**JAVA 2048 DOCUMENTATION**

**Introduction**

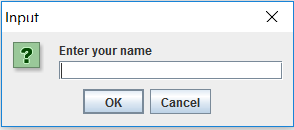
This documentation describes a JAVA implementation of the well-known game names 2048. Like any other game, this one is designed for fun but also requires some strategic thinking.

The purpose of the game is to move the tiles on a 4x4 board in order to obtain the highest ranked tile, whose value is 2048. The game starts with 2 tiles randomly generated anywhere on the board and their value may be 2 or 4 (also randomly chosen).

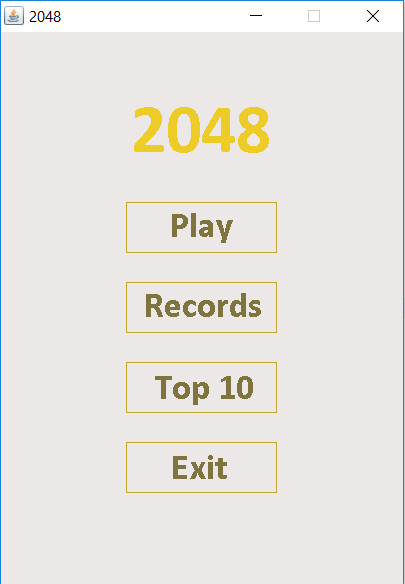
The game only requires arrow keys. When the user presses an arrow key, all the tiles on the board will shift in that direction. When 2 tile of the same value are adjacent on the movement direction, they combine in a single of double value and that value is added to the user’s score. When no more tiles can combine on a direction, there is no free space for spawning a new tile and no 2048 tile created, the game is over. If the user creates a 2048 tile the game is also over but that user is declared winner.

**Use cases**

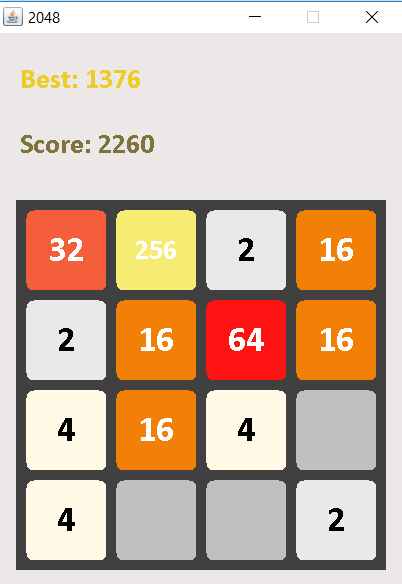
When the app is opened, the user is prompted to enter a name used as an identifier for storing the score.



This step is compulsery. The user cannot access the game without a name. If he chooses not to enter one, a window will pop up tell him that this step can’t be skipped. After a name is chosen, the app menu opens and the user is welcomed by several options.

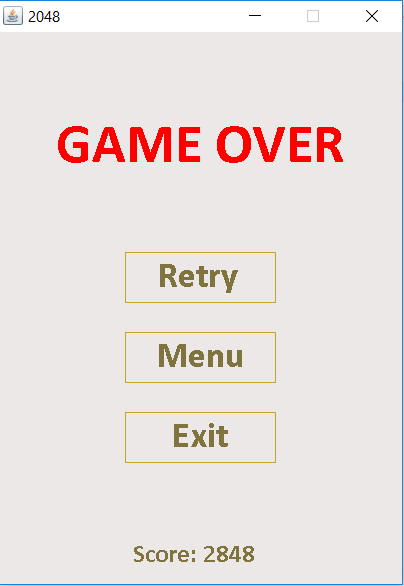


The menu options are pretty self explanatiory. “Play” button simply starts a game session which can end up looking like this:

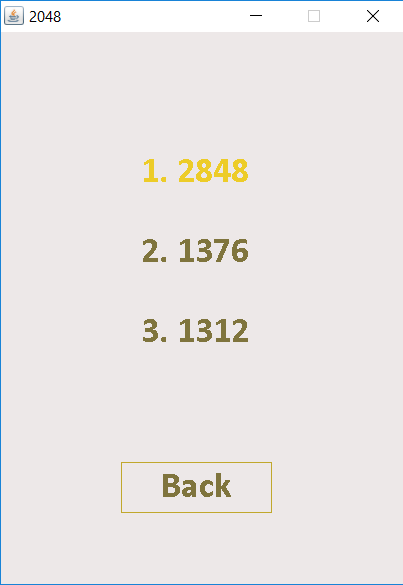
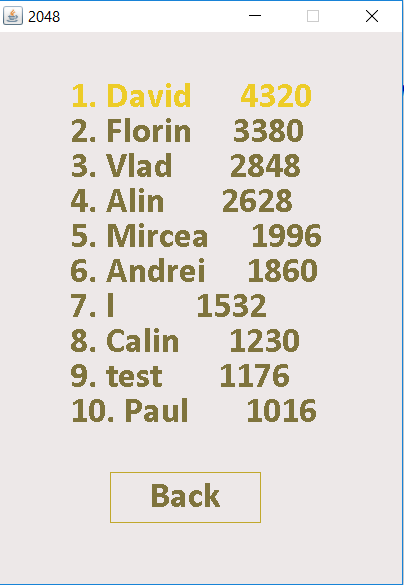


On the upper right corner every user’s best score is printed in gold. Below it there is the score during the current gaming session.

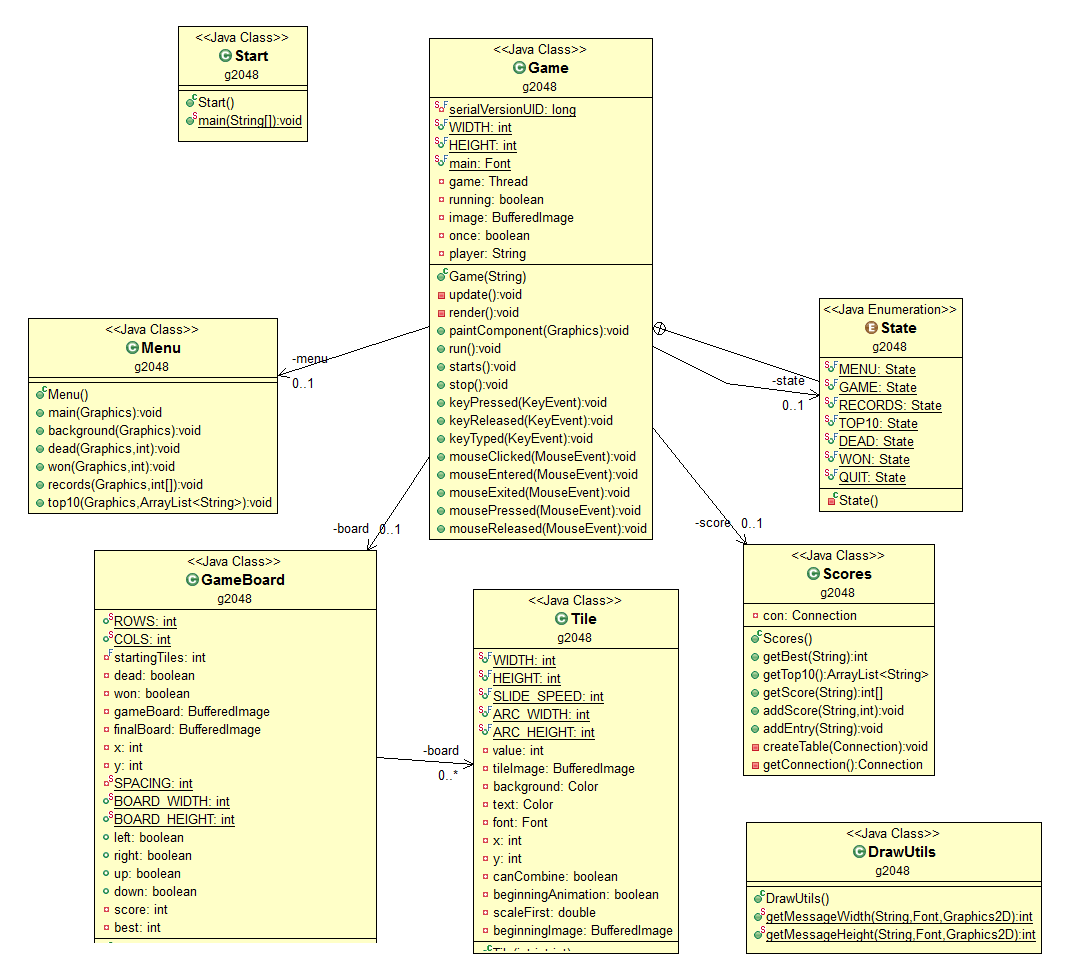
When the user loses the game, the app will automaticaly notice it and print a game over menu. Here the user have the options to play the game once again, go back to the main menu or exit the game. The user is also reminded the score during the current session.



The other 2 options in the main menu are score related. The first one, names “Records” shows the user’s highest 3 scores. The other on, “Top 10”, is a leaderboard of the highest 10 scores ever set along with the players.

**Design & classes**



The above diagram shows all the classes used in development of the game.

Class **Game** is the class which implements the game’s thread and is also responsible with the game loop. Once the function **run()** is automatically called from the thread’s **start()**  function the game loop is enabled, therefore the function **update()** and **render()** are called each other 60 times per second. Function **update()** is responsible with the update of the game logic and function **render()** renders the GUI. It also implements the listeners for the keyboard and mouse.

Class **Menu** is exclusively used to draw each menu’s layout. All the drawing is done on a buffered image which is then painted on the JPanel.

Class **GameBoard** is the class which implements the game’s logic and draw the game board background. Some implementation details will be described in the following section.

Class **Tile** contains each tile’s characteristics (dimensions, background color, value). It is used to draw the tile with the corresponding characteristics and also implements the animations (tile slide speed and spawning animation).

Class **Scores** manages the scores in the connected database. It is responsible for adding new users, storing scores in descending order and creating the leaderboard.

Class **DrawUtils** is used to fit the text in certain graphic context. In this project, DrawUtils is called to fit the text on each tile.

**Implementation details**

The code itself is highly commented and the names are self explanatory. Though the most intresting part of the game is the logic itself and it deserves to be detailed. The table itself is stored in a 4x4 Tile matrix. When the user presses an arrow key, the method **moveTile()** from the class **GameBoard** is called. Based on the direction, the tiles can perform 2 types of movements: horizontal or vertical. By default it is assumed that none of the tile can move. Then, based on which arrow key was pressed, the method iterates through the Tile matrix, calling the function **move()** for each element. The function **move()** simply checks if that tile can move to some new coordinates computed based on the two types of movement (horizontal or vertical) which are determined by the arrow key pressed. A tile can move if it has at least one empty slot on the movement direction or if it can cobine with a tile on that direction. If any of this case is true, the function **move()** returns true indicating that the tile moved and also updates the tile’s characteristics (doubles the value if it combined or stores the new tile’s x and y coordinates. After a movement was performed, the method **move()** spawns a new tile and checks wether the player won or lost the game.

To implement the movement animation, from method **update()** is called **resetPosition()** for each element of the Tile matrix. The using the row and column of the element in the matrix, the function computes the real x and y coordinates of the tile on the screen and compares to the x and y coordinates stored in that tile object. Based on the moving direction, the difference between the coordintes (x stored in object – x computed based on the column and y stored in object and y computed based on the row) may be positive or negative. If it is negative, the tile’s **slide\_speed** is added to that coordinate in order to reach the real position on the scree, otherwise the **slide\_speed** is subtracted. The **slide\_speed** simply tells how many pixels each tile travels during one iterations until in reaches the real coordinates (the ones computed from the row and column number in the Tile matrix).

**Further improvements**

As further improvements we can measure the time needed for a player to build a 2048 block. Also, we can implement an AI which can try to find a solution.