Assignment 4 Specification

Travis Moore

April 12, 2021

This Module Interface Specification (MIS) document contains modules, types and methods used to support implementing the game 2048. The goal of the game is to get the highest score possible or get the tile 2048. You get score by combining tiles with the same number that are next to each other. Every time you connect two tiles into one, the number from the resulting tile will be added to your score. You have four move options. You can either shift all tiles up, down, left or right. A new tile in a randomly generated location will be added every time you successfully move the tiles on the board. The game can be launched in the terminal by typing **make expt** in the A4 folder.

Likely Changes

- Change of the Board Size. Almost every function that worked with the size of the board used the state variable *size* instead of hard coding the current size of the board (4x4).
- My design considers the likely change of the visual display.
- My design considers change of different inputs that the controller will react to.

Overview of The Design

This design uses the Module View Controller (MVC) design pattern, and Singleton design pattern. Controller, BoardT (model module), and UserView (view module) are the following MVC components. The Singleton pattern is implemented in the UserView and Controller modules. Both of those classes can only have one instance. getInstance() has to be called to create the object for both of those classes.

The MVC pattern is implemented by allowing the BoardT module to store the state and current status of the game. UserView is the view module that displays the state of the game board using text-based graphics. Finally, the Controller module is the controller that handles input from the user and connects BoardT with UserView.

UserView Module

Module

UserView

Uses

BoardT

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
getInstance		UserView	
printWelcomeMessage			
printEndingMessage	BoardT		
printBoard	BoardT		
printBoard2	BoardT		

Semantics

Environment Vairables

window: A terminal that will display the game to the computer screen.

State Variables

visual: UserView

State Invariant

None

Assumptions

- The UserView constructor is called before any other access routine is called for a UserView object.

Access Routine Semantics

getInstance():

- transition: visual := visual = null *implies* new UserView()
- out: self
- exception: none

printWelcomeMessage():

- transition: window := Welcome to 2048 message is displayed above board during game.
- exception: none

printEndingMessage(board):

- transition: window := Game over message is displayed, showing the final score.
- exception: none

printBoard(board):

- transition: window := The game board is displayed to the window. Cells are accessed using the getCell method from BoardT. Any cell with the number 0 will not be printed to the screen, instead a blank space will be printed. The terminal is initially cleared to allow easier transitions from every printBoard call.
- exception: none

printBoard2(board):

- transition: window := Similar to printBoard however terminal is not intially cleared which allows the previous board to be shown. Used for when the game is over to show the final state of the game board.
- exception: none

Local Functions

UserView: void $\rightarrow$ UserView
Userview() $\equiv$ new UserView()

Board ADT Module

Module

BoardT

Uses

Services

Syntax

Exported Constants

None

Exported Types

size = 4 // Board size is 4x4

Exported Access Programs

Routine name	In	Out	Exceptions
new BoardT		BoardT	
getCell	$x : \mathbb{Z}$, $y : \mathbb{Z}$	$\mathbb{Z}$	
getScore		$\mathbb{Z}$	
addTile			
gameOver		$\mathbb{B}$	
moveRight			
moveLeft			
moveUp			
moveDown			
setTile	x: $\mathbb{R}$, y: $\mathbb{R}$, number: $\mathbb{Z}$		

Semantics

State Variables

board: Seq of [size, size] of $\mathbb{R}$
score: $\mathbb{Z}$

State Invariant

None

Assumptions

- Size will be a positive number.
- The constructor for BoardT will be called first for each BoardT object before any other access routine is called.
- Assume there is a random function that generates a random number between 0 and 1.
- AddTile() will never be called when the Grid is full with numbers greater than 0.

Access Routine Semantics

new BoardT():

- transition:

  board := $\langle \begin{matrix} \langle 0,0,0,0 \rangle \\ \langle 0,0,0,0 \rangle \\ \langle 0,0,0,0 \rangle \\ \langle 0,0,0,0 \rangle \end{matrix} \rangle \wedge$ addTile() $\wedge$ addTile()

  // two tiles randomly added after

- output: out := self
- exception: none

getCell(x,y):

- transition: none
- output: out := board[x][y]
- exception: none

getScore():

- transition: none
- output: out := score

- exception: none

setTile(x,y,number):

- transition: board[x][y] := number
- output: none
- exception: none

addTile():

- transition: (board[newCellLocation()][newCellLocation()] = 0) $\implies$ board[newCellLocation()][newCellLocation()] := newTileNumber()|True $\implies$ addTile() *// Find a empty tile to put new value in*
- out: none
- exception: none

gameOver():

- transition: none
- out: $\neg\exists(\forall i : \mathbb{N}|i < size \wedge (\forall j : \mathbb{N}|j < size-1) : board[i][j] = board[j][i] \vee board[i][j+1] = board[i][j+1] \vee board[i][j] = 0)$ *// continue game If a cell has 0 or two cell values equal to each other are next to each other*
- exception: none

moveRight():

- transition: board := $\forall i \in [0...size-1]$: shiftRowRight(i) $\wedge$ $\forall j \in [0...size-1]$: board[i][j] = board[i][j-1] $\implies$ score := score + board[i][j]*2 $\wedge$ board[i][j] := board[i][j] + board[i][j-1] $\wedge$ board[i][j-1] := 0 $\wedge$ addTile() *// Move all tiles right, add score if they match and are next to each other, add new tile after*
- out: none
- exception: none

moveLeft():

- transition: board := $\forall i \in [size-1...0]$: shiftRowLeft(i) $\wedge$ $\forall j \in [0...size-1]$: board[i][j] = board[i][j-1] $\implies$ score := score + board[i][j]*2 $\wedge$ board[i][j] := board[i][j] + board[i][j-1] $\wedge$ board[i][j-1] := 0 $\wedge$ addTile()

- out: none
- exception: none

moveUp():

- transition: board := $\forall i \in [size-1...0]$: shiftColumnUp(i) $\wedge$ $\forall j \in [0...size-1]$: board[j][i] = board[j+1][i] $\implies$ score := score + board[j][i]*2 $\wedge$ board[j][i] := board[j+1][i] + board[j][i] $\wedge$ board[j+1][i] := 0 $\wedge$ addTile()

moveDown():

- transition: board := $\forall i \in [size-1...0]$: shiftColumnUp(i) $\wedge$ $\forall j \in [size-1...0]$: board[j][i] = board[j-1][i] $\implies$ score := score + board[j][i]*2 $\wedge$ board[j][i] := board[j+1][i] + board[j][i] $\wedge$ board[j-1][i] := 0 $\wedge$ addTile()

Local functions

shiftRowRight: $\mathbb{R} \to \mathbb{B}$
shiftRowRight(row) $\equiv \forall i \in |[0...size-1]$: shiftNumRight(row) = 1 $\implies$ True
// call shiftNumRight for every i value, if it returns 1 return true

shiftNumRight: $\mathbb{R} \to \mathbb{R}$
shiftNumRight(row) $\equiv \forall i \in [size-1...0]$: board[row][i] = board[row][i-1] $\implies$ 1 $\wedge$ board[row][i] := board[row][i-1] $\wedge$ board[row][i-1] := 0 |True $\implies$ -1
// Shift each number in the row right if possible, return 1 if a tile is shifted

shiftRowLeft: $\mathbb{R} \to \mathbb{B}$
shiftRowLeft(row) $\equiv \forall i \in [0...size-1]$: shiftNumLeft(row) = 1 $\implies$ True

shiftNumLeft(row): $\mathbb{R} \to \mathbb{R}$
shiftNumLeft(row) $\equiv \forall i \in [0...size-1]$: board[row][i] = board[row][i+1] $\implies$ 1 $\wedge$ board[row][i] := board[row][i+1] $\wedge$ board[row][i+1] := 0 |True $\implies$ -1

shiftColumnUp: $\mathbb{R} \to \mathbb{B}$
shiftColumnUp(row) $\equiv \forall i \in [0...size-1]$: shiftNumUp(row) = 1 $\implies$ True

shiftNumUp: $\mathbb{R} \rightarrow \mathbb{R}$
shiftNumUp(column) $\equiv \forall i \in |[0...size-1] :$ board[i][column] = board[i+1][column] $\implies$
$1 \wedge$ board[i][column] := board[i+1][column] $\wedge$ board[i+1[column] := 0 |True $\implies$ -1

shiftColumnDown: $\mathbb{R} \rightarrow \mathbb{B}$
shiftColumnUp(row) $\equiv \forall i \in [0...size-1] :$ shiftNumUp(row) = 1 $\implies$ True

shiftNumDown: $\mathbb{R} \rightarrow \mathbb{R}$
shiftNumDown(column) $\equiv \forall i \in [size-1...0] :$ board[i][column] = board[i-1][column] $\implies$
$1 \wedge$ board[i][column] := board[i-1][column] $\wedge$ board[i-1[column] := 0 |True $\implies$ -1

Services Module

Module

Services

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
newCellLocation		$\mathbb{R}$	
getName		String	

Semantics

State Variables

None

State Invariant

None

Assumptions

We have access to a function that generates a random number between 0 and 1.

Access Routine Semantics

newCellLocation():

- transition: none
- out: RandomNum $< 0.25 \rightarrow 0$ | RandomNum $< 0.5 \rightarrow 1$ | RandomNum $< 0.75 \rightarrow 2$ | 3
- exception: none

newTileNumber():

- transition: none
- out: RandomNum $< 0.90 \rightarrow 2$ | 4
- exception: none

Controller Module

Module

Controller

Uses

BoardT, UserView

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
getInstance	BoardT, UserView	Controller	
initializeGame			
showWelcomeMessage			
showEndMessage			
showBoard			
run2048			

Semantics

Environment Variables

scanner: Scanner(System.in)

State Variables

model: BoardT view: UserView controller: Controller

State Invariant

None

Assumptions

- The Controller constructor is called before any other access routine is called for a Controller object.
- UserView and Boardt instances have been created before calling the Controller constructor

Access Routine Semantics

getInstance(model,view):

- transition: controller := controller = null *implies* new controller(model,view)
- out: self
- exception: none

initializeGame():

- transition: model := new BoardT()
- out: none
- exception: none

showWelcomeMessage():

- transition: view := view.printWelcomeMessage
- out: none
- exception: none

showEndMessage():

- transition: view := view.printEndMessage
- out: none
- exception: none

showBoard():

- transition: view := view.printBoard(model)
- out: none
- exception: none

run2048():

- transition: Runs the game. Game starts with the initial board and welcome message. Then it gets input from the keyboard and executes the specific method bind to the input. "w" will call moveUp from BoardT, "s" will call moveDown from BoardT, "a" will call moveLeft from BoardT, and "d" will call moveRight from BoardT. "p" will end the game, allowing the user to quit earlier if needed to. Besides that, game will run until the player loses. Any other input will be rejected and a prompt asking for correct input will show. At the end of the game the final score and final board will be displayed.
- out: none
- exception: none

Critique of The Design

- I would say the design is mostly essential. However the *setTile*(x,y,num) in BoardT is not essential. I included this method because it made it easier to create high quality test cases. In reality it is not used in the actual game because I made the *addTile()* method as random as possible.
- I would say my Design is general, I accounted for the fact that there are many variations of 2048, the main variation being the size of the game board. The only method that would need to be changed if the size of the board is changed is the newCellLocation() function in Services. It was hard to make this function more generic as I was using percentages to calculate the index. However this could have been avoided If I implemented the addTile() method differently.
- I wouldn't say the design is fully minimal, as there are some access routine with independent services. Unfortunately I ran out of time to fix these methods.
- The design is very consistent, all naming conventions of the methods are the same, as well as the ordering of parameters in functions.
- The design has high cohesion, for example the Controller module works very closely with the UserView module and BoardT module to run the game.
- The design implements information hiding, for example state variables are set as private and local functions are private as well.
- The addTile() method in BoardT was poorly designed in my opinion. I was too focused on making it as random as possible. A better design would have been to keep track of which cells had a value of 0 and pick a random cell from that list, instead I was picking from every cell in the grid. This could lead to very bad runtime when the grid is really big and there are only a couple available spots to put a new number in.
- I could have added a better way of handling the end of the game. For example I could have added a option to restart the game instead of ending the program as soon as the game ends.
- I could have implemented BoardT using Singleton pattern, as I did with the controller and the UserView class.
- I think the design follows the rules of the game well, for example I made sure to test for certain cases where there are 4 of the same tiles in a row/column, resulting in four tiles being added in a same move in the same row or column.

- My design could use more checks to avoid unexpected exceptions, however I had a hard time seeing where exceptions could arise, so I did not have any checks in this design.

Questions

1.

A Code for A4Example.java

```
/**
 * file: A4Example.java
 * Author: Travis Moore (mooret12)
 * Date: April 12th, 2021
 * Description: Runs the game
 */

package src;

import java.util.Arrays;

public class A4Example
{
   public static void main(String[] args) {

         BoardT board = new BoardT();

         UserView view = UserView.getInstance();
         Controller game = Controller.getInstance(board, view);

         game.run2048();

    }

}
```

B Code for AllTests.java

```
/**
 * Author: Travis Moore (mooret12)
 * Revised: April 12th, 2021
 *
 * Description: runs all of the tests.
 */

package src;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    TestBoardT.class
})

public class AllTests
{
}
```

C Code for BoardT.java

```java
/**
 * file: BoardT.java
 * Author: Travis Moore (mooret12)
 * Date: April 12, 2021
 *
 * Description: module representing the game board/grid.
 */

package src;

public class BoardT{


    private int[][] board;
    private int score;
    public final static int size = 4;

    /**
     * @brief Constructor
     * @details generates a board with two randomly inserted tiles
     */
    public BoardT(){

        this.board = new int[size][size];
        this.score = 0;

        for (int i = 0; i < size; i++){

            for (int j = 0; j < size; j++){

                board[i][j] = 0;
            }
        }
        addTile();
        addTile();
    }

    /**
     * @brief Gets the cell number at given x and y
     * @param x - row number
     * @param y - column number
     *
     * @return Integer value of cell/tile.
     */
    public int getCell(int x, int y){

        return board[x][y];
    }

    /**
     * @brief the current score for the game.
     * @return Integer representing the score.
     */
    public int getScore(){

        return score;
    }

    /**
     * @brief Sets a tile number at a given x y coordinate. Used for testing.
     * @param x - row number
     * @param y - column number
     * @param number - value of the tile.
     *
     */
    public void setTile(int x, int y, int number){

        board[x][y] = number;
    }

    /**
     * @brief Adds a tile to the game board. Location and value is random.
     *
     */
    public void addTile(){
```

```
    boolean done = false;

    while (!done){

        int x = Services.newCellLocation();
        int y = Services.newCellLocation();
        int cellValue = Services.newTileNumber();

        if (board[y][x] == 0){

            board[y][x] = cellValue;
            done = true;
        }
    }
}

/**
 * @brief Checks to see if game is over or not. Game is over
 * when you cannot make any more moves and every tile is full (non-zero).
 * @return True if game is over, False if game is not over.
 */
public boolean gameOver(){

    for (int i = 0; i < size; i++){

        for (int j = 0; j < size-1; j++){


            if (board[i][j] == 0 || board[j][i] == 0){

                return false;
            }

            if (board[i][j] == board[i][j+1]){

                return false;
            }

            if (board[j][i] == board[j+1][i] ){

                return false;
            }
        }
    }

    return true;
}
/**
 * @brief Shifts every number to the right if possible. Also adds
 *  tiles that are next together with the same number. Updates score if this
 *  happens.
 */
public void moveRight(){

    boolean changed = false;
    int x;
    int y;
    for (int i = 0; i < size; i++){

        boolean check = shiftrowRight(i);
        if (check == true){

            changed = true;
        }

        for (int j = size-1; j > 0; j--){

            x = size - 1;
            y = size - 2;


            // if we have pair to add
            if (board[i][j] == board[i][j-1] && board[i][j] != 0){

                score += board[i][j] + board[i][j-1];
                board[i][j] += board[i][j];
                board[i][j-1] = 0;
                changed = true;
                shiftrowRight(i);
```

```
        }
        }
    }
    if (changed == true){

        addTile();
    }
}
/**
 * @brief Shifts every number to the Left if possible. Also adds
 *  tiles that are next together with the same number. Updates score if this
 *  happens.
 */
public void moveLeft(){

    boolean changed = false;
    int x;
    int y;
    for (int i = size-1; i >= 0; i--){

        boolean check = shiftrowLeft(i);

        if (check == true){

            changed = true;
        }

        for (int j = 0; j < size-1; j++){


            // if we have pair to add
            if (board[i][j] == board[i][j+1] && board[i][j] != 0){

                score += board[i][j] + board[i][j+1];
                board[i][j] += board[i][j];
                board[i][j+1] = 0;
                changed = true;
                shiftrowLeft(i);

        }
        }
    }
    if (changed == true){

        addTile();
    }
}
/**
 * @brief Shifts every number up if possible. Also adds
 *  tiles that are next together with the same number. Updates score if this
 *  happens.
 */
public void moveUp(){

    boolean changed = false;
    int x;
    int y;
    for (int i = 0; i < size; i++){

        boolean check = shiftColumnUp(i);
        if (check == true){

            changed = true;
        }

        for (int j = 0; j < size-1; j++){

            x = size - 1;
            y = size - 2;


            // if we have pair to add
            if (board[j][i] == board[j+1][i] && board[j][i] != 0){

                score += board[j][i] + board[j][i];
                board[j][i] += board[j][i];
                board[j+1][i] = 0;
                changed = true;
                shiftColumnUp(i);
```

```
        }
        }
    }
    if (changed == true){

        addTile();
    }
}

/**
 * @brief Shifts every number down if possible. Also adds
 *  tiles that are next together with the same number. Updates score if this
 *  happens.
 */
public void moveDown(){

    boolean changed = false;
    int x;
    int y;
    for (int i = 0; i < size; i++){

        boolean check = shiftColumnDown(i);
        if (check == true){

            changed = true;
        }

        for (int j = size-1; j > 0; j--){

            x = size - 1;
            y = size - 2;


            // if we have pair to add
            if (board[j][i] == board[j-1][i] && board[j][i] != 0){

                score += board[j][i] + board[j][i];
                board[j][i] += board[j][i];
                board[j-1][i] = 0;
                changed = true;
                shiftColumnDown(i);

        }
        }
    }
    if (changed == true){

        addTile();
    }
}

private boolean shiftrowRight(int row){

    int answer = -1;

    for (int i = 0; i < size; i++){

        int ans = shiftNumRight(row);

        if (ans == 1 ){

            answer = 1;
        }
    }
    return answer == 1;
}

private int shiftNumRight(int row){

    int ans = -1;

    for (int i = size-1; i > 0; i--){


        if (board[row][i] == 0 && board[row][i-1] != 0){

            board[row][i] = board[row][i-1];
            board[row][i-1] = 0;
            ans = 1;
        }
```

```
    }

    return ans;

    }

private boolean shiftrowLeft(int row){

    int answer = -1;

    for (int i = 0; i < size; i++){

        int ans = shiftNumLeft(row);

        if (ans == 1 ){

            answer = 1;
        }
    }
    return answer == 1;
}

private int shiftNumLeft(int row){

    int ans = -1;

    for (int i = 0; i < size-1; i++){


        if (board[row][i] == 0 && board[row][i+1] != 0){

            board[row][i] = board[row][i+1];
            board[row][i+1] = 0;
            ans = 1;
        }
    }

    return ans;

    }

private boolean shiftColumnUp(int row){

    int answer = -1;

    for (int i = 0; i < size; i++){

        int ans = shiftNumUp(row);

        if (ans == 1 ){

            answer = 1;
        }
    }
    return answer == 1;
}

private int shiftNumUp(int column){

    int ans = -1;

    for (int i = 0; i < size-1; i++){


        if (board[i][column] == 0 && board[i+1][column] != 0){

            board[i][column] = board[i+1][column];
            board[i+1][column] = 0;
            ans = 1;
        }
    }

    return ans;

    }
private boolean shiftColumnDown(int row){

    int answer = -1;

    for (int i = 0; i < size; i++){
```

```
            int ans = shiftNumDown(row);

            if (ans == 1 ){

                answer = 1;
            }
        }
        return answer == 1;
    }

    private int shiftNumDown(int column){

        int ans = -1;

        for (int i = size-1; i > 0; i--){


            if (board[i][column] == 0 && board[i-1][column] != 0){

                board[i][column] = board[i-1][column];
                board[i-1][column] = 0;
                ans = 1;
            }
        }
        return ans;
        }
    }
```

D Code for Controller.java

```
/**
 * file Controller.java
 * Author: Travis Moore (mooret12)
 * Date: April 12, 2021
 *
 * Description: Class for Controller that will handle input from
 *  terminal and connects UserView and BoardT
 */

package src;

import java.util.Scanner;

public class Controller{


    private BoardT model;
    private UserView view;
    private static Controller controller = null;

    private Scanner scanner = new Scanner(System.in);


    /**
     * @brief Constructor
     * @param model - BoardT object
     * @param view -  UserView object
     */
    private Controller(BoardT model, UserView view){
        this.model = model;
        this.view = view;
    }

    /**
     * @brief method for getting a single instance
     * @return a single Controller object
     */
    public static Controller getInstance(BoardT model, UserView view){

        if (controller == null)
            controller = new Controller(model, view);

        return controller;
    }

    /**
     * @brief initializes the game
     */
    public void initializeGame(){

        this.model = new BoardT();
    }

    /**
     * @brief updates the view module to show a welcome message to window/terminal
     */
    public void showWelcomeMessage(){

        view.printWelcomeMessage();
    }

    /**
     * @brief updates the view module to show a end of game message
     */
    public void showEndMessage(){


        view.printEndingMessage(model);
    }

    /**
     * @brief updates the view module to show the board
     */
    public void showBoard(){

        view.printBoard(model);
```

```
    }

    /**
     * @brief runs 2048.
     */
    public void run2048(){

        String move;

        initializeGame();

        view.printBoard2(model);


        while (true){

            move = scanner.next();


            if (move.equals("w")){

                model.moveUp();
                showBoard();

                if (model.gameOver()){

                    break;
                }
            }

            else if (move.equals("s")){

                model.moveDown();
                showBoard();
                if (model.gameOver()){

                    break;
                }

            }

            else if (move.equals("a")){

                model.moveLeft();
                showBoard();
                if (model.gameOver()){

                    break;
                }

            }

            else if (move.equals("d")){

                model.moveRight();
                showBoard();
                if (model.gameOver()){

                    break;
                }

            }

            else if (move.equals("p")){

                System.exit(1);

            }
            else{
                System.out.println("Please enter valid input");
            }


        }
        view.printBoard2(model);
        showEndMessage();
    }

}
```

E Code for Services.java

```
/**
 * file: Services.java
 * Author: Travis Moore (mooret12)
 * Date: April 12, 2021
 *
 * Description: Services module containing functions to help with BoardT
 */

package src;

/**
 * This class contains functions to help create tiles in BoardT. They both deal with using
     Math.random()
 */
public class Services{

    /**
     * @brief Randomly generates a index for a new tile.
     *
     * @return  Integer between 0-3.
     */
    public static int newCellLocation(){

        double num = Math.random();
        if (num < 0.25){

            return 0;

        } else if (num < 0.5) {

            return 1;

        }else if (num < 0.75) {

            return 2;
        }else {

            return 3;
        }
    }
    /**
     * @brief Uses random number from 0 to 1 to decide the new tile's value.
     *
     * @return      either 2 or 4.
     */
    public static int newTileNumber(){

        if (Math.random() < 0.9){

            return 2;
        } else{
            return 4;
        }
    }




}
```

F Code for TestBoardT.java

```
/**
 * Author: Travis Moore (mooret12)
 * Revised: April 12 2021
 *
 * Description: Testing for BoardT class.
 */

package src;
import java.util.*;
import org.junit.*;
import static org.junit.Assert.*;

public class TestBoardT
{

    private BoardT board;

    @Before
    public void setUp(){

        board = new BoardT();
    }

    @After
    public void tearDown(){

        board = null;
    }

    @Test
    public void testGetCell(){

        board.setTile(0,0,128);
        board.setTile(1,1,64);
        assertTrue(board.getCell(0,0) == 128);
        assertTrue(board.getCell(1,1) == 64);
    }

    @Test
    public void testGetScore(){

        assertTrue(board.getScore() == 0);

    }

    @Test
    public void testGetScore2(){

        board.setTile(0,0,64);
        board.setTile(1,0,64);
        board.moveUp();
        assertTrue(board.getScore() >= 128);
    }


    @Test
    public void testGetScore3(){

        board.setTile(0,0,64);
        board.setTile(1,0,64);
        board.setTile(0,1,128);
        board.moveUp();
        board.moveLeft();
        assertTrue(board.getScore() >= 128);
    }

    @Test
    public void testMoveRight(){

        board.setTile(0,0,2);
        board.setTile(0,1,2);
        board.setTile(0,2,2);
        board.setTile(0,3,2);
        board.moveRight();

        assertTrue(board.getCell(0,3) == 4);
```

```
        assertTrue(board.getCell(0,2) == 4);

        board.moveRight();

        assertTrue(board.getCell(0,3) == 8);
    }

    @Test
    public void testMoveRight2(){

        board.setTile(1,0,2);
        board.setTile(1,1,2);
        board.setTile(1,2,2);
        board.setTile(1,3,4);

        board.moveRight();

        assertTrue(board.getCell(1,3) == 4);
        assertTrue(board.getCell(1,2) == 4);
        assertTrue(board.getCell(1,1) == 2);

        board.moveRight();
        assertTrue(board.getCell(1,3) == 8);
    }

    @Test
    public void testMoveLeft(){

        board.setTile(1,0,4);
        board.setTile(1,1,4);
        board.setTile(1,2,4);
        board.setTile(1,3,8);

        board.moveLeft();

        assertTrue(board.getCell(1,0) == 8);
        assertTrue(board.getCell(1,1) == 4);

    }

    @Test
    public void testMoveLeft2(){

        board.setTile(3,0,8);
        board.setTile(3,1,8);
        board.setTile(3,2,8);
        board.setTile(3,3,8);

        board.moveLeft();

        assertTrue(board.getCell(3,0) == 16);
        assertTrue(board.getCell(3,1) == 16);
        assertFalse(board.getCell(3,2) == 8);
    }

    @Test
    public void testMoveUp(){


        board.setTile(0,0,8);
        board.setTile(1,0,8);
        board.setTile(2,0,8);
        board.setTile(3,0,8);

        board.moveUp();

        assertTrue(board.getCell(0,0) == 16);
        assertTrue(board.getCell(1,0) == 16);
        assertTrue(board.getCell(2,0) != 8);

    }

    @Test
    public void testMoveUp2(){


        board.setTile(0,2,2);
        board.setTile(1,2,2);
        board.setTile(2,2,0);
        board.setTile(3,2,0);
```

```
        board.moveUp();

        assertTrue(board.getCell(0,2) == 4);
    }


    @Test
    public void testMoveDown(){


        board.setTile(0,2,2);
        board.setTile(1,2,2);
        board.setTile(2,2,0);
        board.setTile(3,2,0);

        board.moveDown();

        assertTrue(board.getCell(3,2) == 4);
    }

    @Test // special case
    public void testMoveDown2(){


        board.setTile(0,2,4);
        board.setTile(1,2,4);
        board.setTile(2,2,4);
        board.setTile(3,2,4);

        board.moveDown();

        assertTrue(board.getCell(3,2) == 8);
        assertTrue(board.getCell(2,2) == 8);
        assertTrue(board.getCell(1,2) != 8);
    }

    @Test //Previous bug, need to make sure works
    public void testgameOver(){

        board.setTile(0,0,16);
        board.setTile(0,1,2);
        board.setTile(0,2,8);
        board.setTile(0,3,4);

        board.setTile(1,0,16);
        board.setTile(1,1,256);
        board.setTile(1,2,8);
        board.setTile(1,3,0);

        board.setTile(2,0,4);
        board.setTile(2,1,128);
        board.setTile(2,2,32);
        board.setTile(2,3,0);

        board.setTile(3,0,2);
        board.setTile(3,1,4);
        board.setTile(3,2,16);
        board.setTile(3,3,0);


        assertFalse(board.gameOver());

    }

    @Test //Game over situation
    public void testgameOver2(){

        board.setTile(0,0,2);
        board.setTile(0,1,4);
        board.setTile(0,2,8);
        board.setTile(0,3,12);

        board.setTile(1,0,16);
        board.setTile(1,1,32);
        board.setTile(1,2,64);
        board.setTile(1,3,2);

        board.setTile(2,0,64);
        board.setTile(2,1,256);
```

```
        board.setTile(2,2,512);
        board.setTile(2,3,4);

        board.setTile(3,0,2);
        board.setTile(3,1,8);
        board.setTile(3,2,1024);
        board.setTile(3,3,8);


        assertTrue(board.gameOver());

    }

    @Test //Game over situation
    public void testgameOver3(){

        board.addTile();
        board.addTile();
        board.addTile();
        board.addTile();
        board.addTile();
        board.addTile();
        board.addTile();

        assertFalse(board.gameOver());

    }

}
```

G Code for UserView.java

```
/**
 * file UserView.java
 * Author: Travis Moore (mooret12)
 * Date: April 12, 2021
 *
 * Description: Class for view that deals with outputting information
 * to the terminal/window.
 *
 */

package src;

/**
 * This class is for outputting the game to the terminal/window.
 */
public class UserView{


    private static UserView visual = null;

   /**
    * @brief Constructor.
    */
    private UserView(){}

   /**
    * @brief method for getting a single instance
    * @return an UserView object
    */
    public static UserView getInstance(){

        if (visual == null){

            return visual = new UserView();
        }
        return visual;
    }


   /**
    * @brief Displays a welcome message.
    */
    public void printWelcomeMessage(){

        System.out.println("-------------------------------------------------------");
        System.out.println("                    Welcome To 2048                    ");
        System.out.println("-------------------------------------------------------");
        System.out.println(" ");
    }

   /**
    * @brief Displays a end of game message. Also displays final score.
    * @param board - A boardT object.
    */
    public void printEndingMessage(BoardT board){

        System.out.println("-------------------------------------------------------");
        System.out.println("                  Game Over.                           ");
        System.out.printf("                Final Score: " + board.getScore()       );
        System.out.println();
        System.out.println("-------------------------------------------------------");
    }


   /**
    * @brief Displays the current state of the game board. Also clears terminal before displaying
    *  the board. This allows for smooth transition between moves.
    * @param board - A boardT object.
    */
    public void printBoard(BoardT board){

        int size = board.size;
        System.out.print("\033[H\033[2J");
        System.out.flush();
        System.out.println();
        System.out.println();
```

```
    System.out.println();
    System.out.println();
    System.out.println();
    printWelcomeMessage();
    System.out.println();
    System.out.printf("Score: " +  board.getScore());
    System.out.println();
    System.out.println();

    for (int i = 0; i < size; i++){

        for (int j = 0; j < size; j++){

            // System.out.print(board.board[i][j] + "  ");
            if (j == size-1){

                if (board.getCell(i,j) != 0){

                    System.out.print(String.format("%-4s%-4d|", "|", board.getCell(i,j)));

                } else {
                    System.out.print(String.format("%-4s%-4s|", "|", ""));
                }

            } else {

                    if (board.getCell(i,j) != 0){

                        System.out.print(String.format("%-4s%-4d ", "|", board.getCell(i,j)));

                }   else {
                        System.out.print(String.format("%-4s%-4s ", "|", ""));
                }
        }
        }

            System.out.println();

    }

    System.out.println();
}

   /**
    * @brief Displays the current state of the game board. Does NOT clear terminal before
        displaying
    *  the board. This is needed for the end of the game, so the final board with the final tile
        will
    *  be displayed.
    * @param board - A boardT object.
    */
    public void printBoard2(BoardT board){

    printWelcomeMessage();

    int size = board.size;
    System.out.println();
    System.out.printf("Score: " +  board.getScore());
    System.out.println();
    System.out.println();
    for (int i = 0; i < size; i++){

        for (int j = 0; j < size; j++){


            if (j == size-1){

                if (board.getCell(i,j) != 0){

                    System.out.print(String.format("%-4s%-4d|", "|", board.getCell(i,j)));

                } else {
                    System.out.print(String.format("%-4s%-4s|", "|", ""));
                }

            } else {

                    if (board.getCell(i,j) != 0){

                        System.out.print(String.format("%-4s%-4d ", "|", board.getCell(i,j)));
```

```
                    }    else {
                            System.out.print(String.format("%-4s%-4s ", "|", ""));
                    }
            }
            }

                System.out.println();

        }

        System.out.println();
    }

}
```