KING’S COLLEGE LONDON

4CCS1PPA PROGRAMMING PRACTICE AND APPLICATIONS

Third ”Simulation” Coursework (Feb 2025)

Project Name: The Simulation

Student Name:	Mehmet Kutay Bozkurt
Student ID:	23162628

Student Name:	Anas Ahmed
Student ID:	23171444

1 Introduction

This simulation project integrates multiple components to create a dynamic ecosystem that a researcher can use to study predator-prey-plant interactions, with the ability to add or remove any species or environmental factors easily by editing the given JSON file. Infact, almost every aspect of the simulation can be controlled in the JSON file. The simulation is designed to simulate without any grid-based restrictions, allowing entities to move freely in a continuous space. Specifically, the coordinates of the entities are stored as doubles from zero to the width and height of the field. The entities also now have their own genetics, which are inherited from parent(s) and may mutate, all of which are initialised in the JSON file.

The simulation smoothly runs at 60 frames per second by utilising a `QuadTree` to store entities, which allows the entity searching and collision detection to be highly optimised. In each simulation "step," every entity is updated by calling its `update()` method, which handles movement, reproduction, and hunger. Entities make decisions based on other entities in their vicinity (that is, entities that are located inside their `sight` radius) and the current state of the environment (weather and time of day). Additionally, there is a method to handle overcrowding in the simulation, for limiting the growth of entities from being unnaturally rapid.

2 Tasks Lists and Implementation Details

2.1 Base Tasks

Diverse Species: With how the simulation is implemented, adding new species is as easy as adding the species' behavioural data into a JSON file. Each entity species can be of type `Prey`, `Predator`, or `Plant`, and the data is added accordingly. For example, for the predator `Fox`, the following definitions are used:

```
{
  "name": "Fox",
  "multiplyingRate": [0.05, 0.15],
  "maxLitterSize": [1, 4],
  "maxAge": [80, 120],
  "matureAge": [40, 40],
  "mutationRate": [0.01, 0.05],
  "maxSpeed": [4, 6.5],
  "sight": [30, 50],
  "numberOfEntitiesAtStart": 12,
  "eats": ["Rabbit"],
  "size": [3, 6],
  "colour": [230, 20, 40],
  "overcrowdingThreshold": [8, 25],
  "overcrowdingRadius": [10, 15],
  "maxOffspringSpawnDistance": [3, 5]
}
```

The values that are arrays and contain two values (such as `sight`, `size`, or `maxSpeed`) represent the minimum and the maximum values that the entity can have for that genetic trait. In addition, these values can mutate when the entity breeds/multiplies. However, `numberOfEntitiesAtStart` and `eats` are fixed values that are not subject to mutation, as they are not genetic traits and they define what the entity is in the context of the simulation. Finally, the following entities are considered in the simulation:

- `Grass` — Plant.
- `Rabbit` — Prey, eats grass.
- `Squirrel` — Prey, eats grass.
- `Wolf` — Predator, eats rabbit and squirrel.
- `Fox` — Predator, eats rabbit.
- `Bear` — Predator, eats wolf and fox.

Two Predators Competes for the Same Food Source: With the JSON configuration file, it is quite easy to add multiple species for any type of entity. In this case, only two predators (`Wolf` and `Fox`) compete for the same food source (`Rabbit`), which is a prey species. The `Bear` is also added as a predator that eats both wolves and foxes.

Distinguishing Gender: Each animal has their own gender, represented in their genetics, which affects reproduction mechanics. Only animals of opposite genders can reproduce. Specifically, the gender genetic trait is implemented as an Enum with two values: `MALE` and `FEMALE`. Plants do not have gender in the genetics system, meaning that they reproduce asexually.

Tracking Time of Day: An `Environment` class is used to track the time of day and the weather, which governs how both cycles impact entity behaviour. During the night, entities will not move unless they are hungry or there is a predator nearby. Additionally, when sleeping, food consumption is reduced. The day-night cycle can be controlled by the JSON file, and the time of day is displayed on the screen. Additionally, as the day progresses into the night, the screen darkens to represent the time of day, without affecting the text on the screen.

2.2 Challenge Tasks

Adding Plants: Plants have been added, featuring growth and reproduction dynamics. Plants die when they detect too many plants of the same species nearby (as determined by their overcrowding genetics: `overcrowdingThreshold` and `overcrowdingRadius`), which results in natural looking patches of grass.

Adding Weather: As mentioned, weather is added under the `Environment` class, wherein weather conditions influence behavior and visibility, increasing the realism in the simulation environment. There are 4 weather conditions:

- `Clear` — No effect on entities.
- `Raining` — Plants grow faster (by a defined factor in the plant genetics).
- `Windy` — Pushes entities in the wind direction, even when they are sleeping. Wind direction is visualised for the ease of the user.
- `Stormy` — Slows down entities by some factor and has the effect of windy. Different to windy condition, stormy condition is more severe, in the sense that the wind changes directions much more rapidly.

Genetics System: As one of the self-admitted challenges, a genetics system for all of the entities was implemented. As mentioned earlier in the first base task, when reproducing, animals combine their parents' genetics to form their own, with a chance to mutate certain attributes by some mutation factor. Specifically, if $r \in [0, 1]$ is a random number, then the new genetic trait is calculated as:

$$\text{value} = \text{fatherTrait} \times r + \text{motherTrait} \times (1 - r),$$

allowing for a smooth transition between the parents' traits. Then, if we define $s \in \{-1, 1\}$ to be a random value, the mutation factor is applied to the new trait as follows:

$$\text{newTrait} = \text{value} + \text{value} \times \text{mutationFactor} \times s, \quad (1)$$

where the mutation factor is a value, in the range $[0, 1]$, that determines how drastic the mutation is. This system allows for a wide range of genetic diversity in the simulation, which is easily observable when the `mutationFactor` is increased. Lastly, plants reproduce asexually, so they inherit their parent's genetics directly, but these can also mutate according to Equation 1.

JSON Configuration File: Almost every single aspect of the simulation is controlled from this file, including the entities' genetics, the environment, and the simulation parameters. The JSON file is loaded at the start of the simulation, and the simulation is run according to the parameters defined in the file. As well as the entity genetic intervals mentioned above, the following are the parameters that can be controlled in the JSON file:

- `foodValueForAnimals` — Scales the food value when an entity eats an animal.
- `foodValueForPlants` — Scales the food value when entity eats a plant.
- `animalHungerDrain` — Controls the rate of hunger drain over time.
- `animalBreedingCost` — Scales how much food is consumed during breeding (note that food is a value in the range 0 to 1).
- `mutationFactor` — How drastic the mutation changes are.
- `entityAgeRate` — How fast entities age.
- `fieldScaleFactor` — The size of the field, smaller value means more zoomed in.
- `weatherChangeProbability` — The probability of the weather changing at the end of the day.

- `windStrength` — How strong the wind pushes entities.
- `stormMovementSpeedFactor` — How much to slow entities during a storm.
- `dayNightCycleSpeed` — How fast the time passes in the simulation.
- `doDayNightCycle` — Whether the day-night cycle is enabled.
- `doWeatherCycle` — Whether the weather is enabled.
- `showQuadTrees` — Whether to show the debug effect of quadtrees. It just looks really cool.
- `animalHungerThreshold` — The level of hunger when an animal is considered as ”hungry.”
- `animalDyingOfHungerThreshold` — The level when an animal is considered to be ”dying of hunger.”

3 Code Quality Considerations

3.1 Coupling

Engine, Clock, Simulator classes don't rely on each others implementations. Or how Simulator only needs to know about the update() method. Graphics is also decoupled from the rest of the code. And how it is possible to change it to make this simulation a web application (which I will do :D).

3.2 Cohesion

Controller classes for animals. The inheritence model of Entity, Plant, Animal, Prey, Predator. The genetics system. The Mutator class. The Environment class. The Field

3.3 Responsibility-Driven Design

adding QuadTree for optimisation. No need for having really slow code.

3.4 Maintainability

How it's easy to add or change the json file.

4 Final Remarks