Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерных технологий»

Е.А. Цымбалюк, А.А. Шалыто

Разработка искусственного интеллекта на основе Switch-технологии

(реализация интеллекта «Хрюшки» для проекта *Electric Jungle*)

Проектная документация

Санкт-Петербург
2007

Оглавление

Введение

В современной ИТ-индустрии часто проводятся конкурсы программистов. Испытания для молодых талантов могут быть самыми разнообразными, например, олимпиады *ACM*[1], *TopCoder*[2] и т.д. Одной из разновидностей подобных конкурсов является разработка интеллекта танка (*Robocode* [1, 2]) или существа (*Terrarium* [3], *Electric Jungle* [4, 5]). Задание для конкурса выполняется с применением какой-то специальной технологии, для развития которой данное мероприятие и предназначено.

В данной работе описан искусственный интеллект существ для проекта *Electric Jungle*, созданный с применением *SWITCH*-технологии [6]. Игра (процесс жизни существ на игровом поле) является пошаговой стратегией, цель которой – максимальная видовая экспансия. На игровом поле случайным образом расставляется по одному животному каждого вида (в зависимости от режима игры, количество видов животных может отличаться). Игра длиться в течение заранее известного числа ходов. На каждом шаге существо может производить какое-то действие:

- перейти в другую клетку (доступность клетки определяется скоростью существа);
- потребить энергию;
- атаковать другое существо;
- породить другое существо;
- совмещено двигаться и атаковать.

У каждого существа есть масса и энергия. Из названия проекта понятно, что существа – электрические (работают как бы «на батарейках»). У них есть некоторая начальная энергия, которая тратится на все действия: на движение, атаку, порождение новых существ и т.д. Единственное действие, на которое энергия не тратится, это потребление энергии. Для потребления энергии (для того, чтобы существо не погибло) на игровом поле находятся расположенные случайным образом источники энергии. Каждый источник имеет некоторый начальный запас энергии, а также скорость восполнения энергии. Существо, находящееся в одной клетке с источником, может подзарядиться.

Более детальное описание правил можно прочесть на сайте проекта [3].

---

[1] http://www.acm.org

[2] http://www.topcoder.com

Поле игры приведено на рис. 1.

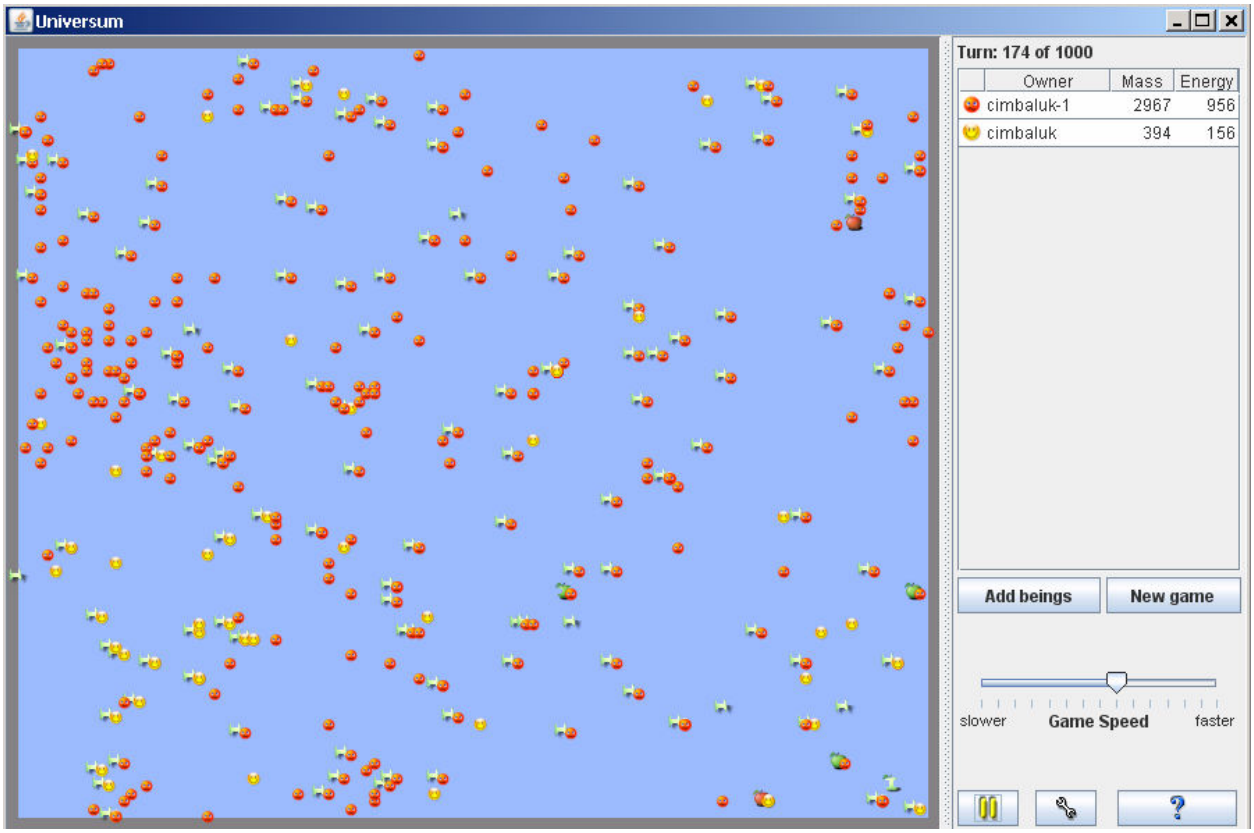


Рис. 1. Поле игры

Слева на рис. 1:

 и  – животных из соответствующих команд на игровом поле;

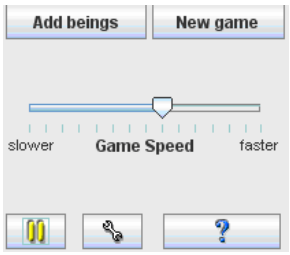
 – источники энергии;

 – богатые источники энергии.

Справа на рис. 1:

 – номер текущего хода;

 – список существ, участвующих в игре (а так же информация о суммарной массе и энергии существ);

 – элементы управления и настройки игры.

1. Постановка задачи

Целью данной работы является разработка интеллекта (если быть точным, группового интеллекта, так как в процессе игры может быть произвольное количество существ, которые должны взаимодействовать друг с другом и т.д.), удовлетворяющий правилам игры в проекте *Electric Jungle.* Существо должно быть способно:

- разведывать (исследовать игровое поле на предмет наличия источников);
- размножаться (увеличивать популяцию своего вида);
- подзаряжаться (контролировать количество энергии, необходимой для выживания, и пополнять ее);
- сражаться (атаковать существ другого вида для максимальной экспансии).

2. Формальное описание

Основываясь на изучении правил и устройства игры *Electric Jungle* [1], выделим следующие состояния, в которых может находиться разрабатываемое существо:

- разведка (начальное состояние существа, которое выполняет разведку игрового пространства и поиск источников);
- кормление (состояние, в котором существо находится у источника и потребляет энергию; набрав определенное количество энергии, существо размножается – данное состояние является основным для увеличения популяции);
- подкармливание (в отличие от «кормления» подкармливание – разовое потребление энергии, оставшейся от погибшего существа, что в процессе игры случается нередко);
- выживание (если у существа слишком мало энергии, либо нечем заняться, то остается лишь ждать, когда все закончится);
- сражение (атака противника, состояние устранения угрозы, нависшей над собственной популяцией).

2.1. Разведка

Как отмечено выше, это состояние является начальным и базовым для разрабатываемых существ. Поэтому именно в нем выполняется анализ происходящего в мире и принятие решений. Правила переходов и действий существа:

- поблизости есть враг – переход в состояние «сражение»;
- размножаемся, если есть достаточно энергии для выживания после порождения нового существа до конца игры и существуют источники, из которых кормятся другие существа (это правило для размножения, необходимое для предотвращения траты энергии на начальных этапах развития);
- поблизости есть источник (если несколько источников, тогда выбираем лучший из них), который позволяет за короткое время породить потомство – переход в состояние «кормление»;
- поблизости есть энергия, которая не удовлетворяет требованиям воспроизводства – переход в состояние «подкармливание»;
- осталось совсем мало энергии (достаточное лишь для выживания до конца игры) – переход в состояние «выживание»;

- поблизости есть неразведанные клетки – отправляемся на разведку;
- есть неразведанные клетки, находящиеся в пределах досягаемости (достаточно энергии для достижения клетки и последующего выживания) – отправляемся на разведку;
- если нечего разведывать – переходим в состояние «выживание».

2.2. Кормление

Данное состояние отвечает за перемещение к источнику для кормления, непосредственно потребление энергии, размножение и выход из состояния кормления при истощении источника:

- если не достигли источника кормления – перемещаемся к нему;
- достаточно энергии для размножения – порождаем новое существо;
- источник истощился – переходим в состояние «выживание»;
- иначе, потребляем энергию.

2.3. Подкармливание

Как и состояние «кормление», данное состояние отвечает за потребление энергии и размножение. Основным отличием является то, что источник энергии одноразовый:

- если не достигли точки подкармливания – движемся к ней;
- можем размножиться – порождаем новое существо;
- не осталось еды – переходим в состояние «разведка»;
- иначе, потребляем энергию.

2.4. Выживание

В этом состоянии существо ничего не делает, если еще есть энергия для выживания, пытается потребить энергию, если стоит на источнике, либо ничего не делает. Также существо пытается перейти в состояние выживания для того, чтобы проверить не появилось ли для существа какого-нибудь применения (например, невдалеке нашелся источник энергии):

- один раз за ход переходим в состояние «разведка» (проверяем, что новых дел не появилось);
- энергия не является критической – ничего не делаем;
- потребляем энергию, необходимую для выживания, если находимся рядом с источником.

2.5. Сражение

Состояние сражения отвечает за атаку вражеских существ, а так же за возврат в состояние разведки:

- невдалеке находится враг – перемещаемся и атакуем его;
- находимся в той же клетке с врагом – атакуем его;
- врага нет поблизости – переходим в состояние «разведка».

3. Описание входных переменных

x1_1 – поблизости есть враг;

x1_2 – есть достаточное количество энергии для выживания после порождения нового существа;

x1_3 – поблизости есть источник, позволяющий за короткие сроки породить потомство;

x1_4 – поблизости есть энергия;

x1_5 – осталось мало энергии (ниже критической);

x1_6 – рядом есть неразведанные клетки;

x1_7 – есть неразведанные клетки, находящиеся в пределах досягаемости;

x2_1 – еще не достигли цели (точки назначения);

x2_2 – энергии достаточно для размножения;

x2_3 – источник истощился;

x3_1 – не осталось еды;

x4_1 – условие выполняется первый раз за ход;

x4_2 – энергия не является критической;

x5_1 – рядом находится враг;

x5_2 – в нашей клетке находится враг.

4. Описание выходных переменных

`Z1` – порождаем новое существо;

`Z2` – перемещаемся в выбранном направлении;

`Z3` – потребляем энергию;

`z4` –пропускаем ход;

`Z5` – потребляем энергию, необходимую для выживания;

`Z6` – одновременно перемещаемся и атакуем врага;

`Z7` – атакуем врага.

5. Схема связей автомата

На рис. 2 приведена схема связей автомата.

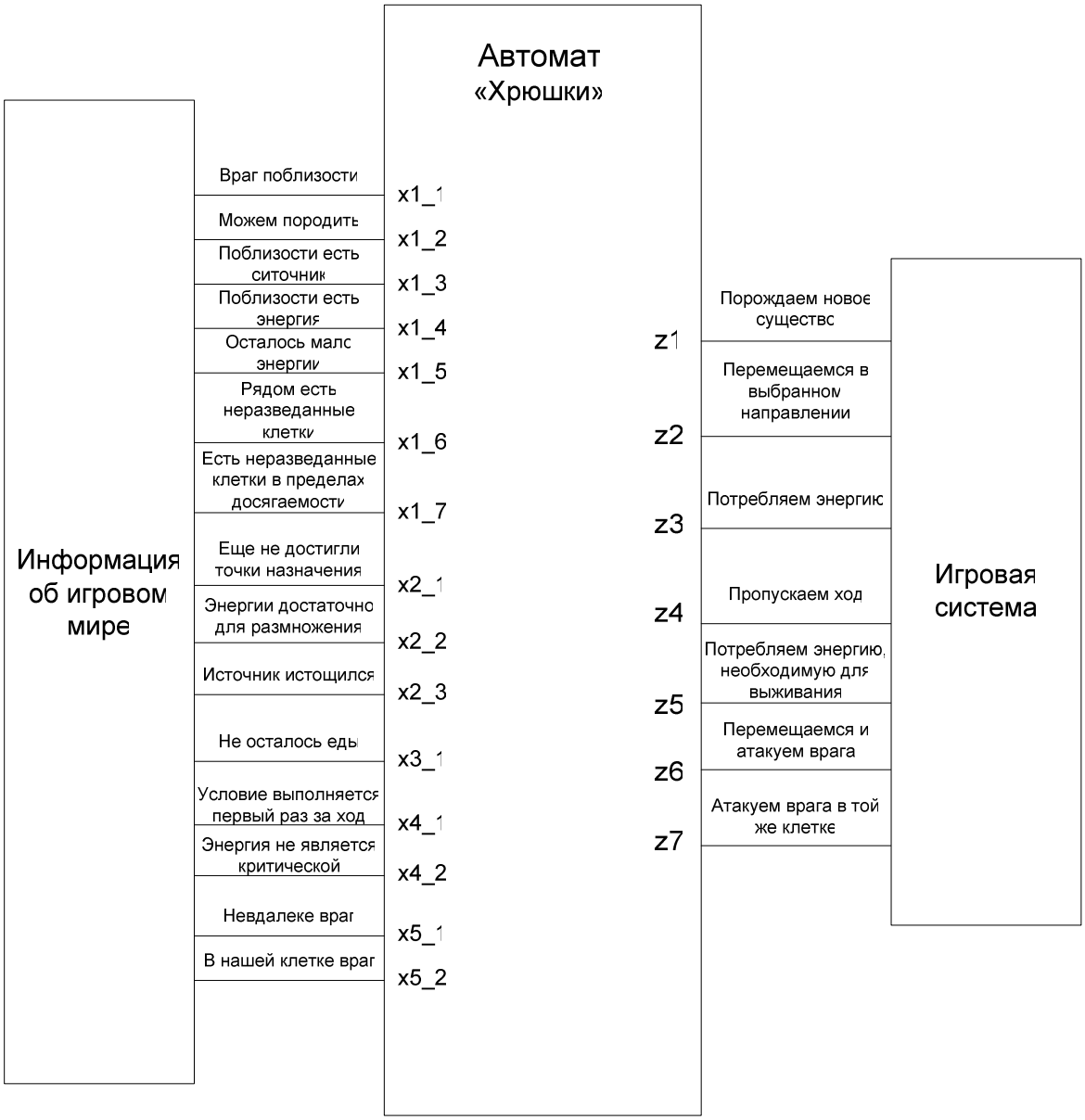


Рис.2. Схема связей автомата

6. Граф переходов

На рис. 3 приведен граф переходов автомата.

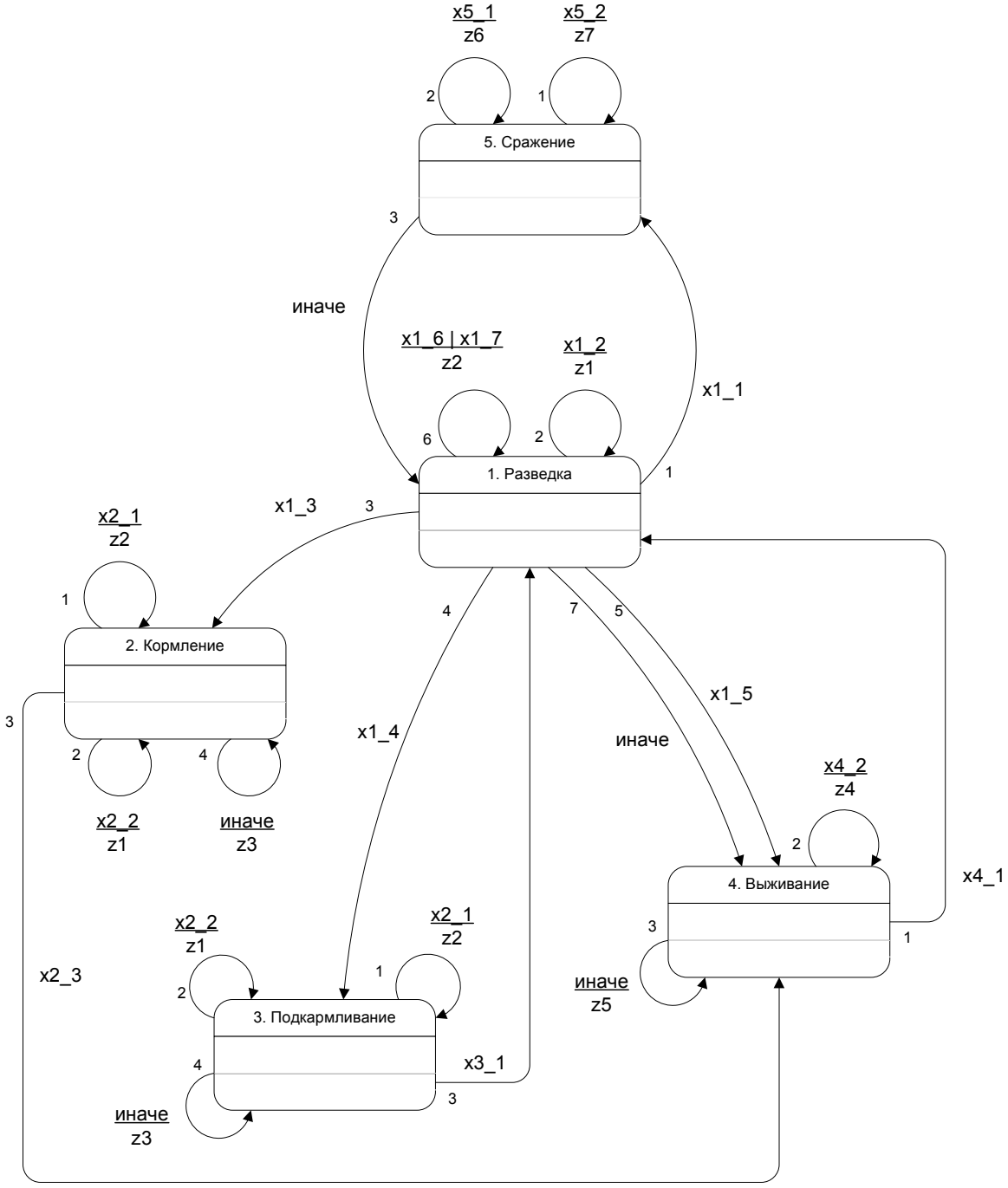


Рис 3. Граф переходов автомата

7. Реализация

Интеллект в виде графа переходов (рис.3) реализован на языке *Java* в соответствии с требованиями проекта *Electric Jungle*. В файле *SuperPiglet.java* (приложение 1) и *SuperHerder.java* (приложение 2) содержатся базовые классы (общие методы и данные) для интеллекта. Файл *Piglet.java* (приложение 3) содержит реализацию автомата, а файл *Herder.java* (приложение 4) – описание класса-контейнера, содержащего общие для всех существ данные и вспомогательные методы. В приложении 5 приведен фрагмент протокола работы автомата.

В данной реализации применены следующие алгоритмы и методы.

1. Алгоритм выбора направления разведки. При этом рассматриваются блоки `9*9`, находящиеся по соседству с текущей позицией. Центр блока, имеющего наибольшее число неразведанных точек, выбирается направлением разведки. Если несколько блоков имеют одинаковое число неразведанных точек, то выбирается случайный блок. Данный рандомизированный подход обеспечивает равномерное распространение существ по игровому полю, что повышает скорость нахождения богатых источников, а, следовательно, скорость размножения.

2. Оптимизация поиска ближайшей неразведанной клетки. При поиске такой клетки применен алгоритм полного обхода по игровому полю. Полностью исследованные столбцы помечаются как исследованные. Таким образом, значительно сокращается количество лишних итераций цикла поиска. Благодаря данной оптимизации существа (количеством более `500`) могут выполнять разведку с приемлемой скоростью, удовлетворяющей требованиям системы.

3. Метод сбора общей игровой информации. Для сбора информации об игровом поле в начале хода каждое существо вызывает специальную процедуру. Она с помощью критической секции [7] и списка рожденных существ позволяет подсчитывать номер текущего шага, который используется при разведке для пометки клеток, в которые уже движется существо. Процедура также позволяет собрать информацию об окружающем мире до того, как существа приняли решения о действиях на данном ходе.

4. Оптимизация скорости доступа к данным. Для повышения производительности при доступе к общим данным (например, к игровому полю) использовалась не критическая секция, а специальный примитив синхронизации *ReadWriteLocker* [7]. Особенностью этого примитива является возможность нескольким потокам осуществлять доступ к данным на чтение, и лишь при изменении данных требовать эксклюзивный

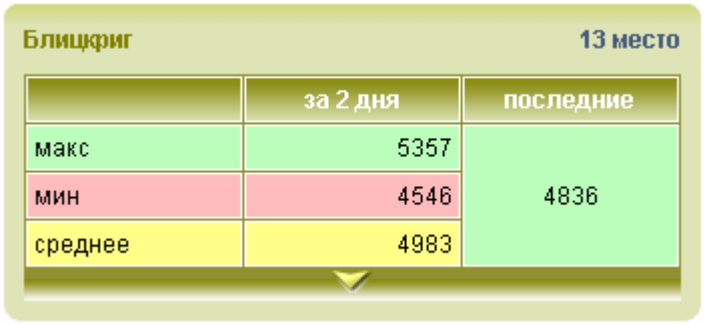
доступ (в то время, как в критической секции доступ всегда эксклюзивный). Данный подход значительно повышает производительность системы (совокупности интеллектов всех существ), так как операции на запись производятся значительно реже, чем на чтение.

Выводы

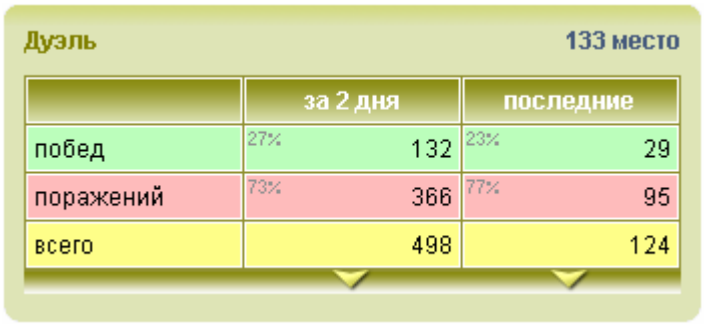
При разработке интеллекта основное внимание уделялось одному из конкурсов – «Блицкриг». При этом одному виду существ дается весь мир. Цель — добиться наивысшего рейтинга путем максимально эффективного использования ресурсов и быстрой разведки за данный временной интервал (200 ходов). Поэтому хороший результат был достигнут именно в этом конкурсе.

Существо участвует в конкурса под названием *Piglets* (автор *E13*) и занимает в конкурсе:

- «Достижения в одиночных играх» http://www.electricjungle.ru:8080/main.jsp?page=fullrating&gamekind=SINGLE **13** место[3].

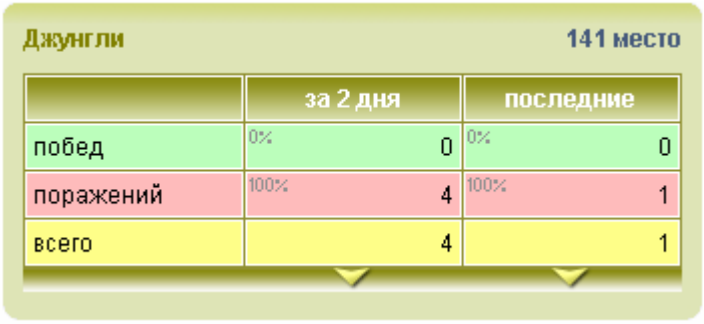


- «Сильнейшие популяции по результатам дуэлей» http://www.electricjungle.ru:8080/main.jsp?page=fullrating&gamekind=DUEL ??? **133** место.



- «Самые живучие существа джунглей» http://www.electricjungle.ru:8080/main.jsp?page=fullrating&gamekind=JUNGLE **141** место.

[3] Данные рейтинга верны на момент написания работы



Разработанный интеллект достаточно стабилен при разведке и размножении существ. При этом под стабильностью понимается, что игровое поле генерируется случайным образом, и поэтому всегда существует элемент удачи при разведке. Таким образом, невозможно гарантировать, что за некоторое небольшое число ходов будет найден богатый источник энергии, и вообще, что источник найдется. Несмотря на это, алгоритм при одиночном режиме развития стабильно развивается в среднем от 4000 до 5000 единиц массы за 200 ходов (правила одиночного режима игры).

Боевая система интеллекта разработана лишь на простейшем уровне. При продолжении работы над интеллектом существ можно повысить степень взаимодействия животных между собой при возникновении угрозы: идти на подмогу, если рядом нашелся враг. Также можно разведывать места (где существа давно не были) для поиска противника. Это может позволить повысить рейтинг в конкурсе.

Источники

1. *Туккель Н.И., Шалыто А.А.* Система управления танком для игры «Robocode». Объектно-ориентированное программирование с явным выделением состояний. Вариант 1. Проектная документация. СПбГУ ИТМО. 2001. http://is.ifmo.ru/projects/tanks/
2. *Кузнецов Д.В., Шалыто А.А.* Система управления танком для игры «Robocode». Объектно-ориентированное программирование с явным выделением состояний. Вариант 2. Проектная документация. СПбГУ ИТМО. 2003. http://is.ifmo.ru/projects/robocode2/
3. *Марков С.М., Шалыто А.А.* Система управления травоядным существом для игры «Terrarium». Проектная документация. СПбГУ ИТМО. 2003. http://is.ifmo.ru/projects/terrarium/
4. *Sun Microsystems*. Конкурс алгоритмов «Электрические Джунгли». http://www.electricjungle.ru
5. *Паньгин А.А.* Электроджунгли. Новый конкурс по программированию на Java // Компьютерные инструменты в образовании. 2006. № 2, с.85–87. http://is.ifmo.ru/elejungle/_CIE_EJ.pdf
6. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/1
7. *Garg V.K.* Concurrent and Distributed Computing in Java. 2004. Wiley-IEEE Press.

Приложение 1. Файл *SuperPiglet.java*

```
package piglets;

import universum.bi.*;

/**
 * Created by IntelliJ IDEA.
 * User: Tanya
 * Date: 02.01.2007
 * Time: 21:06:16
 */
public abstract class SuperPiglet implements Being, Comparable<SuperPiglet> {
    protected float     M;          // mass
    protected float     S;          // speed
    public Integer   ID;         // piglet's id

    public float getM() {
        return M;
    }

    protected DrawWell  ownedWell;

    protected abstract SuperHerder getHerder();

    public String getName() {
        return "Piglet";
    }

    public String getOwnerName() {
        return "cimbaluk";
    }

    private static final float [] search_dx = new float[] { 0, 1, 0, 1, 0, -
1, -1 };
    private static final float [] search_dy = new float[] { 0, 0, 1, 1, -1,
0, -1 };

    public static float distance(Location from, Location to) {
        assert search_dx.length == search_dy.length;

        float x0 = from.getX();
        float y0 = from.getY();
        float x1 = to.getX();
        float y1 = to.getY();

        float minDistance = SuperHerder.INF;

        for (int i=0; i<search_dx.length; ++i) {
            float x = x1 + Constants.getWidth()  * search_dx[i];
            float y = y1 + Constants.getHeight() * search_dy[i];

            float dist = Math.max(Math.abs(x - x0), Math.abs(y - y0));
            minDistance = Math.min(minDistance, dist);
        }

        return minDistance;
    }

    public Location findStep(Location from, Location to) {
        float dist = distance(from, to);
```

```
        if (dist < SuperHerder.EPS) {
            return null;
        }

        if (dist < S + SuperHerder.EPS) {
            return to;
        }

        float x0 = from.getX();
        float y0 = from.getY();
        float x1 = to.getX();
        float y1 = to.getY();

        int minIndex = -1;
        float minDistance = SuperHerder.INF;

        for (int i=0; i<search_dx.length; ++i) {
            float x = x1 + Constants.getWidth()  * search_dx[i];
            float y = y1 + Constants.getHeight() * search_dy[i];

            dist = Math.max(Math.abs(x - x0), Math.abs(y - y0));
            if (dist < minDistance) {
                minDistance = dist;
                minIndex = i;
            }
        }

        assert minIndex != -1;
        x1 += Constants.getWidth()  * search_dx[minIndex];
        y1 += Constants.getHeight() * search_dy[minIndex];

        float x_ar = x1 - x0;
        float y_ar = y1 - y0;

        if (Math.abs(x_ar) > S) {
            x_ar = S * Math.signum(x_ar);
        }
        if (Math.abs(y_ar) > S) {
            y_ar = S * Math.signum(y_ar);
        }

        return new Location(from.getX() + (int)x_ar, from.getY() +
(int)y_ar);
    }

    public void processEvent(BeingInterface bi, Event event) {
        switch (event.kind()) {
            case BEING_BORN:
                BeingParams bp = (BeingParams)event.param();
                M = bp.M;
                S = bp.S;
                ID = bi.getId(this);
                getHerder().register(this);
                break;
            case BEING_DEAD:
                getHerder().unregister(this);
                break;
        }
    }

    public static int normalizeY(int y) {
        return (y + Constants.getHeight()) % Constants.getHeight();
    }
```

```
    public static int normalizeX(int x) {
        return (x + Constants.getWidth()) % Constants.getWidth();
    }

    public int compareTo(SuperPiglet o) {
        return ID.compareTo(o.ID);
    }

    public void reinit(UserGameInfo ugi) {
        getHerder().init(ugi);
//                  getHerder().MAX_TURNS   =  Math.max(getHerder().MAX_TURNS,
ugi.maxTurns);
    }
}
```

Приложение 2. Файл *SuperHerder.java*

```
package piglets;

import universum.bi.*;
import java.util.TreeSet;
import java.util.concurrent.locks.ReentrantReadWriteLock;

public class GladeCell {
    public boolean  explored;
    public int      shouldBeExplored;
    public float    growth;
    public float    count;
    public int      enemy;
}

public class DrawWell implements Comparable<DrawWell> {
    public Location         pos;
    public final SuperHerder herder;
    public final TreeSet<SuperPiglet> owners = new TreeSet<SuperPiglet>();

    public DrawWell(Location pos, SuperHerder herder) {
        this.pos        = pos;
        this.herder     = herder;
    }

    public GladeCell getCell() {
        return herder.gladeCellAt(pos);
    }

    public int compareTo(DrawWell o) {
        int x_cmp = Float.compare(pos.getX(), o.pos.getX());
        if (x_cmp != 0) {
            return x_cmp;
        }
        return Float.compare(pos.getY(), o.pos.getY());
    }
}

public abstract class SuperHerder {
    public static final float INF   = 1e6f;
    public static final float EPS   = 1e-5f;

    public GladeCell[][] glade;
    public final TreeSet<SuperPiglet> piglets = new TreeSet<SuperPiglet> ();
    public final TreeSet<DrawWell> drawWells = new TreeSet<DrawWell> ();
    private final TreeSet<Integer> knownIds = new TreeSet<Integer> ();
```

```
    private int turnCounter = 0;
    private int turn = 0;

    private      final      ReentrantReadWriteLock      locker      =      new
ReentrantReadWriteLock();

    public  final int MAX_TURNS = 200;

    public GameKind gameKind;

    protected void makeTurn(BeingInterface bi) {
        for (SuperPiglet p : piglets) {
            PointInfo curInfo = bi.getPointInfo(p);
            updateWorld(curInfo, p);

            for (PointInfo info : bi.getNeighbourInfo(p)) {
                updateWorld(info, p);
            }
        }
    }

    public void init(UserGameInfo info) {
        this.turn = 0;
        this.turnCounter = 0;
        this.piglets.clear();
        this.knownIds.clear();
        this.drawWells.clear();

        this.gameKind = info.kind;

//        this.MAX_TURNS = 0;

        glade = new GladeCell[Constants.getWidth()][Constants.getHeight()];

        for (int i=0; i< glade.length; ++i) {
            for (int j=0; j< glade[i].length; ++j) {
                glade[i][j] = new GladeCell();
            }
        }
    }

    public final synchronized void updateTurn(BeingInterface bi) {
        if (turnCounter++ == 0) {
            ++turn;
            makeTurn(bi);
        }
        if (turnCounter == piglets.size()) {
            turnCounter = 0;
        }
    }

    public final synchronized int getTurn() {
        return turn;
    }

    public final synchronized void register(SuperPiglet p) {
        piglets.add(p);
        knownIds.add(p.ID);
    }

    public final synchronized void unregister(SuperPiglet p) {
        if (p.ownedWell != null) {
            p.ownedWell.owners.remove(p);
```

```
            --turnCounter;
        }
        piglets.remove(p);
    }

    public final void updateWorld(PointInfo info, SuperPiglet p) {
        lockWrite();

        Location pos    = info.getLocation();
        GladeCell cell  = glade[pos.getX()][pos.getY()];

        cell.explored   = true;
        cell.growth     = info.getGrowthRate(p);
        cell.count      = info.getCount(p);

        if (cell.growth > 0.f) {
            DrawWell drawWell = new DrawWell(pos, this);
            if (!drawWells.contains(drawWell)) {
                drawWells.add(drawWell);
            }
        }

        // check enemies
        for (Integer id : info.getEntities(p, null)) {
            if (!knownIds.contains(id)) {
                cell.enemy = getTurn();
                break;
            }
        }

        unlockWrite();
    }

    public synchronized GladeCell gladeCellAt(Location pos) {
        return glade[pos.getX()][pos.getY()];
    }

    public void lockRead() {
        locker.readLock().lock();
    }

    public void unlockRead() {
        locker.readLock().unlock();
    }

    public void lockWrite() {
        locker.writeLock().lock();
    }

    public void unlockWrite() {
        locker.writeLock().unlock();
    }
}
```

Приложение 3. Файл *Piglet.java*

```
package piglets.delta;

import piglets.SuperPiglet;
import piglets.SuperHerder;
import piglets.DrawWell;
import piglets.GladeCell;
```

```
import universum.bi.*;
import universum.util.Util;

/**
 * Created by IntelliJ IDEA.
 * User: Tanya
 * Date: 03.02.2007
 * Time: 12:20:57
 */

enum PigletState {
    EXPLORE,
    EATING,
    FEEDING,
    SURVIVE,
    ATTACK,
}

class EventException extends Exception {
    public final Event event;

    EventException(EventKind kind, Object arg) {
        this.event = new Event(kind, arg);
    }

    public EventException() {
        this.event = null;
    }
}

public class Piglet extends SuperPiglet {
    public static final float M0                    = 5;
    public static final float S0                    = 3;

    public static final float C_crit                  = Constants.K_emin + 5 *
Constants.K_masscost;
    public static final float C_norm                          = C_crit + 5 *
Constants.K_masscost;
    public static final float C_growt_weight        = 5.f;
    public static final float C_energy_weight       = 0.05f;
    public static final float C_steps_to_born                 =  0.3f  /
Constants.K_bite;
    public static final float C_max_steps_to_dw     = 2.f;

    public static final int C_growth_barrier        = 30;
    public static final int C_fight_barrier         = 1000;
    public static final int C_enemy_history         = 1;

    private PigletState state = PigletState.EXPLORE;

    private float E;
    private Location location;
    private PointInfo pointInfo;
    private BeingInterface beingInterface;

    public Event makeTurn(BeingInterface bi) {
        herder.updateTurn(bi);

        // update piglet params
        beingInterface = bi;
        E = bi.getEnergy(this);
        location = bi.getLocation(this);
        pointInfo = bi.getPointInfo(this);
```

```
// exec automaton
final int MAX_STEPS = 100;
for (int step_ = 0; step_ < MAX_STEPS; ++step_) {
    PigletState prevState = state;

    try {
        switch (state) {
            case EXPLORE:
                if (x1_1()) {
                    log("EXPLORE: x1_1 -> ATTACK");
                    state = PigletState.ATTACK;
                    break;
                }

                if (x1_2()) {
                    log("EXPLORE: x1_2 : z1");
                    z1();
                    break;
                }

                if (x1_3()) {
                    log("EXPLORE: x1_3 -> EATING");
                    state = PigletState.EATING;
                    break;
                }

                if (x1_4()) {
                    log("EXPLORE: x1_4 -> FEEDING");
                    state = PigletState.FEEDING;
                    break;
                }

                if (x1_5()) {
                    log("EXPLORE: x1_5 -> SURVIVE");
                    state = PigletState.SURVIVE;
                    break;
                }

                if (x1_6() || x1_7()) {
                    log("EXPLORE: x1_6 | x1_7 : z2");
                    z2();
                    break;
                }

                log("EXPLORE: otherwise -> SURVIVE");
                state = PigletState.SURVIVE;
                break;

            case EATING:
                if (x2_1()) {
                    log("EATING: x2_1 : z2");
                    z2();
                    break;
                }

                if (x2_2()) {
                    log("EATING: x2_2 : z1");
                    z1();
                    break;
                }

                if (x2_3()) {
                    log("EATING: x2_3 -> SURVIVE");
                    state = PigletState.SURVIVE;
```

```
                break;
            }

            log("EATING: otherwise : z3");
            z3();
            break;

        case FEEDING:
            if (x2_1()) {
                log("FEEDING: x2_1 : z2");
                z2();
                break;
            }

            if (x2_2()) {
                log("FEEDING: x2_2 : z1");
                z1();
                break;
            }

            if (x3_1()) {
                log("FEEDING: x3_1 -> EXPLORE");
                state = PigletState.EXPLORE;
                break;
            }

            log("FEEDING: otherwise : z3");
            z3();
            break;

        case SURVIVE:
            if (x4_1()) {
                log("SURVIVE: x4_1 -> EXPLORE");
                state = PigletState.EXPLORE;
                break;
            }

            if (x4_2()) {
                log("SURVIVE: x4_2 : z4");
                z4();
                break;
            }

            log("SURVIVE: otherwise : z5");
            z5();
            break;

        case ATTACK:
            if (x5_2()) {
                log("ATTACK: x5_2 : z7");
                z7();
                break;
            }

            if (x5_1()) {
                log("ATTACK: x5_1 : z6");
                z6();
                break;
            }

            log("ATTACK: otherwise -> EXPLORE");
            state = PigletState.EXPLORE;
            break;
    }
```

```
            } catch (EventException event) {
                return event.event;
            }

            // break cycle if no action occurs
            if (state == prevState) {
                break;
            }
        }

        return null;
    }

    private boolean x1_1() {
       Location enemyPos = findClosestEnemy();
       float enemyDist = enemyPos != null ? distance(location, enemyPos) :
Herder.INF;
       return enemyDist < S + Herder.EPS;
    }

    private boolean x1_2() {
        return herder.hasEatingDrawWells() &&
            E > M * Constants.K_toborn &&
            (herder.MAX_TURNS - herder.getTurn()) * M * Constants.K_masscost
>
            E * (0.5f - Constants.K_borncost - C_crit);
    }

    private boolean x1_3() {
        DrawWell drawWell = findClosestDrawWell();

        if (drawWell == null) {
            return false;
        }

        eatingDrawWell = drawWell;
        foundDestination = drawWell.pos;

        herder.lockWrite();
        drawWell.owners.add(this);
        herder.addEatingDrawWells(true);
        herder.unlockWrite();
        return true;
    }

    private boolean x1_4() {
        Location next = findClosestFeeding();
        if (next != null) {
            foundDestination = next;
            return true;
        }

        return false;
    }

    private boolean x1_5() {
        return E - M * C_crit < Constants.K_masscost * M * (herder.MAX_TURNS
- herder.getTurn());
    }

    private Location foundDestination;
    private boolean x1_6() {
        foundDestination = findExploreStep();
        return foundDestination != null;
```

```
    }

    private boolean x1_7() {
        foundDestination = findClosestUnexploredCell();
        return          foundDestination          !=          null          &&
checkDistanceConstraints(foundDestination);
    }

    private boolean x2_1() {
        return distance(location, foundDestination) > Herder.EPS;
    }

    private boolean x2_2() {
        return E > M * Constants.K_toborn;
    }

    private boolean x2_3() {
        final    float    canEat    =    eatingDrawWell.getCell().count    /
eatingDrawWell.owners.size();
        if  (eatingDrawWell.owners.size()  >  1  &&  canEat  <  (M  -  E)  /
C_steps_to_born) {
            herder.lockWrite();
            eatingDrawWell.owners.remove(this);
            herder.unlockWrite();

            herder.addEatingDrawWells(false);
            eatingDrawWell = null;

            return true;
        }
        return false;
    }

    private boolean x3_1() {
        final float count = pointInfo.getCount(this);
        final float growth = pointInfo.getGrowthRate(this);
        return count < Herder.EPS || isGrowingPeriod() && (count + growth *
C_steps_to_born < M * Constants.K_toborn - E);
    }

    private boolean tryNotToSurvive = false;
    private boolean x4_1() {
        return tryNotToSurvive = !tryNotToSurvive;
    }

    private boolean x4_2() {
        return E > M * C_crit;
    }

    private boolean x5_1() {
        foundDestination = findClosestEnemy();
        float   dist   =   foundDestination   !=   null   ?   distance(location,
foundDestination) : Herder.INF;

        return foundDestination != null && dist < S + Herder.EPS && dist >
Herder.EPS;
    }

    private boolean x5_2() {
        foundDestination = findClosestEnemy();
        float   dist   =   foundDestination   !=   null   ?   distance(location,
foundDestination) : Herder.INF;
```

```
        return foundDestination != null && dist < S + Herder.EPS && dist <
Herder.EPS;
    }

    private void z1() throws EventException {
        throw new EventException(EventKind.ACTION_BORN, getParams());
    }

    private void z2() throws EventException {
        throw new EventException(EventKind.ACTION_MOVE_TO, findStep(location,
foundDestination));
    }

    private DrawWell eatingDrawWell;
    private void z3() throws EventException {
        if (eatingDrawWell == null) {
            throw new EventException(EventKind.ACTION_EAT, M - E);
        }

        final    float    canEat    =    eatingDrawWell.getCell().count    /
eatingDrawWell.owners.size();
        throw new EventException(EventKind.ACTION_EAT, Math.min(canEat, M -
E));
    }

    private void z4() throws EventException {
        throw new EventException();
    }

    private void z5() throws EventException {
        final float neededToEat = M * C_norm - E;
        throw new EventException(EventKind.ACTION_EAT, neededToEat);
    }

    private void z6() throws EventException {
        throw         new         EventException(EventKind.ACTION_MOVE_ATTACK,
foundDestination);
    }

    private void z7() throws EventException {
        throw new EventException(EventKind.ACTION_ATTACK, foundDestination);
    }

    private boolean isGrowingPeriod() {
        return herder.piglets.size() < C_growth_barrier;
    }

    private DrawWell findClosestDrawWell() {
        herder.lockRead();

        DrawWell found = null;
        float minWeight = Herder.INF;

        for (DrawWell dw : herder.drawWells) {
            GladeCell cell = dw.getCell();

            // first draw well should be good to born
            if    ((cell.count    +    C_steps_to_born    *    cell.growth)    /
(dw.owners.size() + 1) < (M - E) / C_steps_to_born) {
                continue;
            }

            // calc draw well weight
            final float dist = distance(location, dw.pos);
```

```
            if (dist > C_max_steps_to_dw * S0) {
                continue;
            }

            float weight = dist - (cell.count + cell.growth * C_growt_weight)
* C_energy_weight / dw.owners.size();
            if (weight < minWeight) {
                minWeight = weight;
                found = dw;
            }
        }

        herder.unlockRead();

        return found;
    }

    private boolean checkDistanceConstraints(Location next) {
        // distance from current position
        final float dist = distance(location, next);

        final float turnCount = (float)Math.ceil(dist / S);

        // energy at next (consider to move cost and mass-live cost)
        final float E_at_next = E - dist * Constants.K_movecost
                - turnCount * Constants.K_masscost * M;

        return E_at_next > (herder.MAX_TURNS - herder.getTurn() - turnCount)
* M * Constants.K_masscost + M * C_crit;
    }

    private Location findClosestUnexploredCell() {
        if (herder.everythingExplored) {
            return null;
        }

        herder.lockRead();

        Location found = null;
        float minDist = Herder.INF;

        boolean hasUnexploredColumnt = false;
        for (int i=0; i<herder.glade.length; ++i) {
            if (herder.exploredColumn[i]) {
                continue;
            }
            hasUnexploredColumnt = true;

            boolean hasUnexplored = false;
            for (int j=0; j<herder.glade[i].length; ++j) {
                if (!herder.glade[i][j].explored) {
                    hasUnexplored = true;
                }

                // check if cell hasn't explored
                if               (!herder.glade[i][j].explored               &&
herder.glade[i][j].shouldBeExplored != herder.getTurn()) {
                    Location curPos = new Location(i, j);
                    float dist = distance(location, curPos);
                    if (dist < minDist) {
                        found = curPos;
                        minDist = dist;
                    }
```

```
                }
            }

            if (!hasUnexplored) {
                herder.unlockRead();
                herder.lockWrite();
                herder.exploredColumn[i] = true;
                herder.unlockWrite();
                herder.lockRead();
            }
        }

        herder.unlockRead();

        // mark glade as "should ne explored"
        if (found != null) {
            herder.lockWrite();
            herder.glade[found.getX()][found.getY()].shouldBeExplored        =
herder.getTurn();
            herder.unlockWrite();
        }

        if (!hasUnexploredColumnt) {
            herder.lockWrite();
            herder.everythingExplored = true;
            herder.unlockWrite();
        }

        return found;
    }

    private static final int [] EXPLORE_DELTA = new int [] { 0, 1, -1, 2, -2,
3, -3 };
    private Location findExploreStep() {
        herder.lockRead();

        final int VARIATION_LENGTH = 3;

        int max = 0;
        Location best = null;

        for (int ED_i : EXPLORE_DELTA) {
            for (int ED_j : EXPLORE_DELTA) {
                // let's calc cells amount that will be explored
                int unexplored = 0;

                for (int di = 0; di < VARIATION_LENGTH; ++di) {
                    for (int dj = 0; dj < VARIATION_LENGTH; ++dj) {
                        int   x   =   normalizeX(location.getX()   +   ED_i   +
EXPLORE_DELTA[di]);
                        int   y   =   normalizeY(location.getY()   +   ED_j   +
EXPLORE_DELTA[dj]);

                        if           (!herder.glade[x][y].explored           &&
herder.glade[x][y].shouldBeExplored != herder.getTurn()) {
                            ++unexplored;
                        }
                    }
                }

                if (max < unexplored || max == unexplored && Util.frnd() <
0.2f) {
                    max = unexplored;
                    best = new Location(
```

```
                            normalizeX(location.getX() + ED_i),
                            normalizeY(location.getY() + ED_j));
                    }
                }
            }

            herder.unlockRead();

            if (best != null) {
                herder.lockWrite();

                // mark cells that it will be explored
                for (int di = 0; di < VARIATION_LENGTH; ++di) {
                    for (int dj = 0; dj < VARIATION_LENGTH; ++dj) {
                        int x = normalizeX(best.getX() + EXPLORE_DELTA[di]);
                        int y = normalizeY(best.getY() + EXPLORE_DELTA[dj]);

                        herder.glade[x][y].shouldBeExplored = herder.getTurn();
                    }
                }

                herder.unlockWrite();
            }

            return best;
        }

        private Location findClosestEnemy() {
            if (herder.totalMass < C_fight_barrier) {
                return null;
            }

            herder.lockRead();

            float minDist = Herder.INF;
            Location found = null;

            for (int ED_i : EXPLORE_DELTA) {
                for (int ED_j : EXPLORE_DELTA) {
                    final int x = normalizeX(location.getX() + ED_i);
                    final int y = normalizeY(location.getY() + ED_j);

                    final Location pos = new Location(x, y);
                    final float dist = distance(location, pos);
                    if    (herder.glade[x][y].enemy    >=    herder.getTurn()    -
C_enemy_history && dist < minDist) {
                        minDist = dist;
                        found = pos;
                    }
                }
            }

            herder.unlockRead();
            return found;
        }

        private static final int [] FEEDING_DELTA = new int [] { 0, 1, -1, 2, -2,
3, -3, 4, -4, 5, -5, 6, -6 };
        private Location findClosestFeeding() {
            herder.lockRead();

            Location best = null;
            float maxCount = Herder.EPS;
```

```
        for (int FD_i : FEEDING_DELTA) {
            for (int FD_j : FEEDING_DELTA) {
                int x = normalizeX(location.getX() + FD_i);
                int y = normalizeY(location.getY() + FD_j);

                GladeCell cell = herder.glade[x][y];

                if (cell.count > maxCount && cell.growth < Herder.EPS) {
                    maxCount = cell.count;
                    best = new Location(x, y);
                }
            }
        }

        herder.unlockRead();

        return best;
    }

    private boolean isCreated = false;
    public BeingParams getParams() {
        if (!isCreated || isGrowingPeriod()) {
            return new BeingParams(M0, S0);
        }
        return new BeingParams(M * Constants.K_maxbornvariation, S0);
    }

    public void reinit(UserGameInfo userGameInfo) {
        super.reinit(userGameInfo);
    }

    private static final Herder herder = new Herder();
    protected SuperHerder getHerder() {
        return herder;
    }

    public void processEvent(BeingInterface bi, Event event) {
        super.processEvent(bi, event);

        switch (event.kind()) {
            case BEING_BORN:
                isCreated = true;
                break;
        }
    }

    public void log(String s) {
        if (beingInterface != null) {
            beingInterface.log(this, "(" + E + "/" + M + ") " + s);
        }
    }
}
```

Приложение 4. Файл *Herder.java*

```
package piglets.delta;

import piglets.SuperHerder;
import piglets.SuperPiglet;
import universum.bi.BeingInterface;
import universum.bi.UserGameInfo;
import universum.bi.Constants;
```

```
/**
 * Created by IntelliJ IDEA.
 * User: Tanya
 * Date: 03.02.2007
 * Time: 12:19:55
 */
public class Herder extends SuperHerder {
    public boolean [] exploredColumn;
    public boolean everythingExplored;

    public float totalMass;
    protected void makeTurn(BeingInterface bi) {
        super.makeTurn(bi);

        totalMass = 0;
        for (SuperPiglet p : piglets) {
            totalMass += bi.getMass(p, p.ID);
        }
    }

    public void init(UserGameInfo ugi) {
        super.init(ugi);

        eatingDrawWells = 0;
        exploredColumn = new boolean[Constants.getWidth()];
        everythingExplored = false;
    }

    private int eatingDrawWells = 0;
    public synchronized void addEatingDrawWells(boolean add) {
        eatingDrawWells += add ? 1 : -1;
    }

    public synchronized boolean hasEatingDrawWells() {
        return eatingDrawWells > 0;
    }
}
```

Приложение 5. Лог работы программы

```
p[1]: (5.0/5.0) EXPLORE: x1_3 -> EATING
p[1]: (5.0/5.0) EATING: x2_1 : z2
p[1]: (4.967764/5.0) EATING: x2_2 : z1
p[1]: (1.4816458/5.0) EATING: otherwise : z3
p[2]: (2.483882/5.0) EXPLORE: x1_3 -> EATING
p[2]: (2.483882/5.0) EATING: otherwise : z3
p[1]: (1.9794098/5.0) EATING: otherwise : z3
p[2]: (2.9816458/5.0) EATING: otherwise : z3
p[1]: (2.4771736/5.0) EATING: otherwise : z3
p[2]: (3.4794097/5.0) EATING: otherwise : z3
p[1]: (2.9749374/5.0) EATING: otherwise : z3
p[2]: (3.9771736/5.0) EATING: otherwise : z3
p[1]: (3.4727013/5.0) EATING: otherwise : z3
p[2]: (4.474938/5.0) EATING: x2_2 : z1
p[1]: (3.9704652/5.0) EATING: otherwise : z3
p[2]: (1.2352328/5.0) EATING: x2_3 -> SURVIVE
p[2]: (1.2352328/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.2352328/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2352328/5.0) SURVIVE: x4_2 : z4
p[3]: (2.237469/5.0) EXPLORE: x1_6 | x1_7 : z2
p[1]: (4.4682293/5.0) EATING: x2_2 : z1
```

```
p[2]: (1.2329968/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.2329968/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2329968/5.0) SURVIVE: x4_2 : z4
p[3]: (2.2052329/5.0) EXPLORE: x1_6 | x1_7 : z2
p[1]: (1.2318785/5.0) EATING: otherwise : z3
p[2]: (1.2307608/5.0) SURVIVE: x4_1 -> EXPLORE
p[3]: (2.1729968/5.0) EXPLORE: x1_6 | x1_7 : z2
p[4]: (2.2341146/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.2307608/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2307608/5.0) SURVIVE: x4_2 : z4
p[1]: (1.7296425/5.0) EATING: otherwise : z3
p[4]: (2.2018785/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.2285248/5.0) SURVIVE: x4_1 -> EXPLORE
p[3]: (2.1407607/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.2285248/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2285248/5.0) SURVIVE: x4_2 : z4
p[1]: (2.2274063/5.0) EATING: otherwise : z3
p[2]: (1.2262888/5.0) SURVIVE: x4_1 -> EXPLORE
p[4]: (2.1696424/5.0) EXPLORE: x1_6 | x1_7 : z2
p[3]: (2.1085246/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.2262888/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2262888/5.0) SURVIVE: x4_2 : z4
p[1]: (2.7251701/5.0) EATING: otherwise : z3
p[4]: (2.1374063/5.0) EXPLORE: x1_6 | x1_7 : z2
p[3]: (2.0762885/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.2240528/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.2240528/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2240528/5.0) SURVIVE: x4_2 : z4
p[1]: (2.965317/5.0) EATING: otherwise : z3
p[4]: (2.1051702/5.0) EXPLORE: x1_3 -> EATING
p[4]: (2.1051702/5.0) EATING: x2_1 : z2
p[3]: (2.0440524/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.2218168/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.2218168/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2218168/5.0) SURVIVE: x4_2 : z4
p[1]: (3.031945/5.0) EATING: otherwise : z3
p[4]: (2.0729342/5.0) EATING: otherwise : z3
p[2]: (1.2195808/5.0) SURVIVE: x4_1 -> EXPLORE
p[3]: (2.0118163/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.2195808/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2195808/5.0) SURVIVE: x4_2 : z4
p[1]: (3.098573/5.0) EATING: otherwise : z3
p[4]: (2.570698/5.0) EATING: otherwise : z3
p[3]: (1.9795802/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.2173448/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.2173448/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2173448/5.0) SURVIVE: x4_2 : z4
p[1]: (3.165201/5.0) EATING: otherwise : z3
p[2]: (1.2151088/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.2151088/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2151088/5.0) SURVIVE: x4_2 : z4
p[3]: (1.9473441/5.0) EXPLORE: x1_3 -> EATING
p[3]: (1.9473441/5.0) EATING: x2_1 : z2
p[4]: (3.068462/5.0) EATING: otherwise : z3
p[1]: (3.231829/5.0) EATING: otherwise : z3
p[4]: (3.5662258/5.0) EATING: otherwise : z3
p[3]: (1.915108/5.0) EATING: otherwise : z3
p[2]: (1.2128727/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.2128727/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2128727/5.0) SURVIVE: x4_2 : z4
p[1]: (3.298457/5.0) EATING: otherwise : z3
p[4]: (4.06399/5.0) EATING: x2_2 : z1
p[3]: (2.4128718/5.0) EATING: otherwise : z3
p[2]: (1.2106367/5.0) SURVIVE: x4_1 -> EXPLORE
```

```
p[2]: (1.2106367/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2106367/5.0) SURVIVE: x4_2 : z4
p[1]: (3.365085/5.0) EATING: otherwise : z3
p[4]: (1.0297589/5.0) EATING: otherwise : z3
p[5]: (2.031995/5.0) EXPLORE: x1_3 -> EATING
p[5]: (2.031995/5.0) EATING: otherwise : z3
p[3]: (2.9106357/5.0) EATING: otherwise : z3
p[2]: (1.2084007/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.2084007/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2084007/5.0) SURVIVE: x4_2 : z4
p[1]: (3.4317129/5.0) EATING: otherwise : z3
p[4]: (1.5275229/5.0) EATING: otherwise : z3
p[2]: (1.2061647/5.0) SURVIVE: x4_1 -> EXPLORE
p[3]: (3.4083996/5.0) EATING: otherwise : z3
p[5]: (2.529759/5.0) EATING: otherwise : z3
p[2]: (1.2061647/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2061647/5.0) SURVIVE: x4_2 : z4
p[1]: (3.4983408/5.0) EATING: otherwise : z3
p[2]: (1.2039287/5.0) SURVIVE: x4_1 -> EXPLORE
p[4]: (2.025287/5.0) EATING: otherwise : z3
p[2]: (1.2039287/5.0) EXPLORE: x1_5 -> SURVIVE
p[3]: (3.9061635/5.0) EATING: otherwise : z3
p[2]: (1.2039287/5.0) SURVIVE: x4_2 : z4
p[5]: (3.0275228/5.0) EATING: otherwise : z3
p[1]: (3.5649688/5.0) EATING: otherwise : z3
p[4]: (2.5230508/5.0) EATING: otherwise : z3
p[5]: (3.5252867/5.0) EATING: otherwise : z3
p[3]: (4.4039273/5.0) EATING: x2_2 : z1
p[2]: (1.2016927/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.2016927/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.2016927/5.0) SURVIVE: x4_2 : z4
p[1]: (3.6315968/5.0) EATING: otherwise : z3
p[5]: (4.023051/5.0) EATING: x2_2 : z1
p[6]: (2.2019637/5.0) EXPLORE: x1_6 | x1_7 : z2
p[4]: (3.0208147/5.0) EATING: otherwise : z3
p[2]: (1.1994567/5.0) SURVIVE: x4_1 -> EXPLORE
p[3]: (1.1997275/5.0) EATING: otherwise : z3
p[2]: (1.1994567/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.1994567/5.0) SURVIVE: x4_2 : z4
p[1]: (3.6982248/5.0) EATING: otherwise : z3
p[5]: (1.0092893/5.0) EATING: x2_3 -> SURVIVE
p[5]: (1.0092893/5.0) SURVIVE: x4_1 -> EXPLORE
p[3]: (1.6974915/5.0) EATING: otherwise : z3
p[4]: (3.5185785/5.0) EATING: otherwise : z3
p[6]: (2.1697276/5.0) EXPLORE: x1_6 | x1_7 : z2
p[5]: (1.0092893/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.1972207/5.0) SURVIVE: x4_1 -> EXPLORE
p[5]: (1.0092893/5.0) SURVIVE: x4_2 : z4
p[7]: (2.0115254/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.1972207/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.1972207/5.0) SURVIVE: x4_2 : z4
p[1]: (3.7648528/5.0) EATING: otherwise : z3
p[5]: (1.0070533/5.0) SURVIVE: x4_1 -> EXPLORE
p[5]: (1.0070533/5.0) EXPLORE: x1_5 -> SURVIVE
p[5]: (1.0070533/5.0) SURVIVE: x4_2 : z4
p[6]: (2.1374915/5.0) EXPLORE: x1_6 | x1_7 : z2
p[7]: (1.9792893/5.0) EXPLORE: x1_6 | x1_7 : z2
p[4]: (4.0163426/5.0) EATING: x2_2 : z1
p[3]: (2.1952555/5.0) EATING: otherwise : z3
p[2]: (1.1949847/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.1949847/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.1949847/5.0) SURVIVE: x4_2 : z4
p[1]: (3.8314807/5.0) EATING: otherwise : z3
p[4]: (1.0059352/5.0) EATING: otherwise : z3
```

```
p[5]: (1.0048172/5.0) SURVIVE: x4_1 -> EXPLORE
p[5]: (1.0048172/5.0) EXPLORE: x1_5 -> SURVIVE
p[5]: (1.0048172/5.0) SURVIVE: x4_2 : z4
p[3]: (2.6930194/5.0) EATING: otherwise : z3
p[7]: (1.9470532/5.0) EXPLORE: x1_6 | x1_7 : z2
p[6]: (2.1052554/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.1927487/5.0) SURVIVE: x4_1 -> EXPLORE
p[8]: (2.0081713/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.1927487/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.1927487/5.0) SURVIVE: x4_2 : z4
p[1]: (3.8981087/5.0) EATING: otherwise : z3
p[5]: (1.0025812/5.0) SURVIVE: x4_1 -> EXPLORE
p[5]: (1.0025812/5.0) EXPLORE: x1_5 -> SURVIVE
p[5]: (1.0025812/5.0) SURVIVE: x4_2 : z4
p[4]: (1.5036992/5.0) EATING: otherwise : z3
p[6]: (2.0730193/5.0) EXPLORE: x1_6 | x1_7 : z2
p[8]: (1.9759352/5.0) EXPLORE: x1_6 | x1_7 : z2
p[3]: (2.940658/5.0) EATING: otherwise : z3
p[7]: (1.9148171/5.0) EXPLORE: x1_6 | x1_7 : z2
p[2]: (1.1905127/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.1905127/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.1905127/5.0) SURVIVE: x4_2 : z4
p[1]: (3.9647367/5.0) EATING: otherwise : z3
p[5]: (1.0003452/5.0) SURVIVE: x4_1 -> EXPLORE
p[5]: (1.0003452/5.0) EXPLORE: x1_5 -> SURVIVE
p[5]: (1.0003452/5.0) SURVIVE: x4_2 : z4
p[6]: (2.0407832/5.0) EXPLORE: x1_6 | x1_7 : z2
p[7]: (1.882581/5.0) EXPLORE: x1_6 | x1_7 : z2
p[8]: (1.9436991/5.0) EXPLORE: x1_6 | x1_7 : z2
p[3]: (2.9932685/5.0) EATING: otherwise : z3
p[4]: (1.7971247/5.0) EATING: otherwise : z3
p[2]: (1.1882766/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.1882766/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.1882766/5.0) SURVIVE: x4_2 : z4
p[1]: (4.031365/5.0) EATING: x2_2 : z1
p[2]: (1.1860406/5.0) SURVIVE: x4_1 -> EXPLORE
p[2]: (1.1860406/5.0) EXPLORE: x1_5 -> SURVIVE
p[2]: (1.1860406/5.0) SURVIVE: x4_2 : z4
p[3]: (3.0458791/5.0) EATING: otherwise : z3
```