Project

Description

We want to develop a basic ecosystem simulator. We want to simulate a world Where exist two kinds of entities : Vegetals, Consumers I and Consumers II. The World is divided in cases and is toroid.

The Vegetals grow on every part of the World and widespread a specific smell which allows their detection. They cannot move.

The Consumers I eat Vegetals only. They spread a specific smell which allows their detection and they can detect specifically other entities smell. They can move. They are hermaphrodite.

The Consumers II eat Consumers I only. They spread a specific smell which allows their detection and they can detect specifically the smell of Consumers I and Consumers II. They can move. They are hermaphrodite.

The time is divided into turns. The number of turn is increased at the beginning of the turn. During a turn, one of the following actions can be performed (if possible) :

- To move of a number of cases equal to the speed and to eat.
- To move of a number of cases equal to the speed and to reproduce.
- To be born.
- To die.

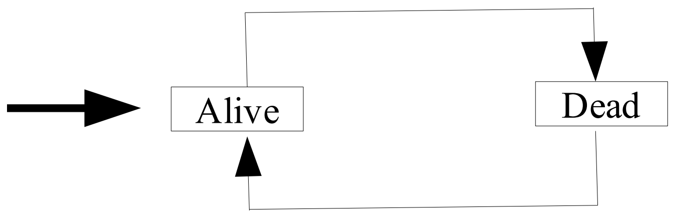
When an "Entity" has lived a number of turn equal to their Life Time, it will die.

Each Entity has a status. Vegetals can be : "Alive, Dead". Consumers can be : "Born, Alive, Dead". When a Consumer is eaten or when its life time is over, its status turns from "Alive" to "Dead". Just after the reproduction, the new Consumer has a status equal to "Born". A new born Consumer can be eaten too.

At the beginning of each turn, the Vegetals status turn to "Alive" if their state was "Dead" during the previous turn and the new Consumers come to life if their status was "Born" during the previous turn. Every "Inactive" turns on "Active"

At the end of each turn, each entity widespreads its smell, all Consumers which status is "dead" are deleted.

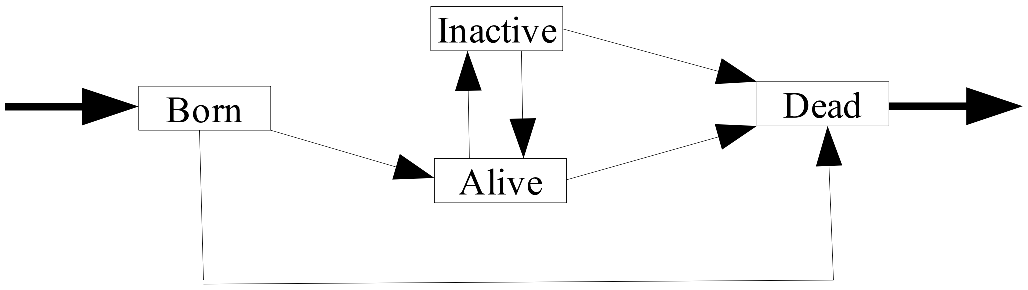
We can sum up these behavior by the following automatas :



Graph of the Vegetals status

During each turn, the consumers which status is "Alive" can move. After a movement (with or without eating), the consumer status turns to "Inactive". After a reproduction between two "Consumers", both are "Inactive". Of course, an "Inactive" can be eaten or die by overpassing its life time. It is impossible to reproduce with a new "Born".

The "Consumers" will also die if they have not eaten during a their maximal time without food.



Graph of the Consumers status

A case is at a distance of one unit of another if it is just adjacent (so each case has 8 neighbors at a distance of 1).

Each turn, each entity leaves its own smell on the cases with an intensity equal to "smell emission" – d, where d is the distance between the case and the entity.

Exemple of a Consumer which leaves an emission of 3 :

1	1	1	1	1
1	2	2	2	1
1	2	3	2	1
1	2	2	2	1
1	1	1	1	1

The emission of each kind of entities are distinct but the emissions of the same kind of entities are cumulative. These emissions can distinctly be detected by Consumers at a maximal distance of "smell emission".

The "Consumers" can move of a number of cases equal or inferior to their speed during each turn. The choice of the destination is defined by the following formula :

Score = $FE*(\frac{TWF}{MTWF})+(STE-PE)*(\frac{MTWF-TWF}{MTWF})$

with :

- FE = Food Emission
- TWF = Time Without Food
- MTWF = Max Time Without Food
- STE = Same Type Emission
- PE = Predator Emission

Each Entity choices the best score.

When an Entity arrives on a free case, nothing happens.

When an entity arrives on a case with only Food, it eats.

When an entity arrives on a case with only Same Type Entities, it reproduces.

If a mix is present, it will choose to eat with a probability of $\frac{TWE}{MTWF}$ or to reproduce with a probability of $\frac{MTWF-TWF}{MTWF}$

If the speed is not enough to reach the destination, the Entity will move toward the best destination. Of course, a new choice will computed each turn.

If two Consumers reproduce, a new Consumer of the same type is created on the same location with the status "Born".

The simulation ends when :

- The Maximal number of turns is reached.

  Or

- There is no Consumers left in the world.

Input

In a file named Vegetal.txt, write in this order :

- The Vegetals life time.
- Maximal life time.
- Smell emission.

In a file named Consumer1.txt, write in this order :

- The Consumers1 life time.
- Maximal life time.
- Smell emission.
- Smell detection.
- Max time without food.
- Time without food.
- Speed.

In a file named Consumer2.txt, write in this order :

- The Consumers1 life time.
- Maximal life time.
- Smell emission.
- Smell detection.
- Max time without food.

- Time without food.
- Speed.

In the call line, as parameter : size of the world in high and length, Maximum number of turn of the similation, number of Consumers 1 and 2, path to the files Vegetal.txt Consumer1.txt Consumer2.txt. (Vegetals are present in all the cases)

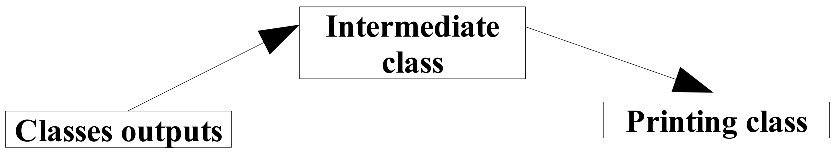
Outputs

The outputs will be on an ASCII form (no GUI). The state of the world will be printed each turn in a xterm.

We must see :

- Each case of the World.
- Each Entity and its state on its position.
- After the print of the World, we must see the total number of Consumer1 and Consumer2.

The outputs must not be linked to the simulator. The simulator will give the informations to a class which give them back to the output class.



Tips

- **Use the classes (Life-Creature) and create a class "Vegetal" inherited from "Life". "Life" is virtual pure.**
- **Specialize the Creatures.**

  - **Write two classes inherited of Creature: "ConsumerI" and "ConsumerII". Creature is virtual pure.**

  - **Write a virtual function "meet" for all Creature such as :**

    - If a ConsumerI meet a ConsumerI, it reproduces itself.
    - If a ConsumerI meet a Vegetal, it eats it.
    - If a ConsumerII meets a ConsumerII, it reproduces itself.
    - If a ConsumerII meets a ConsumerI, it eats it.

  - **Write a class Life which knows :**

  - Its life time (integer number).
  - Its position in a two dimensions world.
  - Maximal life time.
  - Range of smell emission.

- **Write an inherited class "Creature" from "Life" where :**

  - The time without food.
  - The range of smell detection.
  - Speed.
  - Maximal time without food.

- **Write member functions.**

  - Constructors and destructors.
  - Assessors and Mutators.
  - Which print position.
  - Which test if maximal life time is reached.
  - Which test if food is found.
  - Which test if another "Creature" is found.
  - Which compute new coordinates after a move.
  - Which test if the "Life" continues or not.

- **Remember that the World should be unique !**

- **The world knows the time and the set of all positions**

Design Patterns

- **Singleton**

  Singleton principle is to create only one unique object of this class. This design pattern is not perfect it onfen be avoided by multi-threading procedures. I give the general way to code it.

  Meyer's Singleton :

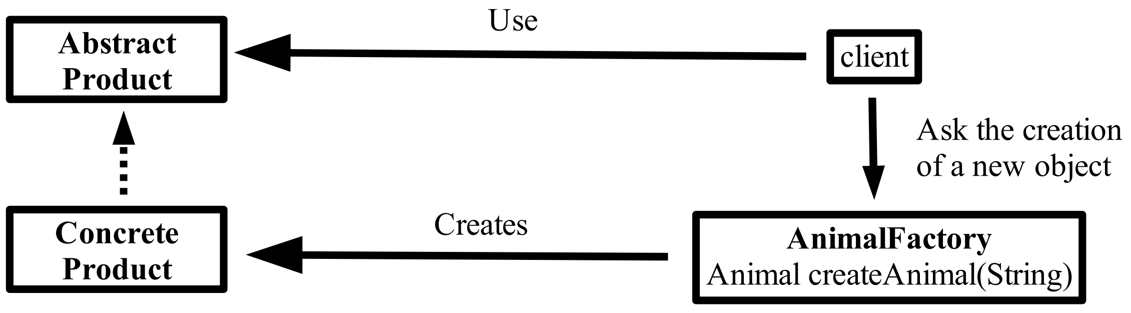
```
template<typename T> class Singleton
{
      public:
            static T& Instance()
            {
                  // suppose T has a default constructor. Otherwise,
                  //modify this line.
                  static T theSingleInstance;
                  return theSingleInstance;
            }
};

class OnlyOne : public Singleton<OnlyOne>
{
      // constructors/destructor of OnlyOne are friends with the Singleton
      friend class Singleton<OnlyOne>;
      //All other method in the interface between singleton and T ...
};
```

Another without templates :

```
class Singleton
{
      public:
            Singleton()
            {
                  if (alreadyCreated) throw logic_error("Only one object
                  of Singleton type !.");

                  // Else, we create the object …

                 alreadyCreated = true;
            }

            ~Singleton()
            {
                  // If we want to recreate an object later, we have to
                  change alreadyCreated.
                  alreadyCreated = false;
            }

      protected:
            static bool alreadyCreated;
};

bool Singleton::alreadyCreated = false;
```

– **Factory pattern**

I give only a sketch to illustrate the principle.



```
public class AnimalFactory {

  Animal createAnimal(String typeAnimal) throws ExceptionCreation {
    Animal animal;
    if("cat".equals(typeAnimal)) {
      animal = new Cat();
    } else if("dog".equals(typeAnimal)) {
      animal = new Dog();
    } else {
      throw new ExceptionCreation("Impossible to create a " + typeAnimal);
    }
    return animal;
  }

}
```

Work :

- Code the simulator described in the project and send the resulting files with their makefile to saule.cedric@uni-bielefeld.de before the 22nd March at 0H01.
- The program must take in entry the input files paths, the number of each Creatures type and the size of the world.
- You have to comment the code in order we can easily modify it.
- You have to give a short report explaining your functions, classes choices and the way to install and launch the program.
- Additionnal features have to be explained in the report. They will be taken into account in the evaluation.
- For people who worked in group, give the personnal contributions (or in case of the program was not split in an easy way, just the global contribution).