Michael Seiranian (20127931), Dvin Amasi Hartoonian (21049232), William Atta (21097986)

Aquatic simulation

PPA Assignment 3

Simulation description

The aquatic simulation is a 2D visual simulation of an aquatic ecosystem adapted and evolved from the Foxes and Rabbits simulation created by Michael Kölling and David J. Barnes. The simulation consists of five animal species and two types of plants. Each animal is either male or female, and breeding only occurs at a certain probability when a male and a female from the same species meet (appear in adjacent cells). Three of the animals are predators (seals, eels and tuna). In the simulation, seals are represented by grey blocks, eels by green and tuna by orange. The other two animal species are prey (lobsters, represented by red and surgeonfish represented by blue. Lobsters and surgeonfish eat seaweed and algae respectively. Two of the predators (seals and eels) compete for the same food source (lobsters) and tuna eat surgeonfish. Every animal has a food level, which is refilled as they find food and is depleted by one unit per step. If they fail to find food in time when their hunger level is triggered and their food level reaches zero, they die. Animals can also die of old age or disease. Plants grow at a certain rate per step, and when eaten by preys, their growth level gets reset to zero and they begin growing again.
Some of the prey and predators are nocturnal and only move, eat and breed during night-time, whereas others only act during the day. Each step of the simulation represents half a day (12 hours), meaning the simulation switches between day and night every step.

Extension tasks

- **At least five acting species with at least two predators and at least two other species (they may eat plants)**

This task was done by creating a class for each animal species. Initially, many instances of these animal classes are initiated in the Simulator class under the “populate” method, where each animal type has a predefined creation probability, and the program iterates through every cell on the field and randomly decides whether to populate the cell and which animal to populate the cell with based on the predefined creation probabilities.

- **At least two predators should compete for the same food source**

Two of the predators (seals and eels) were set to look for the same food source; lobsters. Both of these predators are nocturnal and when they act every other step, they look to move towards a food source if they find one in an adjacent cell. The “findFood” method in the “Seal” and “Eel” classes specifies that if an animal in an adjacent cell is an instance of “Lobster”, the predators move to that cell, the instance of the lobster is set to dead using method “setDead”, and the

food values of the predators is set to a predefined constant value. When multiple seals and eels are adjacent to a lobster, the program determines which predator eats the lobster depending on which instance of the predator acts first during simulation runtime.

- **Some or all species should distinguish male and female individuals. Males and females can breed when they meet (e.g., are in neighbouring cells). Experiment with parameters to create a stable population.**

The superclass “Animal” holds a boolean field “male”, and its value determines whether an instance of an animal is male or female. This is decided randomly by the program, with a 50% chance of an animal being male and a 50% chance being female. This boolean value is inherited by the subclasses of “Animal” (the classes of individual animal species). In the “act” methods of each animal species, a method called “giveBirth” is called, again located in the individual animal classes, which finds free adjacent cells to potentially populate new offspring in and calls the “breed” method in the superclass “Animal”. The breed method along with other methods would have been the same for all animals (checking whether two animals can breed and if a random number meets the probability of breeding), hence it was included in the Animal superclass to reduce code redundancy and it is inherited by the individual animal classes. The breed method calls the boolean method “canBreed” in individual animal classes, where the program checks whether two adjacent animals can breed based on their sex and species type. The boolean value is then retrieved through inheritance and it is used along with a couple of other conditions to decide whether to breed. The offspring are added to a list in the “giveBirth” method, which is returned to the “act” method called by “simulateOneStep” method being called on every step, where the program adds the offspring to the list of all animals to be displayed on the field.
To create a stable population, we used trial and error to tweak many of the constant predefined parameters, and measured success by the number of steps that all species would last. Based on this measurement, we continuously tweaked the values to improve the stability of the population.

- **Keep track of the time of day. At least some creatures should exhibit different behaviour at some time of the day (e.g.: they may sleep at night and not move around during that time).**

The “Simulator” class holds a boolean value “day”, which is true during daytime and false during night-time. Each time the “simulateOneStep” method is called on every step of the simulation, the “changeTimeOfDay” method is called changing the boolean value of “day”. This value is static, so that it can be retrieved by instances of animals. In the “act” methods of all animals, the program checks whether “day” is true or false to decide whether the animals should act or not, depending on whether or not they are nocturnal.

- **Simulate plants. Plants grow at a given rate, but they do not move. Some creatures eat plants. They will die if they do not find their food plant.**

A similar inheritance structure to animals is used for plants. A “Plant” superclass links the individual plant classes “Alga” and “Seaweed”. Instances of plants are randomly initiated on the map, similarly to animals. A plant can coexist in the same cell as an animal, however plants are

not visually represented in the simulation. A 3D array stores all plants and animals to allow the coexisting of a plant and an animal in a single cell. The first two dimensions of the array specify the location of the cell on the map (the row and column), and the third dimension is always either a value of 0 to store an animal or a value of 1 to store a plant in the cell defined by the first two dimensions. The array is called field and is generated in the “Field” class. Plant growth is done by assigning each plant a growth level, and increasing the value on each step at a certain specified rate for each plant type, depending on the weather conditions. Plants stop growing at a certain growth level specified in the individual plant classes. Preys look for specific types of plants to consume in their “findFood” methods when they are hungry, which adds the growth value of the plant multiplied by a constant to their food values. When a plant is consumed, the growth level of the plant is reset to 0. If a prey fails to find enough grown plants and their food value reaches 0 through decrementation on each step, they die through the “setDead” method.

- **Simulate weather. Weather can change, and it influences the behaviour of some simulated aspects.**

A “Weather” class was created specifying three weather conditions; sunny, cloudy and rainy. Three boolean values of sunny, cloudy and rainy are set to true and false depending on which weather condition it is. The “simulateOneStep” method in the “Simulator” class calls the “changeWeather” method from the “Weather” class every two steps. The method randomly picks a weather condition from the three possibilities. This affects plant growth. On sunny weather, the increment growth method for the plants is called twice, on cloudy weather it is called once and on rainy weather it is not called.

- **Simulate disease. Some animals are occasionally infected. Infection can spread to other animals when they meet.**

A boolean value “infected” is stored for every instance of animal in the “Animal” superclass. A probability of 0.5% of being randomly infected during each step is set. When infected, the animal is only set to infect another animal that it can breed with. This ensures that when a spread of disease occurs, the disease does not jump from different species to other species, and only to other animals of the same species, which is usually the case in the real world. Hence, the “infect” method is called in each animal’s “canBreed” method. The “infect” method takes the adjacent animal of the same species and uses a 50% probability to infect the animal, making the transmissibility of the disease 50%. If the adjacent animal is infected and the current animal is not, the disease transmission still occurs in the “infect” method, again with 50% chance. Once infected, the animal’s “infectedCount” variable increments through each step, and once the count reaches 20, the animal is “setDead”, making the animals only be able to withstand the disease for 20 steps before they die. If the count is less than 20, the “setHealthy” method is called each step, which gives them a chance of 5% to recover during each step when infected.

Conclusion

Upon changing the constant values through trial and error to create a stable population, we found certain species would go extinct more than others. Due to this, we removed lobsters' ability to become infected, which aided the longevity of a stable and balanced population of the ecosystem. However, there is room for improvement to ensure a longer lasting balanced population, which could be done easier through data analysis tools to better guide the tweaking of the constant values.