CS202, Spring 2023

Homework 3 - Heaps and Priority Queues

Due: 12/04/2023

Before you start your homework please read the following instructions carefully:

FAILURE TO FULFIL ANY OF THE FOLLOWING REQUIREMENTS WILL RESULT IN A GRADE SCORE OF 0 (zero) WITHOUT ANY CHANCE OF REDEMPTION.

- See the course page for any late submission policies and Honor Code for Assignments.
- Upload your solutions in a single ZIP archive using the Moodle submission form. Name the file as `studentID_name_surname_hw3.zip`.
- Your ZIP archive should contain **only** the following files:
  - Your `.cpp` and `.h` files, as well as a file named `SimulationMgr.cpp`, which includes the main function.
  - Do not forget to put your name, student id, and section number in all of these files. Add a header (see below) to the beginning of each file:

    ```
    /**
     * Author : Name & Surname
     * ID: 12345678
     * Section : 1
     * Homework : 3
     * Description : description of your code
     */
    ```
  - Do not put any unnecessary files such as the auxiliary files generated from your preferred IDE.
- Your code must compile.
- Your code must be complete.
- Your code must run on the **dijkstra.ug.bcc.bilkent.edu.tr** server.
- For any question related to the homework contact your TA: *cihan.erkan@bilkent.edu.tr*

Wildlife Simulation

In this homework, you will simulate the behaviour of a group of simple-minded and greedy creatures. The simulation will start with a number of creatures and food that spawn at various locations. Each creature will know the location of the food with the best quality, and they all will march towards it. The first creature to arrive at the location with food will consume it. At different times during the simulation, we will introduce additional food supplies to the region. The creatures will race and fight against each other to survive by consuming the food resources scattered around.

Food

In this simulation, the food resources will spontaneously spawn at certain locations at certain times. Each food resource will have 4 properties: x and y coordinates of their location, a unique integer ID and their quality. Here, the quality of the food will be denoted by an integer value.

Creatures

The creatures will have 4 properties: A unique integer ID, an integer that represents their health, and x and y coordinates of their location. At each time tick, each creature first will look to fight other creatures. If there are other creatures within a 2 unit distance radius (Euclidean distance between creatures <2) with less than or equal health, it will kill all of them. Then it will look for food. However, since they are greedy, they will only consume the food with the best quality and ignore the rest. Because of that, if its distance to the best food is less than 1, then it will consume the food, otherwise it will not consume anything. If it consumes food, its health will increase based on the quality of the food. So, if a creature with 10 health, consumes a food with 8 quality, its health will become 18. Then, it will move in the direction of the food with the best quality that is currently available. If there are multiple foods with the same quality, creatures should prefer the one with the lowest food ID. If there are no food supplies available, creatures will not move. If they move, the distance covered in a single time tick will depend on the current health of the creature. If the health of the creature is less than or equal to 10, then its survival instincts will kick-in and it will move 1 unit distance. If it is healthy, i.e. its health is greater than 10, then it will only move 10/h unit distance where h is the current health of the creature. Lastly, it will lose 1 health. If its health becomes 0, it will die. Dead creatures do not move, and they do not interact with other creatures and food supplies.

Time

In this homework, you will simulate time using a loop and each iteration of the loop will correspond to a single time tick. At each time tick, following events should happen in exactly this order:

1. Current locations of all creatures should be printed
2. New food resources should be added (if any)
3. Fights between creatures should be resolved (if any)
4. Creatures should consume the best food if it is nearby
5. Health of each creature should decrease by 1

Moreover, to keep different implementations consistent, creatures with a lower ID should take action first. For example, during the fight phase, the creature with lowest ID should check and kill if it can kill anyone, then the creature with the second lowest ID should check, etc. After the creature with the highest ID performs its action for the fight phase, simulation should move to the food consumption phase. You should update the current best food as soon as a creature consumes the best food. So, multiple creatures can consume food during the same time tick. For example, assume that there are two creatures, one with ID 1, located at (0.1,0.1) and another one with ID 2, located at (5.1,5.1). Also assume that, there are 2 food resources located at (0,0) and (5,5) with 10 and 5 quality, respectively. During the feeding phase, creature 1 will consume the food with 10 quality, and after that the best food will become the food with 5 quality. Since creature 2 is near the new best food, it will consume that during the same time tick as well. However, a single creature cannot consume multiple food resources in a single time tick. The clock of the simulation will start from 0; i.e., the first iteration of the loop will be time 0. With each iteration of the loop, you should increment the time by 1. You should run the simulation until all creatures die.

Input

Your simulator should take one command line argument, the name of the input file. The input will be stored as a simple `.txt` file. First line of the input will be a single integer N, denoting the number of creatures that will be in the simulation. The following N lines will hold the creature information and the rest will hold food resource information. The format of the creature information will be:

```
<ID>,<X>,<Y>,<Health>
```

and the format of the food information will be:

```
<ID>,<X>,<Y>,<Quality>,<Spawn Time>
```

Where `<ID>` is the ID of the creature/food, `<X>` and `<Y>` are the starting coordinates of the creature/food, <Health> is the starting health of the creature, `<Quality>` is the quality of the food resource and the `<Spawn Time>` is the time (in terms of number of loop iterations) which you should add the food resource to the simulation. The following is a sample input file:

```
2
1,10,2,10
2,1,1,15
1,7,11,4,0
2,2,2,7,11
3,1,3,11,0
4,9,5,8,0
5,2,12,10,4
6,14,14,18,8
7,1,1,5,5
8,10,2,10,15
```

Output

As mentioned previously, your simulator should output the locations of every creature at the beginning of each time tick, and it should keep outputting them until the simulation terminates when the last creature dies. You should keep printing the location of a creature, even if it is dead. The format of each line should be as following:

```
Creature <ID>: <X>, <Y>
```

You should print exactly two decimal points of both `X` and `Y` coordinates. That is, if `X` is 11/10 you should print it as 1.10 and if `X` is 2/3 then you should print it as 0.67. Your output should not contain any additional information such as the current time tick, simulation start/end, health of creatures etc. The following is a sample output of the simulator. Note that this is not an actual output corresponding to the given input file. This sample only illustrates the expected output format.

```
Creature 1: 10.00, 2.00
Creature 2: 1.00, 1.00
Creature 1: 10.00, 2.00
Creature 2: 1.00, 1.00
Creature 1: 10.00, 2.00
Creature 2: 1.00, 1.00
Creature 1: 10.00, 2.00
Creature 2: 1.00, 1.00
Creature 1: 10.00, 2.00
Creature 2: 1.00, 1.00
```

It is very important that you make sure the format of the output of your program exactly matches the given output format. You will also be provided a .gif file that illustrates how the game progresses with an example, so it's easier to understand the game logic.

Implementation

You should implement this wildlife simulation in an object-oriented manner, making sure you apply the basic principles of class based implementation, information hiding, etc. To manage the simulation components, use a manager class (e.g., `SimulationMgr`) with a `main()` method containing the input, any pre-processing including data structure setups, game loop and output parts.

In order to correctly run the simulation, you obviously need to keep track of the food currently available to creatures and the food resources that you are going to add to the simulation. Note that, during the simulation, you only need to access the food with the highest quality value to determine the behaviour of the creatures. Similarly, when adding new food resources, you only need to know the food resource with the lowest spawn time value. A heap-based priority queue should allow you to do exactly that. Because of that, in this homework, you have to use heap-based priority queues to store both the food that is currently available and the food that you are going to add to the simulation in future. You are free to implement the rest of the homework in any way you prefer.