SPW 4 | **Test Automation & Continuous Delivery** | **ST 21, Exercise 2**

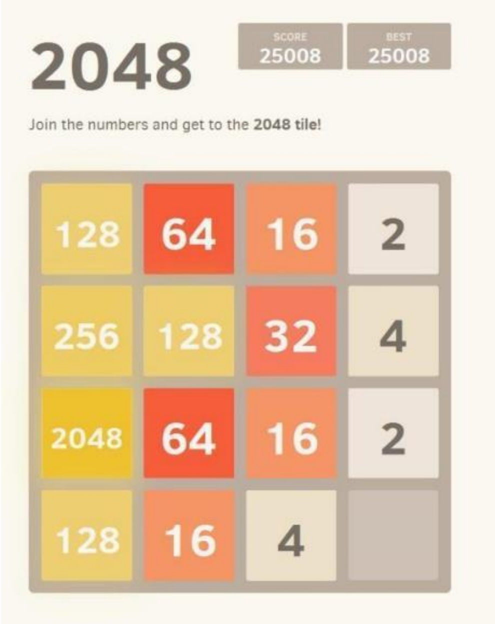
Deadline: 13.06.2022 13:00

Names Angelis Angelos, Bauer Maximilian

Points ________ **Effort in hours** 5

1. 2048 (18 + 6 Points)

The game *2048* (https://play2048.co) is a single-player sliding tile puzzle game. Its objective is to slide numbered tiles on a grid to combine them to larger numbers, eventually creating a tile with the value of 2048. Figure 1 shows a screenshot of the game.



2048

SCORE 25008

BEST 25008

Join the numbers and get to the **2048 tile!**

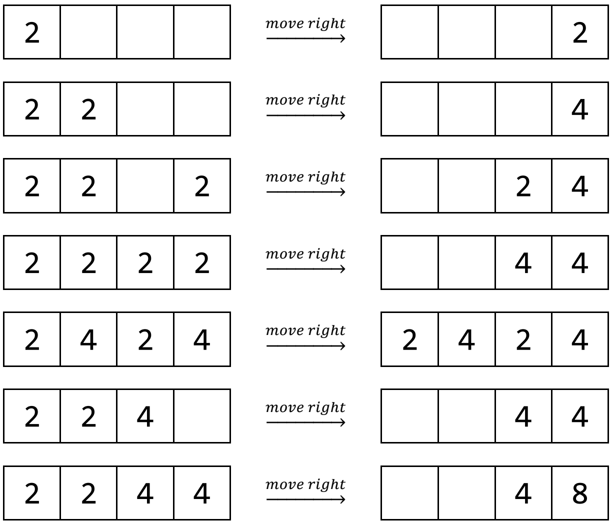
128	64	16	2
256	128	32	4
2048	64	16	2
128	16	4	

Figure 1: Screenshot of the original 2048 game

The rules of the game can be summarized as follows:

- The board consists of 16 tiles (4 x 4).
- When starting the game, the board is initialized with 2 randomly positioned tiles.
- The value of each new tile is randomly chosen from 2 and 4, whereby 2 has a higher probability of 90%.
- At each turn, the player can choose in which direction the tiles should be moved (either *up*, *down*, *left*, or *right*).
- All tiles move in the specified direction as far as they can.
- If two tiles with the same value touch they are merged, and the value is doubled. An already merged tile cannot be merged again in the same move. Tiles which are further in the direction of the move are merged first.
- Each time when two tiles are merged, the score is increased by the value of the merged tiles.
- After each move, a new tile is created at an empty position of the board, which is chosen at random. Again, the value of this new tile is either 2 (90% probability) or 4 (10% probability).
- If two tiles are merged to the value of 2048, the player wins the game.
- If no new tile can be created after a move because there are no empty positions left, the player loses the game.

The following examples illustrate how tiles are moved and merged in a single row when the player chooses to move to the right:



2				$\xrightarrow{move\ right}$				2
2	2			$\xrightarrow{move\ right}$				4
2	2		2	$\xrightarrow{move\ right}$			2	4
2	2	2	2	$\xrightarrow{move\ right}$			4	4
2	4	2	4	$\xrightarrow{move\ right}$	2	4	2	4
2	2	4		$\xrightarrow{move\ right}$			4	4
2	2	4	4	$\xrightarrow{move\ right}$			4	8

In Moodle you find an almost empty project for implementing a CLI version of the game. It contains only a *Main* class and a *main* method with the input loop, an enum *Direction* representing the 4 possible moves, and a skeleton of the *Game* class. Feel free to add additional classes and/or methods as needed.

The following console output shows an exemplary gameplay:

```
Moves: 0     Score: 0
.     .     .     2
.     .     .     .
.     .     .     .
.     .     .     2

command [w, a, s, d, (r)estart, (q)uit, (h)elp] > s

Moves: 1     Score: 4
.     .     .     .
.     .     .     .
.     .     .     .
2     .     .     4

command [w, a, s, d, (r)estart, (q)uit, (h)elp] > d

Moves: 2     Score: 4
2     .     .     .
.     .     .     .
.     .     .     .
.     .     2     4

command [w, a, s, d, (r)estart, (q)uit, (h)elp] > s

Moves: 3     Score: 4
.     .     .     .
.     .     2     .
.     .     .     .
2     .     2     4

command [w, a, s, d, (r)estart, (q)uit, (h)elp] > d
```

```
Moves: 4     Score: 8
.     .     .     .
.     .     .     2
2     .     .     .
.     .     4     4

command [w, a, s, d, (r)estart, (q)uit, (h)elp] > d

Moves: 5     Score: 16
.     .     .     .
2     .     .     2
.     .     .     2
.     .     .     8

command [w, a, s, d, (r)estart, (q)uit, (h)elp] > s

Moves: 6     Score: 20
.     .     .     .
.     .     .     .
.     .     4     4
2     .     .     8

command [w, a, s, d, (r)estart, (q)uit, (h)elp] > d

Moves: 7     Score: 28
.     .     .     .
.     .     .     .
.     .     .     8
2     .     2     8

command [w, a, s, d, (r)estart, (q)uit, (h)elp] >
```

Your tasks consists of two parts:

a) Pair up in teams of 2 and finish the implementation by applying *pair programming* and *test-driven development* (TDD). Focus on a strict TDD loop (write a failing test → make the test pass → refactor) and document your development process in your solution description.

b) After you finished your application, discuss how you experienced test-driven development and highlight your most important insights. Furthermore, critically analyze and review the design of your application and document your findings.

Verlauf

1) Wir haben uns entschieden, mit getMoves anzufangen und sind darauf gekommen, dass wir auf jeden Fall ein paar Variablen brauchen werden, und zwar folgende: „int score, int moves, int board, int Game_won“

2) Dann haben wir uns erstmal entschieden, mit der toString Methode anzufangen. Den Test, den wir uns ausgedacht haben, soll einen nicht leeren String zurückgeben (failed). Jetzt wird ToString implementiert. Wir sind darauf gekommen, dass das Board im Konstruktor initialisiert werden muss. Die Methode wurde dann erfolgreich implementiert.

3) Als Nächstes schauen wir uns die isWon Methode an. In TDT soll man ja am Anfang failed Tests implementieren. Bei isWon ist das in dem Fall ein bisschen schwierig. Um diese Methode und auch andere besser testen zu können haben wir uns entschieden den ctor zu überladen, um das Spiel einfacher in verschiedenen Spielständen testen zu können.

4) Nach dem ctor machen wir mit getMoves weiter. Der failed Test schlägt fehl, weil wir die game.move Methode noch nicht implementiert haben.

5) Wir haben gemerkt, dass eine initialize Methode existiert, um das Board zu initialisieren. Es wird refactored.

6) Es wird ein failed Test für move implementiert. Dafür brauchen wir die getValueAt Methode, weshalb wir als Erstes diese implementieren (und auch ein FailedTest dazu schreiben).

7) Nachdem getValue implementiert wurde, machen wir mit der move Methode weiter. Nach 2 Stunden haben wir die move Methode erfolgreich implementiert und der Testfall ist erfolgreich.

8) Wir haben die Tests für die Move Methode erweitert zu zwei parametrisierten Tests, die horizontal und vertikal das Board testen.

9) Als Nächstes haben wir einen Test für getScore() implementiert. Es wird mithilfe von move getestet und es wird auch wirklich überprüft, ob der Score richtig ist. Nach der Implementierung des Tests wird getScore implementiert.

10) Als Nächstes schreiben wir einen Test für isOver() in dem für ein Board, das in jedem Tile eine verschiedene Zahl hat, überprüft wird, ob das Spiel vorüber ist

11) Beim isWon Test haben wir das Board auf isOver überprüft, weil isOver, isWon beinhaltet.

Diskussion

- Es ist sehr gewöhnungsbedürftig
- Man muss diszipliniert herangehen
- Es ist sehr unpraktisch TDD für getter/setter und an sich eher kleinere Methoden anzuwenden
- Traditionelle Methoden gefallen uns mehr
- Mit TDD heißt nicht automatisch, dass man gute Testfälle schreibt, aber bei uns war dies der Fall
- TDD hat uns geholfen zu definieren, was jede Methode braucht.

Quelltext:

Main.java

```
package spw4.game2048;

import java.util.*;

public class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    String input;

    Game game = new GameImpl();
    game.initialize();
    System.out.println(game);

    while (!game.isOver()) {
      System.out.print("command [w, a, s, d, (r)estart, (q)uit, (h)elp] > ");
      input = scanner.nextLine();

      switch (input) {
        case "w":
          game.move(Direction.up);
          break;
        case "a":
          game.move(Direction.left);
          break;
        case "s":
          game.move(Direction.down);
          break;
        case "d":
          game.move(Direction.right);
          break;
        case "r":
          game.initialize();
          break;
        case "q":
          System.out.println("Ok, bye.");
          return;
        case "h":
          printHelp();
          break;
        default:
          System.out.println("Unknown command");
          break;
      }
      System.out.println(game);
    }
    System.out.println(game.isWon() ? "You win!!! :)" : "You lose. :(");
    System.out.println("Your score: " + game.getScore());
  }

  private static void printHelp() {
    System.out.println();
    System.out.println("Available commands:");
    System.out.println("-------------------");
    System.out.println("w --> move up");
    System.out.println("a --> move left");
    System.out.println("s --> move down");
    System.out.println("d --> move right");
    System.out.println("r --> restart game");
    System.out.println("q --> quit game");
    System.out.println("h --> show help");
  }
}
```

Game.java

```
package spw4.game2048;

public interface Game {
  void initialize();

  void move(Direction direction);

  int getMoves();

  int getScore();

  int getValueAt(int x, int y);

  boolean isOver();

  boolean isWon();
}
```

GameImpl.java

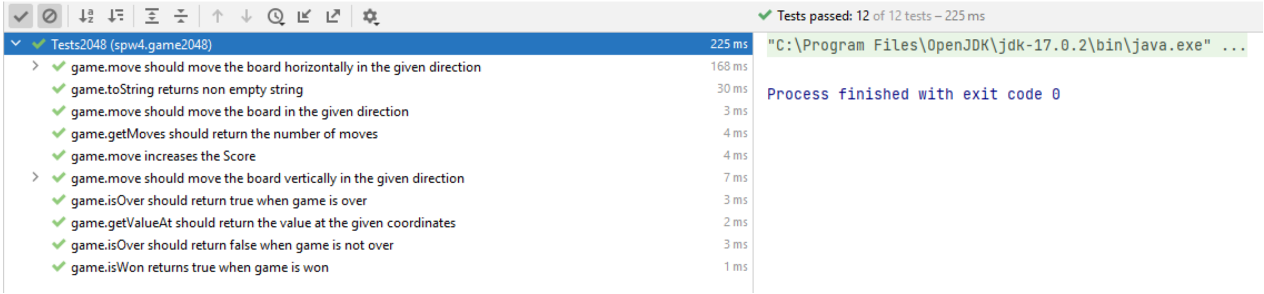
```java
package spw4.game2048;

import java.util.Arrays;
import java.util.Random;

public class GameImpl implements Game {
  private int score = 0;
  private int moves = 0;
  private int[][] board = new int[4][4];
  private final int GAME_WON = 2048;

  public GameImpl() {
  }

  public GameImpl(int[][] board) {
    this.board = board;
    this.score = score;
    this.moves = moves;
  }

  public int getMoves() {
    return moves;
  }

  public int getScore() {
    return score;
  }

  public int getValueAt(int x, int y) {
    return board[x][y];
  }

  public boolean isOver() {
    return !movesAvailable() || isWon();
  }

  public boolean isWon() {
    for (int i = 0; i < board.length; i++) {
      for (int j = 0; j < board[i].length; j++) {
        if (board[i][j] == GAME_WON) {
          return true;
        }
      }
    }

    return false;
  }

  @Override
  public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append(String.format("Moves: %d    Score: %d\n", moves, score));

    for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++) {
        if (board[i][j] == 0) {
          sb.append(".");
        } else {
          sb.append(board[i][j]);
        }
        sb.append("     ");
      }
      sb.append("\n");
    }

    return sb.toString();
  }

  public void initialize() {
    Arrays.stream(board).forEach(a -> Arrays.fill(a, 0));
    spawnTile();
    spawnTile();
  }

  public void move(Direction direction) {
    boolean horizontally = direction == Direction.left || direction ==
Direction.right;
    int row = 0;
    int col = 0;

    ++moves;

    for (int i = 0; i < board.length; i++) {
      int[] numbers = new int[4];
      for (int j = 0; j < board[i].length; j++) {
        row = horizontally ? i : j;
        col = horizontally ? j : i;

        if (horizontally) {
          numbers[col] = getValueAt(row, col);
        } else {
          numbers[row] = getValueAt(row, col);
        }
      }

      shiftTiles(numbers, direction);

      if (horizontally) {
        for (int j = 0; j < numbers.length; j++) {
          board[i][j] = numbers[j];
        }
      } else {
        for (int j = 0; j < numbers.length; j++) {
          board[j][i] = numbers[j];
        }
      }
    }
    spawnTile();
  }

  private void shiftTiles(int[] numbers, Direction direction) {
    int[] merged = {0, 0, 0, 0};
    int count = numbers.length - 1;

    while (count > 0) {
      // swipe left and up
      if (direction == Direction.left || direction == Direction.up) {
        for (int i = 0; i < numbers.length - 1; ++i) {
          if (numbers[i] == 0) {
            if (merged[i + 1] == 1) {
              merged[i] = 1;
              merged[i + 1] = 0;
            }

            numbers[i] = numbers[i + 1];
            numbers[i + 1] = 0;
          } else if (numbers[i + 1] == numbers[i] && merged[i] != 1 &&
merged[i + 1] != 1) {
            numbers[i] *= 2;
            numbers[i + 1] = 0;
            merged[i] = 1;
            score += numbers[i];
            break;
          }
        }
      } else {
        // swipe right and down
        for (int i = numbers.length - 1; i > 0; --i) {
          if (numbers[i] == 0) {
            if (merged[i - 1] == 1) {
              merged[i] = 1;
              merged[i - 1] = 0;
            }

            numbers[i] = numbers[i - 1];
            numbers[i - 1] = 0;
          } else if (numbers[i - 1] == numbers[i] && merged[i] != 1 &&
merged[i - 1] != 1) {
            numbers[i] *= 2;
            numbers[i - 1] = 0;
            merged[i] = 1;
            score += numbers[i];
            break;
          }
        }
      }
      count--;
    }
  }

  private boolean movesAvailable() {
    for (int i = 0; i < board.length; i++) {
      for (int j = 0; j < board[i].length; j++) {
        if (board[i][j] == 0) {
          return true;
        }
      }
    }
    for (int i = 0; i < board.length - 1; i++) {
      for (int j = 0; j < board[i].length - 1; j++) {
        if (board[i][j] == board[i][j + 1] || board[i][j] == board[i + 1][j])
{
          return true;
        }
      }
    }

    return false;
  }

  private void spawnTile() {
    int x = (int) (Math.random() * 4);
    int y = (int) (Math.random() * 4);

    while (board[x][y] != 0) {
      x = (int) (Math.random() * 4);
      y = (int) (Math.random() * 4);
    }

    int min = 0;
    int max = 100;
    Random random = new Random();

    int value = random.nextInt(max + min) + min;

    if (value < 90) {
      board[x][y] = 2;
    } else {
      board[x][y] = 4;
    }
  }
}
```

Direction.java

```
package spw4.game2048;

public enum Direction {
  up,
  down,
  left,
  right
}
```

Tests2048.java

```
package spw4.game2048;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;

import static org.junit.jupiter.api.Assertions.*;

public class Tests2048 {
  @DisplayName("game.getMoves should return the number of moves")
  @Test
  void getMovesShouldReturnNumberOfMoves() {
    Game game = new GameImpl();
    game.initialize();
    game.move(Direction.up);
    game.move(Direction.left);
    game.move(Direction.down);
    game.move(Direction.right);

    assertEquals(4, game.getMoves());
  }

  @DisplayName("game.move increases the Score")
  @Test
  void moveIncreasesScore() {
    int[][] board = new int[][] {
        {2, 2, 0, 2},
        {0, 0, 0, 0},
        {0, 0, 4, 0},
        {0, 2, 0, 0}
    };
    Game game = new GameImpl(board);
    game.move(Direction.up);


    assertEquals(4, game.getScore());
  }

  @DisplayName("game.toString returns non empty string")
  @Test
  void toStringReturnsNonEmptyString() {
    int[][] board = new int[][] {
        {2, 2, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0}
    };
    GameImpl sut = new GameImpl(board);

    assertNotEquals("", sut.toString());
  }

  @DisplayName("game.isWon returns true when game is won")
  @Test
  void isWonReturnsTrueWhenGameIsWon() {
    int[][] board = new int[][] {
        {4, 2, 2, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 2048}
    };
    GameImpl sut = new GameImpl(board);

    assertTrue(sut.isWon());
  }

  @DisplayName("game.move should move the board in the given direction")
  @Test
  void moveShouldMoveTheBoard() {
    int[][] result = new int[][]{
        {4, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0}
    };

    int[][] board = new int[][] {
        {2, 2, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0}
    };
    GameImpl sut = new GameImpl(board);

    sut.move(Direction.left);

    assertEquals(result[0][0], sut.getValueAt(0, 0));
  }

  @DisplayName("game.move should move the board horizontally in the given 
direction")
  @ParameterizedTest(name = "direction = {0}, col = {1}, row = {2}, result = 
{3}")
  @CsvSource({"left, 0, 0, 8", "right, 3, 0, 4"})
  void moveShouldMoveTheBoardHorizontally(Direction direction, int col, int 
row, int result) {
    int[][] board = new int[][] {
        {4, 4, 2, 2},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0}
    };
    GameImpl sut = new GameImpl(board);

    sut.move(direction);

    assertEquals(result, sut.getValueAt(row, col));
  }

  @DisplayName("game.move should move the board vertically in the given
direction")
  @ParameterizedTest(name = "direction = {0}, col = {1}, row = {2}, result =
{3}")
  @CsvSource({"up, 0, 0, 8", "down, 0, 3, 4"})
  void moveShouldMoveTheBoardVertically(Direction direction, int col, int
row, int result) {
    int[][] board = new int[][] {
        {4, 0, 0, 0},
        {4, 0, 0, 0},
        {2, 0, 0, 0},
        {2, 0, 0, 0}
    };
    GameImpl sut = new GameImpl(board);

    sut.move(direction);

    assertEquals(result, sut.getValueAt(row, col));
  }

  @DisplayName("game.getValueAt should return the value at the given
coordinates")
  @Test
  void getValueAtReturnsValueAtGivenCoordinates() {
    int[][] board = new int[][] {
        {2, 2, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0}
    };
    GameImpl sut = new GameImpl(board);

    assertEquals(2, sut.getValueAt(0, 0));
  }

  @DisplayName("game.isOver should return true when game is over")
  @Test
  void isOverShouldReturnTrueWhenGameIsOver() {
    int[][] board = new int[][] {
        {2, 4, 8, 16},
        {32, 64, 128, 256},
        {512, 1024, 2, 4},
        {8, 16, 32, 64}
    };
    GameImpl sut = new GameImpl(board);

    assertTrue(sut.isOver());
  }

  @DisplayName("game.isOver should return false when game is not over")
  @Test
  void isOverShouldReturnFalseWhenGameIsNotOver() {
    int[][] board = new int[][] {
        {4, 2, 2, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 64}
    };
    GameImpl sut = new GameImpl(board);

    assertFalse(sut.isOver());
  }
}
```



✓ Tests passed: 12 of 12 tests – 225 ms

```
✓ Tests2048 (spw4.game2048)                                              225 ms
  ✓ game.move should move the board horizontally in the given direction  168 ms
  ✓ game.toString returns non empty string                                30 ms
  ✓ game.move should move the board in the given direction                 3 ms
  ✓ game.getMoves should return the number of moves                        4 ms
  ✓ game.move increases the Score                                          4 ms
  ✓ game.move should move the board vertically in the given direction      7 ms
  ✓ game.isOver should return true when game is over                       3 ms
  ✓ game.getValueAt should return the value at the given coordinates       2 ms
  ✓ game.isOver should return false when game is not over                  3 ms
  ✓ game.isWon returns true when game is won                               1 ms
```

```
"C:\Program Files\OpenJDK\jdk-17.0.2\bin\java.exe" ...

Process finished with exit code 0
```