PREDATOR & PREY SIMULATION
MULTIAGENT SYSTEM

Subject: Artificial Intelligence Fundamentals

2014
Sergiu Terman

Problem Description

In this problem I am simulating the behavior of multiple *Animals* on a grid. The grid itself represents a forest, a plane with no borders in which a few Animals are placed. This grid has no borders, for instance if I reach position 0 of the x axis -1 is the other side of the grid also known as the maximum x position. The same feature of course also counts for the maximum position plus 1 becoming 0. The simulation happens in turns. In each turn all the Animals will be asked to *make a decision*. Every animal has its own decision making steps, even if the animals in group of predators/preys have a similar behaviour.

There will be *4 types* of Animals. Two predator and two prey type. Two types of species will be able to move (one prey and one predator capable of movement). Each Animal also has a life energy pool, a procreation pool, and an age limit. Animals are able to look to their 8-connected neighboring grid positions for other Animals.

Animal definition

1. Fox (Predator)
    - Can't move
    - Can eat the prey around him
    - After eating, he remains fed for the next 3 rounds. So he doesn't eat during this perioud
    - Can procreate in an empty cell around him, only if the animal is fed
2. Wolf (Predator)
    - Attacks a wolf if there are more than 2 wolves around him
    - The damage done to wolves is in range from $\frac{1}{4} \cdot lifeLimit$ to $\frac{3}{4} \cdot lifeLimit$
    - Can eat the prey around him
    - Can procreate in an empty cell around him, only if the animal is fed

- If none af actions above were made, wolf moves randomly in an empty cell around him

3. Rabbit (Prey)
   - Can't move
   - If no rabbit around him he dies of loneliness
   - If there is rabbit around he procreates in an empty cell around him
4. Deer (Prey)
   - If there are predators around he escapes by moving in a different cell
   - He escapes only if the empty cell is safe, meaning that no predators around that cell
   - He procreates if a deer is around him
   - If none of the actions above were made then he moves in an empty cell around him

Implementation

The implementation of the problem was done in Java programming language. I also have used swing and awt gui libraries for more clear visualization of the system. The source code of the application can be found on this link: `https://github.com/xserjjx/Prey-Predator`

Classes implemented

Animal is the main abstract class. Here all the shared methods between the animals are implemented. Some of them are:

- `List<Animal> lookAround();` returns all the animals in the 8 cell neighbourhood
- `boolean matingProcess();` does all the mating process and returns true if was done successfully

- `List<Coord> getNeighbourhood();` returns the coordinates of cell around the animal
- `abstract void nextStep();` abstract method that is needed to be implemented in the subclasses of the class
- `boolean routineCheck();` a method that checks if the animal is alive or under the age limit. Returns true if everything is fine

Predator abstract class. Is a subclass of *Animal* class. It has additional methods specific to predators

- `boolean eat();` tries to eat the prey around, returns the success of the action
- `List<Prey> getPreyAround();` returns the prey from the neighbourhood

Prey abstract class. Is a subclass of *Animal* class. Has specific method `List<Predator> getFoeAround();` which returns the predators around the prey.

Fox class. Is a subclass of *Predator* class. It only implement the abstract method `void nextStep();`

```
void nextStep() {
        if (!routineCheck())
                return;

        if (hungry >= HUNGRY_LIMIT) {
                this.hasEaten = false;
                eat();
        } else {
                mate();
        }

        age++;
        hungry++;
}
```

As we can see the routine check is done, after which if the fox is hungry then he eats, if he is fed then tries to mate.

Wolf class. Is a subclass of *Predator* class. It has an additional method `boolean attack();` where it tries to attack a wolf around if there are more than 2 in neighbourhood. Now the method `void nextStep();`

```
void nextStep() {
        if (!routineCheck())
                return;

        if (!attack()) {
                if (hungry > HUNGRY_LIMIT) {
                        this.hasEaten = false;
                        if (!eat())
                                moveHelper.moveRandmon();
                } else {
                        if (!mate()) {
                                moveHelper.moveRandmon();
                        }
                }
        }

        age++;
        hungry++;
}
```

So as we can see, he tries to attack first. If no success attacking he eats if he's hungry otherwise he mates. If nothing works he tries to move in an empty cell around

Rabbit class. Is a subclass of *Prey* class. It only implement the abstract method `void nextStep();`

```
void nextStep() {
        if (!routineCheck())
                return;

        List<Animal> bunnyAround =
           getSpecificNeighoburs(this.type);
        if (bunnyAround == null ||
           bunnyAround.size() == 0) {
                life = 0;
                return;
        }

        mate();
}
```

It's checking if there are rabbits around him. If fails than he dies otherwis he tries to mate.

Deer class. Is a subclass of *Prey* class. It has an additional method `boolean avoid();` which tries to escape the predators around him (if it's possible). Now the method `void nextStep();`

```
void nextStep() {
        if (!routineCheck())
                return;

        if (!avoid() && !mate())
                moveHelper.moveRandmon();

        age++;
}
```

As we can see he tries first to avoid, in case if fails he tries to mate. If nothing is working then he moves randomly in an empty cell around him.

Grid class. Is has a 2D array which holds the animals. Based on the position on the grid the animals acquire the coordinates.

Game class. Has the lists of animals and preys. Manages the next generation and the dead animals. Also make's sure that every animal executed their action.

MoveHelper class. Is an additional class used by classes *Deer* and *Wolf* that helps to move in the grid. It has the methods:

- `boolean move(Coord pos);` moves to a specific coordinate in the grid. Returns true if the action was performed successfully, false otherwise
- `boolean moveRandmon();` moves in an empty cell around

Result of implementation

Now let's see the result of the program simulation. In Figure 1 is the initial state where the the animals are placed. From this state the behaviour of system is different in every run, even though it has similarities. In Figure 2 you can observe number of animals during the simulation. Also In Table 1 you can observe how many times the animals procreated in different simulations.

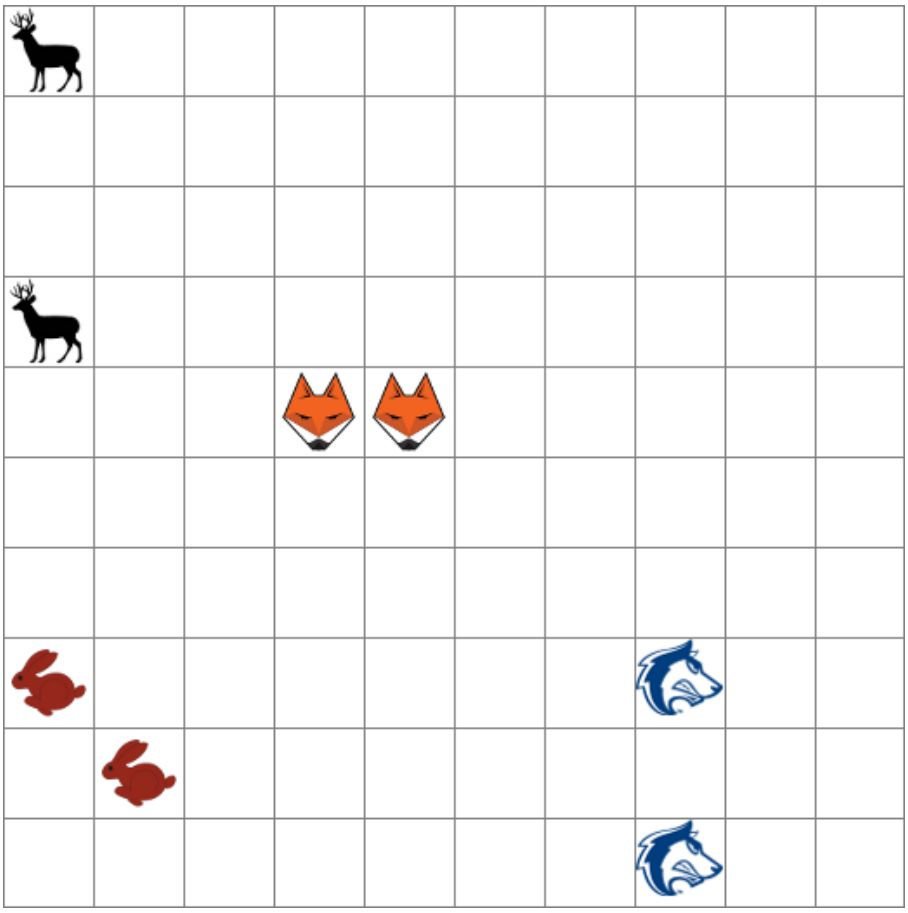

Figure 1: Initial state of animals

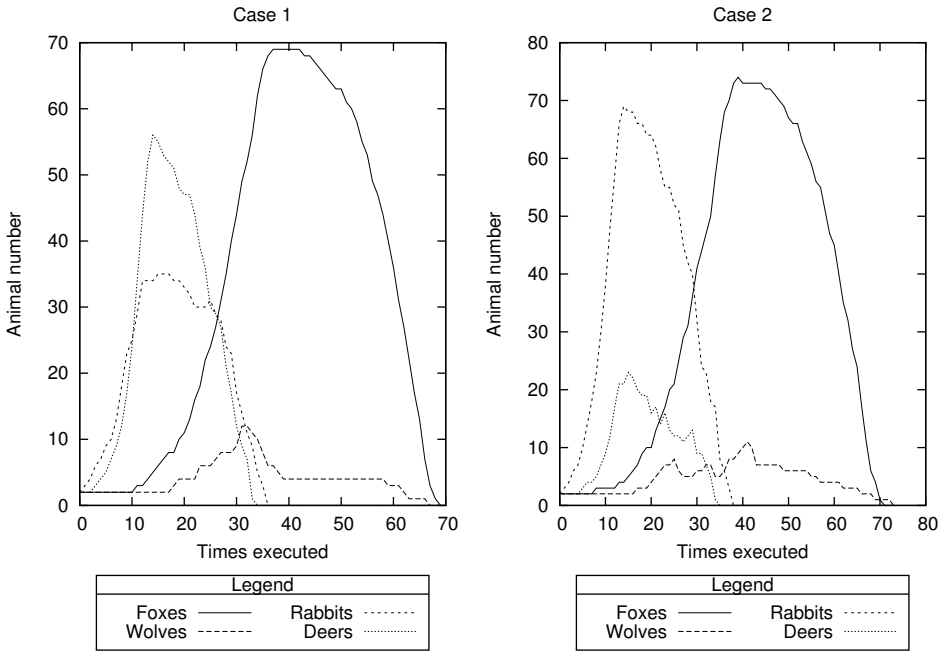

Figure 2: The Behaviour of the animals

Table 1: Procreation of animals

	Predator		Prey	
Case	Fox	Wolf	Rabbit	Deer
1	68	15	57	86
2	90	0	90	54
3	67	17	130	0
4	84	0	131	0
5	70	11	123	0
6	82	0	88	45
7	57	33	140	0
8	80	11	135	0
9	62	20	137	0
10	90	0	70	72