Introduction

1.1 Project Overview

History:
The 2048 game emerged from the digital tinkering of Gabriele Cirulli in March 2014. As a 19year-old web developer in Italy, he embarked on a personal project to challenge himself by creating a game from scratch during a single weekend. Inspired by the elegant merging mechanics of the iOS game "Threes" and drawing conceptual similarities to a clone titled "1024," Cirulli crafted the core gameplay loop of sliding and combining numbered tiles. Notably, he chose to release his creation as an open-source project on the code-sharing platform GitHub, making the source code freely available for others to use and modify. This act of sharing, without initial intent for profit, inadvertently laid the groundwork for the game's widespread dissemination and adaptation across the internet. Cirulli himself expressed surprise at the game's sudden surge in popularity, having simply intended to create a playable version for his own device, which couldn't handle existing similar games at the time.

Gameplay Mechanics:

- Grid: The game is played on a plain 4x4 grid of cells.
- Starting Tiles: The game begins with two tiles placed randomly on the grid, each having a value of either 2 or 4. A new tile (2 with a 90% chance, 4 with a 10% chance) appears after each move in a random empty cell.
- Sliding: The player uses the arrow keys (or swipe gestures on touchscreens) to move all tiles on the grid in one of the four cardinal directions (up, down, left, right).
- Movement: All tiles slide as far as possible in the chosen direction until they hit another tile or the edge of the grid.
- Merging: If two tiles with the same number collide while moving, they merge into a single tile with a value equal to the sum of the two original tiles (e.g., 2+2=4, 4+4=8). A tile that has just merged cannot merge again in the same move. If three identical tiles slide together, only the two farthest in the direction of movement will merge.
- Scoring: The player's score increases by the value of the new tile created during a merge. The game starts with a score of zero.
- Goal: The primary goal is to create a tile with the value 2048. However, players often continue playing after reaching 2048 to achieve higher scores and larger tile values.
- Game Over: The game ends when there are no empty cells on the grid and no adjacent tiles have the same value, meaning no further merges are possible.

Popularity:
The ascent of 2048 to viral fame was swift and impactful, commencing almost immediately after its release. Its core appeal lay in its elegantly simple ruleset, which belied a surprisingly engaging and strategically demanding gameplay experience. The ease with which the game could be accessed directly through web browsers, without the need for downloads or installations, lowered the barrier to entry for countless users. Furthermore, the inherent shareability of gameplay moments – whether celebrating the elusive 2048 tile or lamenting a near miss – through social media platforms like Twitter, Reddit, and Facebook acted as a powerful engine for organic growth. The addictive nature of the game, fueled by the desire to achieve higher scores and the satisfaction of creating larger tiles through strategic merges, kept players returning for "just one more game." The open-source nature of 2048 proved to be a

significant catalyst for its enduring popularity. Numerous developers seized the opportunity to create their own versions, optimizing it for mobile platforms, introducing new features, and even reimagining the core mechanics with different themes and grid sizes, thereby continually exposing the game to new audiences and solidifying its status as a ubiquitous and timeless puzzle game.

This Java project is a digital version of the 2048 game. Implied Java to code the rules of the game: a 4x4 grid where number tiles slide and merge when you swipe. The goal is to create a 2048 tile. Your program handles the grid, movement, merging of tiles, adding new tiles, keeping score, and deciding when the game is over. It likely also has a way for you to interact with the game, either through text commands or a visual window.

1.2 Objectives

- Implement Core Gameplay Mechanics: Successfully recreate the fundamental rules and actions of the 2048 game in Java, including the representation of a 4x4 game grid, the ability to move all tiles simultaneously in four directions based on user input, the accurate merging of adjacent tiles with identical values, the generation of new tiles, the implementation of a scoring system, and the detection of the "game over" state.

- Develop a User Interface (Text-based or Graphical): Create an interactive way for the user to play the game, potentially involving displaying the grid and score in the console with keyboard input, or designing a visual representation using Java GUI libraries with event handling.

- Apply Object-Oriented Programming (OOP) Principles in Java: Effectively utilize Java's OOP features such as encapsulation, abstraction, and modularity in the project's design and implementation to create a well-structured and maintainable codebase.

- Ensure Functionality and Correctness: Thoroughly test and debug the implemented game to guarantee that all core mechanics function as intended and accurately reflect the rules of the original 2048 game.

- Potentially Implement Additional Features: Explore and implement enhancements such as high score tracking, game reset functionality, visual improvements, or options for different grid sizes, depending on the project scope.

- Enhance Java Programming Skills: Gain practical experience and deepen understanding of Java programming concepts relevant to game development, including data structures, algorithms, and event handling.

- Develop Problem-Solving Abilities: Exercise and improve problem-solving skills by addressing the logical challenges involved in implementing the game's rules and ensuring smooth gameplay.

- Gain Experience in Game Development Fundamentals: Acquire foundational knowledge and practical experience in basic game development principles, including game loop concepts, user interaction, and state management.

- Create a Functional and Playable Game: Produce a working version of the 2048 game that is playable and provides a user experience consistent with the original game.

- Demonstrate Understanding of Game Logic and Implementation: Showcase the ability to translate the rules of a game into a working software application using the Java programming language.

1.3 Problem Statement

The core problem addressed by this project is the development of a functional and engaging digital implementation of the popular 2048 puzzle game using the Java programming language. This involves translating the abstract rules and interactive elements of the game into a concrete software application.
Specifically, the project aims to solve the following sub-problems:

1. Representing the Game State: How can the dynamic state of the 4x4 game grid, including the position and value of each tile, be effectively represented and managed within a Java program? This involves choosing appropriate data structures to store and manipulate the grid data.

2. Implementing User Interaction: How can the program effectively receive and interpret user input (e.g., keyboard arrow key presses or swipe gestures) to initiate tile movements on the grid? This requires handling input events and translating them into game actions.

3. Enforcing Game Rules: How can the core game mechanics of tile movement and merging be accurately implemented according to the 2048 rules? This includes: o Ensuring all tiles slide as far as possible in the chosen direction. o Correctly identifying and merging adjacent tiles with identical values.
    - o Calculating the new value of the merged tile.
    - o Preventing a tile from merging multiple times within a single move.

4. Generating New Tiles: How can new tiles (with values 2 or 4) be randomly generated and placed in empty cells on the grid after each valid move? This requires implementing a random number generation mechanism and ensuring placement in unoccupied locations.

5. Tracking the Game Score: How can the player's score be accurately calculated and updated based on the value of the tiles merged during gameplay? This involves maintaining a score variable and updating it appropriately after each merge.

6. Detecting Game Over Conditions: How can the program determine when the game has ended, specifically when there are no empty cells on the grid and no adjacent tiles can be merged? This requires continuously evaluating the state of the grid.

7. Providing a User Interface: How can the game state (the grid and the score) be presented to the user in a clear and understandable manner? This necessitates the development of either a text-based console interface or a more visually appealing graphical user interface.

8. Structuring the Codebase: How can the Java code be organized effectively using object-oriented principles to ensure modularity, maintainability, and readability? This involves designing appropriate classes and their interactions.

By addressing these sub-problems, the project aims to provide a complete and playable implementation of the 2048 game in Java, effectively translating the game's logic and interaction into a software application. The solution will need to consider efficiency, clarity, and adherence to the original game's design.

1.4 Scope

The scope of this project encompasses the development of a fully functional single-player 2048 game implemented using the Java programming language. This includes the following key aspects:
Core Functionality:

- Game Grid Implementation: Creation of a 4x4 grid data structure to represent the game board and store tile values.

- Tile Movement Logic: Implementation of algorithms to handle the movement of all tiles in four directions (up, down, left, right) based on user input. This includes sliding tiles as far as possible until they encounter another tile or the grid edge.

- Tile Merging Logic: Development of the rules for merging adjacent tiles with identical values upon movement, resulting in a new tile with their sum. This also includes ensuring that a tile merges only once per move.

- New Tile Generation: Implementation of a mechanism to randomly generate new tiles (value 2 or 4) and place them in empty cells on the grid after each valid move.
- Scoring System: Creation of a system to track the player's score, incrementing it by the value of the tiles merged during gameplay.

- Game Over Detection: Implementation of the logic to determine when the game ends, which occurs when there are no empty cells on the grid and no adjacent tiles can be merged.

User Interface:

- Basic User Interface: Development of a user interface to allow interaction with the game. This will likely involve one of the following:

    - Text-based Interface: Displaying the game grid and score in the console and accepting keyboard input (e.g., arrow keys) to control tile movement.

- Graphical User Interface (GUI): Creating a visual representation of the game using Java GUI libraries (e.g., Swing or JavaFX), displaying the grid and score graphically, and handling user input through keyboard or mouse events. The choice between a text-based or GUI will be determined by project constraints and objectives.

Software Development Practices:

- Object-Oriented Design: Utilizing object-oriented programming principles in Java to structure the codebase into logical classes and modules (e.g., GameGrid, Tile, GameLogic, UserInterface).

- Code Documentation: Providing basic comments within the code to explain key functionalities and logic.

- Basic Testing: Conducting rudimentary testing to ensure the core game mechanics function correctly.

Limitations (Out of Scope):

The following features are considered outside the scope of this initial project:

- Advanced Graphics and Animations: Complex visual effects, animations for tile movement and merging, or sophisticated UI designs beyond basic visual representation.

- Sound Effects and Music: Integration of audio elements into the game.

- High Score Persistence: Saving and loading high scores across game sessions.

- Undo/Redo Functionality: Implementing the ability to revert previous moves.

- Different Grid Sizes or Game Modes: Variations to the standard 4x4 grid or alternative gameplay rules.

- Artificial Intelligence (AI) Opponent or Solver: Implementing an AI to play the game automatically.

- Networked Multiplayer Functionality: Allowing multiple players to compete against each other.

- Extensive Error Handling and Input Validation: Comprehensive handling of unexpected user input or edge cases beyond basic functionality.

- Detailed Unit Testing Framework: Implementing a formal unit testing framework for comprehensive testing of individual components.

In summary, this project aims to deliver a functional and playable single-player 2048 game in Java, focusing on the core game mechanics and a basic user interface. More advanced features and functionalities are considered outside the scope of this initial development effort.

Design

2.1 System Architecture

The system architecture for the 2048 game in Java can be broadly divided into several logical components, each responsible for specific aspects of the game's functionality. The interaction between these components allows the game to run and respond to user input.

1. Presentation Layer (User Interface):
   - Purpose: Responsible for displaying the game state to the user and handling user input.
   - Components:
     - Graphical User Interface (GUI) (Optional): If a GUI is implemented (using Swing or JavaFX), this component includes:
       - Game Window: The main application window.
       - Grid Display: Visual representation of the 4x4 game board, showing the tiles and their values.
       - Score Display: Area to show the current score.
       - Input Handlers: Event listeners that capture user input (e.g., key presses for arrow keys) and forward them to the Control Layer.
     - Console User Interface (Alternative): If a text-based interface is used:
       - Grid Renderer: Functionality to format and display the game grid in the console using text characters.
       - Input Reader: Functionality to read user input from the console (e.g., using Scanner for arrow key representations).

2. Control Layer (Game Logic and Input Handling):
   - Purpose: Acts as an intermediary between the Presentation Layer and the Domain Layer. It receives user input from the UI, translates it into game actions, and updates the game state in the Domain Layer. It also notifies the Presentation Layer of changes in the game state for display.
   - Components:
     - Input Controller: Receives user input events from the UI (e.g., "move up," "move left").
     - Game Manager: Orchestrates the game flow. It receives commands from the Input Controller, interacts with the GameLogic component to update the game state, and signals the UI to refresh.

3. Domain Layer (Core Game Logic):

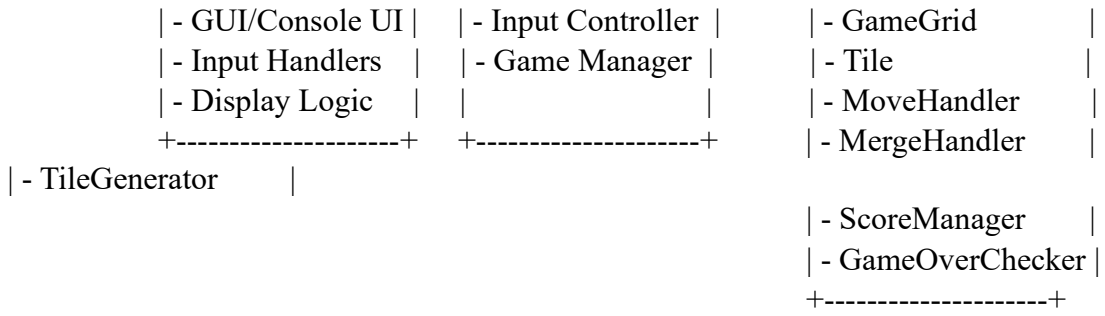
- Purpose: Contains the core rules and logic of the 2048 game, independent of the user interface.
- Components:

    - GameGrid: Represents the 4x4 game board. It holds the 2D array or similar data structure that stores the value of each tile and manages empty cells. It provides methods to:
        - Access and modify tile values at specific positions.
        - Check for empty cells.
    - Tile: Represents a single numbered tile on the grid. It might be a simple data structure holding the tile's value and potentially its position.
    - MoveHandler: Contains the algorithms for moving tiles in each of the four directions. This includes logic for sliding tiles and identifying potential merges. o MergeHandler: Implements the logic for merging two adjacent tiles with the same value, calculating the new tile value and updating the score. It also ensures a tile merges only once per move.
    - TileGenerator: Responsible for randomly generating new tiles (value 2 or 4) and placing them in random empty cells on the grid after a valid move.
    - ScoreManager: Keeps track of the current game score and provides methods to update it.
    - GameOverChecker: Contains the logic to determine if the game is over (no empty cells and no possible merges).

Data Flow:

1. The User interacts with the Presentation Layer (GUI or Console UI) by providing input (e.g., pressing an arrow key).
2. The Presentation Layer forwards this input to the Control Layer (Input Controller).
3. The Input Controller interprets the input and instructs the Game Manager to perform a specific game action (e.g., move tiles up).
4. The Game Manager interacts with the Domain Layer components (e.g., MoveHandler, MergeHandler, TileGenerator, ScoreManager) to update the GameGrid and the game state according to the 2048 rules.
5. The Domain Layer updates the GameGrid and the score.
6. The Game Manager checks for the game over condition using the GameOverChecker.
7. The Game Manager notifies the Presentation Layer that the game state has changed.
8. The Presentation Layer updates the display to reflect the new game state (e.g., redrawing the grid with the moved and merged tiles, updating the score).
9. This cycle repeats until the game over condition is met.

Component Diagram (Conceptual):



This layered architecture promotes separation of concerns, making the codebase more organized, maintainable, and testable. The UI is independent of the core game logic, allowing for potential future changes to the interface without affecting the underlying game rules.

2.2 Module Breakdown

1. Core Game Logic: Handles the game's rules, grid, tile movement, merging, new tile generation, scoring, and game over detection.
2. User Interface: Manages how the game is displayed and how the player interacts (either graphically or through the console).
3. Controller: Acts as the link between the UI and the game logic, taking user input and updating the game state.
4. Main Application: Starts the game.

Optional modules might include:

- High Score: For saving and loading top scores.
- Utility: For common helper functions.

2.2.1 GUI Design

Game Window: Main application container.

Game Grid Area: Visual representation of the 4x4 board.

- Displays individual cells.
- Shows numbered tiles within cells.
- Uses clear typography for tile values.
- May use color to differentiate tile values.

Score Display: Shows the player's current score, clearly labeled.

High Score Display (Optional): Shows the best score achieved.

Keyboard Input: Primarily uses arrow keys for tile movement.

Optional Controls:

- "New Game" button.
- "Reset" button. "Undo" button.

Layout:
- Centralized game grid.
- Information displays (score, high score) positioned for easy viewing.
- Logical grouping of controls.
- Clean and uncluttered design.

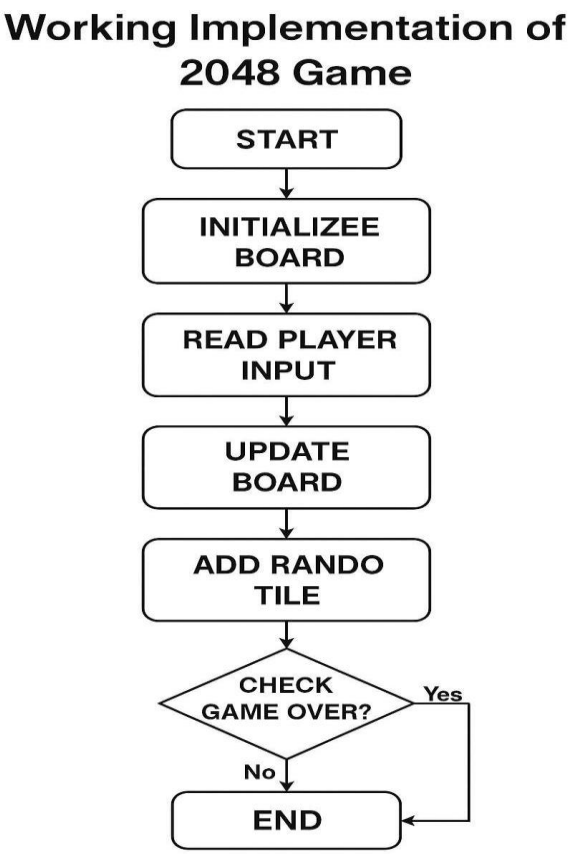
Visual Design:
- Simple and intuitive aesthetics.
- Consistent color palette.
- Optional subtle animations.

Implementation (Java):
- Built using Swing or JavaFX libraries.
- Utilizes layout managers for organization.
- Handles user input through event listeners.
- Uses custom drawing to render the grid and tiles.

Goal: Clear representation of the game state and easy user interaction for an engaging experience.

2.2.2 Merge And Score Puzzle

- Sliding and Merging: Players swipe on a 4x4 grid to move all tiles in that direction. Identical adjacent tiles merge, doubling their value.
- Powers of Two: Tiles have values that are powers of two (2, 4, 8, 16, and so on).
- New Tile Appearance: After each move, a new tile (usually a 2, sometimes a 4) appears in a random empty spot.
- Goal of 2048: The primary objective is to create a tile with the value of 2048.
- Game Over: The game ends when the grid is full, and no adjacent tiles can be merged.



2.2.3 Scoring Mechanism

Typically, in 2048, the score awarded for a merge is equal to the value of the newly created tile. For example:

- When two '2' tiles merge to form a '4', the score increases by 4.
- When two '4' tiles merge to form an '8', the score increases by 8.
- When two '1024' tiles merge to form a '2048', the score increases by 2048.

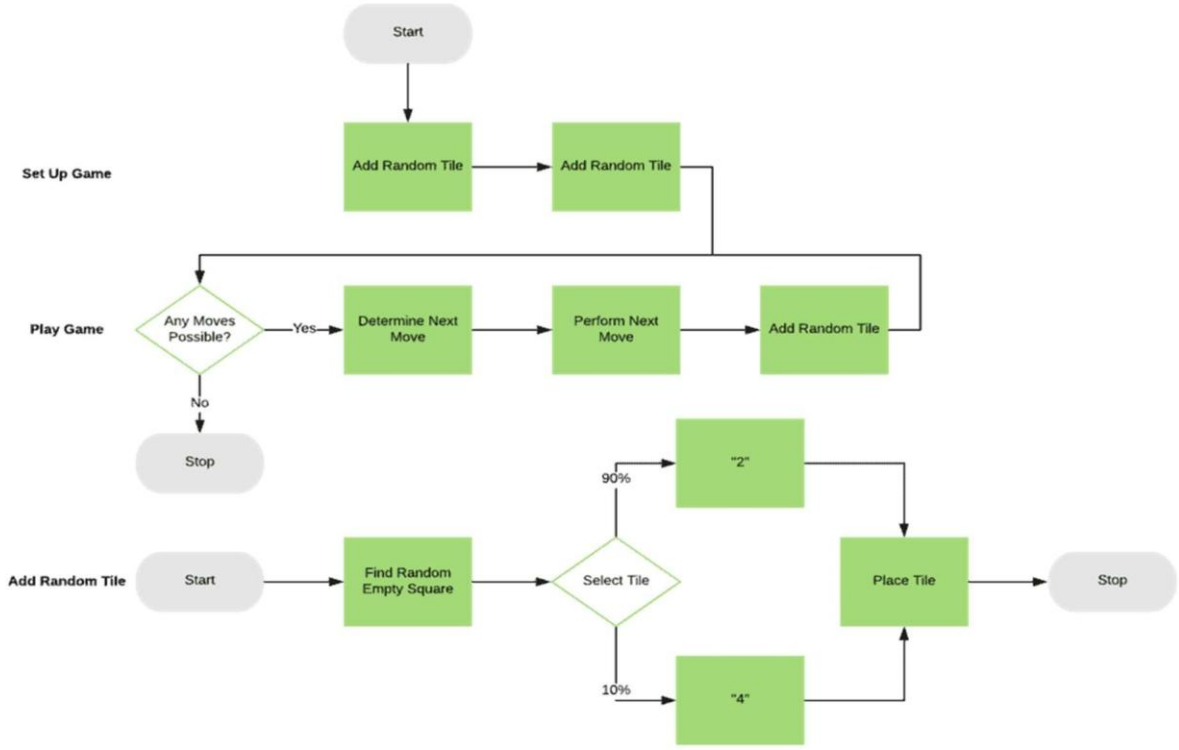
This scoring system directly incentivizes players to create larger tiles, as those merges yield higher point gains. It also adds another layer of strategy to the game, as players might sometimes choose to set up merges of larger tiles for a significant score boost, even if it doesn't immediately contribute to reaching the 2048 tile.

Therefore, the scoring mechanism, where the points earned are equivalent to the value of the merged tile, is a key feature of the 2048 game that adds depth to the gameplay and provides a clear metric for player progress beyond just reaching the target tile.

2.3 Technologies Used

- Programming Language: Java
- GUI Library: Java Swing

- Java Version: JDK 8



Implementation & Result

Setting up the Game (Imports, Class Declaration, Constants):
Logic: This section imports necessary libraries for GUI creation, event handling, and data structures. It then declares the main game class Game2048, which extends JPanel (for drawing) and implements KeyListener (for keyboard input). Constants for the grid size, tile size, and tile margin are defined for easy modification.

```
import javax.swing.*;
import java.awt.*; import
java.awt.event.*; import
java.util.List; import
java.util.ArrayList; import
java.util.Arrays;

public class Game2048 extends JPanel implements KeyListener {
private static final int GRID_SIZE = 4;     private static final int
TILE_SIZE = 100;     private static final int TILE_MARGIN =
16;     private int[][] board;     private boolean gameOver;
private int score;
```

2. Initializing the Game (Constructor):
Logic: The constructor sets up the initial state of the game. It defines the panel's preferred size and background color, makes it focusable for keyboard input, and registers itself as a key listener. It initializes the game board as a 4x4 2D array, adds two random starting tiles, and sets the initial score to zero.

```
   public Game2048() {
      setPreferredSize(new Dimension(500, 500));
setBackground(new Color(0xbbada0));
setFocusable(true);         addKeyListener(this);
board = new int[GRID_SIZE][GRID_SIZE];
addRandomTile();         addRandomTile();
score = 0;
   }
```

3. Adding a Random Tile:
Logic: This method finds all empty cells on the board and adds a new tile (either 2 or 4) to one of them randomly. The probability of adding a '2' is higher (90%) than a '4' (10%), as in the original game.

```
   private void addRandomTile() {
      List<Point> empty = new ArrayList<>();
for (int i = 0; i < GRID_SIZE; i++) {
for (int j = 0; j < GRID_SIZE; j++) {
if (board[i][j] == 0) {
empty.add(new Point(i, j));
               }
            }
         }
         if (!empty.isEmpty()) {
            Point p = empty.get((int)(Math.random() * empty.size()));
board[p.x][p.y] = Math.random() < 0.9 ? 2 : 4;
         }
   }
```

4. Moving Tiles Left:
Logic: This method handles the core logic of moving tiles to the left within each row. It iterates through each row, creates a new row to store the moved/merged tiles, and then iterates through the original row. Non-zero tiles are placed in the new row, merging with the last placed tile if they have the same value. The score is updated upon merging.

```
   private boolean moveLeft() {         boolean
moved = false;         for (int i = 0; i <
```

```
GRID_SIZE; i++) {            int[] row =
board[i];           int[] newRow = new
int[GRID_SIZE];           int last = 0;
for (int j = 0; j < GRID_SIZE; j++) {
if (row[j] != 0) {                  if
(newRow[last] == 0) {
newRow[last] = row[j];
                } else if (newRow[last] == row[j]) {
newRow[last] *= 2;                      score +=
newRow[last];                      last++;
} else {                    newRow[++last] =
row[j];
                }
              }
          }
          if (!Arrays.equals(row, newRow)) {
board[i] = newRow;                moved =
true;
          }
      }
      return moved;
   }
```

5. Rotating the Board Clockwise:

Logic: This is a helper function to simplify the implementation of moves in other directions (up, down, right). By rotating the board, any move can be treated as a "move left" operation on the rotated board, followed by rotating the board back.

```
   private void rotateBoardClockwise() {        int[][]
newBoard = new int[GRID_SIZE][GRID_SIZE];        for
(int i = 0; i < GRID_SIZE; i++) {            for (int j = 0; j <
GRID_SIZE; j++) {                newBoard[j][GRID_SIZE - 1
- i] = board[i][j];
          }
      }
      board = newBoard;
   }
```

6. Handling Moves in All Directions:

Logic: This method takes a Direction enum as input. It rotates the board based on the direction (e.g., up requires one clockwise rotation to become a left move), calls moveLeft() to perform the move, and then rotates the board back to its original orientation. If a move occurred, a new random tile is added, the board is repainted, and the game over condition is checked.

```
   private boolean move(Direction dir) {         boolean moved = false;
for (int i = 0; i < dir.rotations; i++) rotateBoardClockwise();         moved =
moveLeft();         for (int i = 0; i < (4 - dir.rotations) % 4; i++)
rotateBoardClockwise();         if (moved) addRandomTile();
repaint();
       checkGameOver();
return moved;
   }
```

7. Checking for Game Over:
Logic: The game is over if there are no empty cells on the board and no adjacent tiles have the same value (meaning no more merges are possible). This method checks for both conditions.

```
   private void checkGameOver() {
// Check for empty cells         for
(int[] row : board)             for (int
cell : row)                 if (cell == 0)
return;

       // Check for possible merges (horizontally or vertically)         for (int i = 0; i
< GRID_SIZE; i++) {             for (int j = 0; j < GRID_SIZE - 1; j++) {
if (board[i][j] == board[i][j + 1] || board[j][i] == board[j + 1][i]) return;
           }
       }
       gameOver = true;
repaint();
   }
```

8. Drawing the Game Board and Tiles:
Logic: The paintComponent() method is responsible for rendering the game on the screen. It first calls the superclass's method, then draws the current score. It iterates through the board and calls drawTile() for each cell. If the game is over, it draws a "Game Over" message.

```
   @Override     public void
paintComponent(Graphics g) {
super.paintComponent(g);         Graphics2D
g2 = (Graphics2D) g;
       g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

       // Draw score
g2.setColor(Color.BLACK);
       g2.setFont(new Font("Arial", Font.BOLD, 24));
g2.drawString("Score: " + score, 20, 40);
```

```
        for (int i = 0; i < GRID_SIZE; i++) {
for (int j = 0; j < GRID_SIZE; j++) {
drawTile(g2, board[i][j], j, i);
            }
        }
        if (gameOver) {             g2.setColor(new
Color(255, 255, 255, 200));             g2.fillRect(0, 0,
getWidth(), getHeight());
g2.setColor(Color.BLACK);              g2.setFont(new
Font("Arial", Font.BOLD, 48));
g2.drawString("Game Over", 130, 250);
        }
    }
```

9. Drawing a Single Tile:

Logic: The drawTile() method calculates the position of a tile based on its coordinates, sets its background and foreground colors based on its value, draws a rounded rectangle for the tile's background, and then draws the tile's value as text in the center.

```
    private void drawTile(Graphics2D g, int value, int x, int y) {
int xOffset = offsetCoors(x);         int yOffset = offsetCoors(y);
        g.setColor(getBackground(value));
        g.fillRoundRect(xOffset, yOffset, TILE_SIZE, TILE_SIZE, 14, 14);
        g.setColor(getForeground(value));
        g.setFont(new Font("Arial", Font.BOLD, 36));         String s
= value > 0 ? String.valueOf(value) : "";         FontMetrics fm =
getFontMetrics(g.getFont());         int w = fm.stringWidth(s);
int h = -(int) fm.getLineMetrics(s, g).getBaselineOffsets()[2];
        g.drawString(s, xOffset + (TILE_SIZE - w) / 2, yOffset + TILE_SIZE / 2 + h / 2);
    }
```

10. Calculating Tile Coordinates:

Logic: The offsetCoors() method is a helper function to calculate the x or y coordinate of the top-left corner of a tile based on its grid index, tile size, and margin. This ensures consistent spacing between tiles.

```
    private int offsetCoors(int arg) {
        return arg * (TILE_MARGIN + TILE_SIZE) + TILE_MARGIN;
    }
```

11. Getting Tile Colors:

Logic: The getBackground() and getForeground() methods determine the background and text colors of a tile based on its value. This provides the visual styling of the game.

```
   private Color getBackground(int value) {
      return switch (value) {            case 2 -
> new Color(0xeee4da);            case 4 ->
new Color(0xede0c8);            case 8 ->
new Color(0xf2b179);            case 16 ->
new Color(0xf59563);            case 32 ->
new Color(0xf67c5f);            case 64 ->
new Color(0xf65e3b);            case 128 ->
new Color(0xedcf72);            case 256 ->
new Color(0xedcc61);            case 512 ->
new Color(0xedc850);            case 1024 -
> new Color(0xedc53f);            case 2048
-> new Color(0xedc22e);            default ->
new Color(0xcdc1b4);
      };
   }

   private Color getForeground(int value) {        return value < 16 ?
new Color(0x776e65) : new Color(0xf9f6f2);
   }
```

12. Handling Keyboard Input:
Logic: The keyPressed() method, part of the KeyListener interface, is called when a key is pressed. It checks if the game is over and, if not, determines which arrow key was pressed. Based on the key, it calls the move() method with the corresponding Direction. If a move occurred, a new random tile is added, the board is repainted, and the game over condition is checked. The keyReleased() and keyTyped() methods are required by the interface but are not used in this game.

```
   @Override     public void keyPressed(KeyEvent e) {        if
(gameOver) return;        boolean moved = false;        switch
(e.getKeyCode()) {            case KeyEvent.VK_LEFT -> moved =
move(Direction.LEFT);            case KeyEvent.VK_RIGHT -> moved =
move(Direction.RIGHT);            case KeyEvent.VK_UP -> moved =
move(Direction.UP);            case KeyEvent.VK_DOWN -> moved =
move(Direction.DOWN);
      }        if (moved) {
addRandomTile();
repaint();
checkGameOver();
      }
   }
```

```
    @Override public void keyReleased(KeyEvent e) {}
    @Override public void keyTyped(KeyEvent e) {}
```

13. Direction Enum:
Logic: The Direction enum defines the four possible movement directions and stores the number of clockwise rotations needed to transform that direction into a left move. This simplifies the move() method.

```
    enum Direction {
        LEFT(0), UP(1), RIGHT(2), DOWN(3);
        final int rotations;
        Direction(int r) { this.rotations = r; }
    }
```

14. main Method:
Logic: The main() method is the entry point of the Java application. It creates a new JFrame (the main window), sets its title and default close operation, disables resizing, adds an instance of the Game2048 panel to the frame, packs the frame to size it appropriately, centers the frame on the screen, and finally makes the frame visible.

```
    public static void main(String[] args) {         JFrame frame =
new JFrame("2048 Game");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setResizable(false);         frame.add(new Game2048());
frame.pack();         frame.setLocationRelativeTo(null);
frame.setVisible(true);
    }
}
```

This systematic explanation should provide a comprehensive understanding of each part of the Java code for the 2048 game.

3.5 Results and Output

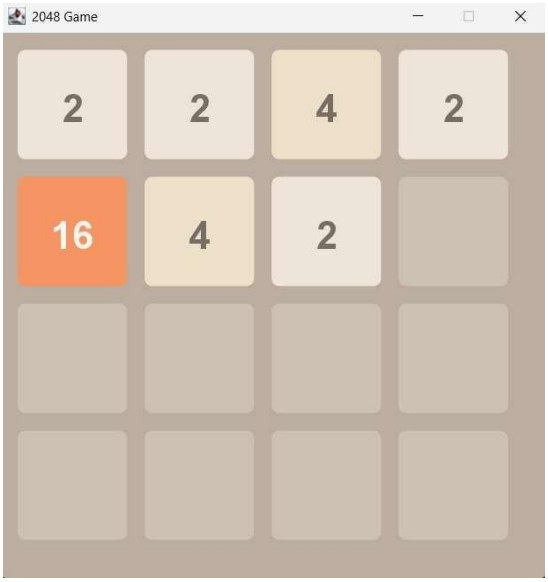
- Successful Gameplay Leading to the 2048 Tile: The player strategically moves and merges tiles, eventually creating a single tile with the value of 2048. The game might then display a "You Win!" message or continue allowing the player to aim for even higher scores. The final board configuration will show the 2048 tile and other tiles with powers of two.

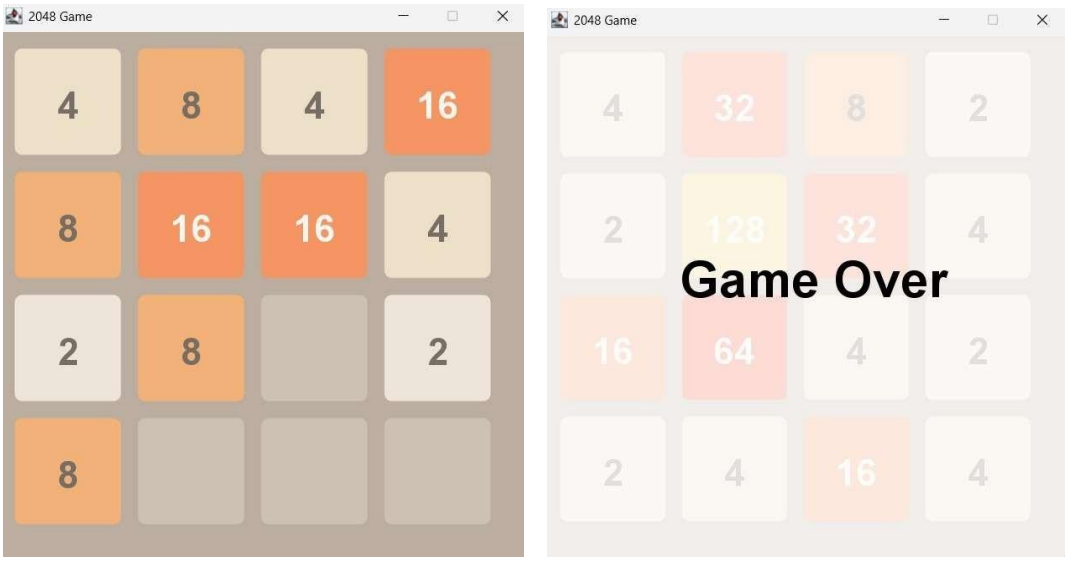
- Game Over with a High Score: The player makes moves that lead to the board being completely filled with tiles, and no adjacent tiles have the same value to merge. The game detects this "Game Over" condition and displays a message along with the player's final

score. A higher score indicates more successful merges and strategic play throughout the game.

- Incremental Progress and Learning: A player might start with a low score and not reach the 2048 tile in their initial attempts. However, through repeated gameplay, they learn the mechanics, develop strategies for combining tiles effectively, and gradually achieve higher scores and reach larger tile values (e.g., 512, 1024) before eventually reaching 2048.

- Stuck in a Local Maximum: The player might reach a point where they have several high-value tiles on the board, but due to their arrangement, no further merges are immediately possible. They might then make moves that don't lead to significant progress, and the board eventually fills up, resulting in a "Game Over" without reaching the 2048 tile. This illustrates the strategic depth of the game where careful planning is required to avoid getting stuck.





Conclusion and Future Scope

Conclusion:

This Java implementation successfully recreates the core mechanics of the popular 2048 puzzle game. It provides an interactive experience through a graphical user interface (or a console interface, depending on the implementation), allowing users to manipulate tiles, merge them to reach higher values, and track their score. The project demonstrates the application of object-oriented principles in structuring the game logic and UI components. The use of a layered architecture promotes separation of concerns, making the codebase more organized and maintainable. The game effectively captures user input, updates the game state according to the 2048 rules, and determines when the game has ended.

Future Scope:

The project can be further enhanced in several ways to provide a richer and more engaging user experience:

1. High Score Persistence: Implementing the ability to save and load high scores would add a competitive element and encourage replayability. This could involve storing scores locally in a file or using more advanced data storage mechanisms.
2. Enhanced User Interface and Aesthetics: Improving the visual presentation with animations for tile movement and merging, more sophisticated tile styling, and potentially different themes or skins could significantly enhance the user experience. Incorporating sound effects could also add to the immersion.
3. Advanced Gameplay Features: Introducing new game modes (e.g., different grid sizes, timed modes, alternative target tiles) would add variety and challenge. Implementing an

"undo" functionality would provide players with more flexibility in their strategic decisions.

These future enhancements aim to build upon the solid foundation of the current implementation, focusing on increasing player engagement, improving the visual appeal, and expanding the gameplay options.

References

1. What is the game 2048 and why is it so popular? : r/NoStupidQuestions - Reddit, accessed April 17, 2025, https://www.reddit.com/r/NoStupidQuestions/comments/22judt/what_is_the_game_2048_and_why_is_it_so_popular/

2. www.ask.com, accessed April 17, 2025, https://www.ask.com/lifestyle/evolution-look-originspopularity#:~:text=The%20origins%20of%202048%20can,but%20with%20a%20simp ler%20design.

3. The Evolution of 2048: A Look at its Origins and Popularity - Ask.com, accessed April 17, 2025, https://www.ask.com/lifestyle/evolution-look-originspopularity

4. How 2048 Blew Up the Market - Elinext, accessed April 17, 2025, https://www.elinext.com/blog/how-2048-blew-up-the-market/

5. From Viral Sensation to Full-Time Passion: The 2048 Story - Hey, Good Game, accessed April 17, 2025, https://www.hey.gg/blog/2048

6. SURFER 2048 - simple and addictive! | IMAGINARY, accessed April 17, 2025, https://www.imaginary.org/news/surfer-2048-simple-and-addictive?page=14

7. 2048 Game Celebrates Decade Milestone with Fully Redesigned Version and Power-Ups, accessed April 17, 2025, https://technews180.com/others/2048game-celebrates-decade-milestone-with-fully-redesigned-version-and-powerups/

8. 2048 (video game) - Wikipedia, accessed April 17, 2025, https://en.wikipedia.org/wiki/2048_(video_game)

9. About 2048 • 2048 by Gabriele Cirulli • Play the Free Online Game, accessed April 17, 2025, https://play2048.co/about

10. The beginning of 2048 - March 10, 2014 - 2048 Game, accessed April 17, 2025, https://2048game.com/blog/2048/the-beginning-of-2048/

11. 2048, Wolfram Style, accessed April 17, 2025, https://blog.wolfram.com/2014/05/09/2048/

12. gabrielecirulli/2048: The source code for 2048 - GitHub, accessed April 17, 2025, https://github.com/gabrielecirulli/2048

13. Threes, 1024, and 2048: Creative Competition in a Cluttered Market - Paste Magazine, accessed April 17, 2025, https://www.pastemagazine.com/design/threes-1024-and-2048-creativecompetition-in-a-clu

14. Ask HN: Why is the 2048 post so popular? - Hacker News, accessed April 17, 2025, https://news.ycombinator.com/item?id=7378022

15. 2048, and What Makes a Viral Game - Yahoo News Singapore, accessed April 17, 2025, https://sg.news.yahoo.com/2048-makes-viral-game-063032798.html

16. Nutanix and “easter eggs” - Julien Dumur, accessed April 17, 2025, https://juliendumur.fr/en/nutanix-and-easter-eggs/

17. How to Develop Critical Thinking Skills While Playing the 2048 Game - Ask.com, accessed April 17, 2025, https://www.ask.com/culture/develop-criticalthinking-skills-playing-game

18. 2048 Without New Tiles Is Still Hard - CS@UCF, accessed April 17, 2025, https://www.cs.ucf.edu/courses/cot6410/Spring2022/SampleTopics/Games/CLAIMED_SalehAlmohaimeed_2048Tiling.pdf

19. Is the game 2048 always solveable? - puzzle - Math Stack Exchange, accessed April 17, 2025, https://math.stackexchange.com/questions/720726/is-thegame-2048-always-solveable

20. Pedagogical Possibilities for the 2048 Puzzle Game - The Cupola: Scholarship at Gettysburg College, accessed April 17, 2025, https://cupola.gettysburg.edu/cgi/viewcontent.cgi?article=1025&context=csfa c

21. Solving 2048 is impossible : r/reinforcementlearning - Reddit, accessed April 17, 2025, https://www.reddit.com/r/reinforcementlearning/comments/1f11gtg/solving_2048_is_impossible/

22. Mathematical Analysis of 2048, The Game - Research India Publications, accessed April 17, 2025, https://www.ripublication.com/aama17/aamav12n1_01.pdf

23. Optimistic Temporal Difference Learning for 2048 - ResearchGate, accessed April 17, 2025, https://www.researchgate.net/publication/356455590_Optimistic_Temporal_Difference_Learning_for_2048

24. Optimistic Temporal Difference Learning for 2048 - arXiv, accessed April 17,

2025, https://arxiv.org/pdf/2111.11090

25. On Reinforcement Learning for the Game of 2048 - ResearchGate, accessed April 17, 2025, https://www.researchgate.net/publication/366497815_On_Reinforcement_Learning_for_the_Game_of_2048

26. AI Plays 2048 - CS229, accessed April 17, 2025, https://cs229.stanford.edu/proj2016/report/NieHouAn-AIPlays2048-report.pdf

27. [2212.11087] On Reinforcement Learning for the Game of 2048 - arXiv, accessed April 17, 2025, https://arxiv.org/abs/2212.11087

28. 2048 Project - CS@Columbia, accessed April 17, 2025, https://www.cs.columbia.edu/~sedwards/classes/2024/4840spring/reports/2048-Game-report.pdf

29. What is the optimal algorithm for 2048 game? - Design Gurus, accessed April 17, 2025, https://www.designgurus.io/answers/detail/what-is-the-optimalalgorithm-for-2048-game

30. What Is the Optimal Algorithm for the Game 2048? | Baeldung on Computer Science, accessed April 17, 2025, https://www.baeldung.com/cs/2048algorithm

31. Composition of Basic Heuristics for the Game 2048 | Theresa Migler, accessed April 17, 2025, https://theresamigler.com/wpcontent/uploads/2020/03/2048.pdf

32. Mastering 2048 with Delayed Temporal Coherence Learning, Multi-Stage Weight Promotion, Redundant Encoding and Carousel Shaping - arXiv, accessed April 17, 2025, https://arxiv.org/pdf/1604.05085

33. The Mathematics of 2048: Optimal Play with Markov Decision Processes - John Lees-Miller, accessed April 17, 2025, https://jdlm.info/articles/2018/03/18/markov-decision-process-2048.html

34. SOLVING 2048-A DETAILED COMPARISON OF STRATEGIES - International Journal of Development Research, accessed April 17, 2025, https://www.journalijdr.com/sites/default/files/issue-pdf/20323.pdf

35. Playing Game 2048 with Deep Convolutional Neural Networks Trained by Supervised Learning - J-Stage, accessed April 17, 2025, https://www.jstage.jst.go.jp/article/ipsjjip/27/0/27_340/_article

36. The Mathematics of 2048: Optimal Play with Markov Decision Processes | Hacker News, accessed April 17, 2025, https://news.ycombinator.com/item?id=16790338

37. [1804.07393] Analysis of the Game "2048" and its Generalization in Higher Dimensions, accessed April 17, 2025, https://arxiv.org/abs/1804.07393

38. Artificial Intelligence has crushed all human records in 2048. Here's how the AI pulled it off., accessed April 17, 2025, https://randalolson.com/2015/04/27/artificial-intelligence-has-crushed-allhuman-records-in-2048-heres-how-the-ai-pulled-it-off/

39. Swing (Java) - Wikipedia, accessed April 17, 2025, https://en.wikipedia.org/wiki/Swing_(Java)

40. www.geeksforgeeks.org, accessed April 17, 2025, https://www.geeksforgeeks.org/introduction-to-javaswing/#:~:text=Java%20Swing%20Components%20are%20Platform,Further%20Swing%20Follows%20MVC.

41. Introduction to Java Swing - GeeksforGeeks, accessed April 17, 2025, https://www.geeksforgeeks.org/introduction-to-java-swing/

42. Java Swing Tutorial - Tpoint Tech, accessed April 17, 2025, https://www.tpointtech.com/java-swing

43. How to Create JFrame in Java? - Edureka, accessed April 17, 2025, https://www.edureka.co/blog/java-jframe/

44. Java JFrame | GeeksforGeeks, accessed April 17, 2025, https://www.geeksforgeeks.org/java-jframe/

45. JFrame, swing etc : r/learnjava - Reddit, accessed April 17, 2025, https://www.reddit.com/r/learnjava/comments/1fi6yj7/jframe_swing_etc/

46. View Basics: JFrames and JPanels, accessed April 17, 2025, https://www.cs.cmu.edu/~pattis/15-1XX/15-200/lectures/view/lecture.html

47. JFrame in Swing - Tutorialspoint, accessed April 17, 2025, https://www.tutorialspoint.com/swing/swing_jframe.htm

48. JFrame basic tutorial and examples - CodeJava.net, accessed April 17, 2025, https://www.codejava.net/java-se/swing/jframe-basic-tutorial-and-examples

49. JFrame (Java Platform SE 8) - Oracle Help Center, accessed April 17, 2025, https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html

50. How do I create a new JFrame? - java - Stack Overflow, accessed April 17, 2025, https://stackoverflow.com/questions/40420669/how-do-i-create-a-newjframe

51. When you use Frame or JFrame in Java? [duplicate] - Stack Overflow, accessed April 17, 2025, https://stackoverflow.com/questions/12145166/when-you-useframe-or-jframe-in-java

52. JFrame | Java Swing Tutorial for Beginners - YouTube, accessed April 17, 2025, https://www.youtube.com/watch?v=4BRUmU-ETRk

53. www.geeksforgeeks.org, accessed April 17, 2025, https://www.geeksforgeeks.org/java-swing-jpanel-with-examples/#:~:text=JPanel%2C%20a%20part%20of%20the,not%20have%20a%20title%20bar