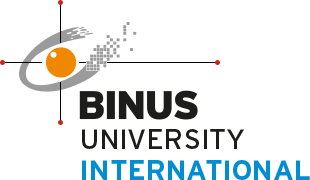
**Object Oriented Programming Final Report:**

**“2048 Game Using Java”**

****

**Lecturer: JUDE JOSEPH LAMUG MARTINEZ, M.C.S**

**Prepared By:**

**Felix Ferdinand -2602108842 - L2AC**

**Binus University 2023**

**Tables of Content Pages**

**Background…………………………………………………………………………...3**

**Solution Design………………….…………………………………………………….4**

**Program Analysis……………….…………………………………………………….5**

**Code Explanation…………..………………………………………...……….5**

**Screenshots Of Code………………………………………………………….7**

**Evidence……………………………………………………….……………………..18**

**Screenshots of Application…...……………………………………….……..18**

**Conclusion……………………………………………………………...…….19**

**Background**

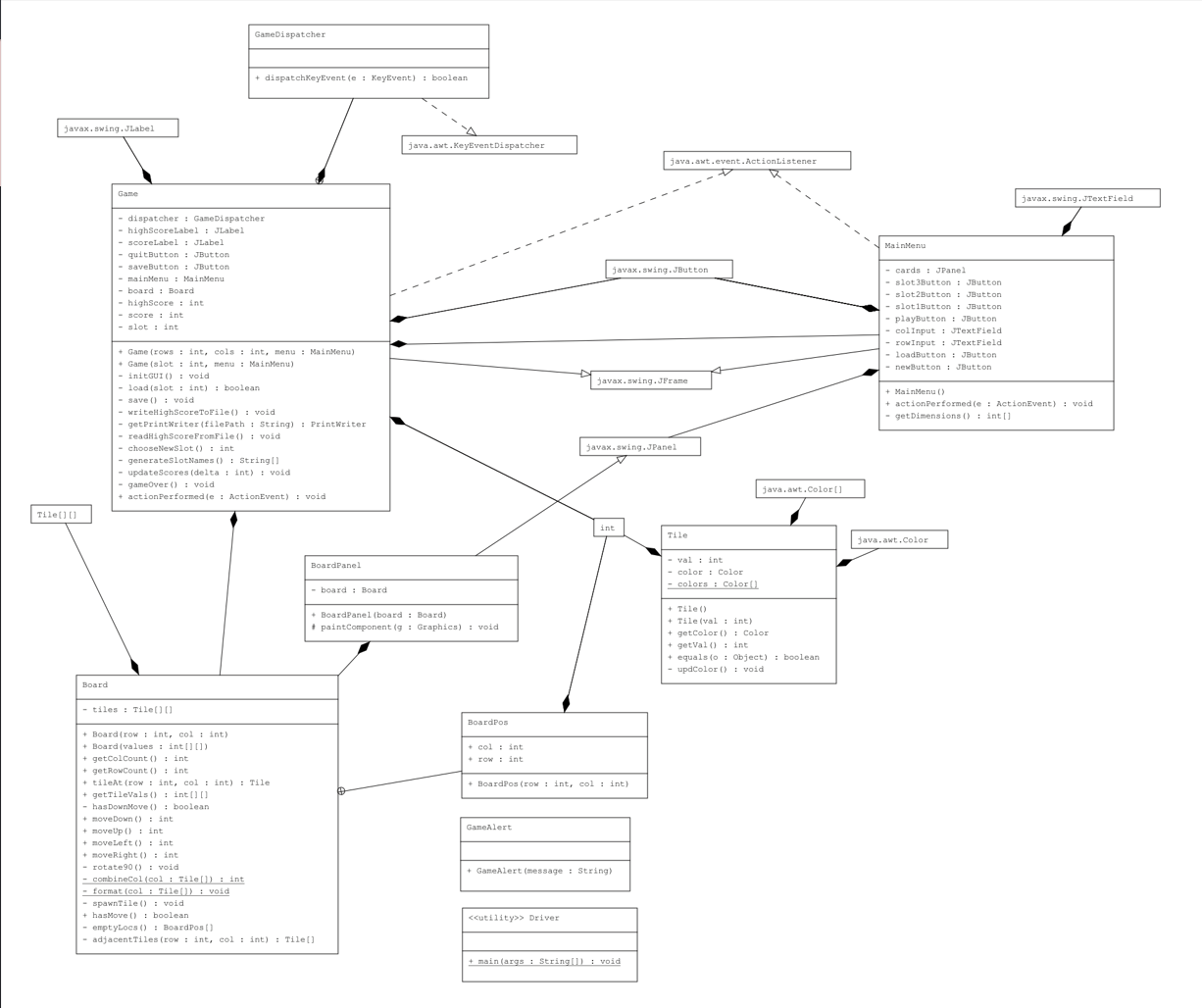
“2048” is a tile-sliding puzzle game that was first released during the march of 2014 by an Italian web developer named Gabriel Cerulli that was famous for its presence in the mobile game industry. The creator of this game had originally implemented a simple objective to the game itself, to combine like-numbered tiles until the player could reach the desired and coveted “2048” tile. The game has offered several features for the quality-of-life experience designed for the benefit of the players such as a scoring system which keeps track of the user’s current high score and achievements alongside a redo button which resets the game to the player’s last tile move.

**Personal Experience**

Having played the original “2048” game on my mobile device for over 8 years, it is safe to assume that I had an avid experience in creating strategies and techniques that would help me reach the 2048 tile or even more. Overtime, the original game has been proved as stale and boring due to its limited features and mechanics that are offered in the game. In this project, I am to create a similar implementation of the game while adding several features that would help increase its overall gameplay and experience for the original players.  
  
Project Goals:

1. Mechanics: The mechanics of this implementation would be relatively the same as compared to the original game to keep its simplistic and addictive nature at touch.
2. Features: There are several features that I would have added for a more engaging and lifelike approach at connecting the players with the game itself.
3. Aesthetics: There would be a little feature inside the code that would change the overall look and aesthetic of the game to keep players from getting bored while playing the game.

**Solution Design**

****

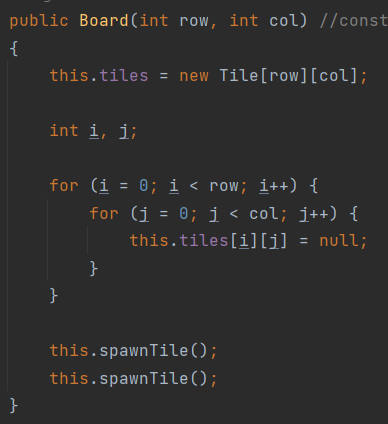
The solution design is shown as a class diagram in order to capture and define each structure of classes and functions. This would also highlight the relationship each class had with its respective functions.

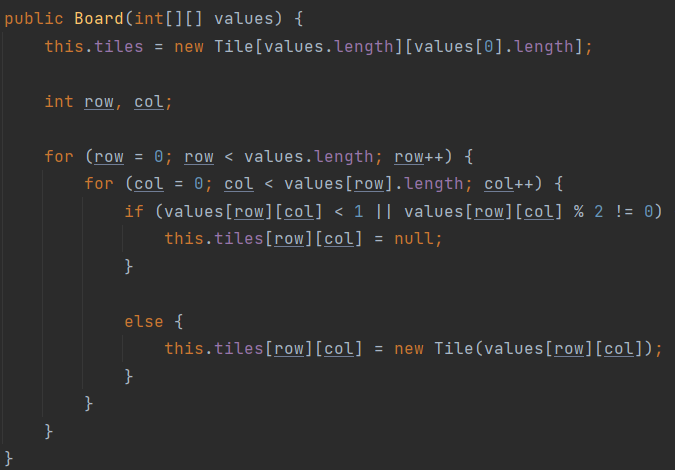
**Program Analysis**

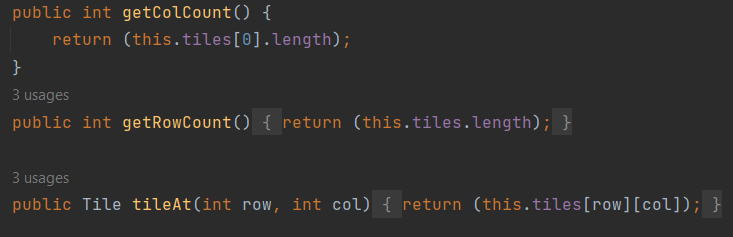
The program analysis will cover each functions that are available in the code itself, in addition, each functions and code block that is essential to the program itself will be discussed in the analysis below:

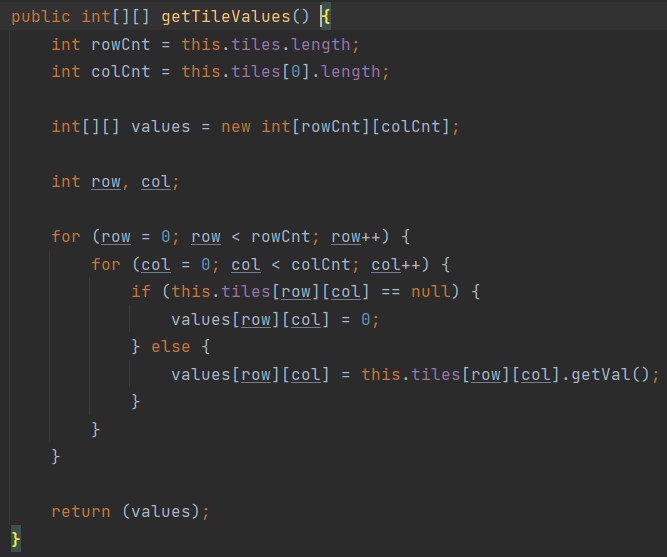
* Board(int row, int col): This is a constructor for the board class. The board works by initialising a game grid using a 2-dimensional array. A nested for loop will then be iterated over the parameters in order to generate the empty grid based on the size of the rows and columns. The spawnTile() will then called to randomly add two tiles into the empty grid
* Board(int[][] values): This is another constructor for the board class which takes a 2-dimensional array called values as a parameter. A nested for-loop will iterate over the length of the value array to generate the rows and the values[rows] to initialise the column. The if clause will return null if the row and col index inside of the tile array = 1 or if it's perfectly divisible by true otherwise the Tile will be initialised based on the previous array value.
* getColCount(): returns the number of column inside the game grid
* getRowCount(): This function returns the number of rows inside the game grid
* tileAt(int row, int col): returns the tile object at the specified row and column that is inside the game grid.
* getTileValues(): This function returns a 2-dimensional array representing the current values of all the tiles that are currently available in the game grid. It first converts the tile object to the corresponding value if the specified rows and columns are not equal to a null value.
* hasDownMove(): This is a private function that checks whether there is a valid “down” move available on the game grid. The function works by checking whether a tile is mergeable or not, if tile is mergeable, the function returns true or if there are empty spaces in the grid, the function returns true otherwise the function would return false.
* moveDown(): this function moves all of the tiles into the down direction. It combines tiles that have the same value and updates the score accordingly. In conclusion, this function returns the total score earned from merging the tiles together.
* moveUp(): this function moves all the available tiles that are on the game grid in the “up” direction by rotating the grid 180 degrees and performing a down move. The grid will then be rotated back 180 degrees to its original form, the function returns the total score that is earned from merging tiles.
* moveLeft(): Similar to the moveUp() function, this function moves all of the tiles into the left direction by rotating the grid by 90 degrees clockwise and performing a down move and rotating the grid back to its original form. This function returns the total score earned from merging the tiles due.
* moveRight(): This function moves all of the available tiles into the right direction by rotating the grid by 90 degrees counterclockwise and performing a down move which then rotates the grid back 90 degrees clockwise. This function returns the total score earned from merging the tiles.
* rotate90(): This function rotates the grid by 90 degrees clockwise by swapping the rows and columns of the grid.
* combineCol(Tile[] col): This function takes the col array as a parameter and combines all of the tiles in a single column of the game grid. This function merges all the adjacent tiles with the same value and returns a total score earned from merging the tiles.
* format(Tile[] col): This function is a helper function that takes the col array as a parameter and moves all of the tiles in a single column of the grame down, which eliminates every empty space.
* spawnTile(): This is a function that first checks the empty locations that are available in the game grid, if there are any, the function would spawn either a 2 or a 4 number tile randomly on the game grid.
* hasMove(): This is an important function that determines whether any moves are possible to execute by checking for like-numbered adjacent tiles that are able to be merged or any empty locations that are available in the game grid.
* emptyLocations(): This is a private helper function that returns an array of empty locations that are available in the game grid by scanning the entire rows and columns of the grid using a for-loop.
* adjacentTiles(): This function is a helper function that returns any available adjacent tiles by checking every possible axis that is available on the grid, if there are any, the function returns an array of adjacentTiles to the specified rows and columns inside the game grid.
* BoardPos(): This is a constructor for the boardPos class that marks or indicates a position in the game grid using rows and columns.

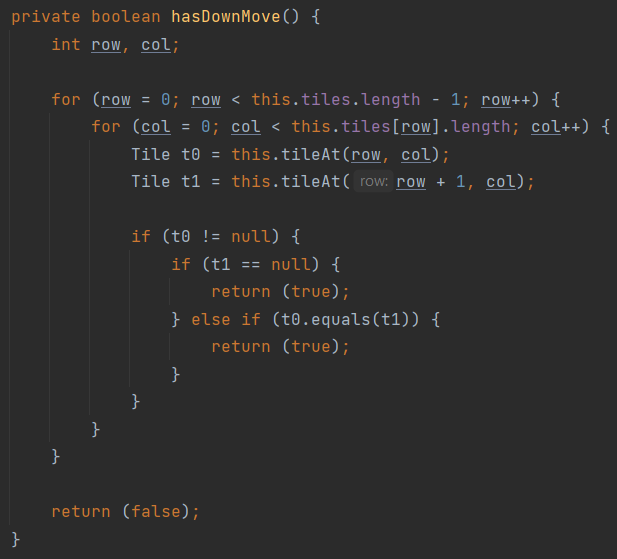
Screenshots of the code:

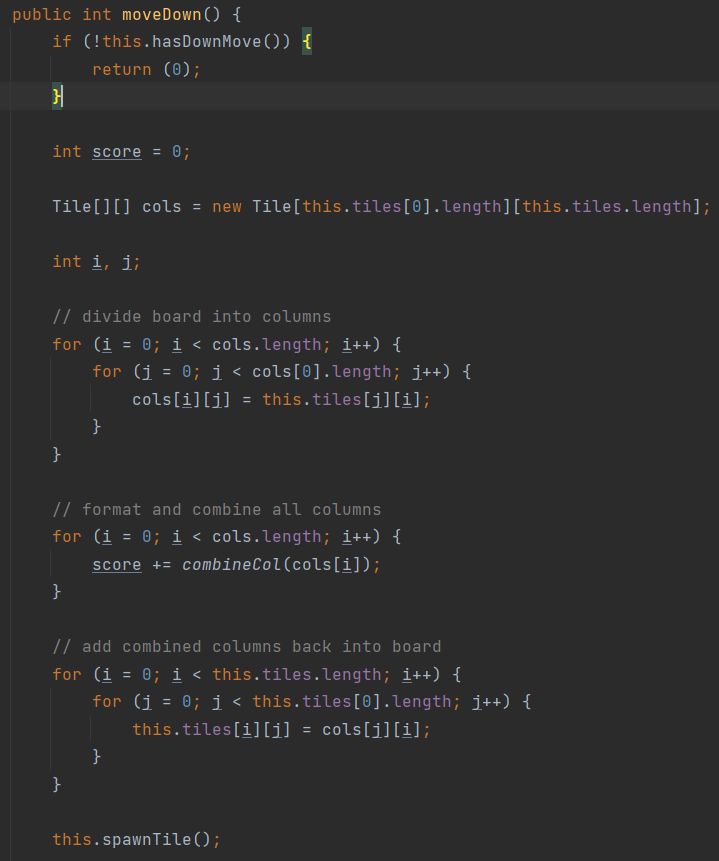


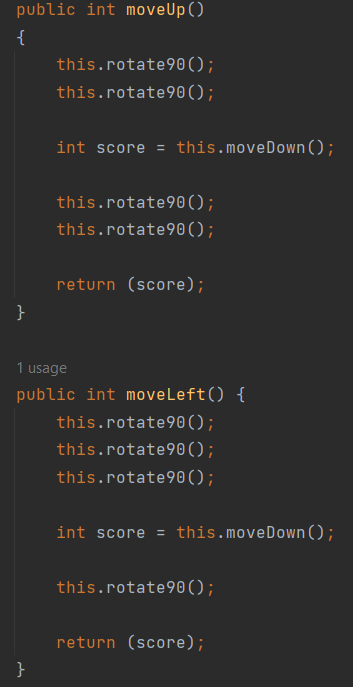
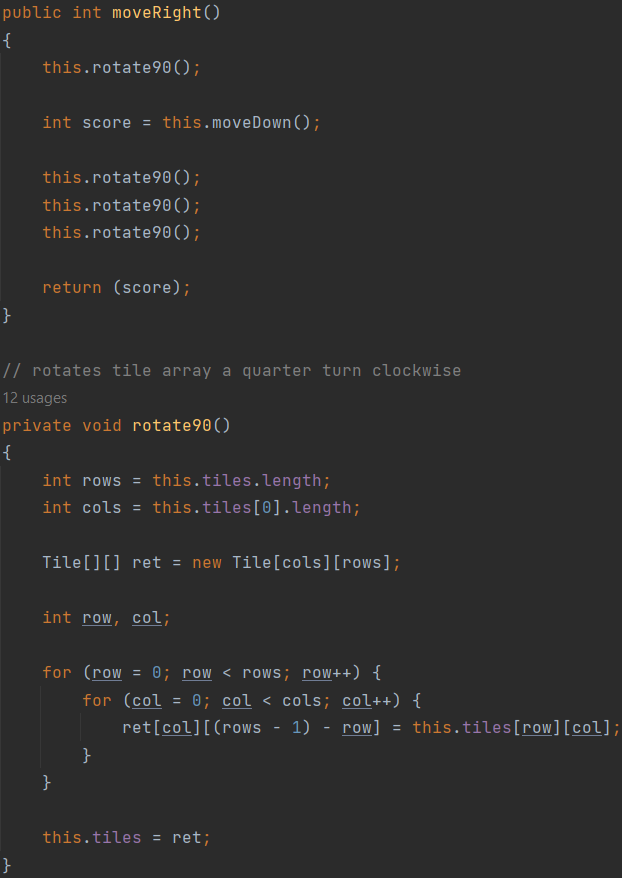


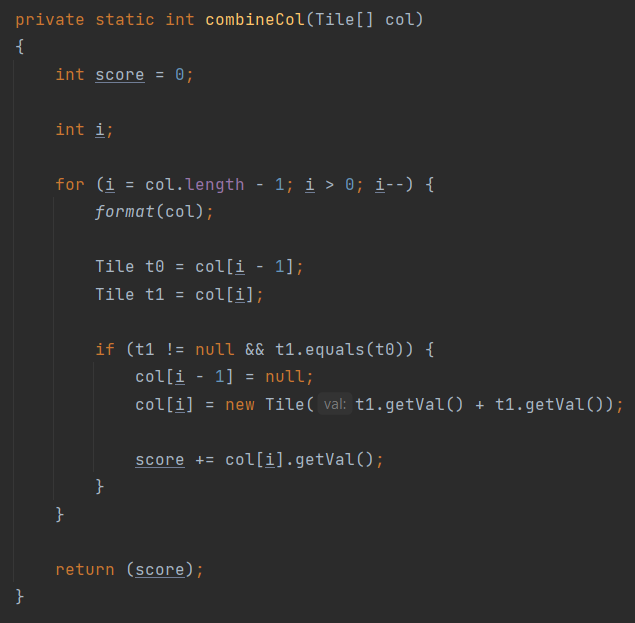


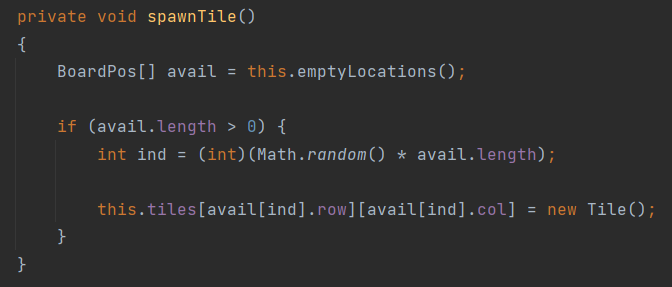




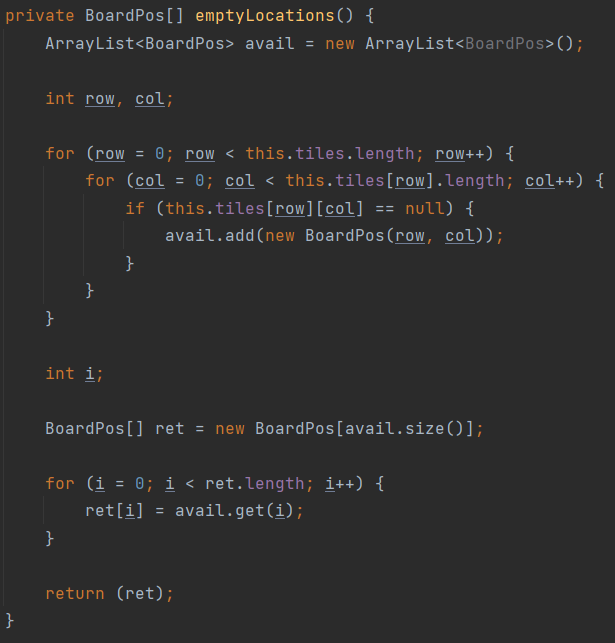


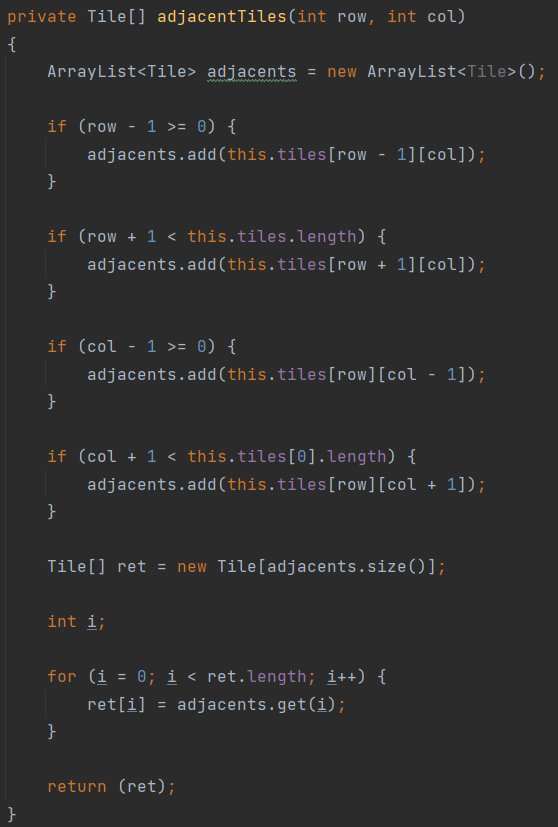
 











**Evidence**

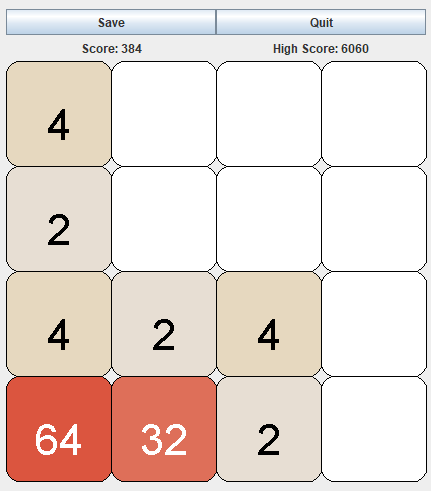
This section of the report displays the evidence of a working application, a conclusion and also a github link to the code.

Github link: <https://github.com/ferd78/2048-OOP-FP>

Video Demo:

https://drive.google.com/file/d/1nt2MG5rUlxn-d4RuWTI4RzLuy20fEoCf/view?usp=drive\_link

Application:





**Conclusion**

In conclusion, I have learned and understand important aspects of object oriented programming in order to implement correct usage regarding the use of polymorphism and inheritance. In addition, I have strengthened my knowledge regarding the use of the Java programming language and its essential libraries such as Java Swing, in order to create window-based applications.