# Data Structure and Algorithm Analysis B Project 2022 Spring: RGBY Cell Life

## 1. Introduction

### 1.1 Basic description of the problem

In this project, you are required to simulate some simplified virtual single-celled life on a rectangular 2d planar space. All of the cells are prefect circles.

Each cell has the following properties:

- **Identity** or Id. An interger represent the uniqueness of this cell. This does **NOT** change during the life cycle of the cell.
- **Radius**. The unit is meter. This does **NOT** change during the life cycle of the cell.
- **Position**. Represented by a real number pair (x, y), where x is the distance between the left border of the plain and the center of the cell (in meters), and y is the distance between the bottom border of the plain and the center of the cell (in meters). Suppose there is a cell $c_0$ with position ($x_0$, $y_0$), and radius $r_0$, then the set of all points: [$(x, y)|(x - x_0)^2 + (y - y_0)^2 \leq r_0^2$] are said to be "within the cell $c_1$".
- **Color**. Either **RED**, **GREEN**, **BLUE** or **YELLOW**. The color of a cell can change however, but it will always be one of the RGBY color.
- **Perception range**. Represented by a real number in meters. A cell can know the number of red, green and blue cells in its neighborhood. To be more specific, suppose there is a cell denoted as $c_1$, and its $x$, $y$ coordinates and perception range are denoted as $x_1$, $y_1$ and $p_1$ correspondingly. There is another cell $c_2$. If there is a point ($x$, $y$) that is "within the cell $c_2$" and satisfies $x_1 - p_1 \leq x \leq x_1 + p_1$ and $y_1 - p_1 \leq y \leq y_1 + p_1$, then we say "$c_2$ is within the perception range of $c_1$". A cell knows how many red cells are within its perception rage. The same for green, blue and yellow cells. **The perception range is set as a square in order to simplify the question.**

Some other aspects of our 2d space and the cells:

- The cells are in a two dimensional space. There are four walls around the space. The cells cannot go outside of the walls. They **CANNOT** overlap with the walls.
- The walls do not move, so that you can simulate a finite size of a rectangular space where $x \in$ [0, width] and $y \in$ [0, height] instead of from negative infinity to positive infinity.

- The cells **CANNOT** overlap with each other. That means for every two cells $c_1$ and $c_2$ at any time, $(x_1 - x_2)^2 + (y_1 - y_2)^2 >= (r_1 + r_2)^2$.

Your program should display the result using a graphic window.

## 1.2 Simulation process

The simulation starts at time 0. After 1/15 seconds, each cell moves to a new position. And then checks other cells in its neighborhood, and decides whether it should change color. Then this repeats in every 1/15 seconds.

The process is like following (where cell[i] is the cell whose id is i):

```
while(true)
    wait for 1/15 seconds.
    for( int i = 0; i < size; i ++ )
        move cell[i] to new position.
    for( int i = 0; i < size; i ++ )
        scans cell[i]'s perception range, calculate the number of rgb cells.
    for( int i = 0; i < size; i ++ )
        change cell[i]'s color with the rules if necessary.
```

We assume that cells move one step every 1/15 seconds and stand still for 1/15 seconds. Let $(x_t, y_t)$ be the position of a cell at the $n^{th}$ step. We set the moving rule as follows:

1. **Red** cells move upward at each step, i.e., $(x_{t+1}, y_{t+1}) = (x_t, y_t + 1/15)$
2. **Green** cells move downward at each step, i.e., $(x_{t+1}, y_{t+1}) = (x_t, y_t - 1/15)$
3. **Blue** cells move to the left at each step, i.e., $(x_{t+1}, y_{t+1}) = (x_t - 1/15, y_t)$
4. **YELLOW** cells move to the right at each step, i.e., $(x_{t+1}, y_{t+1}) = (x_t + 1/15, y_t)$

Note that every cell cannot overlap with each other, so when a cell "hits" another cell during its movement, it stop. In this case, the cell will stop in this step.

When color mutation process begin, every cell scans its neighborhood and then they make the decision. Then all of them change color together at the same time. This happens instantly.

The rules for color change is as simple as follows:

- For red cells, if there are at least 3 red cells in its perception range (excluding itself), and the percentage of red cells in its perception range is greater than 70% (excluding itself), then it turns green. Otherwise, if there are at least 1 yellow cell in its perception range, and the percentage of yellow cells in its perception range is lesser than 10%, then it turns yellow.

- For green cells, if there are at least 3 green cells in its perception range (excluding itself), and the percentage of green cells in its perception range is greater than 70% (excluding itself), then it turns blue. Otherwise, if there are at least 1 red cell in its perception range, and the percentage of red cells in its perception range is lesser than 10%, then it turns red.
- For blue cells, if there are at least 3 blue cells in its perception range (excluding itself), and the percentage of blue cells in its perception range is greater than 70% (excluding itself), then it turns yellow. Otherwise, if there are at least 1 green cell in its perception range, and the percentage of green cells in its perception range is lesser than 10%, then it turns green.
- For yellow cells, if there are at least 3 yellow cells in its perception range (excluding itself), and the percentage of yellow cells in its perception range is greater than 70% (excluding itself), then it turns red. Otherwise, if there are at least 1 blue cell in its perception range, and the percentage of blue cells in its perception range is lesser than 10%, then it turns blue.

# 2. Project Requirement

The actural work of the project contains many parts:

- Write the system described above. (30 Points)
- Performance optimization. (10 points)
- Test. (20 points)
- Write report. (10 points)
- Evaluation. (30 points)
- Bonus. (5 points)

## 2.1 Write the system described above

The program should at least be able to run. There are also some requirements of the project:

- The program should be able to start in "GUI" mode and "Terminal" mode (descripted in a later section).
- The program should be able to run and show the movement of the cells with GUI.
- The cells should be able to change color.

## 2.2 Performance optimization

The program may be quite slow when there are too many cells. Try to improve the running time of your program. The data structures you learned in class may not be enough. You are encouraged to read some materials.

## 2.3 Test

- Show that your program can produce the correct result.
- Show the performance of your program.
- Show the performance improvement of the optimization technique you used.

Programming mistakes can happen anywhere in such a project. Some mistakes will make your program produce the wrong result but still be able to run. Try to prove that your program can produce the right result. This might be hard, so you will be given some test data.

You can list out some special case you may meet during your programming, and show how you deal with them. Also you can make a diagram to compare the optimization and before optimization.

## 2.4 Write report

Include everything you want us to know about your project. Such as the basic structure of your project, the data structures you have used in the project, the way you optimize the performance, and anything that makes your project different from others.

For everything mentioned in 2.1, 2,2, 2.3, etc, if you actually did it. write it in report.

**Clearly state how to run your program in your report**. It's necessary when we test your program.

You may also include some statistics of your program, such as running time.

However, we have no requirement on the length of the report. Do **NOT** try to make it unnecessarily long. Remember to include group members and student numbers. Also remember to upload your report in "pdf" format.

If you insert some diagrams in the report, mark the name to show the meaing below every diagram and make sure they are located where the corresponding content is.

## 2.5 Evaluation

You may noticed that the above parts add up to 70 points. In the last we will run some tests we have prepared on your program. That is the last 30 points of your project. This can only be done after your code is submitted. We'll test your program by some cases.

The difficulty of each test cases is different. Brute force algorithms may not pass all of them, and you can try to minimize the time complexity of your algorithm.

## 2.6 Bonus

Try to generate some test data by yourselves. Try to modify the rules of cell moving or color mutation to generate interesting results.

## 2.7 Restriction on 3rd party library

**There are restrictions on what you can use and cannot use in this project.**

You could use various tools to advance your project, but third party physics engines, game engines are not allowed. For instance, you could use OpenGL, and OpenCL, but Unity and Unreal are not allowed to use in the project.

You are allowed to use algs4 library. If you want to use some library and you are not sure about it, you can ask us.

## 3. Input & Output Standard

The program running argument is either --gui or --terminal. Here **--terminal** means you should not show a window. Instead, you should print the required information in the terminal. And **--gui** means you should show a window.

So if your program is invoked like this:

```
java your.package.name.YourClassName --terminal
```

Then it means start your program in terminal mode. The gui mode is the default mode, which means **if this argument is NOT set, run the program in gui mode.**

The input data of the initial state is read from the standard input. Here's a template input file in the block shown as below.

Sample Input:

```
40 50
3
0.5 1.5 0.5 6.1 r
1.1 0.5 0.3 5   g
1.8   1 0.1 4   b
3
4   2
10  0
14  1
```

The numbers in the 1st line indicates the width and height of the rectangle space, all the objects are supposed be in this space. For example, the 1st line contains 40 and 50, then the x coordinate of everything in this space is in [0, 40], and y coordinate of everything in this space is in [0, 50].

The number "num" in the 2nd line is the number of objects.

For each of the next "num" lines, there are 5 elements represent the initial state of one object. They are:

- x-coordinate
- y-coordinate
- radius
- perception range
- color of the square "r" for red, "g" for green, "b" for blue.

The unit of x, y, radius and perception range is meter.

The objects are indexed. The "ID" of the first object is the 0 and the second object is the 1,..., the last object is num-1.

**It is guaranteed that the cells will NOT overlap with each other in the input data.**

The number in the next line "query" indicates the number of queries. For each of the next "query" lines, there are 2 numbers. The first is the time "t" in seconds and the second is the index of a cell. You should print a line containing the x, y and color of the cell at the time "t" for each query.

**It is guaranteed that time "t" in queries with be in accending order.** Please print the color **after** the color mutation if necessary.

Sample Output:

```
0.1 1 b
0.5 10.5 r
1.8 0.3 b
```

Please follows strictly the above requirement. When we test your program (as mentioned in the last section, 30 points), we will run your program with our input data and compare your result with the standard output data. If your program does not follow the above restriction and prints something else, the comparison will fail and it will not be good.

It you have any question about the input output format, you can ask.

## Reference

N-Body Simulation:
https://introcs.cs.princeton.edu/java/assignments/nbody.html

N-Body Simulation, with Barnes-Hut tree:

https://introcs.cs.princeton.edu/java/assignments/barnes-hut.html

QuadTree:
https://www.geeksforgeeks.org/quad-tree/

https://baike.baidu.com/item/%E5%9B%9B%E5%8F%89%E6%A0%91/8557650