become both challenging and enjoyable. It demonstrates how basic concepts, when applied correctly, can result in a fully functional and engaging application.