Programmer's Guide For "2584"

Author: Ryan Kelly

Language Used: Java

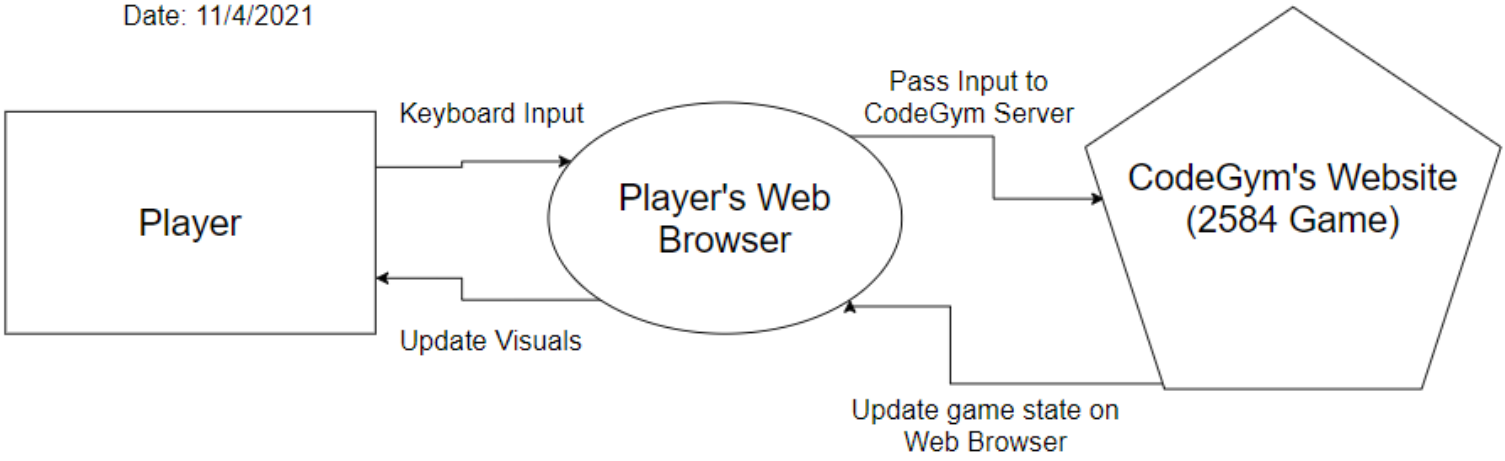
Software: CodeGym's Online Java Compiler

Assets Utilized: CodeGym's Game Class in Java
https://codegym.cc/groups/posts/138-games-section-on-codegym-game-engine

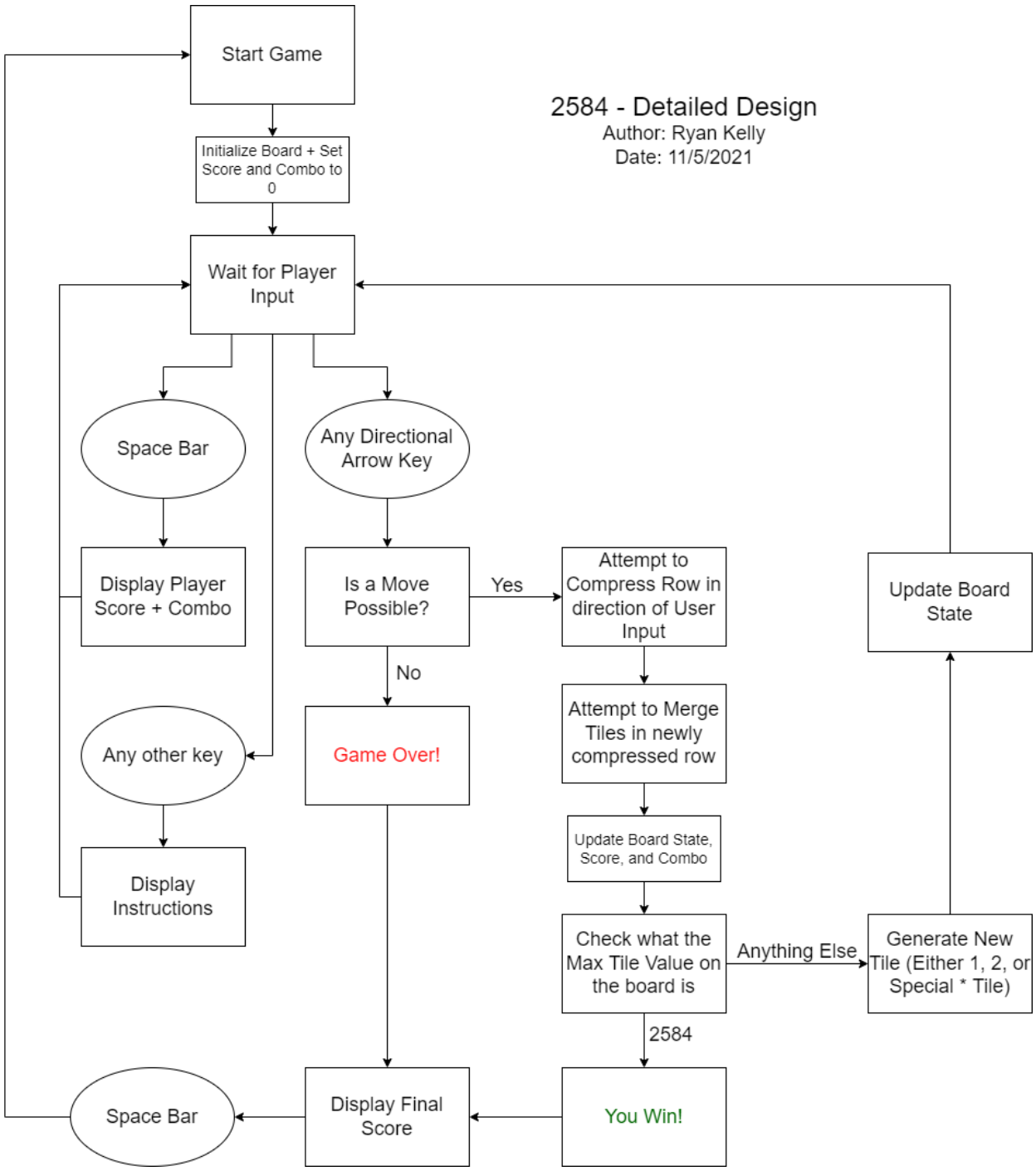
2584 - High Level Design

2854 - High Level Design

Author: Ryan Kelly
Date: 11/4/2021



Detailed Design - 2584 Game



Where to Play

Location:

2584 is available to play on Windows, Mac, and Linux systems by following the link provided.
https://codegym.cc/projects/apps/64229

Script Pseudocode

Pseudocodes for important functions in 2584 are included here.

```
createGame Function Pseudocode {
        Set score and combo to zero
        For every tile on the gamefield {
                Initialize their values to zero
        }
        Set 2 random tiles to either 1, 2, or *
        Display starting message to player
}

 createNewNumber Function Pseudocode {
        X position equals random number between 1 and 4
        Y position equals random number between 1 and 4
        If gamefield at XY equals 0 {
                Number equals random number between 1 and 100
                If number equals 99 {
                        Gamefield at XY equals Special * Tile
                }
                If number is less than 99 and greater than 10 {
                        Gamefield at XY equals 1
                }
                Else {
                        Gamefield at XY equals 2
                }
```

```
            Integer max equals the result of getMaxTileValue function
            If max equals 2584 {
                You Win!
            }
        }
}
```

```
onKeyPress Function Pseudocode {
        Call canUserMove function to see if a move is possible
        If a move is not possible {
            Game Over!
        }
        If key pressed is space and the game has ended in game over or you win {
            Start a new game
        }
        If key pressed is left arrow and the game has not ended {
            Call moveLeft function
        }
        If key pressed is right arrow and the game has not ended {
            Call moveRight function
        }
        If key pressed is up arrow and the game has not ended {
            Call moveUp function
        }
        If key pressed is down arrow and the game has not ended {
            Call moveDown function
        }
```

```
        If key pressed is space and the game has not ended {
                Display current score and combo
        }
        Else {
                Do nothing
        }
        Update the gamefield
}


moveLeft function Pseudocode {
        Boolean moved equals false
        Boolean mergeThisMove equals false
        For every row in the gamefield {
                Boolean compressed equals true if the row can compress, false otherwise
                Boolean merged equals true if tiles in the row can merge, false otherwise
                If merged is true {
                        Compress the current row
                        Set mergeThisMove to true
                }
                If compressed or merged equals true {
                        Moved equals true
                }
        }
        If moved equals true {
                Create a new tile on the board
        }
        If mergeThisMove is false {
```

```
                Set combo to 0
        }
}
```

```
compressRow Function Pseudocode {
        Integer insertPosition equals 0
        Boolean result is set to false
        For every tile in the current row {
                If current tile is greater than zero {
                        If current tile is not at the insertPosition {
                                Previous tile equals current tile
                                Current tile equals zero
                                Result is set to true
                        }
                        Add 1 to insertPosition
                }
        }
        Return result
}

mergeRow Function Pseudocode {
        Boolean result is set to false
```

For every tile in the current row {

For every value in the fibonacci sequence until 2584 {

If current tile doesn't equal 0 and current tile equals a fibonacci value and the next tile equals a fibonacci value that can merge with current tile and a combo isn't happening {

Current tile merges with next tile

Next tile is set equal to zero

Result is set to true

Combo is set to true

Current Combo is incremented by 1

Add sum of merged tiles to score

}

Else if current tile doesn't equal 0 and current tile equals a fibonacci value and the previous tile equals a fibonacci value that can merge with current tile and a combo isn't happening {

Current tile merges with next tile

Previous tile is set equal to zero

Result is set to true

Combo is set to true

Current Combo is incremented by 1

Add sum of merged tiles to score

}

/*This if-else statement is rather long, but these first two checks basically outline what two situations can occur when a merge happens. 1, a combo starts, or 2, a combo continues. The other if else statements check the tiles in the reverse order, and there has to be separate checks for the special * Block for merging*/

}

```
        }
        Return true is a merge occurred, false otherwise
}
```

Actual Program Code

```
/*
Author: Ryan Kelly
Class: Computer Science 4880 - Independent Research Study
School: University of Missouri: Saint Louis
Date: 10/31/2021
Language Used: Java

Description: For my Independent Research Project, I am creating an educational math game.
This program is a variation of the popular game 2048, that uses fibonacci numbers instead of
base 2 numbers. The original program was written following CodeGym's 2048 Game Task
Tutorial. I modified several functions to change how the game is played. As mentioned
previously, I changed the tiles to be fibonacci numbers. The goal of the game now is
to reach a tile with the value 2584. In addition, combos have been added to reward
the player for consecutive moves that merge tiles, leading to higher scores. Finally,
I implemented a special asterisk (*) tile that merges with any block.
*/
package com.codegym.games.game2048;

import com.codegym.engine.cell.*;

public class Game2048 extends Game {
    //Variable declarations
    private static final int SIDE = 4;
    private int[][] gameField = new int[SIDE][SIDE];
    private boolean isGameStopped = false, result = false, curCombo = false;
    private int score = 0, combo = 0;
    //This value is used as it isn't part of the Fibonacci sequence, and just denotes the special * block
    private static int EX = 100;
    //Array of color values for blocks
    private Color[] colors = Color.values();
    private int[] fibValues = new int[]{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, EX};

    //Initialize is run when the program is started, and creates the game screen.
    @Override
    public void initialize() {
        setScreenSize(SIDE, SIDE);
        createGame();
        drawScene();
    }
```

```
//Initializes the gamefield with 0s, create 2 random tiles, and display the starting message.
private void createGame() {
    score = 0;
    combo = 0;
    setScore(score);
    for (int y = 0; y < SIDE; y++) {
        for (int x = 0; x < SIDE; x++) {
            gameField[y][x] = 0;
        }
    }
    createNewNumber();
    createNewNumber();
    Color cell = Color.GREEN;
    Color text = Color.BLACK;
    int textSize = 25;
    String message = "Try to reach 2584!" + "\n Press the I Key to check Instructions!";
    showMessageDialog(cell, message, text, textSize);
}

//Update gamefield
private void drawScene() {
    for (int y = 0; y < SIDE; y++) {
        for (int x = 0; x < SIDE; x++) {
            setCellColoredNumber(x, y, gameField[y][x]);
        }
    }
}

//Adds a new tile to the gamefield, having either the value 1, 2, or the special * blocks
//The * is only a 1% chance, 1 is an 89% chance, and 2 is a 10% chance
//Everytime createNewNumber is called (which is every move) it calls
//getMaxTileValue to see if the player meets the win condition
private void createNewNumber() {
    int x = getRandomNumber(SIDE);
    int y = getRandomNumber(SIDE);
    if ( gameField[y][x] == 0 ) {
        int number = getRandomNumber(100);
        if (number == 99) {
            gameField[y][x] = fibValues[18];
        }
        else if (number < 99 && number > 10) {
            gameField[y][x] = fibValues[1];
        }
```

```
        else {
            gameField[y][x] = fibValues[2];
        }
    }
    else {
        createNewNumber();
    }
    int max = getMaxTileValue();
    if (max == 2584)
    {
        win();
    }
}

//Sets a tiles value, as well as it's color
private void setCellColoredNumber(int x, int y, int value) {
    Color color = getColorByValue(value);
    String str;
    if (value == 0) {
        str = "";
    }
    else if (value == EX) {
        str = "*";
    }
    else {
        str = "" + value + "";
    }
    setCellValueEx(x, y, color, str);
}

//Determines which color to make the cell by it's value
private Color getColorByValue(int value) {
    if (value == 1) {
        return Color.PLUM;
    }
    else if (value == 2) {
        return Color.SLATEBLUE;
    }
    else if (value == 3) {
        return Color.DODGERBLUE;
    }
    else if (value == 5) {
        return Color.DARKTURQUOISE;
```

```
}
else if (value == 8) {
    return Color.MEDIUMSEAGREEN;
}
else if (value == 13) {
    return Color.LIMEGREEN;
}
else if (value == 21) {
    return Color.DARKORANGE;
}
else if (value == 34) {
    return Color.SALMON;
}
else if (value == 55) {
    return Color.ORANGERED;
}
else if (value == 89) {
    return Color.DEEPPINK;
}
else if (value == 144) {
    return Color.MEDIUMVIOLETRED;
}
else if (value == 233) {
    return Color.PURPLE;
}
else if (value == 377) {
    return colors[7];
}
else if (value == 610) {
    return Color.DARKRED;
}
else if (value == 987) {
    return colors[73];
}
else if (value == 1597) {
    return colors[13];
}
else if (value == 2584) {
    return colors[89];
}
else if (value == EX) {
    return Color.YELLOW;
}
```

```
    else {
        return Color.WHITE;
    }
}

//Attempted function to animate tile movements
//public static void wait(int ms)
//{
//    try
//    {
//        Thread.sleep(ms);
//    }
//    catch(InterruptedException ex)
//    {
//        Thread.currentThread().interrupt();
//    }
//}

//Function that moves tiles in the direction the user presses
private boolean compressRow(int[] row) {
    int insertPos = 0;
    boolean result = false;
    for (int x = 0; x < SIDE; x++) {
        if (row[x] > 0) {
            if (x != insertPos) {
                row[insertPos] = row[x];
                row[x] = 0;
                result = true;
            }
            insertPos++;
        }
    }
    return result;
}

//Function that combines tiles if they are next to each other on the fibonacci sequence
private boolean mergeRow(int[] row) {
    result = false;

    for (int x = 0; x < SIDE - 1; x++) {
        for(int i = 1; i < fibValues.length - 1; i++) {
            if (row[x] != 0 && row[x] == fibValues[i] && row[x + 1] == fibValues[i + 1] && curCombo == false) {
                row[x] += row[x + 1];
```

```
    row[x + 1] = 0;
    result = true;
    curCombo = true;
    combo = combo + 1;
    score += row[x];
    setScore(score);
}
else if (row[x] != 0 && row[x] == fibValues[i] && row[x + 1] == fibValues[i - 1] && curCombo == false) {
    row[x] += row[x + 1];
    row[x + 1] = 0;
    result = true;
    curCombo = true;
    combo = combo + 1;
    score += row[x];
    setScore(score);
}
else if (row[x] != 0 && row[x] == fibValues[i] && row[x + 1] == fibValues[i + 1] && curCombo == true) {
    row[x] += row[x + 1];
    row[x + 1] = 0;
    result = true;
    curCombo = true;
    combo = combo + 1;
    score = score + (row[x] * combo);
    setScore(score);
}
else if (row[x] != 0 && row[x] == fibValues[i] && row[x + 1] == fibValues[i - 1] && curCombo == true) {
    row[x] += row[x + 1];
    row[x + 1] = 0;
    result = true;
    curCombo = true;
    combo = combo + 1;
    score = score + (row[x] * combo);
    setScore(score);
}
else if (row[x] != 0 && row[x] == fibValues[i] && row[x + 1] == fibValues[18] && curCombo == false) {
    row[x] += fibValues[i - 1];
    row[x + 1] = 0;
    result = true;
    curCombo = true;
    combo = combo + 1;
    score += row[x];
    setScore(score);
}
```

```
            else if (row[x] != 0 && row[x] == fibValues[i] && row[x + 1] == fibValues[18] && curCombo == true) {
                row[x] += fibValues[i - 1];
                row[x + 1] = 0;
                result = true;
                curCombo = true;
                combo = combo + 1;
                score += row[x];
                setScore(score);
            }
        }
    }
    return result;
}

//Reads user input. If the game is over and the user presses space, start a new game.
//If the game is running and the user presses space, display score.
//If the user presses an arrow key, check if a move is possible and either move, or give a game over.
@Override
public void onKeyPress(Key key) {
    boolean movePossible = canUserMove();
    if (movePossible == false) {
        gameOver();
        //return;
    }

    if (key == Key.SPACE && isGameStopped == true) {
        isGameStopped = false;
        createGame();
        drawScene();
    }

    if (key == Key.UNKNOWN && isGameStopped == false) {
        Color cell = Color.GREEN;
        Color text = Color.BLACK;
        int textSize = 15;
        String message = "Goal: Make a tile with 2584 points!\nControls: Arrow Keys shift tiles. \nSpace displays score and
restarts on a game over.\n The I Key shows instructions!";
        showMessageDialog(cell, message, text, textSize);
    }

    if (key == Key.LEFT && isGameStopped == false) {
        moveLeft();
    }
```

```
    else if (key == Key.RIGHT && isGameStopped == false) {
        moveRight();
    }
    else if (key == Key.UP && isGameStopped == false) {
        moveUp();
    }
    else if (key == Key.DOWN && isGameStopped == false) {
        moveDown();
    }
    else if (key == Key.SPACE && isGameStopped == false) {
        Color cell = Color.LIGHTBLUE;
        Color text = Color.WHITE;
        int textSize = 25;
        String message = "Score: " + score + " Combo: " + combo;
        showMessageDialog(cell, message, text, textSize);
    }
    else {
        return;
    }
    drawScene();
}

//Function that checks if you need to compress, merge, or both based off your input
//If a merge doesn't occur, combo returns to 0
private void moveLeft() {
    boolean moved = false;
    boolean mergeThisMove = false;

    for (int[] row : gameField) {
        boolean compressed = compressRow(row);
        boolean merged = mergeRow(row);
        if (merged) {
            compressRow(row);
            mergeThisMove = true;
        }
        if (compressed || merged) {
            moved = true;
        }
    }
    if (moved) {
        createNewNumber();
    }
    if(!mergeThisMove)
```

```
    {
        curCombo = false;
        combo = 0;
    }
}

//By rotating the gameField, we can create the same result as a moveRight function using moveLeft
private void moveRight() {
    rotateClockwise();
    rotateClockwise();
    moveLeft();
    rotateClockwise();
    rotateClockwise();
}

private void moveUp() {
    rotateClockwise();
    rotateClockwise();
    rotateClockwise();
    moveLeft();
    rotateClockwise();
}

private void moveDown() {
    rotateClockwise();
    moveLeft();
    rotateClockwise();
    rotateClockwise();
    rotateClockwise();
}

//Rather than coding a moveRight, moveUp, and moveDown, we can
//rotate the gamefield until moveLeft will produce the intended results
private void rotateClockwise() {
    int[][] rotatedGameField = new int[SIDE][SIDE];

    for (int i = 0; i < SIDE; i++) {
        for (int j = 0; j < SIDE; j++) {
            rotatedGameField[j][SIDE - 1 - i] = gameField[i][j];
        }
    }
    gameField = rotatedGameField;
}
```

```
//Check gamefield for the largest current value
private int getMaxTileValue() {
    int maxValue = 0;
    for (int i = 0; i < SIDE; i++) {
        for (int j = 0; j < SIDE; j++) {
            if (maxValue < gameField[i][j]) {
                maxValue = gameField[i][j];
            }
        }
    }
    return maxValue;
}

//When called, win stops the game and displays "You win!" plus your score.
private void win() {
    isGameStopped = true;
    Color cell = Color.LIGHTBLUE;
    Color text = Color.WHITE;
    int textSize = 25;
    String message = "You win! Score: " + score;
    showMessageDialog(cell, message, text, textSize);
}

//Function to check if a move is possible. If not possible, return false
//This leads to a gameOver
private boolean canUserMove() {
    for (int i = 0; i < SIDE; i++) {
        for (int j = 0; j < SIDE; j++) {
            for (int x = 1; x < fibValues.length; x++) {
                if (gameField[j][i] == 0) {
                    return true;
                }
                else if (j < SIDE - 1 && gameField[j][i] == fibValues[x] && gameField[j + 1][i] == fibValues[x + 1]) {
                    return true;
                }
                else if (j < SIDE - 1 && gameField[j][i] == fibValues[x] && gameField[j + 1][i] == fibValues[x - 1]) {
                    return true;
                }
                else if (i < SIDE - 1 && gameField[j][i] == fibValues[x] && gameField[j][i + 1] == fibValues[x + 1]) {
                    return true;
                }
                else if (i < SIDE - 1 && gameField[j][i] == fibValues[x] && gameField[j][i + 1] == fibValues[x - 1]) {
```

```
                    return true;
                }
            }
        }
    }
    return false;
}

//Function to stop the game. Displays "You lose!" plus your final score.
private void gameOver() {
    isGameStopped = true;
    Color cell = Color.RED;
    Color text = Color.GREY;
    int textSize = 25;
    String message = "You lose! Score: " + score;
    showMessageDialog(cell, message, text, textSize);
}
}
```

Testing

9/29 - Tested new base for the game, Fibonacci numbers instead of base 2. Tiles combined properly all the way to 2584, which is the win condition. Combos and scores were also tested, but combo never updated from zero. Needs some fixing.

10/6 - During a test run to test displaying score, I noticed the final score was not displaying properly. Corrected the order of function calls in gameOver() to prevent this error.

10/14 - Tested the gaming using the new Fibonacci tile that can combine with any tile to increase its value to the next in the Fibonacci sequence. On initial implementation, it actually increased the value two up the Fibonacci sequence instead of one. This made the Fibonacci tile a bit too powerful and not very intuitive, so it was corrected. Game is still beatable/losable.

11/10 - One last button was introduced, pressing anything besides the arrow keys or space bar will display the instructions for the game.

11/26 - The game was published on CodeGym for others to play. I sent it to two of my classmates. They were both able to play it on their respective browsers (FireFox and Opera GX). Both noted that the game is quite a bit longer than the original 2048, and is a bit difficult to beat. The lose screen worked for both of them, but neither reached the win screen. They suggested making the Fibonacci Tile more common. Each directional arrow, the space bar, and the I button work as intended.

2584

User Manual

By Ryan Kelly

11/2/2021

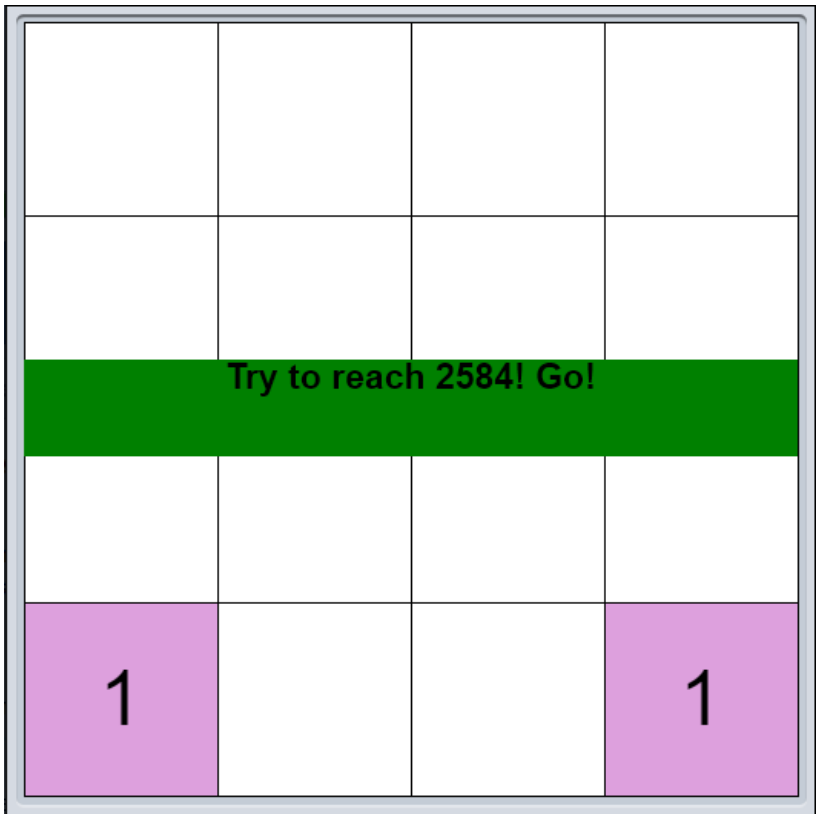


Table Of Contents:

What is 2584?

2584 is a simple tile-based math puzzle game. Your goal is to have one of these tiles reach 2584, the namesake of the game. It is based on the popular game 2048, though instead of base two, players will be working with fibonacci numbers. Try to reach a high score by chaining together moves!

Controls

Up Arrow - Shift all tiles towards the top of the board.
Down Arrow - Shift all tiles towards the bottom of the board.
Left Arrow - Shift all tiles towards the left side of the board.
Right Arrow - Shift all tiles towards the right side of the board.
Space - If a game is currently running, press Space to see your score and combo!
If the game is over, press Space to restart the game!

System Requirements

2584 runs on browsers, so requirements match that of internet browsers.

Minimum System Requirements (Google Chrome):

Windows

- Windows 7, Windows 8, Windows 8.1, Windows 10 or later
- An Intel Pentium 4 processor or later that's SSE3 capable

Mac

- OS X El Capitan 10.11 or later

Linux

- 64-bit Ubuntu 18.04+, Debian 10+, openSUSE 15.2+, or Fedora Linux 32+

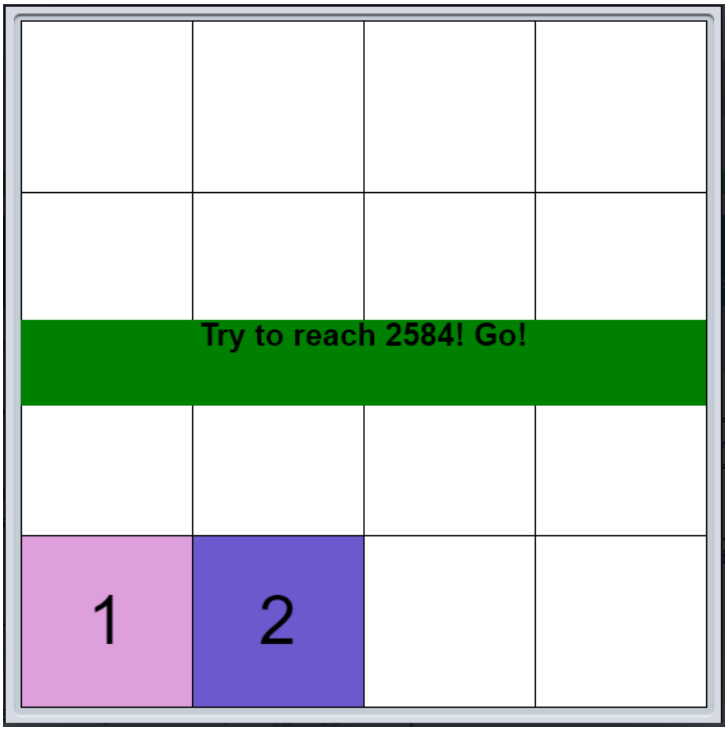
Playing the Game

1. **How to start**

Currently, 2584 is available on CodeGym.cc by using this link.
https://codegym.cc/projects/apps/64229
I hope to learn how to move the CodeGym packages into another Java editor to allow users to download the game. Once you follow the link, press the Run button to start the game.

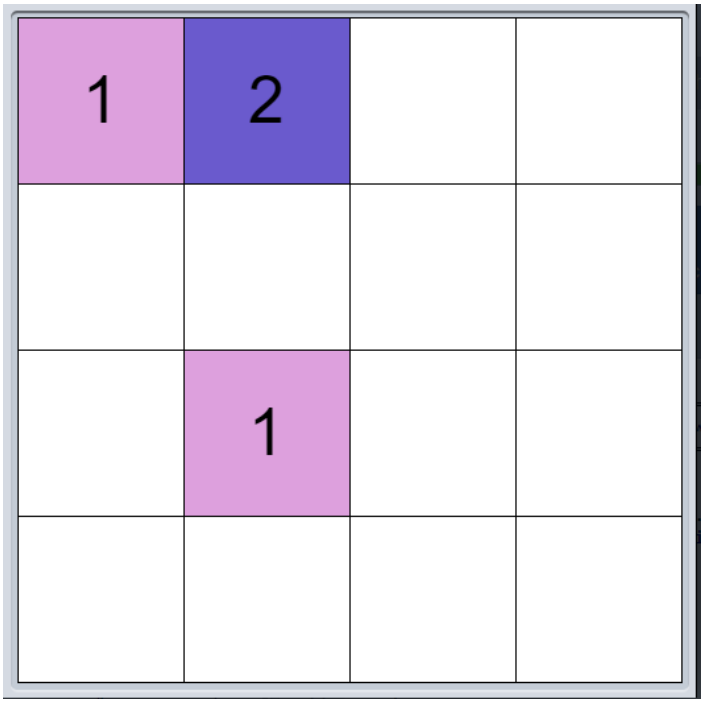
2. **How to play**

Once the game is running, players will be greeted with a screen similar to this.



Every board begins with two tiles in two random locations on the board. The first two tiles are always ones or twos. All white spaces are considered empty.

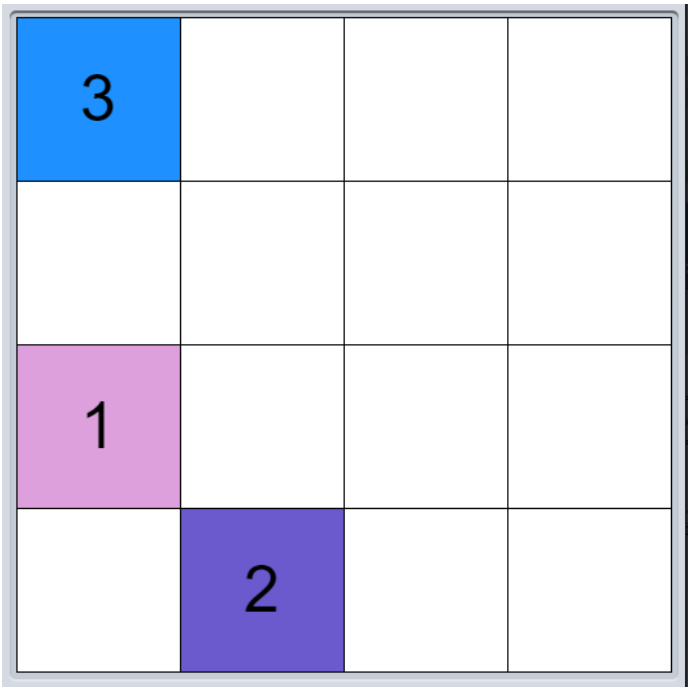
By pressing any of the arrow keys, the tiles will shift in that direction until they hit a wall, or another block. Lets try pressing the up arrow to see what happens!



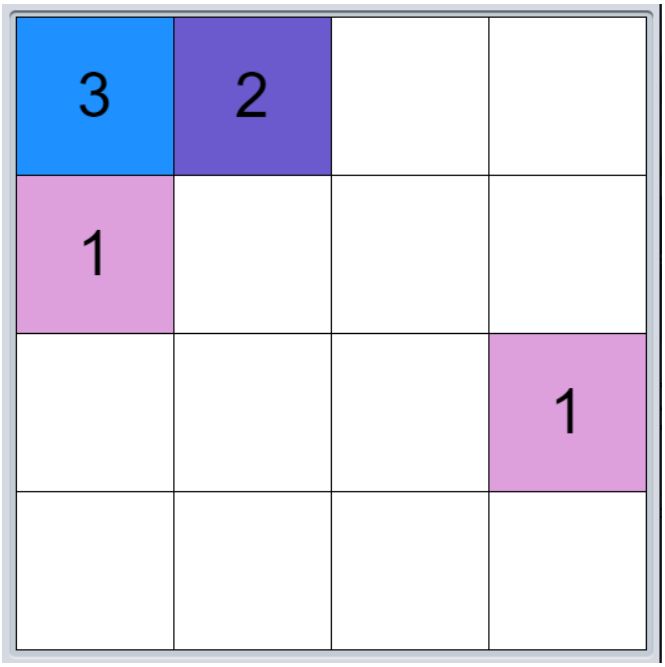
By pressing up, our one and two tiles shifted to the top of the board! After every move, one new tile is created.

If you are not familiar with the Fibonacci Sequence, it is a sequence of numbers that starts with a zero and a one, and continues based on the rule that each number is equal to the sum of the two preceding numbers. The sequence goes 0, 1, 1, 2, 3, 5, 8, 13, and so on. Because one and two are next to each other in the sequence, they can be merged into three!

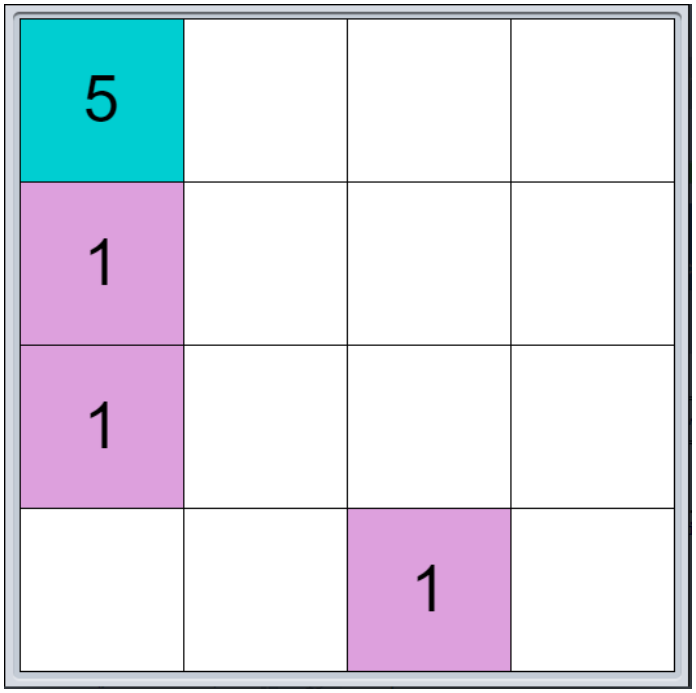
Let’s try pressing the left arrow!



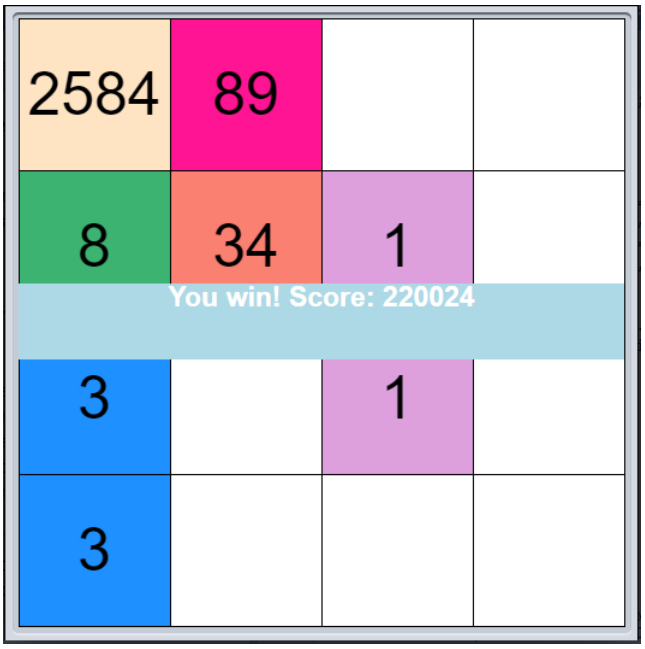
Our one and two merged in the top corner, creating a three! Our one tile shifted to the left, and a new two tile was created. The next number in the sequence after 3 is 5, so let's combine the 3 and 2! First, we press the up arrow…



Then, we press the left arrow…

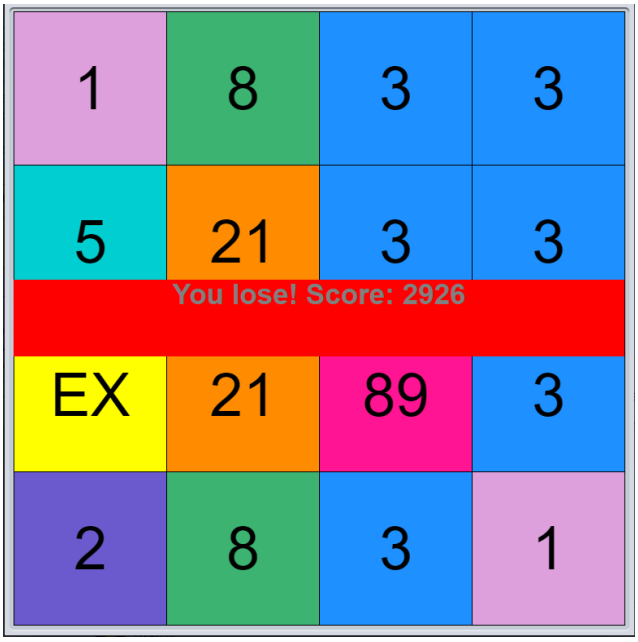


And we get our five! Keep combining these fibonacci numbers until you reach 2584 and win the game!



Hit space to start a new game!

If there is no whitespace left on the board and no possible moves left, the game will display a “You lose!” message.



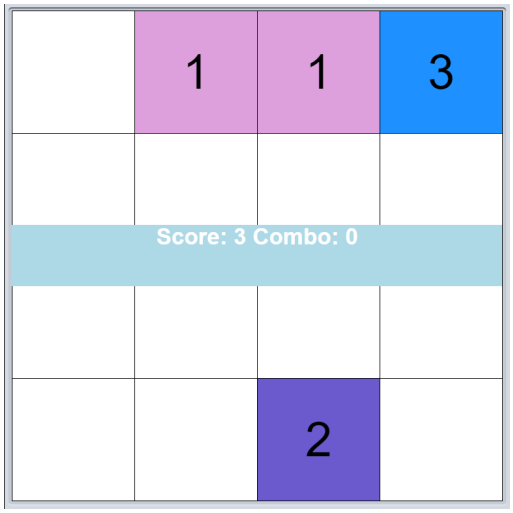
Hit space to start a new game!

3. **Score / How to combo!**

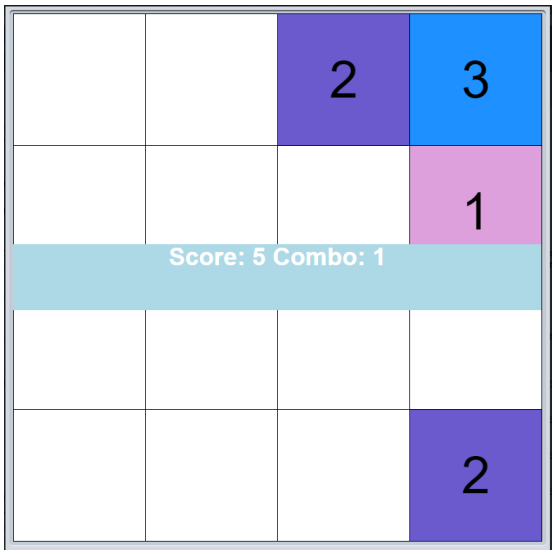
When tiles are merged together, your score increases by the sum of the two tiles. For example, merging at 1 and 1 will increase your score by 2.

If you are able to merge two tiles, and then merge that resulting tile with another tile the next move, you will start a combo! For example, if we take the situation from the above paragraph, and combine our 2 with a 3 on the turn after we created the 2, we will have a combo of two! When you have a combo, the sum added to your score gets multiplied by the combo number! So, (2 + 3) * 2 = 10! Keep comboing your moves to get the highest score possible!

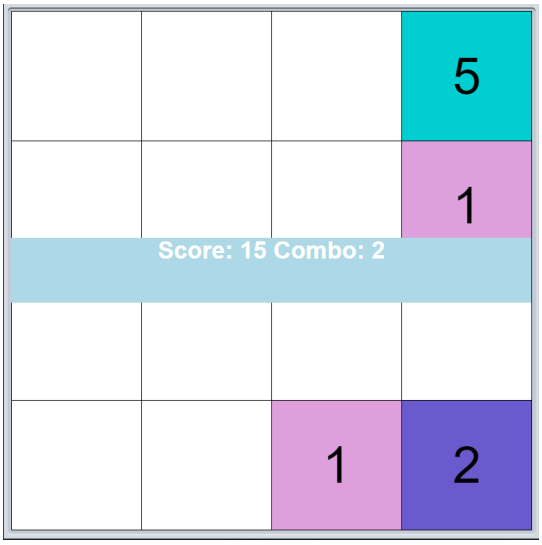
Combo Example:
Starting Score: 3



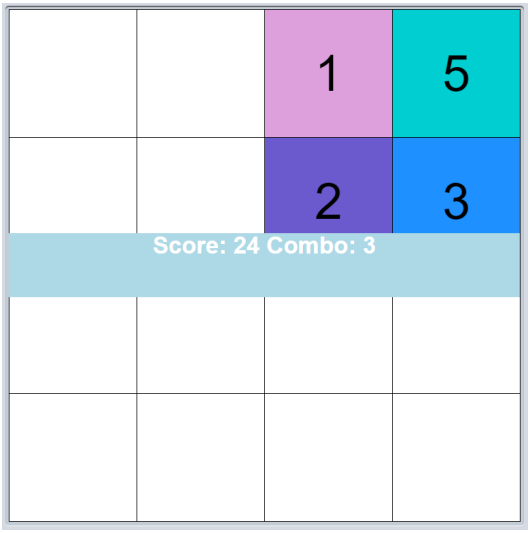
Move: Right
Combo: 1
Score: 3 + ((1 + 1) *1) = 5



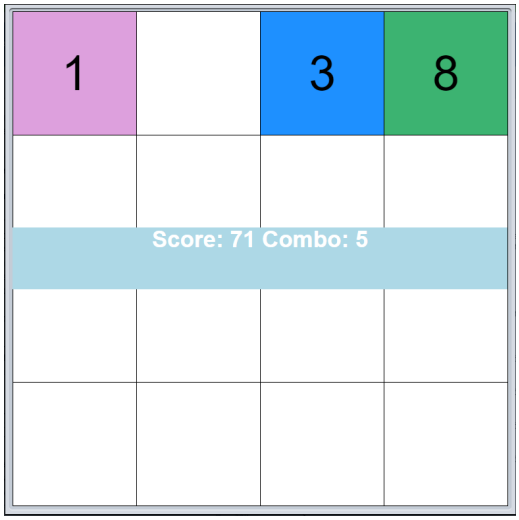
Move: Right
Combo: 2
Score: 5 + ((3 + 2) * 2) = 15



Move: Up
Combo: 3
Score: 15 + ((1 + 2) * 3) = 24



Move: Up
Combo: 5! (We combine 1 +2 and 5+3!)
Score: 24 + ((3 + 5) * 4) = 56 + ((1 + 2) * 5) = 71



Keep comboing to reach the highest possible score!

4. **EX Blocks**

Keep your eyes peeled for EX Blocks! These rare golden blocks have a 1% chance of spawning every turn, and can combine with any other block! Plan your moves carefully so you can maximize its value!