

CSCI-UA.0101-005: Project 1

Instructor: Teseo SCHNEIDER
New York University

Part 1: Warmup *40 points*

In this first part you will extend the “game of life” code to support additional features.

- (1) (10 points) Refactor the method `countAliveNeighs`. Instead of having a loop to retrieve the x, y -coordinates of the 8 neighbors; add a new method `int[] [] getNeighs(boolean[] [] alive, int i, int j)` to retrieve them. This change *should* affect only the body of `countAliveNeighs`.
- (2) (20 points) Extend the code to support different rules. Change the `update` method to also receive two arrays containing the numbers of required neighbors for a cell to survive or die. Example where a cell is born if it has exactly 3 neighbors, survives if it has 2 or 3 living neighbors, and dies otherwise:

```
int[] born = {3};
int[] surviving = {2, 3};

while(true)
{
    print(alive);
    update(alive, born, surviving);
    ...
}
```

- (3) (10 points) Change the program to receive *command line arguments* to decide the size of the grid and the rules for a cell to survive or die. For instance, `java GOL 10 B3/S23` runs the game of life on a 10×10 grid with the rule B3/S23. That is, a cell is born if it has exactly 3 neighbors and survives if it has 2 or 3 living neighbors.

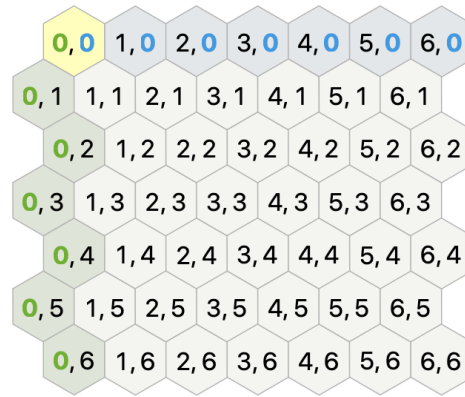
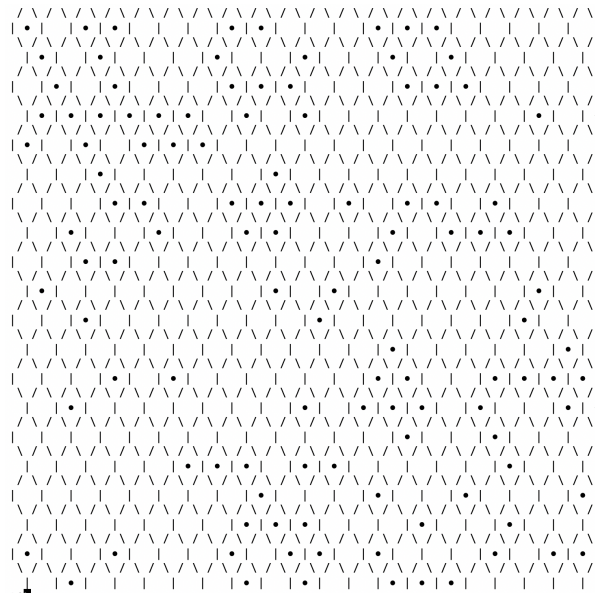


Figure 1: Example of an hexahedral grid.

Figure 2: Example of a print of 20×20 grid.

Part 2: Hexagonal game of life: *80 points*

In this second part you will extend the “game of life” to run on a hexagonal grid. A hexagonal grid can be stored as a normal grid where every row is “shifted by half” left and right (Figure 1). For more information visit this [page](#). To start, just copy your current “game of life” code.

- (1) (40 points) Change the `print` code to visualize a hexagonal grid. Use the characters `/`, `\`, and `|` to visualize the grid and use `•` to illustrate if a cell is alive (Figure 2).
- (2) (30 points) Change the method `int[][] getNeighs(boolean[][] alive, int i, int j)` to retrieve the x, y -coordinates of the 6 neighbors of a cell at position i, j (instead of 8 as in the square grid).
- (3) (10 points) Now your code supports input grid size and rules for both a hexagonal and square grid. Implement a new class to accept an optional user parameter `hex` to decide which grid to run, e.g., `java GOL 30 B2/S2 hex` will run on an hexagonal grid while `java GOL 30 B2/S2` on the square one. *Important*, the new class must call the `main` methods of the other two classes.

Question	Points	Bonus Points	Score
Warmup <i>40 points</i>	40	0	
Hexagonal game of life: <i>80 points</i>	80	0	
Total:	120	0	