МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа №4**

по дисциплине «Технология программирования»

на тему: «Многопотоковые приложения»

|  |  |
| --- | --- |
| Группа: АВТ-808 | Преподаватель: Михайленко Дмитрий Анатольевич |
| Студент: Кузьмина Александра |  |
| Вариант: 1 |  |
|  |  |

Новосибирск, 2020

**Цель работы**

Изучить особенности реализации и работы потоков в Java, управлением приоритетами потоков и синхронизацией потоков.

**Задание**

Доработать программу, созданную в лабораторной работу № 3:

1. Создать абстрактный класс BaseAI, описывающий «интеллектуальное поведение» объектов. Класс должен создавать поток, обеспечивающий движения объектов коллекции;
2. Реализовать класс BaseAI для каждого из видов объекта, включив в него поведение, описанное в индивидуальном задании по варианту;
3. Синхронизовать работу потоков расчета интеллекта объектов, их рисования и генерации новых объектов. Рисование должно остаться в основном потоке;
4. Добавить в панель управления кнопки для остановки и возобновления работы интеллекта каждого вида объектов. Реализовать через засыпание/пробуждение потоков (методы wait() и notify());
5. Добавить в панель управления выпадающие списки для выставления приоритетов каждого из потоков.

**Вариант 1**

1. Муравьи-рабочие двигаются в один из углов области их обитания (например, [0;0]) по прямой со скоростью V, а затем возвращаться обратно в точку своего рождения с той же скоростью.
2. Муравьи-воины двигаются по окружности с радиусом R со скоростью V.

**Ход работы**

В ходе решения поставленных задач доработки программы были добавлены новые классы и внесены соответствующие изменения в ранее разработанные:

1. Создан абстрактный класс **BaseAI**, наследник класса **Thread**. В данном описывается “интеллектуальное поведение” объектов и создается поток, который обеспечивает движение объектов.

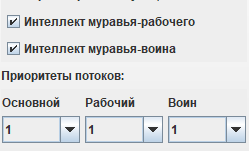


2. Класс BaseAI реализован для двух видов объекта муравья: **WorkerAI** и **WarriorAI**. В конструкторах данных классов запускается собственный поток для интеллекта каждого вида муравья, а в переопределенном методе run() происходит вызов метода move(), отвечающего за движение объектов данного вида.





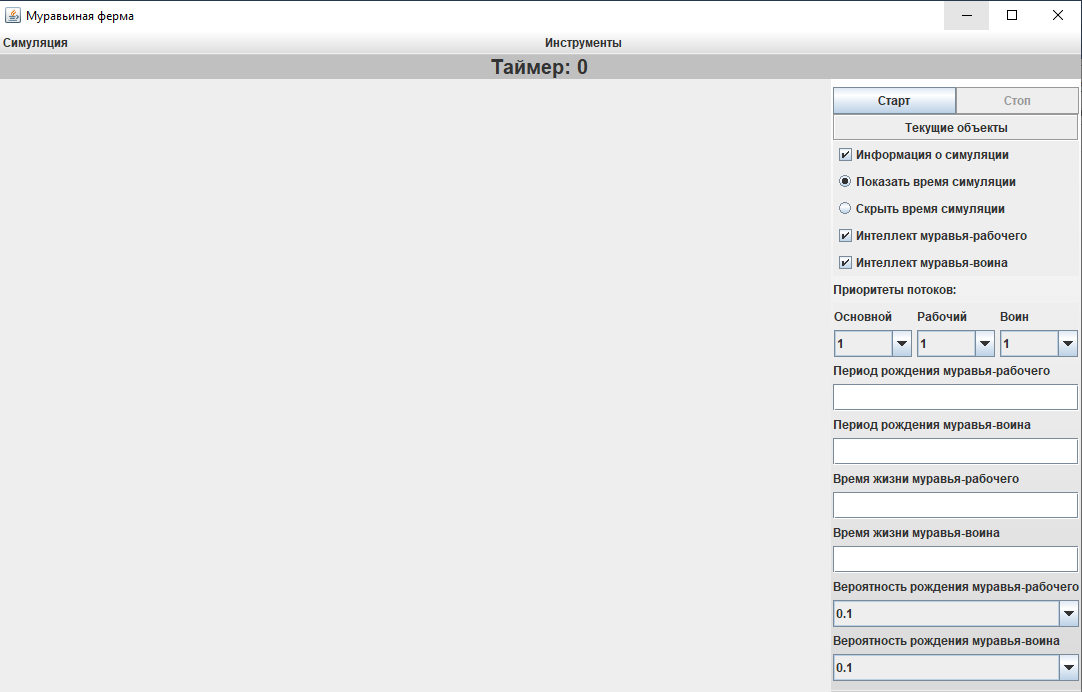
3. На панель управления добавлены выпадающие списки для выставления приоритета каждого потока и кнопки для остановки/возобновления работы интеллекта каждого вида объекта.



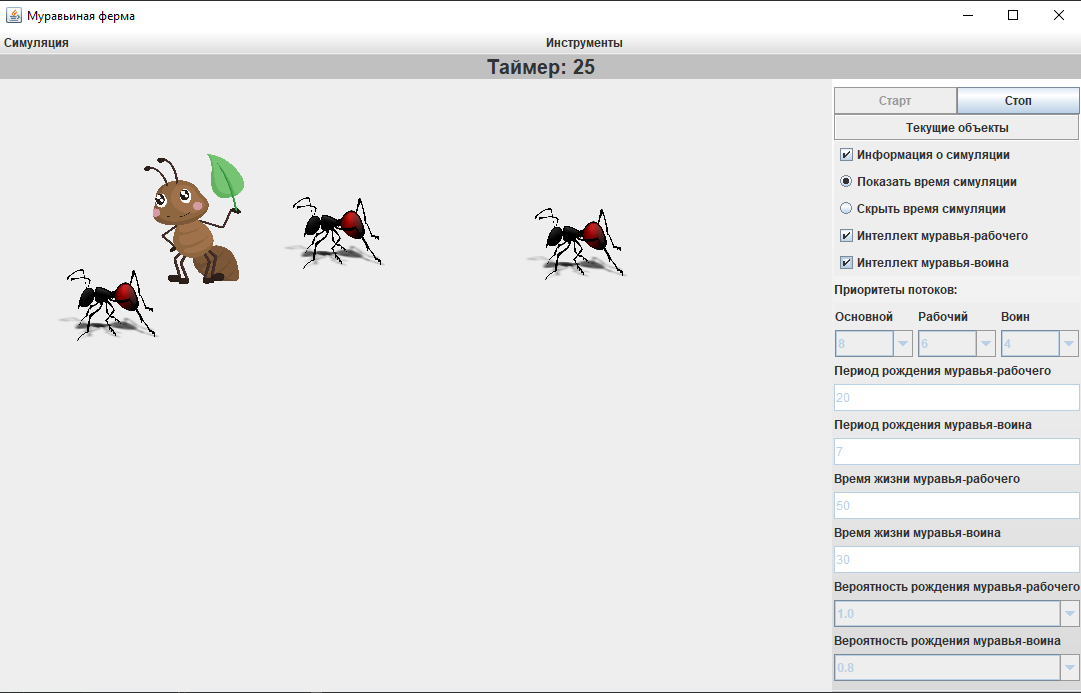
Остальные классы программы были переработаны в соответствии с новыми задачами.

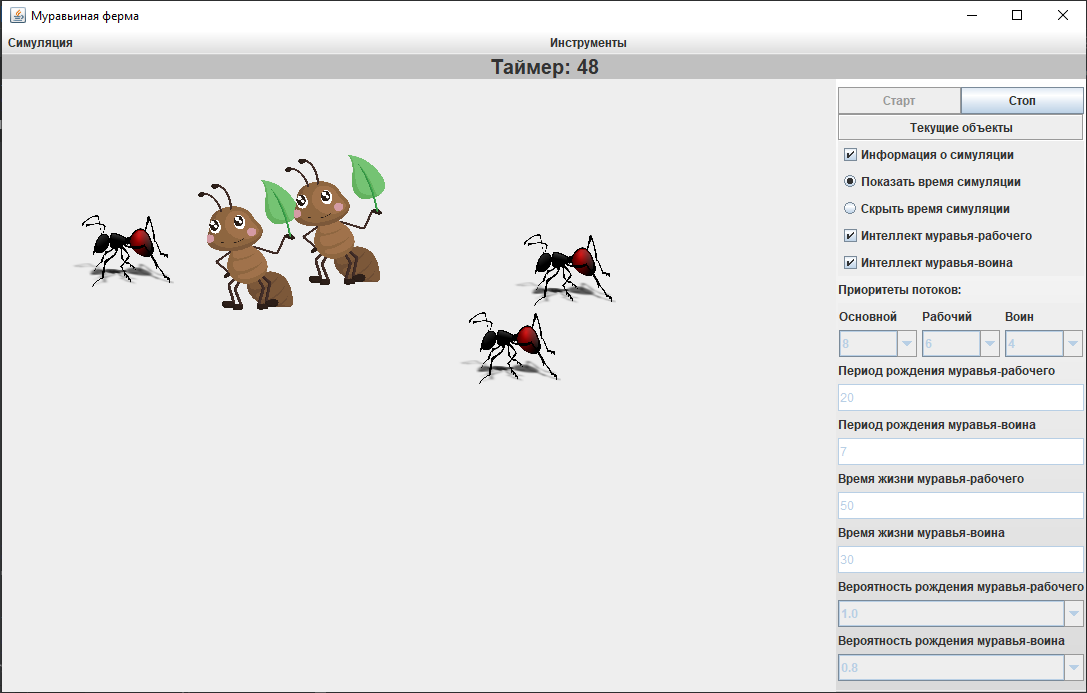
**Скриншоты работы программы**

Окно до запуска симуляции:



Окно во время симуляции:





**Вывод**

При выполнении данной работы были изучены особенности реализации многопоточности и синхронизации потоков в Java. Был расширен интерфейс программы, созданной в предыдущих лабораторных работах; теперь каждый вид объекта имеет свой вид поведения и движется в области симуляции.

**Листинг программы**

**BaseAI**

package AntFarm;  
  
public abstract class BaseAI extends Thread  
{  
 public synchronized void startAI() {}  
 public synchronized void stopAI() {}  
}

**WorkerAI**

package AntFarm;  
  
public class WorkerAI extends BaseAI  
{  
 public static Thread *thread*;  
 public boolean isMoving;  
 Singleton ants = Singleton.*getSingleton*();  
  
 WorkerAI()  
 {  
 *thread* = new Thread(this, "WorkerAI");  
 isMoving = true;  
 *thread*.start();  
 }  
  
 @Override  
 public void run() {  
 while (true) {  
 synchronized (this)  
 {  
 while (!isMoving)  
 {  
 try  
 {  
 wait();  
 }  
 catch (InterruptedException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 if (ants.getArraySize()!=0)  
 {  
 for (int i=0; i<ants.getArraySize();i++)  
 {  
 if (ants.*getAnts*().get(i).getClass() == AntWorker.class)  
 ants.*getAnts*().get(i).move();  
 }  
 }  
 try  
 {  
 Thread.*sleep*(500);  
 }  
 catch (InterruptedException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 @Override  
 public synchronized void startAI()  
 {  
 isMoving = true;  
 notify();  
 }  
  
 @Override  
 public void stopAI()  
 {  
 isMoving = false;  
 }  
}

**WarriorAI**

package AntFarm;  
  
public class WarriorAI extends BaseAI  
{  
 public static Thread *thread*;  
 public boolean isMoving;  
 Singleton ants = Singleton.*getSingleton*();  
  
 WarriorAI()  
 {  
 *thread* = new Thread(this, "WarriorAI");  
 isMoving = true;  
 *thread*.start();  
 }  
  
 @Override  
 public void run() {  
 while (true) {  
 synchronized (this)  
 {  
 while (!isMoving)  
 {  
 try  
 {  
 wait();  
 }  
 catch (InterruptedException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 if (ants.getArraySize()!=0)  
 {  
 for (int i=0; i<ants.getArraySize();i++)  
 {  
 if (ants.*getAnts*().get(i).getClass() == AntWarrior.class)  
 ants.*getAnts*().get(i).move();  
 }  
 }  
 try  
 {  
 Thread.*sleep*(500);  
 }  
 catch (InterruptedException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 @Override  
 public synchronized void startAI()  
 {  
 isMoving = true;  
 notify();  
 }  
  
 @Override  
 public void stopAI()  
 {  
 isMoving = false;  
 }  
}