PPA Assignment 3 Report
Name: Shin-Hung Lin, Max Chang, Ting-Chen Chen
Student Number: K21068460, K21003979, K20076342

Description of Simulation:

The setting of the simulation is in a farm. There are mainly six kinds of species in this farm with some interactions. Some of the species will be the predators and some of the species will be the preys. Through the simulator, we included the different probabilities that one kind of species can be created in given grid position. Also, we indicate that different species will have different color to represent. For example, sheep's color will represent in pink dots and the chickens will be represented in orange dots, etc. In this project, we will add in five different kinds of acting species, treat plants to simulate their growth. Also, at least there will be two species that hunting the same species. For example, the racoons and the foxes will hunt and eat the same species of chicken. In addition, this simulation will have different gender in different species for breeding to stable the total number of populations. This simulation of farm also represents some of the creatures will have different behavior in different time. Wolves will be moving only at night. The plants will be grown in specific rate and some creatures will eat them to survive.

Additional description updated:

In our final program, there are only four species (wolves, foxes, chicken, and grass), since the ecosystem of our simulation isn't stable, and four species can make the population stable. However, our coding functions do work. If all the species implement into the program, some of the creatures will die fast (by few steps only). In addition, to maintain the ecosystem, we changed food of some of the predators. Species of wolves will be the predator of chicken and chicken will be eating grass.

Types of species simulating:

- Racoons
- Chickens
- Sheep
- Cows
- Foxes
- Wolves
- Grass
- Grain

Each of the species' behavior and interaction:

- Racoons (predator)eat chicken
- Chickens(predator)eat grain
- Sheep(prey) eat grass
- Cows(prey) eat grass
- Foxes(prey) eat chicken
- Wolves (predator)eat sheep and cows

Description of all extension tasks that we implemented:

Basic task

- Create at least five different species

  We created new classes in this project, including racoons, wolves, foxes, chicken, and sheep. Besides those classes for the animals, we have created two abstract classes for predator and preys. Different species will be represented in different color of dots. To do this, we add the setColor method in SimulatorView class. How the species move is determined by the Findfood method with the while loop will determine what kind of predator will eat what kind of species of prey. The animal will get hungry by walking one step the Food value will decrease in one. For the code, this behavior will be mainly determined by Findfood method. Overriding technique can be used for a predator that eats two kinds of species of animal.

```
@Override
protected Location findFood(){
    Field field = getField();
    List<Location> adjacent = field.adjacentLocations(getLocation());
    Iterator<Location> it = adjacent.iterator();
    while(it.hasNext()) {
        Location where = it.next();
        Object animal = field.getObjectAt(where);
        if(animal instanceof Sheep) {
            Sheep sheep = (Sheep) animal;
            if(sheep.isAlive()) {
                sheep.setDead();
                foodLevel = SHEEP_FOOD_VALUE;
                return where;
            }
        }
        else if(animal instanceof Cow){
            Cow cow = (Cow) animal;
            if(cow.isAlive()){
                cow.setDead();
                foodLevel = COW_FOOD_VALUE;
                return where;
            }
        }
    }
    return null;
}
```

- At least two predators will compete the same species

  To execute this section, foxes and racoons will be set as predators and the common prey is the chicken. For example, in the class of fox, we insert chickenfoodvalue method from superclass. If the fox eats the chicken with the food value of 9, the fox's food value will have an increment of 9.

- Gender distinguishes, breeding from meeting each other in a specific distance

  By doing this task, we use Boolean to determine the gender. Male will be set as true, and female will be set as false. First, we created the isMale protected variable in the class of animal. In the findpartner method, we apply isMale variable, and by canbreed method, the animals will breed if the animals are in mature age and there's a mate next to them with different gender.

```
protected boolean findsPartner()
{
    Field field = getField();
    List<Location> adjacent = field.adjacentLocations(getLocation());
    Iterator<Location> it = adjacent.iterator();
    while(it.hasNext()) {
        Location where = it.next();
        Object entity = field.getObjectAt(where);
        if(entity instanceof Animal) {
            Animal mate = (Animal) entity;
            if (this.getClass().equals(mate.getClass())
            && mate.isAlive() && this.isMale != mate.isMale) {
                return true;
            }
        }
    }
    return false;
}
```

- Track of the time of day, and species will have different behavior in different time

  We created a new Time class and put isNight variable in the constructor. The ChangeDay method will work with the isNight variable. If the timing is night, it will be false. The counter will be count as every five steps, at first, it will be daytime, and from the fifth step, it will turn into night. From entity class, we insert gettime method before act method since the time will affect different species of behavior.

```
public void simulateOneStep()
{
    step++;
    // Provide space for newborn entities.
    List<Entity> newEntities = new ArrayList<>();
    // Let all entities act.
    if (step % 5 == 0) {
        time.changeDay();
    }

    for(Iterator<Entity> it = entities.iterator(); it.hasNext(); ) {
        Entity entity = it.next();
        entity.getTime(time.getTime());
        entity.act(newEntities);
        if(! entity.isAlive()) {
            it.remove();
        }
    }
```

Challenge Tasks

- Simulate the plants

  In the class of plant, we add the breedingprobability variable to set the growth rate. The plants won't move is because of the setting the new location method will just make the plants finding spaces to breed but not moving.

```
public void act(List<Entity> newPlants)
{
    super.act(newPlants);
    if (!isNight || rand.nextDouble() <= NIGHT_GROW_PROBABILITY) {
        if(isAlive()) {
            giveBirth(newPlants);
            Location newLocation = getField().freeAdjacentLocation(getLocation());
            if (newLocation == null) {
                setDead();
            }
        }
    }
}
```

Problems that we have faced/ bugs that we have

- Any animal can give birth to any animal
  We put give birth method into the specific class of animal instead of putting into the animal abstract class to fix the bug.

- Population stability problem

We found out that all our creatures in the simulation died too fast in less than 100 steps or some of the species of preys died too fast cause to some of the predators died too fast, so we need to try out different value of mature age, maximum age, breeding probability, maximum litter size, and food value of the preys to maintain the population in our farm simulation. Since there are too many variables, it is difficult to make the whole population stable.

- More population maintainability related problem/bugs
  From the method of populate in class of simulator, the order of the species that we put into the for loop will also affect the creature's creation probability. This issue will cause to instability of population, although the coding function is working.
  Below we can see that we put fox creation probability in the very front of the if else statement, that means that the fox might be created first and more with higher probability.

```
Random rand = Randomizer.getRandom();
field.clear();
for(int row = 0; row < field.getDepth(); row++) {
    for(int col = 0; col < field.getWidth(); col++) {
        if(rand.nextDouble() <= FOX_CREATION_PROBABILITY) {
            Location location = new Location(row, col);
            Fox fox = new Fox(true, field, location);
            entities.add(fox);
        }
        else if(rand.nextDouble() <= CHICKEN_CREATION_PROBABILITY) {
            Location location = new Location(row, col);
            Chicken chicken = new Chicken(true, field, location);
            entities.add(chicken);
        }
        else if(rand.nextDouble() <= RACOON_CREATION_PROBABILITY){
            Location location = new Location(row, col);
            Racoon racoon = new Racoon(true, field, location);
            entities.add(racoon);
        }
        else if(rand.nextDouble() <= SHEEP_CREATION_PROBABILITY){
            Location location = new Location(row, col);
            Sheep sheep = new Sheep(true, field, location);
            entities.add(sheep);
        }
        else if(rand.nextDouble() <= COW_CREATION_PROBABILITY){
            Location location = new Location(row, col);
            Cow cow = new Cow(true, field, location);
            entities.add(cow);
        }
        else if(rand.nextDouble() <= WOLF_CREATION_PROBABILITY){
            Location location = new Location(row, col);
            Wolf wolf = new Wolf(true, field, location);
            entities.add(wolf);
        }
        else if(rand.nextDouble() <= GRASS_CREATION_PROBABILITY) {
            Location location = new Location(row, col);
```