OOP2 homework 3
Maya Yagan 200101124
06.06.2023

Class SlideGame

Explanation:
This class represents the sliding game 2048.

```
> SlideGame game = new SlideGame()
//a new instance has been created which means that the constructor is working
properly
> int[][] x1 = {{0,0,1,1},{1,1,0,0},{2,0,2,2},{4,2,0,2}}
//array x1 will be used to test method move left
> game.moveLeft(x1)
//testing method moveLeft. It should move all the elements to the left then add
any adjacent equal elements.   0,0,1,1 -> 2,0,0,0
                                       1,1,0,0 -> 2,0,0,0
                                       2,0,2,2 -> 4,2,0,0
                                       4,2,0,2 -> 4,4,0,0
{ { 2, 0, 0, 0 }, { 2, 0, 0, 0 }, { 4, 2, 0, 0 }, { 4, 4, 0, 0 } }
//returned as expected ☑
> int[][] x2 = {{0,0,0,0},{0,0,0,8},{0,1,1,2},{8,8,8,8}}
//we will test some other combinations of elements
> game.moveLeft(x2)
//expected results:                    0,0,0,0 -> 0,0,0,0
                                       0,0,0,8 -> 8,0,0,0
                                       0,1,1,2 -> 2,2,0,0
                                       8,8,8,8 -> 16,16,0,0
{ { 0, 0, 0, 0 }, { 8, 0, 0, 0 }, { 2, 2, 0, 0 }, { 16, 16, 0, 0 } }
//returned as expected ☑
> int[][] x3 = {{8,0,1,4},{2,1,1,8},{2,0,0,2},{0,4,1,4}}
// more combinations
> game.moveLeft(x3)
//expected results:                    8,0,1,4 -> 8,1,4,0
                                       2,1,1,8 -> 2,2,8,0
                                       2,0,0,2 -> 4,0,0,0
                                       0,4,1,4 -> 4,1,4,0
{ { 8, 1, 4, 0 }, { 2, 2, 8, 0 }, { 4, 0, 0, 0 }, { 4, 1, 4, 0 } }
//returned as expected ☑
> int[][] x4 = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}}
> game.moveLeft(x4)
//the case of a full array with no moves
                                       1,2,3,4 -> 1,2,3,4
                                       5,6,7,8 -> 1,2,3,4
                                       9,10,11,12 -> 1,2,3,4
                                       13,14,15,16 -> 1,2,3,4
{ { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16
} }//returned as expected ☑
```

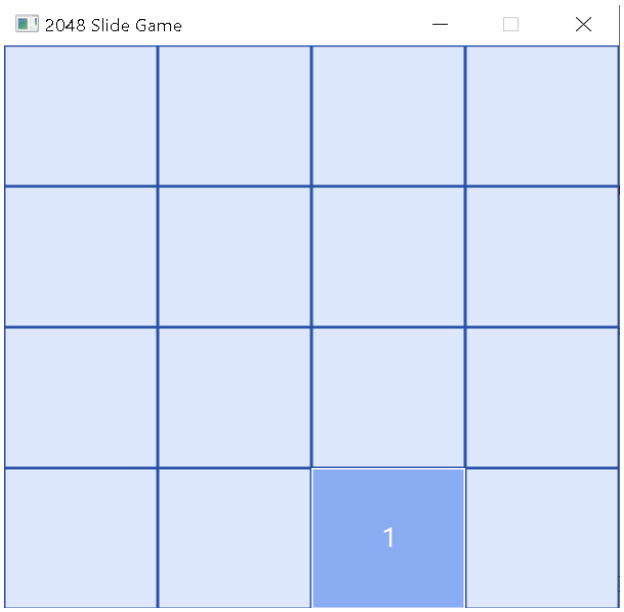
```
> int[][] x1 = {{0,0,1,1},{1,1,0,0},{2,0,2,2},{4,2,0,2}}
> game.moveRight(x1)
//testing method moveRight. It should move all the elements to the right then
add any adjacent equal elements.     0,0,1,1 -> 0,0,0,2
                                     1,1,0,0 -> 0,0,0,2
                                     2,0,2,2 -> 0,0,2,4
                                     4,2,0,2 -> 0,0,4,4
{ { 0, 0, 0, 2 }, { 0, 0, 0, 2 }, { 0, 0, 2, 4 }, { 0, 0, 4, 4 } }
//returned as expected ☑
> int[][] x2 = {{0,0,0,0},{0,0,0,8},{0,1,1,2},{8,8,8,8}}
> game.moveRight(x2)
//expected results:                  0,0,0,0 -> 0,0,0,0
                                     0,0,0,8 -> 0,0,0,8
                                     0,1,1,2 -> 0,0,2,2
                                     8,8,8,8 -> 0,0,16,16
{ { 0, 0, 0, 0 }, { 0, 0, 0, 8 }, { 0, 0, 2, 2 }, { 0, 0, 16, 16 } }
//returned as expected ☑
> int[][] x3 = {{8,0,1,4},{2,1,1,8},{2,0,0,2},{0,4,1,4}}
> game.moveRight(x3)
//expected results:                  8,0,1,4 -> 0,8,1,4
                                     2,1,1,8 -> 0,2,2,8
                                     2,0,0,2 -> 0,0,0,4
                                     0,4,1,4 -> 0,4,1,4
{ { 0, 8, 1, 4 }, { 0, 2, 2, 8 }, { 0, 0, 0, 4 }, { 0, 4, 1, 4 } }
//returned as expected ☑
> int[][] x4 = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}}
> game.moveRight(x4)
//expected results:                  1,2,3,4 -> 1,2,3,4
                                     5,6,7,8 -> 1,2,3,4
                                     9,10,11,12 -> 1,2,3,4
                                     13,14,15,16 -> 1,2,3,4
{ { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16
} }//returned as expected ☑
> int[][] x1 = {{0,0,1,1},{1,1,0,0},{2,0,2,2},{4,2,0,2}}
> game.moveUp(x1)
//testing method moveUp. It should move all the elements upward then add any
adjacent equal vertical elements.    0,0,1,1 -> 1,1,1,1
                                     1,1,0,0 -> 2,2,2,4
                                     2,0,2,2 -> 4,0,0,0
                                     4,2,0,2 -> 0,0,0,0
{ { 1, 1, 1, 1 }, { 2, 2, 2, 4 }, { 4, 0, 0, 0 }, { 0, 0, 0, 0 } }
//returned as expected ☑
> int[][] x2 = {{8,0,0,0},{4,2,0,8},{4,1,0,8},{8,8,8,8}}
> game.moveUp(x2)
//expected results:                  8,0,0,0 -> 8,2,8,16
                                     4,2,0,8 -> 8,1,0,8
                                     4,1,0,8 -> 8,8,0,0
```

```
                                    8,8,8,8 -> 0,0,0,0
{ { 8, 2, 8, 16 }, { 8, 1, 0, 8 }, { 8, 8, 0, 0 }, { 0, 0, 0, 0 } }
//returned as expected ☑
> int[][] x3 = {{0,0,0,0},{4,1,0,8},{0,1,0,1},{4,2,8,1}}
> game.moveUp(x3)
//expected results:                 0,0,0,0 -> 8,2,8,8
                                    4,1,0,8 -> 0,2,0,2
                                    0,1,0,1 -> 0,0,0,0
                                    4,2,8,1 -> 0,0,0,0
{ { 8, 2, 8, 8 }, { 0, 2, 0, 2 }, { 0, 0, 0, 0 }, { 0, 0, 0, 0 } }
//returned as expected ☑
> int[][] x4 = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}}
> game.moveUp(x4)
//expected results:                 1,2,3,4 -> 1,2,3,4
                                    5,6,7,8 -> 1,2,3,4
                                    9,10,11,12 -> 1,2,3,4
                                    13,14,15,16 -> 1,2,3,4
{ { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16
} }//returned as expected ☑
> int[][] x1 = {{0,0,1,1},{1,1,0,1},{2,0,2,2},{4,2,0,2}}
> game.moveDown(x1)
//testing method moveDown. It should move all the elements downward then add
any adjacent equal vertical elements.
                                    0,0,1,1 -> 0,0,0,0
                                    1,1,0,1 -> 1,0,0,0
                                    2,0,2,2 -> 2,1,1,2
                                    4,2,0,2 -> 4,2,2,4
{ { 0, 0, 0, 0 }, { 1, 0, 0, 0 }, { 2, 1, 1, 2 }, { 4, 2, 2, 4 } }
//returned as expected ☑
> int[][] x2 = {{2,4,1,1},{1,1,0,8},{1,1,2,8},{0,2,0,0}}
> game.moveDown(x2)
//expected results:                 2,4,1,1 -> 0,0,0,0
                                    1,1,0,8 -> 0,4,0,0
                                    1,1,2,8 -> 2,2,1,1
                                    0,2,0,0 -> 2,2,2,16
{ { 0, 0, 0, 0 }, { 0, 4, 0, 0 }, { 2, 2, 1, 1 }, { 2, 2, 2, 16 } }
//returned as expected ☑
> int[][] x3 = {{8,2,0,1},{0,0,0,1},{8,2,2,8},{0,0,2,0}}
> game.moveDown(x3)
//expected results:                 8,2,0,1 -> 0,0,0,0
                                    0,0,0,1 -> 0,0,0,0
                                    8,2,2,8 -> 0,0,0,2
                                    0,0,2,0 -> 16,4,4,8
{ { 0, 0, 0, 0 }, { 0, 0, 0, 0 }, { 0, 0, 0, 2 }, { 16, 4, 4, 8 } }
//returned as expected ☑
> int[][] x4 = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}}
> game.moveDown(x4)
```

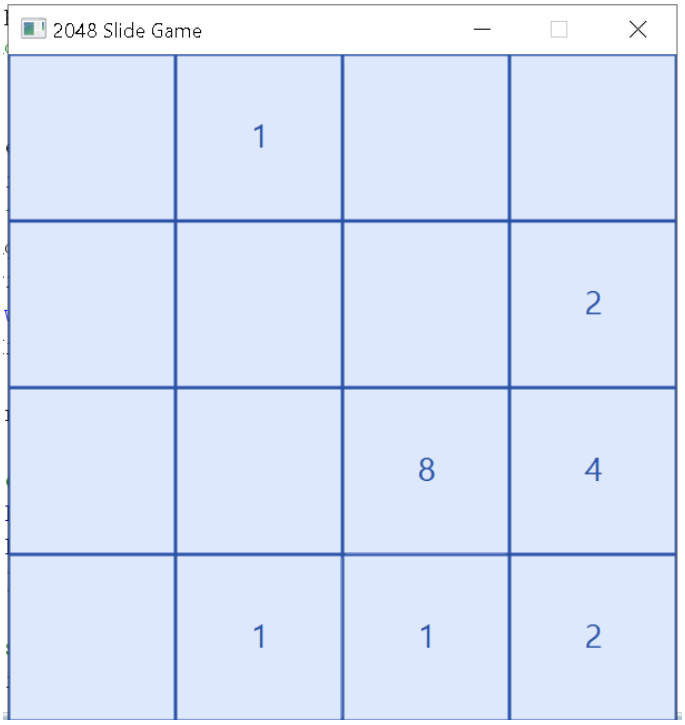
```
//expected results:                 1,2,3,4 -> 1,2,3,4
                                    5,6,7,8 -> 1,2,3,4
                                    9,10,11,12 -> 1,2,3,4
                                    13,14,15,16 -> 1,2,3,4
{ { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16
} }//returned as expected ☑
> int[][] array = {{0,0,1,1},{0,2,0,2},{0,1,1,1},{1,0,1,1}}
> game.slideElements(array, 'L')
//testing helper method slideElements. This method just moves the elements in
the given direction (no addition is done in this method). This method will be
tested only once for each direction because we have already tested this method
by testing the above 4 move methods
The directions are: Left = L, Right = R, Up = U, Down = D
                                    0,0,1,1 -> 1,1,0,0
                                    0,2,0,2 -> 2,2,0,0
                                    0,1,1,1 -> 1,1,1,0
                                    1,0,1,1 -> 1,1,1,0
{ { 1, 1, 0, 0 }, { 2, 2, 0, 0 }, { 1, 1, 1, 0 }, { 1, 1, 1, 0 } }
//returned as expected ☑
> game.slideElements(array, 'R')
                                    1,1,0,0 -> 0,0,1,1
                                    2,2,0,0 -> 0,0,2,2
                                    1,1,1,0 -> 0,1,1,1
                                    1,1,1,0 -> 0,1,1,1
{ { 0, 0, 1, 1 }, { 0, 0, 2, 2 }, { 0, 1, 1, 1 }, { 0, 1, 1, 1 } }
//returned as expected ☑
> int[][] array = {{0,0,1,1},{0,2,0,2},{0,1,1,1},{1,0,1,1}}
> game.slideElements(array, 'U')
//expected results:                 0,0,1,1 -> 1,2,1,1
                                    0,2,0,2 -> 0,1,1,2
                                    0,1,1,1 -> 0,0,1,1
                                    1,0,1,1 -> 0,0,0,1
{ { 1, 2, 1, 1 }, { 0, 1, 1, 2 }, { 0, 0, 1, 1 }, { 0, 0, 0, 1 } }
//returned as expected ☑
> game.slideElements(array, 'D')
//expected results:                 1,2,1,1 -> 0,0,0,1
                                    0,1,1,2 -> 0,0,1,2
                                    0,0,1,1 -> 0,2,1,1
                                    0,0,0,1 -> 1,1,1,1
{ { 0, 0, 0, 1 }, { 0, 0, 1, 2 }, { 0, 2, 1, 1 }, { 1, 1, 1, 1 } }
//returned as expected ☑
> int[][] array = {{0,0,1,1},{0,2,0,2},{0,1,1,1},{1,0,2048,1}}
//this array has a 2048, meaning that the player has won
> game.isGameWon(array)
//testing method isGameWon. It should return true if one element of the array
is 2048 otherwise false. The above array has 2048, so it should return true
true
//returned as expected ☑
```

```
> int[][] array = {{0,0,1,1},{0,2,0,2},{0,1,1,1},{1,0,0,1}}
> game.isGameWon(array)
//the case where there isn't a 2048 in the array. It should return false
false
//returned as expected ☑
> int[][] x1 = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}}
//this array doesn't have any zeros and or adjacent equal elements in any
direction
> game.isGameOver(x1)
//testing method isGameOver. The game is over when there are no more moves
which means all the elements are non zeros and there aren't any adjacent equal
elements in any direction. The above array has no more moves so it should
return true
true
//returned as expected ☑
> int[][] x2 = {{0,1,1,1},{0,2,0,2},{0,1,1,1},{1,0,0,1}}
//this array has zero elements and available moves
> game.isGameOver(x2)
//testing the case where the array has zero elements and available moves. It
should return false for the array above
false
//returned as expected ☑
> int[][] x3 = {{2,1,1,1},{2,2,4,2},{2,1,1,1},{1,8,8,1}}
//this array has no zero elements, but there are available moves
> game.isGameOver(x3)
//testing the case where there are no zeros in the array but there are some
moves in the game. It should return false for the above array
false
//returned as expected ☑
> int[][] x = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}
//this array is all zeros just to make it easy to notice where a new element
has been added
> game.addOne(x)
//testing method addOne. This is a helper method that adds one to the array
randomly. It should return the same array but with a 1 placed somewhere
{ { 0, 0, 0, 0 }, { 0, 0, 0, 0 }, { 0, 0, 0, 0 }, { 0, 1, 0, 0 } }
//returned as expected ☑
> game.addOne(x)
//just testing one more time
{ { 0, 0, 0, 0 }, { 0, 0, 0, 0 }, { 0, 0, 0, 0 }, { 0, 1, 1, 0 } }
//returned as expected ☑
> int[][] x = {{0,0,0,0},{4,4,0,4},{1,1,1,1},{0,2,0,2}}
//this array has 9 non zero numbers
> game.numberNonZero(x)
//testing method numberNonZero. This is a helper method that returns the number
of non zero elements in an array. It should return 9 here
9 //returned as expected ☑
```

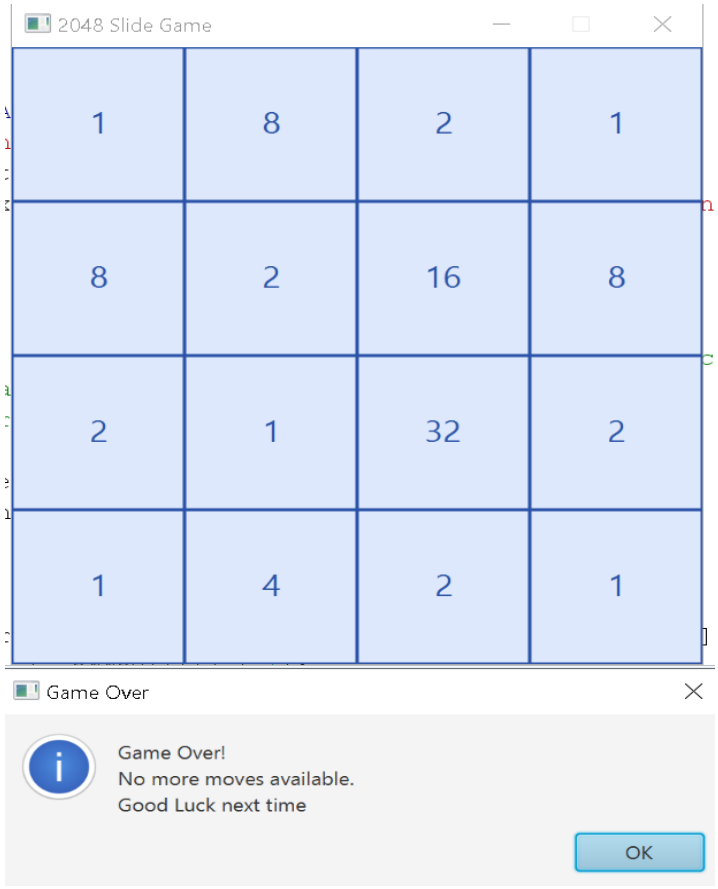
//I will not test the two helper methods intToButton and buttonToInt as it is hard to show how they work from the interaction pane. These methods are used in the GUI parts of the code to convert from int to button and vice versa. I will show how the GUI parts are working. Thus I will have tested these methods at the same time.
By testing the GUI I will have tested the following as well:
start(), gameWon(), gameOver(), intToButton(), buttonToInt(), main()



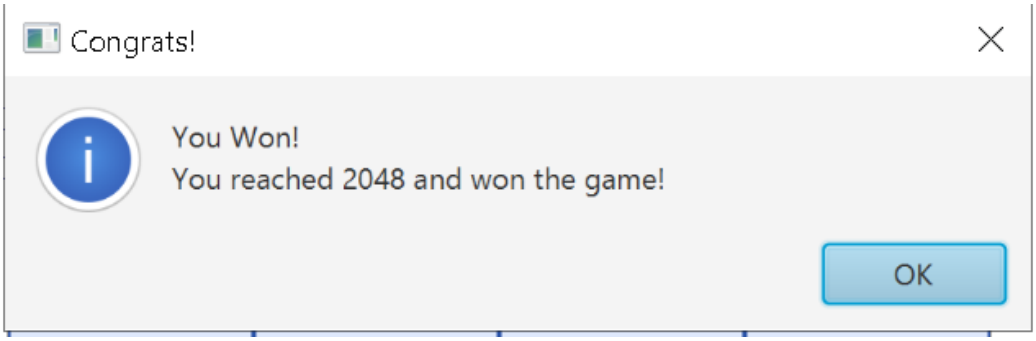
This is what will be displayed when we run the game. At this point we will have tested start(), intToButton(), buttonToInt(), main().



After playing the game for a while we will be testing gameWon() and gameOver() at each move.



This is what will happen if we lose the game. Method gameOver() will work by popping the game over window out and we won't be able to keep playing anymore.



If we win the game, the method gameWon() will work by popping this window out and we won't be able to keep playing anymore.