INFORMATIONS

GitHub link of the project : https://github.com/luddoz-c/war-of-blobs

Please feel free to open an issue if you detect some problems, want to suggest any improvement or if you want to give us some advice.

CHANGELOG

24/04/19

Create `reset_grid` function to reset the grid when needed, in order to start a new one without having to launch the program again.

13/04/19

Update the `export_file` function (`blob.blobs` instead of `for` loop).

10/04/19

Several changes :

- Renamed `change_pos` method to `move_pos` : the position passed as parameter will be the relative move from the blob's position, not absolute anymore.
- Add `move` method that moves the blob towards the biggest one and, in case there are several ones (same size), towards the nearest.
- Add `import_file` and `export_file` functions to import and export some configurations.
- Add the details of each step (when we go to the next step, every changes are printed : merges, moves...)
- Converted the `change_pos` method to `move_function` so that the blob moves to the position `x, y` from its actual position instead of going to the absolute position `x, y`.
- Add docstrings and comments for every function/method.
- Replace `blob.grille` variable by `grille` (as a global variable), because it is more convenient.
- Add some code into `if __name__ == "__main__":` so that we can now launch a simulation from a terminal.
- Add `create_grid` function to create a grid of `n` columns and `m` rows.

09/04/19

- Fix the `check_blobs` function (priority bugs). Sometimes the blobs either didn't merge or merged with the wrong priority.
- Also other fixes on the `check_blobs` function. (2 commits made).

08/04/19

Add the other diagonal priority (from top right to bottom left).

04/04/19

Creation of the method that checks the priority of each blob, in order to know how to merge them.

31/03/19

Creation of the `next` function that simulates the next step of the simulation. However, this function is **TEMPORARY** as it is random (which is not the purpose of this simulation).

Addition of the colorization of the blobs : for that, we used the module `ansicolors`, which we imported as `colors`. Now, every blob's weight is colorized in the console, depending on the blob's color attribute (that we had to convert from values in `[0:1]` to values in `[0:255]`)

27/03/19

Definition of the `changePos` method, which is going to change the value of the blob's position (`self.pos`), edit the grid (list of lists) to move the blob to the right position (erase its old position and define its new one), and also check whether or not there is already a blob to the position where we want to move the blob.

Also, definition of a function that is going to generate a blob randomly (name, position...), and that checks whether or not there is already a blob at the generated position for the blob. Also, we created a function `generate_blobs` that uses the previous function to generated several blobs.

26/03/19

Creation of the `draw_grid` function to create the grid to be printed on the console.

We defined 2 grids :

- The first one is just the interface to be printed thanks to our previous function
- The second one is where all the informations are stored (positions of blobs, empty cells...). It is is a list of lists, containing empty strings where there are no blobs and a `blob` object where there is one blob.

Each blob has 2 names: one for the interface, which is `b + its number + its weight:` (ex : `b1(45)`), and the other one, its actual name (generated randomly), like `Mu`, `Ga`, `Yo`... (not printed in the console).

A blob has also a color, defined as a tuple `(r, g, b)` and a size (which is actually its weight). The `r`, `g` and `b` values are between `0` and `1`, with at most 2 decimals.

20/03/19

Creation of the class blob that we're going to use for every new blob that is created. This way, it will be easier to get its position, to change it, to edit its name, get the number of blobs on the grid...

HOW TO USE ?

From a terminal, try these commands:

1. Launch the program typing :

```
python3 main.py
```

2. When asked to create a random simulation, type `y` (yes), setup your configuration and see what happens.

3. In a python shell, execute the program. When asked to create a random simulation, type `n` (no).

4. Then, setup up your configuration. If you want a grid of `n` columns and `m` rows, type in :

```
>>> create_grid(n, m)
```

5. Display your configuration by typing :

```
>>> print(draw_grid())
```

6. Create blobs by using one (or more) of the commands below :

```
>>> blob([name], [weight], [color], [position]) # To create a blob manually
>>> generate_blob(W) # To create a blob randomly, of weight less than or equal to W
>>> generate_blobs(n, W) # To create n blobs of weight less than or equal to W
```

7. Display your configuration by typing

```
>>> print(draw_grid())
```

8. When you're done, export your configuration :

```
>>> export_file(filename)
```

where filename is the name of the file where you want your configuration to be saved.

> **IMPORTANT** : the name of the file has to end with `.json` **!**

9. Either continue adding/removing blobs or close the program, then open it again and import your configuration using:

```
>>> import_file(filename)
```

where filename is the name of the file where you want your configuration to be loaded from.

10. Display the imported configuration (`print(draw_grid())`) and use `wob_next()` to go to the next step of the simulation.

> If you don't want to type `wob_next` manually for each step, just type in : `while len(blob.blobs) > 1 : wob_next()`.