**ICS3U Final Project - 2048**

**1. Game Background**

|  |  |
| --- | --- |
| 2048 is a single-player online and mobile game created in March 2014 by 19-year-old Italian web developer Gabriele Cirulli, in which the objective is to slide tiles on a grid to combine them and create a tile with the number 2048. Cirulli created the game in a single weekend as a test to see if he could program a game from scratch.  The following link can help you understand the game more:  <https://www.youtube.com/watch?v=8Jl5EKm4kUg> |  |

**2. About the Game**

|  |
| --- |
| Rules:   * Played on a 4x4 grid. * When user pressed any of the four keys (*up*, *down*, *left* or *right*), all the tiles will shift to that direction. * If two tiles of the same number hit while moving, they will merge into a tile with the total value of the two tiles collided. * Every turn (each directional key pressed), a new tile will randomly appear in an empty spot on the board with a value of either 2 or 4.   Ending Condition:   * *Winning*: Player creates a tile of value 2048. * *Losing*: When player has no legal moves (there are no empty spaces and no adjacent tiles with the same value).   Score:   * Game will keep track of player’s score. Points will be given when tiles merge and will increment by the combined tile value. |

**3. Project Specification**

|  |  |  |  |
| --- | --- | --- | --- |
| **game\_config.txt**  This file contains information that the program will read in and configure the game accordingly.  The **first line** of the configuration the number system the game will be using. There are two settings:   * **VALUE\_GRID** – The grid array that keeps track of the game will be represented by the value itself. (Example, value 2, 4, 8, 16 and etc. will be stored in the array) * **INDEX\_GRID** – The grid array that keeps track of the game will be represented by the index that of the level (Example, index 0 = 2, index 1 = 4, index 2 = 8, index 3 = 16 and etc.)   The **second line** represents the winning condition, where the level number indicates the level the user must reach in order to win. This number must set according to the setting you have on the **first line**.   * If **VALUE\_GRID** setting is used, you must indicate the winning level condition correspond to the values on the grid. * If **INDEX\_GRID** setting is used, you must indicate the winning level condition correspond to the level index.   *Sample*:   |  | | --- | | VALUE\_GRID  2048 |   or   |  | | --- | | INDEX\_GRID  10 |   **gui\_config.txt**  This file contains information that the program will read in and configure the game GUI accordingly.  The **first line** of the configuration file is the filename of the logo image to be used.  The **second line** is the number of game tiles the game will be using. For example, if 2048 is the winning value, from 2 to 2048, there are 11 tile values. So use 11 to indicate that.  The **subsequent lines** are the filenames of the slot values image. The images are in the images folder. If 11 were specified on the second line, the configuration must have 11 lines of image filenames after the second line.  *Sample:*   |  | | --- | | images/iconLogo.jpg  11  images/icon2.jpg  images/icon4.jpg  images/icon8.jpg  images/icon16.jpg  images/icon32.jpg  images/icon64.jpg  images/icon128.jpg  images/icon256.jpg  images/icon512.jpg  images/icon1024.jpg  images/icon2048.jpg |   **Game2048Listener.java**  This class handles the key pressed from the user. You do not have to touch this file at all. However this class will use the following methods in **Game2048**, which you must implement. The methods are the following:   * newSlot() * play(int direction)   **Game2048GUI.java**  The class creates the GUI of the game. It represents the game board using a 2D array with NUMROW rows and NUMCOL columns. **First index** represents the row and **second index** represents the column. Index [0][0] representing the top left slot and [0][NUMCOL - 1] representing the top right slot.  This class provides several methods for **Game2048** to update the GUI. Please refer to the API for the descriptions of the methods. However the most important methods **Game2048** will use are:   * clearSlot(int row, int col) – Erase a specific slot on the grid. * setGridByIndex(int[][] grid) – It takes in the 2D array grid and having the GUI displays it. The array must represent the grid with slot index (their corresponding levels). * setGridByValue(int[][] grid) – It takes in the 2D array grid and having the GUI displays it. The array must represent the grid with slot value (their corresponding numbers). * setNewSlotBySlotIndex(int row, int col, int index)– Set a single slot on the grid with a specified slot value represented by the level index. * setNewSlotBySlotValue(int row, int col, int value)– Set a single slot on the grid with a specified slot value represented by the value. * showGameOver() – Show the game over textbook. * showGameWon() – Show the game won textbook. * setScore(long score) – Set the score to be displayed on the GUI.     You must complete the **initConfig()** method. Initialize the following variables with information reading from the configuration file. The filename of the configuration will be in the GUID\_CONFIG\_FILE variable.   * logoIconFile (String): the filename of the logo graphic of the game. * slotIconFile (String[]): the filenames of the slot graphics files of each tile number.   Don’t forget to create the configuration file to test your code.  **Game2048.java**  The file contains the implementation of the logic of the game. You are responsible to write this.    You are provided with the variable declarations. Please read over the documentation in the code for detailed descriptions.  You are also provided with the signatures of two methods:   * **Game2048 constructor**: it contains the initialization of variables. Creations of grid array, scores, and etc should be added here. (In particular, the winning level should be initialized with the value on game\_config.txt”) It should also insert the first number tile. * **newSlot()**: This method places a new number tile on a random slot on the grid. It is called every time a key is released. Make sure to check if a new tile should be inserted first (a slide or a tile combination has occurred previously) * **play(int direction)**: This method is the core of the game. It implements the action to be taken after an arrow key of the specified direction is pressed.   A Game2048GUI object named gui is created in this class. Make sure to call the appropriate methods of gui in newSlot and play to update the GUI. For example gui.setNewSlotBySlotValue(0, 0, 2) will set the slot at the top left corner (row #0, column #0) to the tile corresponding to the value 2.  The core of the game logic would be inside the play() method, but remember to separate tasks into different methods. So in Game2048, create methods when necessary to improve your code. |

**4. Extra Information**

|  |  |
| --- | --- |
| **Keywords and Conventions:**  **C:\Users\Kevin\Desktop\GameInfo.jpg** | **“images” folder:**  The folder contains the images of the slots as well as their respective .psd file. You can use Photoshop to modify the images to suit your style.  **“api” folder:**  This folder contains the API of Game2048GUI.java and Game2048Listener.java classes. |

**5. Design Document**

|  |
| --- |
| To produce any quality software, the design phase is done before the implementation phase (*coding*). The design phase allows clients to confirm their ideas with the development team to avoid confusion and wasted implementation time.  **Document Format**  The document must be in the following format:   * Cover Page   + - Project logo     - Project name     - Document title (Design Document)     - Your name     - Date     - Course code     - Teacher’s name * Table of Content * Footer   + - Document title     - Project name     - Page number (page x of y), starting on 2nd page   **Document Content**  The document must include the following sections:  **Screens**  Provide a detailed sketch of the screen. Note that use of copyrighted pictures is not permitted. Although we are not all artists, drawing your own pictures that are of high quality will yield a better implementation mark than downloading existing work.  **Graphic Pictures**  In this section, include a snapshot of each one of the graphic pictures that you will include in your game. The size (in pixel) of the player pieces (which are squares in shape) is specified as constant in the Game2048GUI.java.  *Note*: The logo’s image is 216x150 pixels.  **Methods Organization**  This section should list all the methods to be implemented and a brief description for each of them. |

**6. Assessment**

**A) Design Document (20%):**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Levels** | **0** | **1** | **2** | **3** | **4** |
| **Communication** | | | | | |
| **Clarity, Grammar & Spelling** | * many errors in spelling and grammar * difficult to comprehend explanations | * some attention to spelling and grammar * some explanations are comprehensible | * few spelling or grammar errors * majority of the explanations are comprehensible | * very few spelling or grammar errors * most explanations are comprehensible | * no spelling or grammar errors * explanations are very comprehensible |
| **Layout & Formatting** | * at least one formatting specification met * attempt to make layout easy to follow | * met some of the formatting specifications * some of the layout is easy to follow | * met half of the formatting specifications * half of the layout is easy to follow and aesthetically appealing | * met most formatting specifications * most of the layout is easy to follow and aesthetically appealing | * met all formatting specifications * layout is very easy to follow and aesthetically appealing |
| **Content** | | | | | |
| **Content** | * One section completed | * Equivalent of two sections complete | * Content is almost complete, there is signs of creativity and good planning | * Content is complete, shows creativity and good planning | * Content is complete, very creative and well planned |
| **Method Organization** | * No evidence of any planning on method organization | * Limited evidence of planning on method organization | * Some evidence of planning on method organization | * Methods are well planned but not all details are included | * All methods are well planned and all details are included |

**B) Code Implementation (80%):**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Levels** | **0** | **1** | **2** | **3** | **4** |
| **Functionality** | | | | | |
| **Initialize by config file** | Unable to initialize the game with the config files | Initializes the game with the config files with some error | Correctly initializes the game with the config files |  |  |
| **Insert new tile** | Unable to insert a new tile | Able to place a tile of randomized value on a non-randomized slot (*or vice versa*) | Able to place a tile of randomized value on a randomized slot | Insert the new tile only in appropriate situation (after tiles are slided or combined) |  |
| **Validation of move (slidable / combinable)** | Unable to validate a move | Correctly validate moves in one direction | Correctly validate moves in two directions | Correctly validate moves in three directions | Correctly validate moves in all directions |
| **Slot Sliding** | Does not slide correctly at all | Only one direction slides correctly | Two directions slide correctly | Three directions slide correctly | All directions slide correctly |
| **Slot Combining** | Does not combine correctly at all | Slots do not combine and increment value correctly most of the time | Slots combine and increment their value correctly half of the time | Slots combine and increment their value correctly with one error. (*Either unable to increment or unable to combine occasionally*) | Slots combine and increment their values correctly |
| **Winning the game** | Unable to determine if player wins. | Occasionally able to determine if play wins the game. | Able to determine if player wins the game |  |  |
| **Losing the game** | Unable to determine if player loses. | Occasionally able to determine if play loses the game. | Able to determine if player loses the game |  |  |
| **Coding Style** | | | | | |
|  |  |  |  |  |  |
| **Constants** | Unable to use constants appropriately | Little constants used appropriately | Some constants used appropriately | Most constants used appropriately | All constants used appropriately |
| **Methods** | No methods were created and used | Little methods created and used effectively | Some methods created and used effectively | Most methods created and used effectively | All methods created and used effectively |
| **Documentation** | | | | | |
| **Program Header** | No documentations were used | Documentations were used |  |  |  |
| **Method Header** | No methods has documents header | Minimal methods has documents header | Some methods has documents header | All methods has documents header |  |
| **Description of Algorithm** | No descriptions through the code | Minimal descriptions through the code | Some descriptions through the code | Abundant descriptions through the code |  |
| **Interface** | | | | | |
| **Game Logo** | No logo or using the default logo | Unoriginal log used. (eg: From internet) | Original and creative game logo used |  |  |

**Due Date**

Wednesday January 13, 2016

* Design Documentation
* All graphics files are created and sized properly

Monday January18, 2016

* initConfig method in Game2048GUI.java completed (as a result, the game grid with the logo will be displayed)
* newSlot method completed
* all initializations are completed in the Game2048 constructor (as a result, the initial number tile is placed on the grid and the game is ready to begin)

Wednesday, January 20, 2016

* validation of move - test if a move to the different directions is possible (slidable / combinable)

Thursday, January 21, 2016

* slide the grid in all directions

Friday, January 22, 2016

* combine number tiles
* update the score

Monday, January 25, 2016

* check if player won the game
* check if player loss the game
* Final Product completed