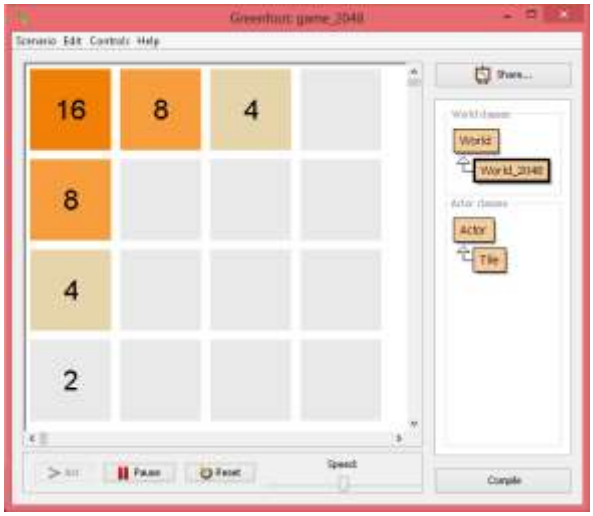
2048

2048 is number puzzle game created in March 2014 by 19-year-old Italian web developer Gabriele Cirulli, in which the objective is to slide numbered tiles on a grid to combine them and create a tile with the number 2048.



The game is played on a 4×4 grid, with numbered tiles that slide smoothly when a player moves them using the four arrow keys. Every time the grid changes (either by tiles merging or moving), a new tile will randomly appear in an empty spot on the board with a value of 2.

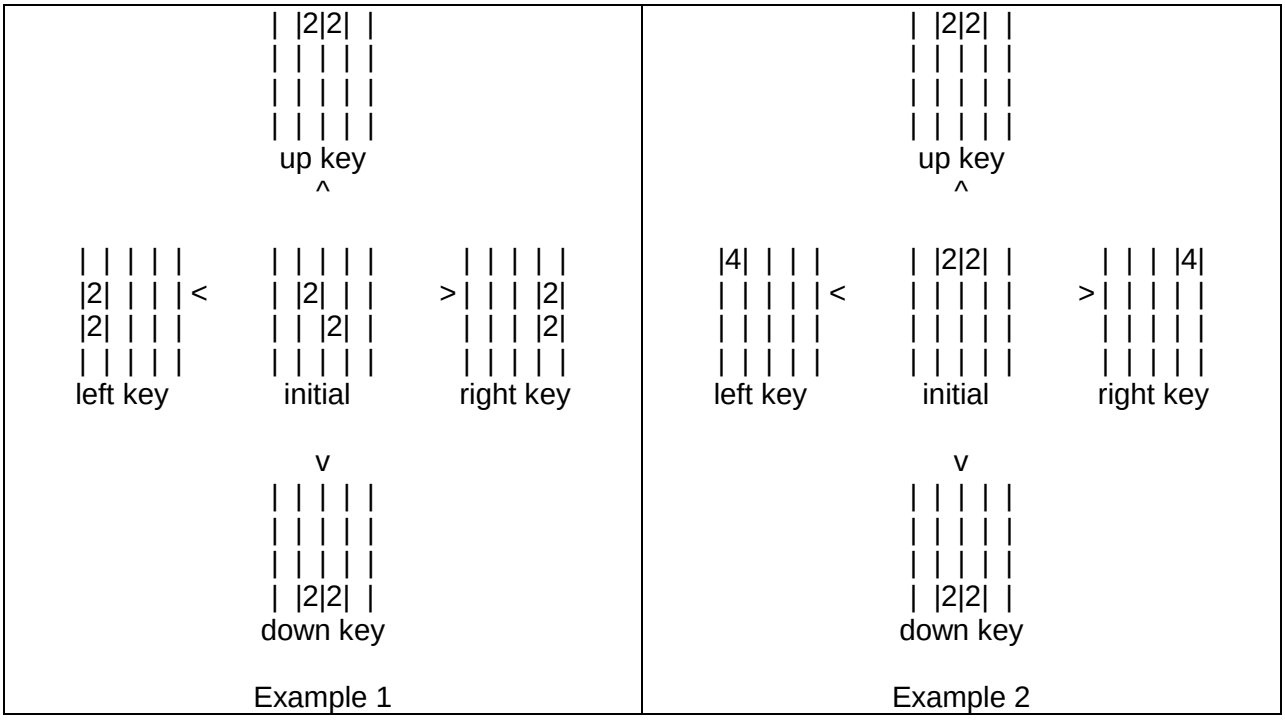
If two tiles of the same number collide while moving, they will merge into a tile with the total value of the sum of two tiles that collided.

Example: 2+2=4 ... 4+4=8 ...

The resulting tile cannot merge with another tile again in the same move. When a 2048 tile is created, the player wins.

Try it online at (http://gabrielecirulli.github.io/2048/) if you haven't played it to help you understand how to play it.

The diagram below shows two examples where the center grid represents the initial state surrounded by the ending state of the grid after a certain arrow key is pressed.



Tiles that show no numbers actually have the number zero but it is not displayed in the above diagram.

Here is what you need to understand in order to build the game in Greenfoot.
1- How to create a 4x4 grid using a two dimensional array.
2- A tile has a number which is a multiple of 2 (0, 2, 4, 8, 16, 32, 64, 128, 256 , 512, 1024, 2014,... and beyond if you wish to go after getting 2048.)
3- A tile's position is specified by the row and column of the grid.

So to create the initial grid start with all 16 cells having the number 0 then randomly choose two cells and set their number to 2.

Task one: create the grid and two randomly selected tiles having the number 2.
Placing a tile having number 2 at row 1 column 2 is done by this line of code:

grid[row][col]=value; // grid[1][2]= 2;

The two dimensional grid array can be declared as a field accessible to all methods.

Write a **printGrid()** method to print the grid for you to check and for debugging. It should display all numbers including the 0's. eg:

0	0	0	0
0	0	0	0
0	0	0	0
0	2	2	0

You can write most of your code in the World_2048 class.

The entire game can be played with only ASCII characters printed, but we will enhance it later to look somewhat like the real game.

eg: the above grid will look as follows if the **left** arrow key is pressed:

0	0	0	0
0	0	0	0
0	0	**2**	0
4	0	0	0

The two 2 tiles merge into a **4** tile and move left replacing the 0 tile near the left edge.
A random 0 tile is selected and its value changed from 0 to **2** (in the above example at grid[2][2])

mergeLeft()
When the left arrow key is pressed, we need to merge left each of the above four rows. Let's look at the bottom row as an example.

It is easier to copy the contents of a grid row into a one dimensional array of size 4, manipulate it then copy it back to the grid.

Example1:If the bottom row was
|0|2|2|0|

when merged left, it becomes
|0|**4**|0|0| because a 2 next to 2 merge left into **4**
and the 4 then moves further **left**.
|**4**|0|0|0| because the 0 to the left of the 4. A number can move left if a 0 is to the left of it.

Example2:If the bottom row was
|8|2|2|0|

when merged left, it becomes
|8|**4**|0|0| because a 2 next to 2 merge into **4**. The 4 stays because there is an 8 (non 0) blocking it from moving further to the left.

Example3:If the bottom row was
|4|0|2|2|

when merged left, it becomes
|4|0|**4**|0|
then because of the 0 to its left it moves to the left
|4|**4**|0|0|
it cannot merge any further .
The **4** stays because there is 4 (non 0) blocking it from moving further to the left.

If the left arrow is again pressed
|4|4|0|0|
then the two 4 tiles merge into an 8
|**8**|0|0|0|

Two things happen when the left arrow is pressed:
1- if there is a 0 to the left of a number, the number moves left replacing the 0 and a 0 takes its place.
Example1:
|4|0|4|0|
0 to the left of 4 allows the 4 to move left and a 0 tile replaces it.
|4|**4**|**0**|0|

Example2:
|0|0|0|**2**|
|0|0|**2**|**0**| The 2 moves to the left because there is a 0 to its left.
|0|**2**|**0**|0| The 2 moves again for the same reason.
|**2**|**0**|0|0| The 2 moves again for the same reason.

So you need a method **stripLeftZeroes(**rowArray**)** that modifies rowArray
|0|0|0|**2**|
to
|**2**|0|0|0|

You need to develop another method **mergeLeft(**rowArray**)** to merge similar adjacent numbers like
|4|0|2|2|

to
|4|0|**4**|0|

If the left arrow key is pressed again,
you will need to call **stripLeftZeroes(**rowArray**)** to get
|4|4|0|0|
then call **mergeLeft(**rowArray**)** to get
|**8**|0|0|0|

When processing is done after a left arrow key is press, and if the resulting grid changes (either by tiles merging or replacing adjacent 0 tiles) , a new tile with number **2** is randomly created.

To select a random position for the new tile, we need to know which tiles have a 0 value.
We can store the row and col numbers of 0 tiles in two parallel arrays, row_array to hold the row values and col_array to hold the column values.
Eg: lets say we have 4 empty tiles having a 0 value (x here means non zero):

x	x	x	x
0	x	x	0
x	0	x	x
x	x	0	x

grid[1][0], grid [1][3], grid[2][1] and grid[3][2]. We would store these row and column values as:
row_array = {1,1,2,3}
col_array. = {0,3,1,2}

We can then pull a random number less than the length of row_array (here its length is 4 so between 0 and 3) and use it as an index to select an element from row_array and col_array.
We can then change the selected tile value to 2 and reprint the grid.

If our random number was say 1 then we would have selected grid[1][3] (1 is element 1 of row_array and 3 is element 1 of col_array)
row_array = {1,**1**,2,3}
col_array. = {0,**3**,1,2}

and then change its value to **2 (**the actual game places 2's 90% of the time and 4's 10% of the time).
x	x	x	x
0	x	x	**2**
x	0	x	x
x	x	0	x

To process the entire grid when a left arrow key is pressed we use the following algorithm in pseudo code. Call the method **moveLeft()**.

for each row of the 2 dimensional grid
    copy the row into a new 1 dimensional array **rowArray**
    stripLeftZeroes(rowArray**)**
    mergeLeft(rowArray**)**
    copy rowArray back to the row of the 2 dimensional grid

You can then test your program by creating an empty grid with two randomly selected tiles set to 2
Here are 5 examples:

(a)	(b)	(d)	(e)	(f)
|0|0|0|0|	|2|0|0|0|	|0|0|2|0|	|0|0|0|0|	|0|0|0|0|
|0|2|0|0|	|0|0|0|0|	|0|0|0|0|	|0|0|0|0|	|2|0|0|0|
|0|0|0|0|	|0|0|0|2|	|0|0|0|0|	|0|2|0|2|	|2|0|0|0|
|0|0|2|0|	|0|0|0|0|	|0|2|0|0|	|0|0|0|0|	|0|0|0|0|

Press the left arrow key to get

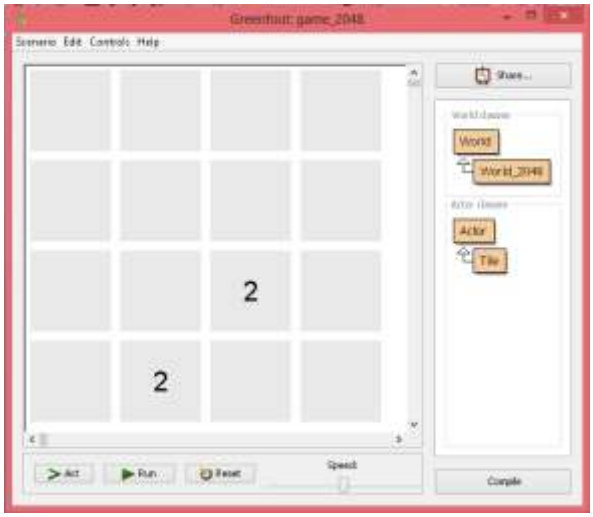
(a)	(b)	(d)	(e)	(f)
|0|0|0|0|	|**2**|0|0|0|	|**2**|0|0|0|	|0|0|0|0|	|0|0|0|0|
|**2**|0|0|0|	|0|2|0|0|	|0|0|0|0|	|0|0|0|0|	|2|0|0|0|
|0|0|0|2|	|**2**|0|0|0|	|0|0|2|0|	|**4**|0|0|0|	|2|0|0|0|
|**2**|0|0|0|	|0|0|0|0|	|**2**|0|0|0|	|0|0|2|0|	|0|0|0|0|

The 2's not in bold are the randomly generated 2's added because the grid changed.
In (f), pressing the left arrow did not cause the grid to change, therefore no new 2 was added.,

One way of figuring out if the grid changes is to save a copy of the grid before calling **moveLeft()**, and after the call to **moveLeft()** compare the saved copy to the current grid. This way you can find out if the grid has changed.

Creating the real look:
After we create the initial grid with two cells having 2 as in this example:
0	0	0	0
0	0	0	0
0	0	2	0
0	2	0	0

We can create a method **paintGrid()** that would actually draw the tiles.



After each arrow key press, if the grid changes, you can remove all objects from the world (wipe all objects out) bollowed by a call to **paintGrid().**

To remove all objects from the world you can use the following code:

removeObjects(getObjects(null)); // check http://www.greenfoot.org/topics/203

paintGrid() does the following:
For each row and column of the grid get the value (number) of the tile.
create a tile with the same number and place it in the World at the same row and column
Tile tile = new Tile(grid[row][col],row,col);
addObject(tile,col,row);
Notice:
The values of col and row are between 0 and 3. To make it easier to place the new tile without having to calculate the actual coordinates in the world, you can create the world as follows:

super(400, 400, 100);

This statement calls the World constructor

`World(int worldWidth, int worldHeight, int cellSize)`

where

`worldWidth`	is set to 400
`worldHeight`	is set to 400
`cellSize`	is set to 100

The last argument `cellSize` being 100 , allows us to specify the grid row and column instead of numbers like 200 and 100 if `cellSize` was instead set to 1.

So for example, if someone wanted to place a **2** tile at row 2 and column 1:

0	0	0	0
0	0	0	0
0	**2**	0	0
0	0	0	0

They would code it as
addObject(tile,2,1);
instead of
addObject(tile,200,100);

Otherwise you can multiply the row and col values by 100 to get the same result.

The Tile class
A tile is a Greenfoot image 100x100 with a string like "0" or "2" drawn on it. The tile color depends of the drawn string "0" or "2".

So when a tile is created its constructor needs to know the image that is to be drawn on it. Its color is then set depending on what string "0" or "2" or another value is passed as argument.

The Java code for the Tile class will be provided to you by your instructor.

In addition to **moveRight**() you will need to develop

moveLeft()
moveUp()
moveDown()

moveLeft is very similar to moveRight except for the direction.
moveUp and moveDown effect the columns of the grid not the rows.

After you develop these three methods you may want to eliminate some repeated code.

Enhancements may include a running score that you may wish to add.

Have Fun!