Санкт−Петербургский государственный университет информационных технологий, механики и оптики

Кафедра "Компьютерные технологии"

Е.В. Калугин

# Существо *Yeti* для проекта *Электрические джунгли*

Проектная документация

Проект создан в рамках

«Движения за открытую проектную документацию»

**http://is.ifmo.ru**

Санкт-Петербург
2007

# Оглавление

# Введение

В ходе магистерской практики была поставлена задача – создать существо для проекта *Электрические Джунгли* (http://www.electricjungle.ru).

*Электрические Джунгли (ЭД)* – это состязание, в котором игроки программируют поведение населяющих особый мир виртуальных существ, борющихся за ограниченный ресурс – «пищу». Целью игрока является программирование поведения своих существ таким образом, чтобы его вид как целое максимально преуспел. Разработка должна вестись на языке *Java.*

Для решения данной задачи была использована *SWITCH*-технология, так как она в полной мере подходит для данного типа задач.

# 1. Правила проекта «*Электрические Джунгли*»

## *1.1. Обозначения*

- **X% МЭ** — количество энергии в процентах от максимальной энергии (МЭ) существа;
- **`K_someconst`(5)** — постоянная, которая определяет значение одного параметра игры (все постоянные определены в исходном тексте игры). В скобках указано текущее значение. Для большинства констант эти значения во время всего конкурса сохранятся, однако возможны и переопределения.

## *1.2. Понятия и цели*

В мире *ЭД* основная цель — максимальная видовая экспансия. Победившим признается тот игрок, чей вид за отведенное количество ходов добьется максимальной массы – суммарная масса всех особей которого будет наибольшей.

### 1.2.1. Пространство и время

В *ЭД* пространство устроено как замкнутое двумерное дискретное поле из клеток, закольцованное по краям (так, что получается тор). Впрочем, в движке поддерживаются и другие топологии и размерности, но победитель будет выявлен на стандартной тороидальной топологии 140x120. Игра состоит из последовательности ходов. За один ход каждому существу дается шанс совершить свое действие. Когда все существа совершат свои действия — начинается следующий ход. В общем случае порядок выполнения действий не определен, разные существа могут совершать действия одновременно.

### 1.2.2. Масса и скорость

Каждое существо в *ЭД* обладает двумя базовыми характеристиками: массой и скоростью, а также в каждый момент времени характеризуется энергетическим уровнем. Масса совпадает с максимальной энергией (МЭ), которой может обладать существо. Если энергетический уровень падает ниже `K_emin`(15)% МЭ — существо умирает и вся его оставшаяся энергия доступна для потребления другими существами. Масса не может быть больше `K_maxmass`(1000) и меньше `K_minmass`(0.1). Скорость не может быть больше `K_maxspeed`(10) и меньше `K_minspeed`(1).

Общая масса вида (сумма масс всех живых существ данного вида) определяет количество очков игрока. Масса существа определяет его энергоемкость, энергопотребление за ход и повреждение, наносимое в битве.

Скорость определяет максимальное расстояние, на которое может передвигаться существо за один ход. Однако чем больше скорость, тем энергетически дороже передвижение существа. Независимо от скорости существо может получать информацию только о точке, где оно находится, а также о соседних точках.

### 1.2.3. Энергия

Энергия является основой существования в *ЭД*. Любое действие (даже просто выживание) требует потребления некоторого количества энергии. В начале игры на поле случайным образом размещаются источники энергии, из которых существа могут черпать энергию (не более 10% МЭ). Энергия в источниках постепенно восполняется. Начальное количество энергии и скорость ее роста случайны и различны для разных источников. На поле имеется NUM_REGULAR(130) обычных источников и NUM_GOLDEN(3) золотых источников, производящих гораздо больше энергии. Кроме этого, в точке, где рождается первое существо каждого игрока даётся BORN_BONUS(100) единиц энергии, и за каждый ход прирастает BORN_BONUS_GROWTH(1) единиц энергии. Хотя в различных играх энергия и прирост по-разному распределены на поле, суммарная энергия и ее прирост всегда одинаковы.

### 1.2.4. Информация о внешнем мире

Существо может получить информацию о внешнем мире при помощи *API* `BeingIntreface` и `PointInfo`. Доступна следуюшая информация:

- количество энергии самого существа;
- владелец и масса любого существа, находящегося достаточно близко (на той же клетке или на примыкающих клетках);
- количество энергии, скорость ее прироста и максимальная емкость в клетке;
- список всех объектов в данной точке (в частности, других существ);
- суммарная масса всех живых существ в данной точке.

## 1.3. Действия и события

За один ход существо может совершить одно действие, а также каждое существо получает оповещения о происходящих с ним событиях. Конкретный порядок выполнения действий не специфицирован. Доступны следующие действия:

- `ACTION_MOVE_TO` — перейти в другую клетку. Доступность клетки определяется скоростью существа. Стоит `K_movecost`(1)% скорости.
- `ACTION_EAT` — потребить энергию. Количество энергии, которую можно потребить за ход, но общая энергия не может превышать энергоемкости (массы) и потребленная энергия не может быть больше `K_bite`(10)% МЭ. Бесплатно.
- `ACTION_GIVE` — передать энергию другому существу. Бесплатно. Существа должны быть на одной и той же клетке.
- `ACTION_ATTACK` — Атаковать другое существо, нанося повреждение в `K_fight`(20)% МЭ. При этом теряется `K_fightcost`(1)% собственной МЭ + `K_retaliate`(5)% МЭ атакуемого существа.
- `ACTION_BORN` — породить другое существо, возможно с немного отличающимися (не более чем на `K_minbornvariation`(0.8)/`K_maxbornvariation`(1.2)) массой и скоростью (существо с массой 100 и скоростью 2 может порождать существа с массой от 80 до 120 и скоростью от 1.6 до 2.4). Энергия при этом делится пополам. Рождение возможно только тогда, когда у существа достаточно энергии – не менее `K_toborn`(80%) МЭ. Акт рождения стоит `K_borncost`(20)% МЭ. При этом существу можно передать "генокод" — произвольный *Java* объект.
- `ACTION_MOVE_ATTACK` — совмещено двигаться и атаковать, при этом наносится меньше `K_fightmovepenalty` (0.75) повреждений. Стоимость совпадает с суммарной стоимостью атаки и передвижения.

Также существо получает оповещения о следующих событиях:

- `BEING_BORN` — первое оповещение, которую существо получает после рождения. Можно инициализировать различные параметры, специфичные для данного существа.
- `BEING_DEAD` — последнее оповещение, передается после смерти существа.
- `BEING_ATTACKED` — существо атаковано кем-то, идентификатор аттакующего передается как параметр.
- `BEING_ENERGY_GIVEN` — Кто-то передал нам энергию.

## 1.4. Состязание

Состязание начинается с помещения одного существа каждого игрока в случайную точку пространства, при этом один раз вызывается метод `reinit()`, информирующий о текущих условиях игры и позволяющий заново инициализировать статические переменные. Масса и энергия изначального существа определяется самим игроком и может быть произвольной. Параметры первого существа определяются возвращаемым значением метода `Being.getParams()`. Параметры всех остальных существ определяются параметрами действия `ACTION_BORN`. После этого у каждого существа есть возможность попытаться добиться победы в заданных условиях, как описано далее.

## 1.5. Виды игр

1. **Блицкриг — `SINGLE`**

Одному виду существ дается весь мир. Цель — добиться наивысшего рейтинга путем максимально эффективного использования ресурсов и быстрой разведки за данный временной интервал (200 ходов).

2. **Дуэль — `DUEL`**

Главный вид состязания в Электрических Джунглях. Два конкурирующих вида существ сражаются на протяжении 1000 ходов.

3. **Джунгли — `JUNGLE`**

До восьмими конкурирующих видов сражаются на одном поле 2000 ходов.

## 1.6. Графическое представление игры

Поле игры приведено на рис. 1.

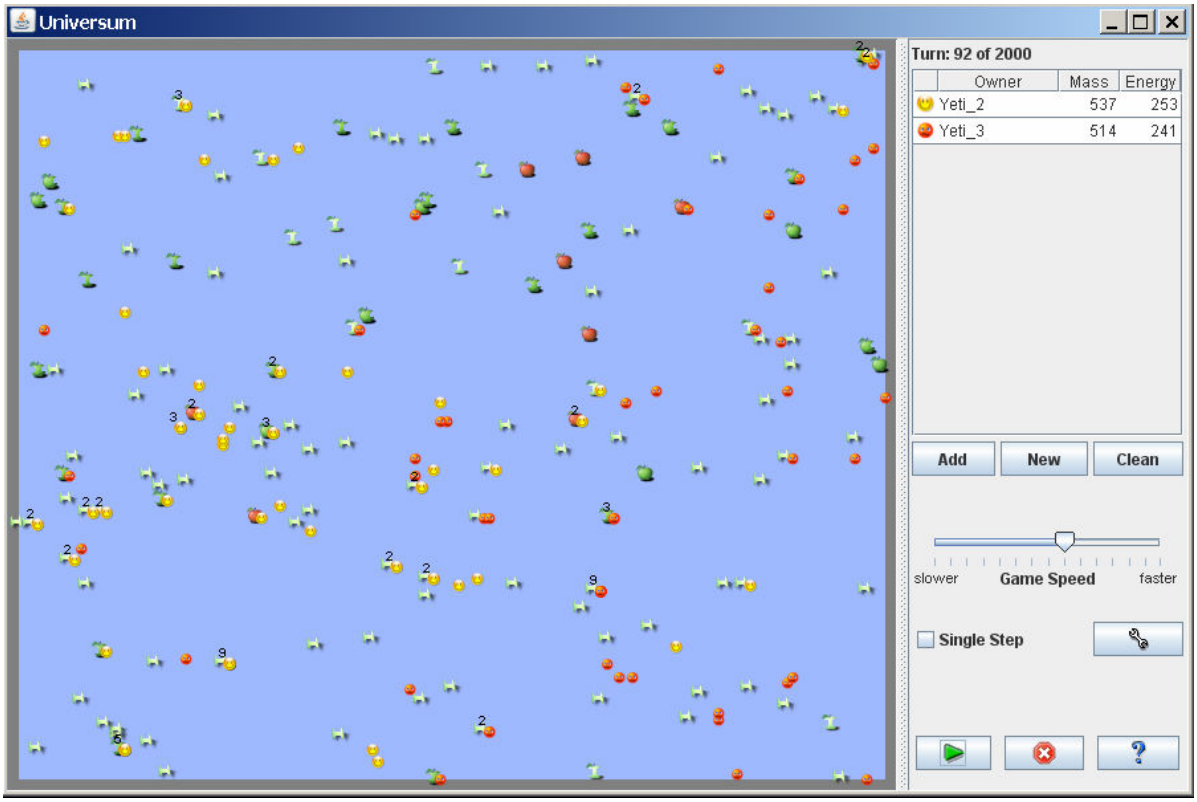


Рис. 1. Поле игры

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.

# 2. Описание существа

## *2.1. Общее описание существа*

Для проекта Электрические Джунгли было написано существо, которое было названо *Yeti*. На настоящий момент оно заточено только на один вид игры – дуэль, поэтому вся нижеописанная стратегия и тактика относится именно к этому виду игры. Существо *Yeti* имеет начальную массу 10 и скорость три.

## *2.2. Стратегия ведения игры*

Общая стратегия игры достаточно проста – вначале захватить как можно больше источников энергии, затем, набрав достаточное количество сил поэтапно отвоёвывать у противника все оставшиеся ценные источники. После этого довести игру до победы уже не составляет труда.

Для реализации этой стратегии были разработаны три вида существ, с различными линиями поведения.

\

## 2.3. Виды существ

### 2.3.1. Исследователь

Основная задача исследователя – разведка карты и особенно поиск ресурсов. Причём делать это он может двумя способами, просто двигаясь в сторону ближайшей неисследованной территории, либо следуя по контрольным точкам (waypoints). Контрольные точки задаются один раз в начале игры и охватывают всю карту, однако редкий исследователь проходит маршрут целиком – как правило, они погибают по дороге. Во многом с этим связано то, что в последней версии *Yeti* исследователь осуществляет разведку только первым способом.

Как только исследователь находит ресурс он “отъедается” на нём, порождая новых существ и лишь, после того как вся энергия будет вычерпнута из ресурса, исследователь продолжает разведку.

Также, помимо разведки исследователь атакует заведомо более слабых противников, в случае если случайно на них натыкается.

На рис. 2 показаны исследователи в поиске ресурсов (обведены синим, а стрелка указывает направление движения).

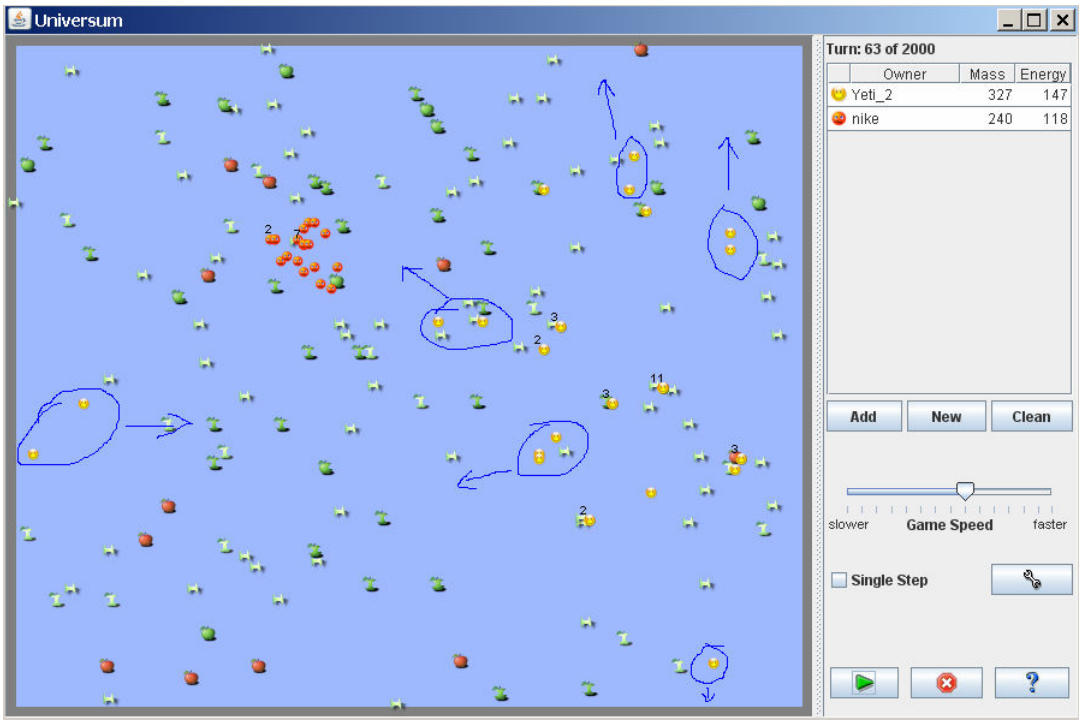


Рис. 2. Действия исследователей по поиску ресурсов

Поведение исследователя полностью задаётся конечным автоматом, представленным на рис. 3.

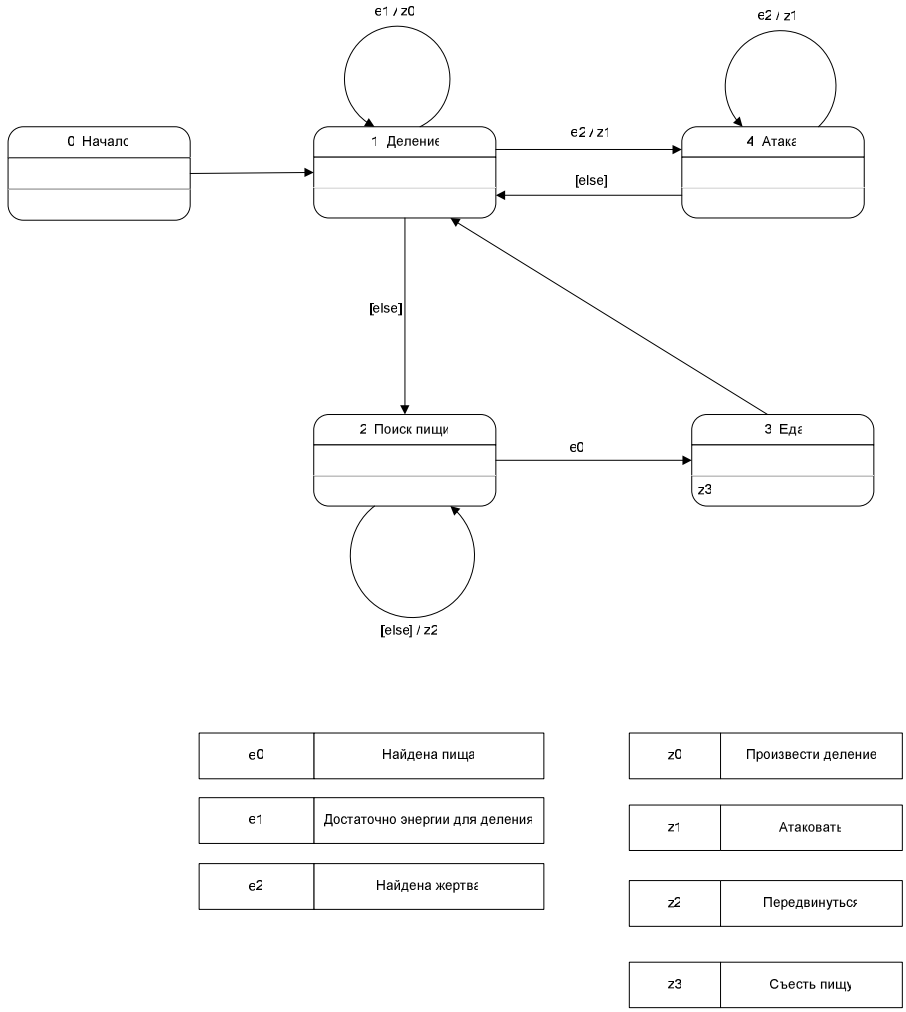


Рис. 3. Граф переходов, описывающий работу исследователя

### 2.3.2. Стражник

Задача стражника – охранять ресурсы от противника. Непременным атрибутом стражника является объект охраны – точка пространства, к которой он не должен подпускать противника. В ходе игры этой точкой, как правило, оказывается источник энергии. Количество стражников на каждом ресурсе тем больше, чем более ёмким является этот ресурс.

Как только существо *Yeti* переходит в режим стражника, оно начинает движение к объекту охраны (если, конечно, объект охраны не расположен в той точке, где существо было рождено), не отвлекаясь на еду и не обращая внимания на противников. Достигнув объекта охраны, стражник начинает охранять объект – атаковать любое

слабое вражеское существо вблизи объекта и атаковать любое, в том числе и сильное, существо на самом объекте. Если поблизости нет врагов, то существо употребляет пищу (энергию) и, если есть возможность, размножается.

На рис. 4 показаны стражники, охраняющие ресурсы (обведены синим).

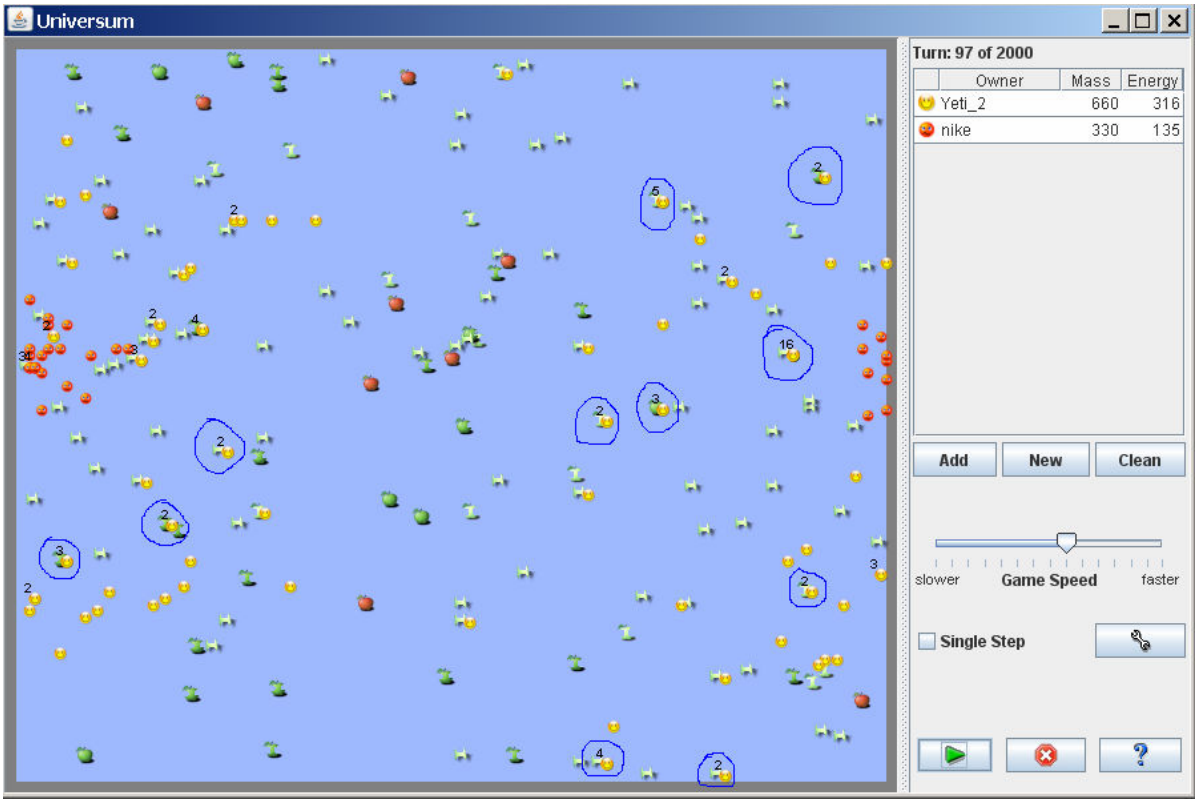


Рис. 4. Стражники, охраняющие ресурсы

Поведение стражника полностью задаётся конечным автоматом, который представлен на рис. 5.

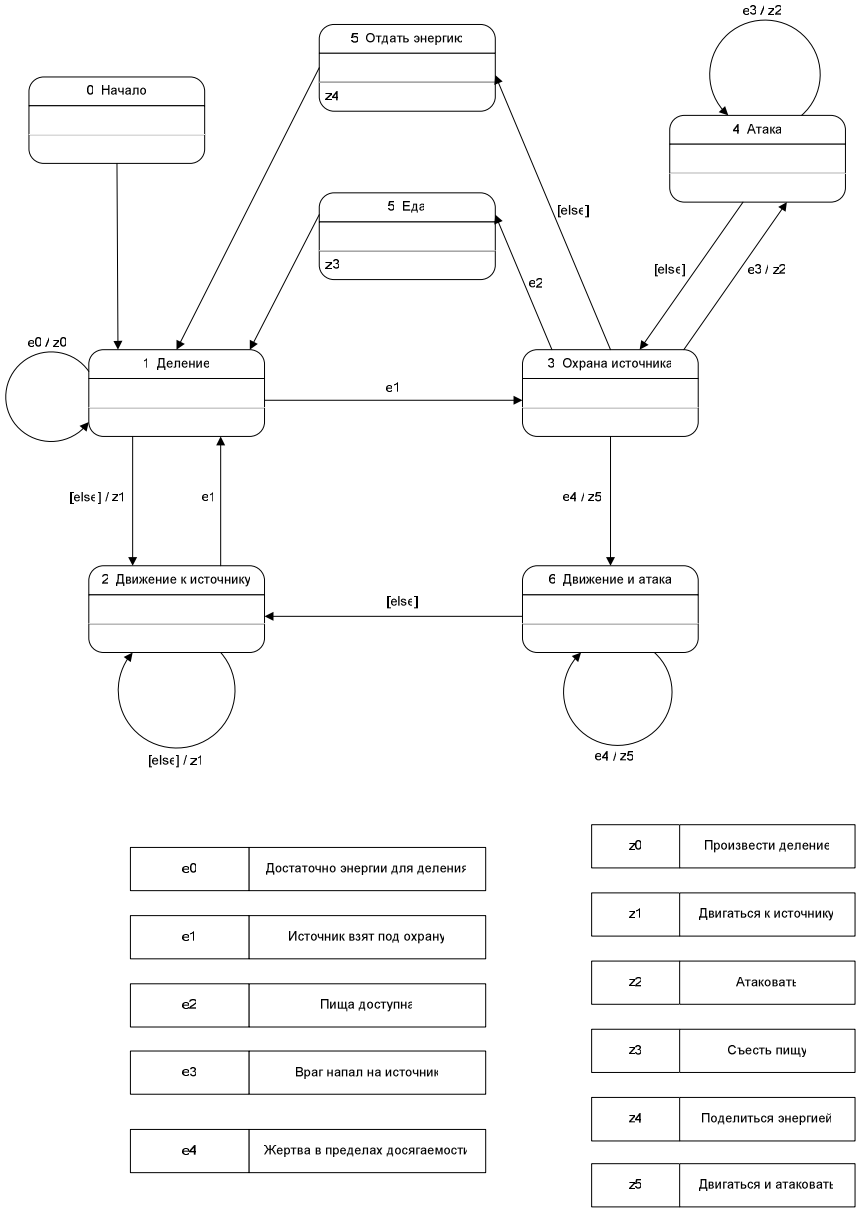


Рис. 5. Граф переходов, описывающий работу стражника

### 2.3.3. Воин

В ходе исследования было выяснено, что если стражники по одиночке, один за другим, идут охранять источник, который уже захвачен врагом, то это, как правило, не только не даёт никакого эффекта, но и является своеобразной подпиткой для оккупировавшего источник врага. В связи с этим был разработан ещё один вид существа – воин. Особенностью данного вида является то, что воины должны консолидировано

действовать на карте. Поэтому их имеет смысл рассматривать только в рамках отряда, описанного ниже.

Основная задача воинов – отбивать у врага захваченные ресурсы. Для этого отряду воинов передаются координаты оккупированного ресурса, который надо отбить и так называемая точка сбора. Точка сбора – это место на карте, желательно вне зоны видимости противника, с которого можно за один ход попасть на оккупированный ресурс. Воины собираются в точке сбора и, как только их суммарная масса достигает некоторого порогового значения, им посылается сигнал об атаке, после чего они одновременно перемещаются на оккупированный источник.

Полностью поведение воина задаётся конечным автоматом, который представлен на рис. 6.

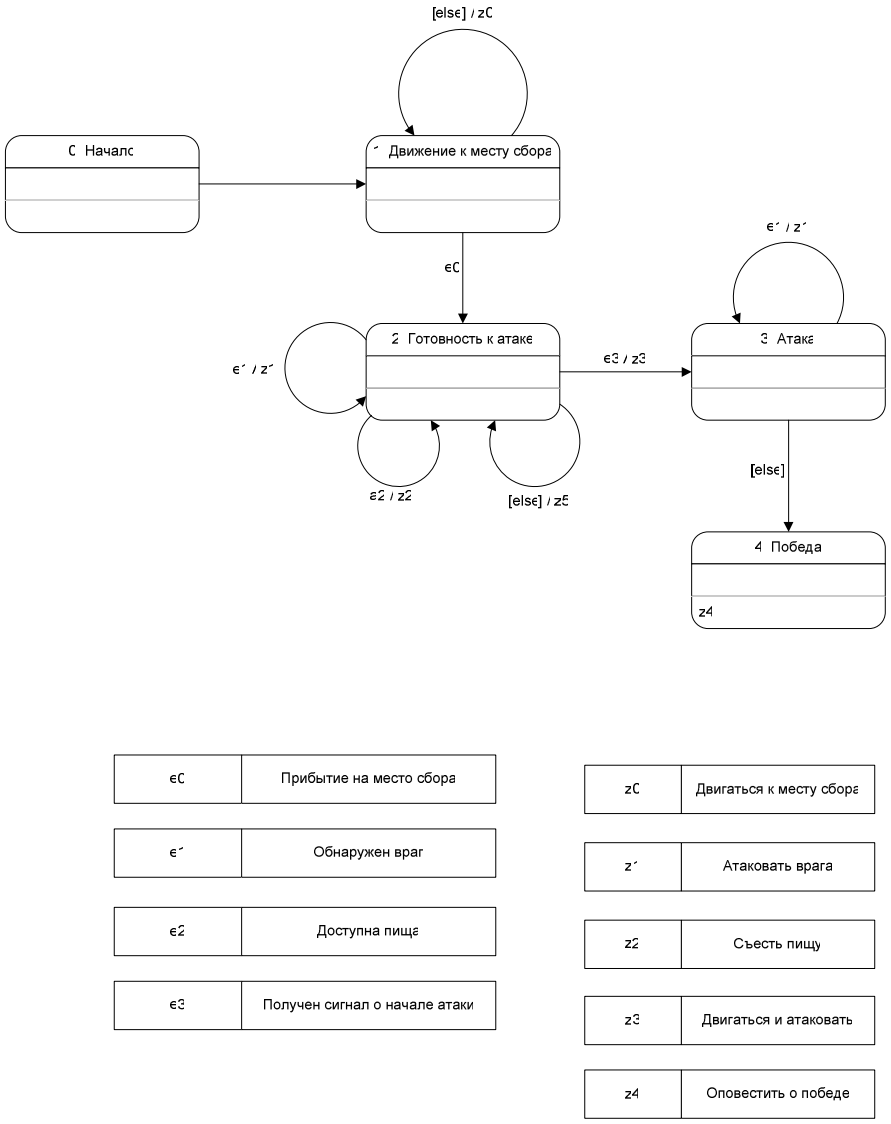


Рис. 6. Граф переходов, описывающий работу воина

### 2.3.3.1. Боевой отряд

Для координации действий воинов был создан специальный класс – боевой отряд. В задачу боевого отряда входит доставка воинов к месту сбора, разведка позиций противника и, наконец, синхронная атака всеми воинами, входящими в отряд.

Для разведки позиций из отряда выделяется один боец, который посылается на занятую противником позицию. Там он, естественно, погибает, однако он успевает узнать точную численность противника в данной точке.

На рис. 7 показан боевой отряд, собирающий силы для атаки оккупированного источника (синим обведены воины, собирающиеся в боевой отряд).

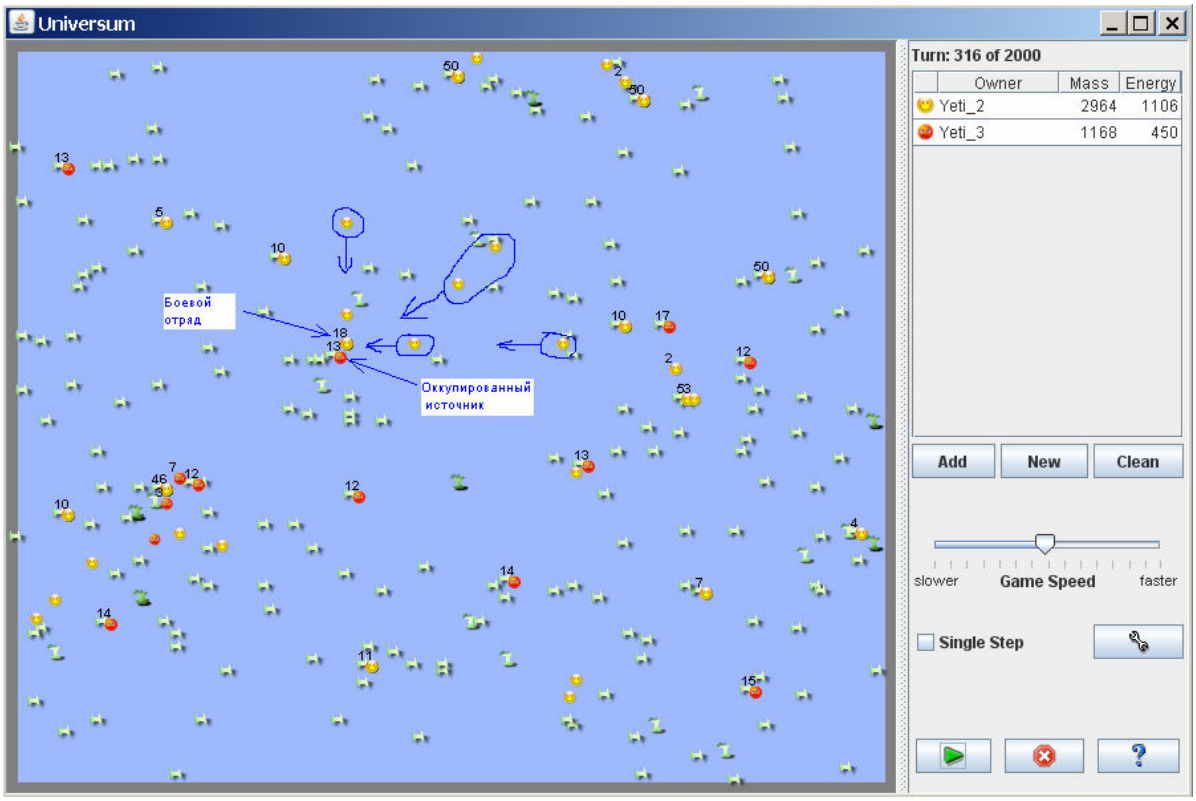


Рис. 7. Боевой отряд, собирающий силы для атаки оккупированного источника

Полностью поведение боевого отряда задаётся конечным автоматом, который представлен на рис. 8.

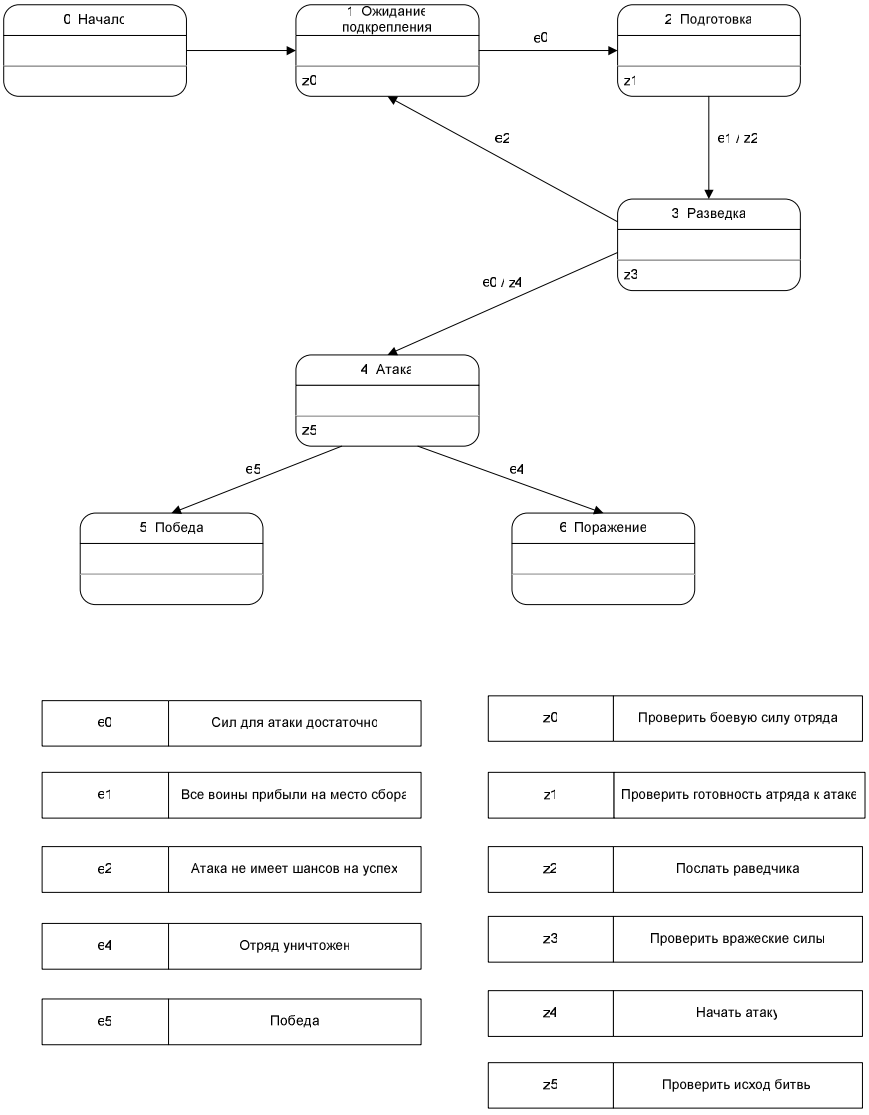


Рис. 8. Граф переходов, описывающий работу боевого отряда

## *2.4. Управление*

Общее управление существами производится посредством “глобального разума” – статического класса, к которому в процессе поединка имеют доступ все существа *Yeti*. “Разум” содержит в себе всю информацию о текущем положении дел у популяции:

- список всех существ популяции;
- список всех боевых отрядов;

- информацию о том, какие участки карты исследованы, а какие нет;
- информацию о найденных ресурсах и их ёмкости;
- информацию об оккупированных источниках и примерных силах врага расположившегося на них.

“Разум” предоставляет существам всю топологическую информацию, вычисляет оптимальный путь до заданной точки и т.д. Также он занимается общей координацией боевых действий – определяет, какие источники требуют захвата в первую очередь, и формирует боевые отряды для их захвата. Однако непосредственно управлением отряда “разум” не занимается.

Также “разум” определяет несколько стадий развития популяции. В зависимости от текущей стадии, при рождении нового существа, производится выбор – существом какого из трёх видов оно станет. Таким образом, регулируется численность различных видов на каждой из стадий. На данный момент существуют три стадии:

- STAGE_0 – на этой стадии популяция производит в основном разведчиков. В случае если на этой стадии были найдены золотые ресурсы, то для каждого из них популяция производит по одному стражнику. Переход в следующую стадию осуществляется по истечении некоторого количества ходов (в текущей версии после 100 ходов);
- STAGE_1 – на этой стадии осуществляется более сильная охрана уже найденных источников, но в то же время все лишние силы пускаются на разведку. Переход в следующую стадию осуществляется по истечении некоторого количества ходов (в текущей версии после 150 ходов), либо в случае если исследовано 95% карты;
- STAGE_2 – на этой стадии ещё больше усиливается охрана. Также перестают производиться разведчики, а все существующие разведчики “заново рождаются” (Производится изменение их состояния так, как будто они только что родились) и становятся либо стражниками, либо воинами. На этой стадии популяция начинает вести боевые действия.

## 2.5. Структура программы

Главный класс программы *Yeti_2* описан в файле `Yeti_2.java`. Именно методы этого класса вызывает игровая оболочка и именно из него игровой оболочке передаются ходы существа. В этом файле описаны три автомата: исследователь, стражник и воин. Для хранения информации специфической для каждого вида существ используется класс `BeingKind` (файл `BeingKind.java`) .

В файле `Yeti_3.java` описан класс `Yeti_3`, который наследуется от класса `Yeti_2` и является его точной копией. Единственное, чем он отличается от своего родителя, это начальной массой, которая у него равна шести.

В файле `Consciousnss.java` описан ”Разум”, который осуществляет управление действиями популяции в целом (подробнее описано в разд. 2.4).

В начале игры, исследователи могут двигаться как до ближайшей не исследованной территории, так и по строго заданному маршруту. В последнем случае управление движением по маршруту выполняется в классе `Task` (файл `Task.java`)

Вспомогательный класс `Field.java` используется “Разумом” для хранения информации об игровом поле, и, в частности, об исследованной и неисследованной его части, об источниках энергии, а также о врагах, на которых когда-либо натыкались существа популяции.

За клетки игрового поля отвечает класс `MyPointInfo` (файл `MyPointInfo.java`), а за определение расстояния между клетками класс `LocationComparator` (файл `LocationComparator.java`).

В файле `WarriorDetachment.java` содержится описание боевого отряда – его численность, источник который требуется захватить, точка сбора и другая служебная информация. Также в нем содержится еще один управляющий автомат.

В ходе описания боевых действий возникла следующая проблема – за один ход все действия существ осуществляются одновременно: сначала игровой оболочке передаются те ходы, которые хочет совершить популяция, а лишь затем игровая оболочка их все выполняет. Таким образом, в случае синхронной атаки, атакующее существо не знает об уроне, уже нанесенном другими существами жертве, которую оно собирается атаковать. Это крайне негативно влияло на эффективность боевых действий. Для повышения эффективности синхронной атаки или обороны было решено запоминать все атакующие действия, прогнозировать причиненный урон врагу и, исходя из этого, предлагать атакующему существу того врага, который еще не убит его собратьями. Решение этой задачи реализовано в классе `Operation` (файл `Operation.java`).

## Заключение

При разработке существа основное внимание уделялось главному виду состязаний – дуэли. В этом конкурсе два конкурирующих вида существ, сражаются на протяжении 1000 ходов. К сожалению правила конкурса, постоянно претерпевают изменения, так, например, сейчас все участники соревнований разбиты на четыре лиги. На данный момент существо *Yeti* находится в первой лиге, однако в ближайшее время, скорее всего, выйдет в высшую лигу (был период, когда существо занимало 11 место в ”Дуэли”, попадая в двадцатку в двух остальных соревнованиях).

В конкурсе принимают участия два существа *Yeti* с названиями *Yeti_2* и *Yeti_3*. Эти два существа отличаются между собой лишь массой – у *Yeti_2* она равна 10, а у *Yeti_3* равна шести.

Рейтинг соревнований можно посмотреть здесь:
http://www.electricjungle.ru:8080/main.jsp?page=rating

- Сильнейшие популяции по результатам дуэлей

| 1 лига | | | |
|---|---|---|---|
| 44 | catz | com.being.sunday.SundayVast | 123 |
| 45 | Avenger | Beings.Avenger.Mutagen | 110 |
| 46 | jenizz | universum.beings.Yeties.Yeti_2 | 108 |
| 47 | LeZhiK | lezhik.Ant | 107 |
| 48 | jenizz | universum.beings.Yeties.Yeti_3 | 102 |
| 49 | vvp | universum.beings.SecondBeing | 102 |
| 50 | imanic | universum.beings.imanic.One | 102 |

- Самые живучие существа *Джунглей*

| | | | |
|---|---|---|---|
| 28 | tylor | tylor.Tiger | 3 |
| 29 | LeZhiK | lezhik2.Ant | 3 |
| 30 | jenizz | universum.beings.Yeties.Yeti_3 | 2 |
| 31 | vvp | universum.beings.SecondBeing | 2 |
| 32 | jarmaine | com.jarmaine.snoopies.Snoopy | 2 |
| 51 | catz | com.being.sunday.SaturdayVast | 1 |
| 52 | kruz | universum.beings.RuslanusBeing | 1 |
| 53 | jenizz | universum.beings.Yeties.Yeti_2 | 1 |
| 54 | tormal | com.ionidea.TormalBeing | 1 |
| 55 | GreenTea | GreenTea.Locust | 1 |

- Достижения в Одиночных играх

| | | | |
|---|---|---|---|
| 54 | VladimirTTT | universum.beings.Tbeing | 4143 |
| 55 | Dyakonov | universum.beings.Sapsan | 4140 |
| 56 | jenizz | universum.beings.Yeties.Yeti_2 | 4133 |
| 57 | DHelper | yargroup.EasyLight | 4128 |
| 58 | LeZhiK | lezhik.Ant | 4120 |
| 60 | Sauron | creature.Infusorium | 4110 |
| 61 | tivv | com.gtechua.being.b1.B2 | 4069 |
| 62 | jenizz | universum.beings.Yeties.Yeti_3 | 4046 |
| 63 | Putnik | universum.beings.Life | 4013 |
| 64 | mulya | universum.beings.Murashy_beta | 3980 |

## Источники


1. *Sun Microsystems*. Конкурс алгоритмов «Электрические Джунгли». http://www.electricjungle.ru
2. *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
3. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/1
4. *Электрические джунгли.* http://is.ifmo.ru/elejungle/

# Приложение 1. Исходный текст программы

## *1. Файл* `Yeti_2.java. Существо` ***`Yeti`*** `с массой 10`

```
/*
 * Yeti_2.java
 *
 * Created on 13 Январь 2007 г., 18:54
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package universum.beings.Yeties;


import universum.bi.*;
import universum.util.*;
import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;

import universum.beings.Yeties.*;

/**
 *
 * @author kalugin
 */
public class Yeti_2 implements Being  {

    static Object bossHandle = null;

    static List<Yeti_2> population;

    static Consciousness razum;

    public boolean log_on = false;


    // initial parameters
    // mass
    public float M0 = 10f;
    // and speed
    public float S0 = 3f;
    // when energy level here is less than this - should start looking for better place
    static final float E_min = 0.5f;
    // which energy level is acceptable to select this point as new location
    static final float E_min2 = 0.1f;
    // minimum of energy one should have before bearing baby
    static final float E_born = 0.8f;

    // only if our total mass multiplicated on this coef is bigger than
    // enemy total mass, we will try to attack

    static final float E_attack_coef = 1.25f;

    static final float GUARDIAN_ENERGY_GIVE_THRESHOLD = 0.2f;

    // local copy of params
    public float M;
    public float S;

    public final float E_guard_attck_coef = 0.51f;

    // threshold to eat
    private float E_eat;
```

```
// array to store info about other beings
Integer[] others;
// our own id
Integer id;

public Event take_action = null;

// population control
static int maxCount = 200;
static int currentCount = 0;

public static int current_turn = -1;



enum States {
    y0_INITIAL,
    y1_EXPLORER_BORN,
    y2_EXPLORER_SEARCH_FOOD,
    y3_EXPLORER_EAT,
    y4_EXPLORER_ATTACK,


    y1_GUARDIAN_BORN,
    y2_GUARDIAN_MOVE_TO_SOURCE,
    y3_GUARDIAN_GUARD_SOURCE,
    y4_GUARDIAN_FIGHT,
    y5_GUARDIAN_EAT,
    y6_GUARDIAN_MOVE_AND_ATTACK,
    y7_GUARDIAN_TRY_TO_GRANT,

    y1_WARRIOR_MOVE_TO_ASSEMBLY,
    y2_WARRIOR_READY_TO_ATTACK,
    y3_WARRIOR_ATTACK,
    y4_WARRIOR_SWITCH_TO_GUARDIAN
};


public BeingKind being_kind;

public States being_state  = States.y0_INITIAL;


/** Creates a new instance of Yeti_2 */
public Yeti_2() {
}

public void setTask( Task task ) {
    being_kind.setTask( task );
}

public Task getTask() {
    return being_kind.getTask();
}

public void log( BeingKind.CurrentType type, BeingInterface bi, String message ) {
   bi.log( this, type.toString() + bi.getLocation( this ) + ": " + message );
}

// this one is the essence of your being - it's invoked once a turn
// and lets your being do something
public synchronized Event makeTurn(BeingInterface bi) {
    // information about point we're in
    PointInfo pi = bi.getPointInfo(this);
    // how much energy do we have now
    float e = bi.getEnergy(this);
    // how much energy available at cour current location
    float avail = pi.getCount(this) - 1;


    if( bi.getTurnsCount() > current_turn ) {
```

```
            current_turn = bi.getTurnsCount();
            razum.initializeNextTurn( bi );
        }


        List<PointInfo> all_points = new ArrayList();
        all_points.addAll( bi.getNeighbourInfo( this ) );
        all_points.add( pi );

        razum.analyzePoints( all_points, bi, this );


        take_action = null;

        // select action
        switch( being_kind.getCurrent() ) {
            case EXPLORER:
                a1_explorer( bi );
                break;
            case GUARDIAN:
                a2_guardian( bi );
                break;
            case WARRIOR:
                a3_warrior( bi );
                break;
        }

        return take_action;
    }


    private void a1_explorer( BeingInterface bi ) {
        // information about point we're in
        PointInfo pi = bi.getPointInfo(this);
        // how much energy do we have now
        float e = bi.getEnergy(this);
        // how much energy available at cour current location
        float avail = pi.getCount(this) - 1;

        // array of all entities here
        Integer[] ids = pi.getEntities( this, null );
         // index of potential victum
        Integer victum = 0;


        boolean e0_FOOD_AVAILABLE = false;
        boolean e1_ENOUGH_ENERGY_TO_BORN = false;
        boolean e2_VICTUM_AVAILABLE = false;


        if( avail > ( M * E_min ) && ( avail > pi.getGrowthRate( this ) ) ) {
//            bi.log( this, bi.getLocation( this ) + " food available " +
String.valueOf( avail ) + " g.r. = " +
//                    String.valueOf( pi.getGrowthRate( this ) ) );
            e0_FOOD_AVAILABLE = true;
        } else {
//            bi.log( this, bi.getLocation( this ) + " food NOT available " +
String.valueOf( avail ) + " g.r. = " +
//                String.valueOf( pi.getGrowthRate( this ) ) );

        }

        if( e >= ( M * E_born ) ) {
            e1_ENOUGH_ENERGY_TO_BORN = true;
        }


        if( ids.length > 1 ) {
            List<Integer> enemy = new ArrayList();
            List<Integer> friend = new ArrayList();
            for( int i = 0; i < ids.length; i++ ) {
```

```
                if( bossHandle != bi.getOwner(this, ids[i]) ) {
                    enemy.add( ids[i] );
                } else {
                    friend.add( ids[i] );
                }
            }

            if( enemy.size() != 0 ) {

                int enemy_mass = 0;
                int friend_mass = 0;

                float curr_enemy_mass;
                float min_enemy_mass = 10000f;

                for( int i = 0; i < enemy.size(); i++ ) {
                    curr_enemy_mass = bi.getMass( this, enemy.get( i ).intValue() );
                    enemy_mass += curr_enemy_mass;
                    if( curr_enemy_mass < min_enemy_mass ) {
                        min_enemy_mass = curr_enemy_mass;
                        victum = enemy.get( i );
                    }
                }

                for( int i = 0; i < friend.size(); i++ ) {
                    friend_mass += bi.getMass( this, friend.get( i ).intValue() );
                }

                if( friend_mass * E_attack_coef > enemy_mass/* &&
                  ( ( e / M - Constants.K_fightcost - Constants.K_emin ) * M  >
                        min_enemy_mass * Constants.K_retaliate )*/ ) {
                    e2_VICTUM_AVAILABLE = true;
                }

            }
        }

        // Finite-State Machine. Автомат "Исследователь"

        while( true ) {
            switch( being_state ) {
                default:
                    being_state = States.y1_EXPLORER_BORN;
                    log( BeingKind.CurrentType.EXPLORER, bi, "переход из состояния НАЧАЛО в
ДЕЛЕНИЕ" );
                    continue;
                case y3_EXPLORER_EAT:
                    log( BeingKind.CurrentType.EXPLORER, bi, "переход из состояния ЕДА в
ДЕЛЕНИЕ" );
                    z3_explorer_eat( bi );
                    log( BeingKind.CurrentType.EXPLORER, bi, "действие z3" );
                    being_state = States.y1_EXPLORER_BORN;



                    break;
                case y1_EXPLORER_BORN:
                    if( e1_ENOUGH_ENERGY_TO_BORN ) {
                        log( BeingKind.CurrentType.EXPLORER, bi, "событие e1, остаёмся в
состоянии ДЕЛЕНИЕ " );
                        z0_explorer_born( bi );
                        log( BeingKind.CurrentType.EXPLORER, bi, "действие z0" );
                        break;
                    }

                    if( e2_VICTUM_AVAILABLE ) {
                        log( BeingKind.CurrentType.EXPLORER, bi, "событие e2, переход из
состояния ДЕЛЕНИЕ в АТАКА" );
                        being_state = States.y4_EXPLORER_ATTACK;
                        z1_explorer_attck( bi, victum );
                        log( BeingKind.CurrentType.EXPLORER, bi, "действие z1" );
```

```
                        break;
                    }

                    log( BeingKind.CurrentType.EXPLORER, bi, "[else], переход из состояния
ДЕЛЕНИЕ в ПОИСК ПИЩИ" );

                    being_state = States.y2_EXPLORER_SEARCH_FOOD;

                    continue;
                case y2_EXPLORER_SEARCH_FOOD:

                    if( e0_FOOD_AVAILABLE ) {
                        log( BeingKind.CurrentType.EXPLORER, bi, "событие e0, переход из
состояния ПОИСК ПИЩИ в ЕДА" );
                        being_state = States.y3_EXPLORER_EAT;
                        continue;
                    }

                    log( BeingKind.CurrentType.EXPLORER, bi, "[else], остаёмся в состоянии
ПОИСК ПИЩИ " );

                    z2_explorer_searchFood( bi );

                    log( BeingKind.CurrentType.EXPLORER, bi, "действие z2" );

                    break;

                case y4_EXPLORER_ATTACK:

                    if( e2_VICTUM_AVAILABLE ) {
                        log( BeingKind.CurrentType.EXPLORER, bi, "событие e2, остаёмся в
состоянии АТАКА" );
                        z1_explorer_attck( bi, victum );
                        log( BeingKind.CurrentType.EXPLORER, bi, "действие z1" );
                        break;
                    }
                    log( BeingKind.CurrentType.EXPLORER, bi, "[else], переход из состояния
АТАКА в ДЕЛЕНИЕ" );
                    being_state = States.y1_EXPLORER_BORN;
                    continue;
            }
            break;
        }

    }

    private void z0_explorer_born( BeingInterface bi ) {
        BeingParams bp = getParams();
        bp.parameter = id;


        bp.M = this.M * Constants.K_minbornvariation;

        if( bp.M < Constants.K_minmass ) {
            bp.M = Constants.K_minmass;
        }

        take_action = new Event(EventKind.ACTION_BORN, bp);
    }

    private void z1_explorer_attck( BeingInterface bi, Integer victum ) {
        take_action = new Event(EventKind.ACTION_ATTACK, victum);
    }

    private void z2_explorer_searchFood( BeingInterface bi ) {


        List<PointInfo> ns = bi.getNeighbourInfo(this);

        for (PointInfo n : ns) {
            // if one of our neighbour locations have enough evergy -
```

```
        // move there
        if ( ( ( n.getCount(this) -  1 ) > ( M * E_min2 ) ) &&
                ( ( n.getCount(this) -  1 ) > n.getGrowthRate( this ) ) ) {
            // OK, go this way and hope for the best
            take_action = new Event(EventKind.ACTION_MOVE_TO, n.getLocation());
            return;
        }
    }

    Location loc = razum.whereToGo( bi, this );

    if( loc != null ) {
        take_action = new Event( EventKind.ACTION_MOVE_TO, loc );
    } else {
        take_action = null;
    }

}

private void z3_explorer_eat( BeingInterface bi ) {
    take_action = new Event(EventKind.ACTION_EAT, M);
}



private void a2_guardian( BeingInterface bi ) {

    // information about point we're in
    PointInfo pi = bi.getPointInfo(this);
    // how much energy do we have now
    float e = bi.getEnergy(this);
    // how much energy available at cour current location
    float avail = pi.getCount(this) - 1;

    // array of all entities here
    Integer[] ids = pi.getEntities( this, null );
     // index of potential victum
    Integer aggressor = 0;
    Integer victum = 0;



    boolean e0_born_available = false;
    boolean e1_at_source = false;
    boolean e2_eat_avail = false;
    boolean e3_enemy_here = false;
    boolean e4_victum_avail = false;

    if( e >= ( M * ( E_born ) ) ) {
        e0_born_available = true;
    }

    if( pi.getLocation() == being_kind.getGuardObject() ) {
        e1_at_source = true;
    }

    if( avail > ( M * E_min ) ) {
        e2_eat_avail = true;
    }

    if( ids.length > 1 ) {
        List<Integer> enemy = new ArrayList();
        List<Integer> friend = new ArrayList();
        for( int i = 0; i < ids.length; i++ ) {
            if( bossHandle != bi.getOwner(this, ids[i]) ) {
                enemy.add( ids[i] );
            } else {
                friend.add( ids[i] );
            }
        }
```

```
    if( enemy.size() != 0 ) {

        float curr_enemy_mass;
        float min_enemy_mass = 10000f;

        for( int i = 0; i < enemy.size(); i++ ) {
            curr_enemy_mass = bi.getMass( this, enemy.get( i ).intValue() );
            if( curr_enemy_mass < min_enemy_mass ) {
                min_enemy_mass = curr_enemy_mass;
                aggressor = enemy.get( i );
            }
        }

        // !!!!!!!!!!
        int idx = Util.rnd( enemy.size() );
        aggressor =  enemy.get( idx );
        // !!!!!!!!!!!

        e3_enemy_here = true;
    }
}

List <PointInfo> neighbours = bi.getNeighbourInfo( this );

for( PointInfo neighbour : neighbours ) {
    // array of all entities here
    Integer[] neighbour_ids = neighbour.getEntities( this, null );
     // index of potential victum
    Integer neighbour_victum = 0;

    if( neighbour_ids.length > 1 ) {
        List<Integer> enemy = new ArrayList();
        List<Integer> friend = new ArrayList();
        for( int i = 0; i < neighbour_ids.length; i++ ) {
            if( bossHandle != bi.getOwner(this, neighbour_ids[i]) ) {
                enemy.add( neighbour_ids[i] );
            } else {
                friend.add( neighbour_ids[i] );
            }
        }

        if( enemy.size() != 0 ) {

            int enemy_mass = 0;
            int friend_mass = 0;

            float curr_enemy_mass;
            float min_enemy_mass = 10000f;

            for( int i = 0; i < enemy.size(); i++ ) {
                curr_enemy_mass = bi.getMass( this, enemy.get( i ).intValue() );
                enemy_mass += curr_enemy_mass;
                if( curr_enemy_mass < min_enemy_mass ) {
                    min_enemy_mass = curr_enemy_mass;
                    victum = enemy.get( i );
                }
            }

            for( int i = 0; i < friend.size(); i++ ) {
                friend_mass += bi.getMass( this, friend.get( i ).intValue() );
            }

            if( friend_mass * E_guard_attck_coef > enemy_mass &&
              ( ( e / M - Constants.K_fightcost - Constants.K_emin ) * M  >
                    min_enemy_mass * Constants.K_retaliate ) ) {
                e4_victum_avail = true;
            }

        }
    }
```

```
        }

        // Finite-State Machine. Автомат "Стражник"


        while( true ) {
            switch( being_state ) {

                default:
                    log( BeingKind.CurrentType.GUARDIAN, bi, "переход из состояния НАЧАЛО в
ДЕЛЕНИЕ" );
                    being_state = States.y1_GUARDIAN_BORN;
                    continue;

                case y1_GUARDIAN_BORN:
                    if( e0_born_available ) {
                        log( BeingKind.CurrentType.GUARDIAN, bi, "событие e0, остаёмся в
состоянии ДЕЛЕНИЕ" );
                        z0_guardian_born( bi, e1_at_source );
                        log( BeingKind.CurrentType.GUARDIAN, bi, "действие z0" );
                        break;
                    }
                    if( e1_at_source ) {
                        log( BeingKind.CurrentType.GUARDIAN, bi, "событие e1, переход из
состояния ДЕЛЕНИЕ в ОХРАНА ИСТОЧНИКА" );
                        being_state = States.y3_GUARDIAN_GUARD_SOURCE;
                        continue;
                    }
                    log( BeingKind.CurrentType.GUARDIAN, bi, "[else], переход из состояния
ДЕЛЕНИЕ в ДВИЖЕНИЕ К ИСТОЧНИКУ" );
                    being_state = States.y2_GUARDIAN_MOVE_TO_SOURCE;
                    z1_guardian_move_to_source( bi );
                    log( BeingKind.CurrentType.GUARDIAN, bi, "действие z1" );
                    break;

                case y2_GUARDIAN_MOVE_TO_SOURCE:
                    if( e1_at_source ) {
                        log( BeingKind.CurrentType.GUARDIAN, bi, "событие e1, переход из
состояния ДВИЖЕНИЕ К ИСТОЧНИКУ в ДЕЛЕНИЕ" );
                        being_state = States.y1_GUARDIAN_BORN;
                        continue;
                    }
                    z1_guardian_move_to_source( bi );
                    log( BeingKind.CurrentType.GUARDIAN, bi, "действие z1" );
                    break;

                case y3_GUARDIAN_GUARD_SOURCE:
                    if( e3_enemy_here ) {
                        log( BeingKind.CurrentType.GUARDIAN, bi, "событие e3, переход из
состояния ОХРАНА ИСТОЧНИКА в АТАКА" );
                        being_state = States.y4_GUARDIAN_FIGHT;
                        z2_guardian_fight( bi, aggressor );
                        log( BeingKind.CurrentType.GUARDIAN, bi, "действие z2" );
                        break;
                    }

                    if( e2_eat_avail ) {
                        log( BeingKind.CurrentType.GUARDIAN, bi, "событие e2, переход из
состояния ОХРАНА ИСТОЧНИКА в ЕДА" );
                        being_state = States.y5_GUARDIAN_EAT;
                        continue;
                    }

                    if( e4_victum_avail ) {
                        log( BeingKind.CurrentType.GUARDIAN, bi, "событие e4, переход из
состояния ОХРАНА ИСТОЧНИКА в ДВИЖЕНИЕ и АТАКА" );
                        being_state = States.y6_GUARDIAN_MOVE_AND_ATTACK;
                        z5_guardian_move_and_attack( bi, victum ); //!!!!!!!!!!!
                        log( BeingKind.CurrentType.GUARDIAN, bi, "действие z5" );
                        break;
                    }
```

```
                    log( BeingKind.CurrentType.GUARDIAN, bi, "[else], переход из состояния
ОХРАНА ИСТОЧНИКА в ОТДАТЬ ЭНЕРГИЮ" );
                    being_state = States.y7_GUARDIAN_TRY_TO_GRANT;
                    continue;

                case y4_GUARDIAN_FIGHT:
                    if( e3_enemy_here ) {
                        log( BeingKind.CurrentType.GUARDIAN, bi, "событие e3, остаёмся в
состоянии АТАКА" );
                        z2_guardian_fight( bi, aggressor );
                        log( BeingKind.CurrentType.GUARDIAN, bi, "действие z2" );
                        break;
                    }
                    log( BeingKind.CurrentType.GUARDIAN, bi, "[else], переход из состояния
АТАКА В ОХРАНА ИСТОЧНИКА" );
                    being_state = States.y3_GUARDIAN_GUARD_SOURCE;
                    continue;

                case y5_GUARDIAN_EAT:
                    log( BeingKind.CurrentType.GUARDIAN, bi, "переход из состояния ЕДА В
ДЕЛЕНИЕ" );
                    being_state = States.y1_GUARDIAN_BORN;
                    z3_guardian_eat( bi );
                    log( BeingKind.CurrentType.GUARDIAN, bi, "действие z3" );
                    break;

                case y6_GUARDIAN_MOVE_AND_ATTACK:
                    if( e4_victum_avail ) {
                        log( BeingKind.CurrentType.GUARDIAN, bi, "событие e4, остаёмся в
состоянии ДВИЖЕНИЕ И АТАКА" );
                        z5_guardian_move_and_attack( bi, victum ); // !!!!!!!!!!!!
                        log( BeingKind.CurrentType.GUARDIAN, bi, "действие z5" );
                        break;
                    }
                    log( BeingKind.CurrentType.GUARDIAN, bi, "[else], переход из состояния
ДВИЖЕНИЕ И АТАКА в ДВИЖЕНИЕ К ИСТОЧНИКУ" );
                    being_state = States.y2_GUARDIAN_MOVE_TO_SOURCE;
                    continue;

                case y7_GUARDIAN_TRY_TO_GRANT:
                    log( BeingKind.CurrentType.GUARDIAN, bi, "переход из состояния ОТДАТЬ
ЭНЕРГИЮ в ДЕЛЕНИЕ" );
                    being_state = States.y1_GUARDIAN_BORN;
                    z4_guardian_try_to_grant( bi );
                    log( BeingKind.CurrentType.GUARDIAN, bi, "действие z4" );
                    break;
            }
            break;
        }

         if( log_on ) {
            bi.log( this, being_kind.getCurrent().toString() +
                    "    being_state=" + being_state.toString() );
            bi.log( this, " M=" + String.valueOf( M ) + " e=" + String.valueOf( e ) + " S=" +
String.valueOf( S ) );
            bi.log( this, " e0_born_available=" + String.valueOf( e0_born_available ) );
            bi.log( this, " e1_at_source=" + String.valueOf( e1_at_source ) );
            bi.log( this, " e2_eat_avail=" + String.valueOf( e2_eat_avail ) );
            bi.log( this, " e3_enemy_here=" + String.valueOf( e3_enemy_here ) );
            bi.log( this, " e4_victum_avail=" + String.valueOf( e4_victum_avail ) );
            bi.log( this, "-----------------------------------" );

        }

    }

    private void z0_guardian_born( BeingInterface bi, boolean is_at_source ) {
        BeingParams bp = getParams();
        bp.parameter = id;
        if( is_at_source ) {
            bp.M = this.M * Constants.K_maxbornvariation;
```

```
//            bp.S = this.S * Constants.K_minbornvariation;

            if( bp.M > Constants.K_maxmass ) {
                bp.M = Constants.K_maxmass;
            }

//            if( bp.S < 2 ) {        //!!!!!!!!!!
//                bp.S = 2;
//            }
        }
        take_action = new Event(EventKind.ACTION_BORN, bp);
    }

    private void z1_guardian_move_to_source( BeingInterface bi ) {
        take_action = new Event( EventKind.ACTION_MOVE_TO,
                razum.moveTo( bi, this, being_kind.getGuardObject() ) );
    }

    private void z2_guardian_fight( BeingInterface bi, Integer victum ) {
        take_action = new Event(EventKind.ACTION_ATTACK, victum);
        razum.achtung( bi, this );


    }

    private void z3_guardian_eat( BeingInterface bi ) {
        take_action = new Event(EventKind.ACTION_EAT, M);
    }

    private synchronized void z4_guardian_try_to_grant( BeingInterface bi ) {

//        if( ( bi.getEnergy( this ) / this.M ) < GUARDIAN_ENERGY_GIVE_THRESHOLD ) {
//            take_action = null;
//            return;
//        }

//
//        Location curr_loc = bi.getLocation( this );
//
//        List <Yeti_2> all_guards = razum.getGuardiansList( curr_loc );
//
//        List <Yeti_2> all_guards_here = new ArrayList();
//
//        if( all_guards == null ) {
//            take_action = null;
//            System.out.println( "gogno sluchilos'" );
//            return;
//        }
//
//        for( Yeti_2 yeti : all_guards ) {
//            if( ( bi.getLocation( yeti ) == curr_loc ) && ( yeti != this ) ) {
//                all_guards_here.add( yeti );
//            }
//        }
//
//        Yeti_2 most_strong = null;
//
//        float most_strong_val = this.M;
//
//        for( Yeti_2 yeti : all_guards_here ) {
//            if( ( yeti.M > most_strong_val ) && ( bi.getEnergy( yeti ) < 0.8 ) ) {
//                most_strong = yeti;
//                most_strong_val = yeti.M;
//            }
//        }
//
//        if( most_strong != null ) {
//            take_action = new Event(EventKind.ACTION_GIVE, bi.getId( most_strong ),
Math.min( bi.getEnergy( this ), ( most_strong.M - bi.getEnergy( most_strong ) ) ) );
//            all_guards.remove( most_strong );
//            return;
```

```
//        }

        take_action = null;
    }

    private void z5_guardian_move_and_attack( BeingInterface bi, Integer victum ) {
        take_action = new Event(EventKind.ACTION_MOVE_ATTACK, victum );
    }



    private void a3_warrior( BeingInterface bi ) {
        // information about point we're in
        PointInfo pi = bi.getPointInfo(this);
        // how much energy do we have now
        float e = bi.getEnergy(this);
        // how much energy available at cour current location
        float avail = pi.getCount(this) - 1;

        // array of all entities here
        Integer[] ids = pi.getEntities( this, null );



        boolean e0_at_assembly_point = false;
        boolean e1_enemy_here = false;
        boolean e2_eat_avail = false;
        boolean e3_signal_to_attack = false;
        boolean e4_freed = false;



        if( pi.getLocation() == being_kind.getAssemplyPoint() ) {
            e0_at_assembly_point = true;
        }

        if( ids.length > 1 ) {
            List<Integer> enemy = new ArrayList();
            List<Integer> friend = new ArrayList();
            for( int i = 0; i < ids.length; i++ ) {
                if( bossHandle != bi.getOwner(this, ids[i]) ) {
                    enemy.add( ids[i] );
                } else {
                    friend.add( ids[i] );
                }
            }

            if( enemy.size() > 0 ) {

                float curr_enemy_mass;
                float total_enemy_mass = 0;
                float min_enemy_mass = 10000f;

                for( int i = 0; i < enemy.size(); i++ ) {
                    curr_enemy_mass = bi.getMass( this, enemy.get( i ).intValue() );
                    total_enemy_mass += curr_enemy_mass;
                    if( curr_enemy_mass < min_enemy_mass ) {
                        min_enemy_mass = curr_enemy_mass;
                    }
                }

                e1_enemy_here = true;

            } else if( pi.getLocation() == being_kind.getTargetPoint() ) {
                e4_freed = true;
            }
        }


        if( avail > 0 ) {
```

```
        e2_eat_avail = true;
    }

    // signal to attack
    e3_signal_to_attack = being_kind.getAttackFlag();


    // Finite-State Machine. Автомат "Воин"

    while( true ) {
        switch( being_state ) {

            default:
                being_state = States.y1_WARRIOR_MOVE_TO_ASSEMBLY;
                continue;

            case y1_WARRIOR_MOVE_TO_ASSEMBLY:
                if( e0_at_assembly_point ) {
                    being_state = States.y2_WARRIOR_READY_TO_ATTACK;
                    continue;
                }
                z0_warrior_move_to_assembly( bi );
                break;
            case y2_WARRIOR_READY_TO_ATTACK:
                if( e3_signal_to_attack ) {
                    being_state = States.y3_WARRIOR_ATTACK;
                    z3_warrior_move_and_attack( bi ); // !!!!!!!!!
                    break;
                }
                if( e1_enemy_here ) {
                    z1_warrior_fight( bi );
                }
                if( e2_eat_avail ) {
                    z2_warrior_eat( bi );
                    break;
                }

                z5_warrior_wait( bi );
                break;
            case y3_WARRIOR_ATTACK:
                if( e4_freed ) {
                    being_state = States.y4_WARRIOR_SWITCH_TO_GUARDIAN;
                    continue;
                }
                // !!!!!!!!!!!!!!!!
                if( pi.getLocation() != being_kind.getTargetPoint() ) {
                    if( pi.getLocation() == being_kind.getAssemplyPoint() ) {
                        z0_warrior_move_to_assembly( bi );
                        break;
                    }
                    z6_warrior_move_to_target( bi );
                    break;
                }
                // !!!!!!!!!!!!!!!!!!

                z1_warrior_fight( bi );
                break;
            case y4_WARRIOR_SWITCH_TO_GUARDIAN:
                z4_warrior_switch_to_guardian( bi );
                break;
        }
        break;
    }

}

private void z0_warrior_move_to_assembly( BeingInterface bi ) {
    take_action = new Event( EventKind.ACTION_MOVE_TO,
            razum.moveTo( bi, this, being_kind.getAssemplyPoint() ) );
}
```

```
private Integer warrior_select_victim( BeingInterface bi ) {

    List<Integer> victum_list = razum.getEnemyIdList( being_kind.getTargetPoint() );

    if( ( victum_list != null ) && ( victum_list.size() > 0 ) ) {
        for( Integer vic : victum_list ) {
            if( razum.getEnemyMass( being_kind.getTargetPoint() , vic ) == 0 ) {
                continue;
            } else {
            }

            if( razum.commitAttackOpearation( bi, vic, being_kind.getTargetPoint(), this,
                    Constants.K_fight * this.M ) ) {
                return  vic;
            }
        }
    }


    int idx = Util.rnd( victum_list.size() );
    return victum_list.get( idx );
}

private void z1_warrior_fight( BeingInterface bi ) {

    Integer victim = warrior_select_victim( bi );

    if( victim != null ) {
        take_action = new Event( EventKind.ACTION_ATTACK, victim );
    } else {
        take_action = null;
    }

}

private void z2_warrior_eat( BeingInterface bi ) {
    take_action = new Event(EventKind.ACTION_EAT, M);
}

private void z3_warrior_move_and_attack( BeingInterface bi ) {

    Integer victim = warrior_select_victim( bi );

    if( victim != null ) {
        take_action = new Event( EventKind.ACTION_MOVE_ATTACK, victim );
    } else {
        z6_warrior_move_to_target( bi );
    }

}

private void z6_warrior_move_to_target( BeingInterface bi ) {
        take_action = new Event( EventKind.ACTION_MOVE_TO,
            razum.moveTo( bi, this, being_kind.getTargetPoint() ) );

}

private void z4_warrior_switch_to_guardian( BeingInterface bi ) {
    being_kind.setVictoryFlag( true );
}

private void z5_warrior_wait( BeingInterface bi ) {
    take_action = null; // need to implement
}


public void upgradeBeingKind( BeingKind bk ) {
    being_state = States.y0_INITIAL;
    being_kind = bk;
}
```

```
    public BeingKind getBeingKind( ) {
        return being_kind;
    }


    public void analyzeEnemyPower( BeingInterface bi ) {
        // information about point we're in
        PointInfo pi = bi.getPointInfo(this);
        // array of all entities here
        Integer[] ids = pi.getEntities( this, null );

        if( ids.length > 1 ) {
            List<Integer> enemy = new ArrayList();
            List<Integer> friend = new ArrayList();
            for( int i = 0; i < ids.length; i++ ) {
                if( bossHandle != bi.getOwner(this, ids[i]) ) {
                    enemy.add( ids[i] );
                } else {
                    friend.add( ids[i] );
                }
            }

            if( enemy.size() != 0 ) {

                int enemy_mass = 0;
                int friend_mass = 0;

                float curr_enemy_mass;
                float min_enemy_mass = 10000f;

                for( int i = 0; i < enemy.size(); i++ ) {
                    curr_enemy_mass = bi.getMass( this, enemy.get( i ).intValue() );
                    enemy_mass += curr_enemy_mass;
                    if( curr_enemy_mass < min_enemy_mass ) {
                        min_enemy_mass = curr_enemy_mass;
                    }
                }

                for( int i = 0; i < friend.size(); i++ ) {
                    friend_mass += bi.getMass( this, friend.get( i ).intValue() );
                }

                if( friend_mass * 0.8 < enemy_mass ) {
                    razum.setOccupied( bi.getLocation( this ) );
//                    bi.log( this, "Set Occupied " +  bi.getLocation( this ).toString() );
                }
                razum.setEnemyPowerInfo( bi, this, bi.getLocation( this ), enemy_mass, enemy );

            } else {
                 // not occupied
                 razum.clearOccupied( bi.getLocation( this ) );
//                 bi.log( this, "Clear Occupied " +  bi.getLocation( this ).toString() );
            }
        }
    }


    public synchronized void processEvent(BeingInterface bi, Event e) {
        switch (e.kind()) {
            case BEING_BORN:
                // increase population counter
                currentCount++;
                init(bi, bi.getId(this), (BeingParams)e.param());
                break;
            case BEING_DEAD:
                switch( being_kind.getCurrent() ) {
                    case EXPLORER:
                        razum.delExplorer( bi, this );
                        break;
                    case GUARDIAN:
```

```
                        razum.delGuardian( bi, this, being_kind );
                        break;
                    case WARRIOR:
                        razum.delWarrior( bi, this );
                        break;
                }

                // gonna get dead, decrease population counter
                currentCount--;
//                bi.log(this, "dying...");
                analyzeEnemyPower( bi );
                population.remove( this );
                break;
            case BEING_ATTACKED:
                // now just print if one attacks us
//                bi.log(this, "attacked by "+e.sender()+" of "+
//                       bi.getOwner(this, e.sender())+" damage: "+e.param() +
//                       " my_cuurent_energy_level=" + ( bi.getEnergy( this ) / this.M ) );
//                bi.analyzeEnemyPower( bi );
                break;
            case BEING_ENERGY_GIVEN:
                // ... or gives a gift
//                bi.log(this, "got "+e.param()+" energy from "+e.sender());
                break;
        }
    }

    public String getName() {
        return "yeti_2";
    }

    // this method is invoked once per game, to make sure all static are properly inited
    public void reinit(UserGameInfo info) {
        currentCount = 0;
        bossHandle = null;
       Util.log("game "+info.kind+ " maxTurns="+info.maxTurns);
    }

    // return our parameters
    public BeingParams getParams() {
        BeingParams bp = new BeingParams(M0, S0);
        return bp;
    }

    // who this entity belongs to (and score accounted for)
    public String getOwnerName() {
        return "Yeti_2";
    }

    private void init(BeingInterface bi, Integer id, BeingParams bp) {
         this.id = id;
         this.M  = bp.M;
         this.S  = bp.S;
         this.E_eat = 0.6f * M;
         this.others = new Integer[1];


         if (bossHandle == null) {
             bossHandle = bi.getOwner(this, id);
             population = new ArrayList();
             razum = new Consciousness( bi, this, population );
         }


         being_kind = razum.chooseNewBornBeingKind( bi, this );

         population.add( this );
    }
}
```

## *2. Файл* Yeti_3.java. Существо *Yeti* с массой 6. Наследуется от класса *Yeti_2*

```
/*
 * Yeti_3.java
 *
 * Created on 23 Январь 2007 г., 14:21
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package universum.beings.Yeties;

/**
 *
 * @author kalugin
 */

import universum.beings.Yeties.*;

public class Yeti_3 extends Yeti_2 {

    /** Creates a new instance of Yeti_3 */
    public Yeti_3() {
        this.M0 = 6;
    }
    public String getName() {
        return "yeti_3";
    }

    public String getOwnerName() {
        return "Yeti_3";
    }


}
```

## *3. Файл* Consciousness.java. Разум, но половина логики, зависящей от состояний, "вынута" и перенесена в автоматы.

```
/*
 * Consciousness.java
 *
 * Created on 11 Январь 2007 г., 15:36
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package universum.beings.Yeties;

import java.util.List;
import java.util.Random;
import java.util.concurrent.*;
import universum.bi.*;
import universum.util.*;


import java.util.ArrayList;

import java.lang.Math;

import java.util.Arrays;
/**
```

```
 *
 * @author kalugin
 */
public class Consciousness {


    public final int ARROWS = 35; //35
    public final int STEP = 3; //3
    public final int DIMENSION = 3; //3
    public final int MAX_EXPLORERS_GROUP = 0; //3

    public final int MAX_DETACHMENTS = 3;

    public final int STAGE_0_MAX_TURN = 50;
    public final int STAGE_1_MAX_TURN = 180; //150

    public final int MAX_GOLDEN_GUARD_STAGE_0 = 2;
    public final int MAX_MEDIUM_GUARD_STAGE_0 = 1;
    public final int MAX_SIMPLE_GUARD_STAGE_0 = 0;


    public final int MAX_GOLDEN_GUARD_STAGE_1 = 5; //10
    public final int MAX_MEDIUM_GUARD_STAGE_1 = 2;
    public final int MAX_SIMPLE_GUARD_STAGE_1 = 0;


    public final int MAX_GOLDEN_GUARD_STAGE_2 = 50;
    public final int MAX_MEDIUM_GUARD_STAGE_2 = 10;
    public final int MAX_SIMPLE_GUARD_STAGE_2 = 2;

    public final int ENOUGH_GOLDEN_GUARD_STAGE_2 = 15;
    public final int ENOUGH_MEDIUM_GUARD_STAGE_2 = 5;
    public final int ENOUGH_SIMPLE_GUARD_STAGE_2 = 1;

    public final float ENEMY_KILLED_ENERGY_THRESHOLD = 0.35f;



//    public PointInfo map[][];
    public boolean is_explored[][];

    public List<Entity> initial_explorers;

    public int total_initial_explorers = 0;

    public List<Task> initial_tasks;

    public static Field field;

    public List<Yeti_2> population;

    public List<Operation> operations = null;


    enum Stage {
        STAGE_0,
        STAGE_1,
        STAGE_2,
        STAGE_3
    };

    public Stage stage;

    public List<Location> occupied_list;

//    public boolean war_state = false;
//    public float my_forces_power = 0;
//    Location assembly_point;
//    Location attack_point;
    public ArrayBlockingQueue<WarriorDetachment> detachments;
```

```
    /** Creates a new instance of Consciousness */
    public Consciousness( BeingInterface bi, Entity me, List<Yeti_2> population_list ) {
//        this.map = new PointInfo[ Constants.getWidth() ][ Constants.getHeight() ];
        this.is_explored = new boolean[ Constants.getWidth() ][ Constants.getHeight() ];
        this.initial_explorers = new ArrayList();
        this.initial_tasks = new ArrayList();
        this.population = population_list;

        this.occupied_list = new ArrayList();
//        this.detachments = new ArrayList();
        this.detachments = new ArrayBlockingQueue( 50 );

        this.field = new Field( Constants.getWidth(), Constants.getHeight() );


        for( int i = 0; i < Constants.getWidth( ); i++ ) {
            for( int j = 0; j < Constants.getHeight(); j ++ ) {
                this.is_explored[i][j] = false;
            }
        }

        int x0 = bi.getLocation( me ).getX();
        int y0 = ( Constants.getHeight() + bi.getLocation( me ).getY() - (STEP * 5) ) %
Constants.getHeight();
//                int y0 = bi.getLocation( me ).getY();


        int total_rounds = Constants.getHeight() / (ARROWS * STEP);

        for( int i = 0; i < ARROWS; i++ ) {
            Task task = new Task();

            int level = i * STEP;

            for( int rounds = total_rounds; rounds >= 0; rounds-- ) {
                int t = level + ( STEP * ARROWS * rounds );

                int x ;
                int y ;
                Location wploc;

                for( int k = 0; k <= 16; k++ ) {
                    x = (int)( Constants.getWidth() * ( 1.0 * (k - 1) /16 ) );
                    y = t;
                    wploc = bi.createLocation( ( 2 * Constants.getWidth() + ( x0 + x ) ) %
Constants.getWidth() ,
                                               ( 2 * Constants.getHeight() + ( y0 + y ) ) %
Constants.getHeight() );
                    task.pushWP( wploc );
                }



            }


            initial_tasks.add( task );
        }

        switchToStage( Stage.STAGE_0 );
    }

    private synchronized boolean addNewWarrior( BeingInterface bi, Yeti_2 yeti ) {
//        System.out.println( "Adding new warrior" );
        if( occupied_list.size() == 0 ) {
            return false;
        }

        for( WarriorDetachment group : detachments ) {
```

```
            if( group.getCurrentStatus() == WarriorDetachment.Status.NEED_MORE_POWER ) {
                group.addNewWarrior( yeti );
                return true;
            }
        }

        if( detachments.size() < MAX_DETACHMENTS ) {

            List<Location> still_not_attacked = new ArrayList();

            still_not_attacked.addAll( occupied_list );

            for( WarriorDetachment group: detachments ) {
                still_not_attacked.remove( group.attack_point );
            }

            if( still_not_attacked.size() == 0 ) {
                return false;
            }


            Location curr_loc = null;

            for( Location loc : still_not_attacked ) {
                int most_weak = Integer.MAX_VALUE;
                int power;

                if( field.isGolden( loc ) ) {
                    power = (int)field.getTotalEnemyMass( loc );
                    if( power < most_weak ) {
                        most_weak = power;
                        curr_loc = loc;
                    }
                }
            }

            if( curr_loc != null ) {
                letsStartAWar( curr_loc, yeti, bi );
                return true;
            }


            for( Location loc : still_not_attacked ) {
                int most_weak = Integer.MAX_VALUE;
                int power;

                if( true ) {
                    power = (int)field.getTotalEnemyMass( loc );
                    if( power < most_weak ) {
                        most_weak = power;
                        curr_loc = loc;
                    }
                }
            }

            if( curr_loc != null ) {
                letsStartAWar( curr_loc, yeti, bi );
                return true;
            }

        }

        return false;
    }

    public synchronized void letsStartAWar( Location attack_point, Yeti_2 comander,
BeingInterface bi ) {
        WarriorDetachment wg = new WarriorDetachment( attack_point, field, bi );
        wg.addNewWarrior( comander );
        detachments.add( wg );
        for( Yeti_2 yeti : population ) {
```

```
            if( yeti.being_kind.getCurrent() == BeingKind.CurrentType.EXPLORER ) {
                if( initial_explorers.contains( yeti ) ) {
                    initial_explorers.remove( yeti );
                    total_initial_explorers--;
                }
                wg.addNewWarrior( yeti );
            }
            if( yeti.being_kind.getCurrent() == BeingKind.CurrentType.GUARDIAN ) {
                if( isEnoughGuarded( yeti.being_kind.getGuardObject() ) ) {
                    this.delGuardian( bi, yeti, yeti.being_kind );
                    wg.addNewWarrior( yeti );
                }
            }
        }
        wg.forceAttack( true ); // !!!!!!!!!!!!!
    }

    public boolean isEnoughGuarded( Location loc ) {
        if(  field.isGolden( loc ) && ( field.getGuardsNumber( loc ) >
ENOUGH_GOLDEN_GUARD_STAGE_2 ) )  {
            return true;
        }
        if(  field.isMedium( loc ) && ( field.getGuardsNumber( loc ) >
ENOUGH_MEDIUM_GUARD_STAGE_2 ) )  {
            return true;
        }

        if(  field.isSimple( loc ) && ( field.getGuardsNumber( loc ) >
ENOUGH_SIMPLE_GUARD_STAGE_2 ) )  {
            return true;
        }
        return false;
    }

    public synchronized void delWarrior( BeingInterface bi, Yeti_2 yeti ) {
        for( WarriorDetachment group : detachments ) {
            if( group.isMyWarrior( yeti ) ) {
                group.deleteWarrior( yeti );
                return;
            }
        }
    }

    private synchronized void processWar( BeingInterface bi ) {
//        System.out.println( "detachments = " + detachments.size() );
        List<WarriorDetachment> remove_candidates = new ArrayList();
        for( WarriorDetachment group : detachments ) {
            group.nextTurn( bi );
            if( group.getCurrentStatus() == WarriorDetachment.Status.WIN ) {
                // win!!!!!!!!!!!
                System.out.println( "War finished. We win!!!!!!!!!!!!!!!1" );
                clearOccupied( group.attack_point );
                remove_candidates.add( group );
//                disbandDetachment( bi, group );
                break;
            } else
            if( group.getCurrentStatus() == WarriorDetachment.Status.LOSE) {
                // loose.....................
                System.out.println( "War finished. We loose....................." );
                remove_candidates.add( group );
//                disbandDetachment( bi, group );
                break;
            }
        }
        for( WarriorDetachment group : remove_candidates ) {
            disbandDetachment( bi, group );
        }
    }

    private synchronized void disbandDetachment( BeingInterface bi, WarriorDetachment group ) {
        List<Yeti_2> warrior_list = group.getWarriorList();
```

```
        detachments.remove( group );
        System.out.println( "removing detachment" );
        for( Yeti_2 yeti : warrior_list ) {
            yeti.being_kind.switchTo( BeingKind.CurrentType.EXPLORER );
//            System.out.println( "produced new " + yeti.being_kind.getCurrent() );
        }

        for( Yeti_2 yeti : warrior_list ) {
            yeti.being_kind = chooseNewBornBeingKind( bi, yeti );
            System.out.println( "produced new " + yeti.being_kind.getCurrent() );
        }
    }

    public synchronized BeingKind chooseNewBornBeingKind( BeingInterface bi, Yeti_2 me ) {
        BeingKind new_born = new BeingKind();

        Location loc;

        switch( stage ) {
            case STAGE_0:
////                System.out.println( "born in stage_0, max_golden = " +
String.valueOf( field.MAX_GOLDEN_GUARDS ) );
//                if( field.atGolden( bi, me ) ) {
//                    loc = field.addGoldenGuard( bi, me );
//                    if( loc != null ) {
//                        new_born.switchTo( BeingKind.CurrentType.GUARDIAN );
//                        new_born.setGuardObject( loc );
//                        System.out.println( "born golden in stage_0" );
//                        return new_born;
//                    }
//                }
//
//                if( total_initial_explorers < ( ARROWS * MAX_EXPLORERS_GROUP ) ) {
//                    new_born.switchTo( BeingKind.CurrentType.EXPLORER );
//                    addExplorer( bi, me, new_born );
//                }
//
//                break;

// !!!!!!!!!
            case STAGE_1:

                if( field.atGolden( bi, me ) ) {
                    loc = field.addGoldenGuard( bi, me );
                    if( loc != null ) {
                        new_born.switchTo( BeingKind.CurrentType.GUARDIAN );
                        new_born.setGuardObject( loc );
                        return new_born;
                    }
                }

                if( total_initial_explorers < ( ARROWS * MAX_EXPLORERS_GROUP ) ) {
                    new_born.switchTo( BeingKind.CurrentType.EXPLORER );
                    addExplorer( bi, me, new_born );
                }

                if( field.atMedium( bi, me ) ) {
                    loc = field.addMediumGuard( bi, me );
                    if( loc != null ) {
                        new_born.switchTo( BeingKind.CurrentType.GUARDIAN );
                        new_born.setGuardObject( loc );
                        return new_born;
                    }
                }

                new_born.switchTo( BeingKind.CurrentType.EXPLORER );
                return new_born;

            case STAGE_2:
                        {
```

```
                            if( field.atMedium( bi, me ) ) {
                                loc = field.addMediumGuard( bi, me );
                                if( loc != null ) {
                                    new_born.switchTo( BeingKind.CurrentType.GUARDIAN );
                                    new_born.setGuardObject( loc );
//                                        System.out.println( "1111" );
                                    return new_born;
                                }
                            }


                            loc = field.addGoldenGuard( bi, me );
                            if( loc != null ) {
                                new_born.switchTo( BeingKind.CurrentType.GUARDIAN );
                                new_born.setGuardObject( loc );
                                return new_born;
                            }



                            if( addNewWarrior( bi, me ) ) {
                                return me.being_kind;
                            }

                            loc = field.addMediumGuard( bi, me );
                            if( loc != null ) {
                                new_born.switchTo( BeingKind.CurrentType.GUARDIAN );
                                new_born.setGuardObject( loc );
//                                  System.out.println( "1111" );
                                return new_born;
                            }


                            loc = field.addSimpleGuard( bi, me );
                            if( loc != null ) {
                                new_born.switchTo( BeingKind.CurrentType.GUARDIAN );
                                new_born.setGuardObject( loc );
                                return new_born;
                            }


                            new_born.switchTo( BeingKind.CurrentType.EXPLORER );
//                          System.out.println( "5555" );
                            return new_born;
                        }

        }

        new_born.switchTo( BeingKind.CurrentType.EXPLORER );
        return new_born;
    }



    public synchronized void analyzePoints( List<PointInfo> list, BeingInterface bi, Yeti_2
me ) {
        field.analyzePoints( list, me );

        processWar( bi );

        switch( stage ) {
            case STAGE_0:
                if( bi.getTurnsCount() > STAGE_0_MAX_TURN ) {
                    switchToStage( Stage.STAGE_1 );
                }
                break;
            case STAGE_1:
                if( field.enoughGoldFound() || field.enoughExplored() || bi.getTurnsCount() >
STAGE_1_MAX_TURN ) {
                    switchToStage( Stage.STAGE_2 );
                }
```

```
                break;
            case STAGE_2:
                if( field.enoughExplored() ) {
                    if( me.being_kind.getCurrent() == BeingKind.CurrentType.EXPLORER ) {
                        if( initial_explorers.contains( me ) ) {
                            initial_explorers.remove( me );
                            total_initial_explorers--;
                        }
                        me.being_kind = chooseNewBornBeingKind( bi, me );
                    }
                }
                break;
        }

    }

    public void switchToStage( Stage new_stage ) {
        stage = new_stage;
        System.out.println( " Switch to " + new_stage.toString() );

        switch( new_stage ) {

            case STAGE_0:
                field.setMaxGoldedGuard( MAX_GOLDEN_GUARD_STAGE_0 );
                field.setMaxMediumGuard( MAX_MEDIUM_GUARD_STAGE_0 );
                field.setMaxSimpleGuard( MAX_SIMPLE_GUARD_STAGE_0 );
                break;
            case STAGE_1:
                field.setMaxGoldedGuard( MAX_GOLDEN_GUARD_STAGE_1 );
                field.setMaxMediumGuard( MAX_MEDIUM_GUARD_STAGE_1 );
                field.setMaxSimpleGuard( MAX_SIMPLE_GUARD_STAGE_1 );
                break;
            case STAGE_2:
                field.setMaxGoldedGuard( MAX_GOLDEN_GUARD_STAGE_2 );
                field.setMaxMediumGuard( MAX_MEDIUM_GUARD_STAGE_2 );
                field.setMaxSimpleGuard( MAX_SIMPLE_GUARD_STAGE_2 );
                break;
        }
    }

    public synchronized void setEnemyPowerInfo( BeingInterface bi, Yeti_2 me, Location loc, int
total_mass, List<Integer> enemy_list ) {
        field.setEnemyPowerInfo( bi, me, loc, total_mass, enemy_list );
    }

    public List<Integer> getEnemyIdList( Location loc ) {
        return field.getEnemyIdList( loc );
    }

    public float getEnemyMass( Location loc, Integer enemy_id ) {
        return field.getEnemyMass( loc, enemy_id );
    }


    public List<Yeti_2> getGuardiansList( Location loc ) {
        return field.getGuards( loc );
    }

    public synchronized void setOccupied( Location loc ) {
        if( (!field.isGolden( loc )) && (!field.isMedium(loc)) ) {
            if( occupied_list.size() != 0 ) {
                return;
            }
        }
        field.setOccupied( loc );
        if( ! occupied_list.contains( loc ) ) {
            occupied_list.add( loc );
        }
    }

    public synchronized void clearOccupied( Location loc ) {
```

```
        field.clearOccupied( loc );
        if( occupied_list.contains( loc ) ) {
            occupied_list.remove( loc );
        }
    }

    public synchronized void delGuardian( BeingInterface bi, Being me, BeingKind guardian ) {
        field.delGuard( bi, me, guardian.getGuardObject() );
    }

    private void addExplorer( BeingInterface bi, Yeti_2 yeti, BeingKind explorer ) {
        explorer.setTask( initial_tasks.get( total_initial_explorers % ARROWS ) );
        total_initial_explorers++;
        initial_explorers.add( yeti );
    }

    public void delExplorer( BeingInterface bi, Yeti_2 killed ) {
        if( initial_explorers.contains( killed ) ) {
            total_initial_explorers--;

            Yeti_2 last_explorer = (Yeti_2)initial_explorers.get( initial_explorers.size() -
1 );
            last_explorer.setTask( killed.getTask() );

            initial_explorers.remove( killed );
        }
    }




    public void setInfo( List<PointInfo> list ) {
        for( PointInfo info : list ) {
//            if( ! is_explored[ info.getLocation().getX() ][ info.getLocation().getY() ] ) {
//                map[ info.getLocation().getX() ][ info.getLocation().getY() ] = info;
              is_explored[ info.getLocation().getX() ][ info.getLocation().getY() ] = true;
//            }
        }
    }


    public synchronized void initializeNextTurn( BeingInterface bi ) {
        int guards_count[][]= new int[ Constants.getWidth() ][ Constants.getHeight() ];
        List<Yeti_2> guards[][] = new
ArrayList[ Constants.getWidth() ][ Constants.getHeight() ];

        List<Yeti_2> remove_candidates = new ArrayList();

        operations = new ArrayList();


        for( int i = 0; i < Constants.getWidth(); i++ ) {
            for( int j = 0; j < Constants.getHeight(); j++ ) {
                guards_count[i][j] = 0;
                guards[i][j] = null;
            }
        }


        for( Yeti_2 yeti : population ) {

            if( yeti.getBeingKind().getCurrent() == BeingKind.CurrentType.GUARDIAN ) {
                int x,y;

                x = yeti.getBeingKind().getGuardObject().getX();
                y = yeti.getBeingKind().getGuardObject().getY();

                guards_count[x][y]++;

                if( guards[x][y] == null ) {
                    guards[x][y] = new ArrayList();
```

```
                }

                guards[x][y].add( yeti );
            }
            try {
                yeti.analyzeEnemyPower( bi );
            }
            catch ( NullPointerException e ) {
//                population.remove( yeti );
                remove_candidates.add( yeti );
                System.out.println( "exception happened " + yeti.id );
            }

        }

        for( Yeti_2 yeti : remove_candidates ) {
            population.remove( yeti );
        }


        field.setGuardsCountList( guards_count );
        field.setGuardsList( guards );
    }

    public synchronized boolean commitAttackOpearation( BeingInterface bi, Integer vicimID,
Location where, Yeti_2 me, float damage ) {
        float current_damage = 0;

        for( Operation op : operations ) {
            if( ( op.getOperationKind() == Operation.OperationKind.ATTACK ) &&
                 ( op.getAnimal().intValue() == vicimID.intValue() ) ) {
                current_damage += op.getValue();
//                System.out.println( vicimID.toString() + " is alredy damaged, its damage=" +
current_damage + " its mass=" + getEnemyMass( where, vicimID ) );
            }
        }

        if( current_damage > ENEMY_KILLED_ENERGY_THRESHOLD * getEnemyMass( where, vicimID ) ) {
//            System.out.println( vicimID.toString() + " is alredy killed, its damage=" +
current_damage + " its mass=" + getEnemyMass( where, vicimID ) );
            return false;
        }

        Operation new_operation = new Operation( Operation.OperationKind.ATTACK );

        new_operation.attackOperation( vicimID, damage );

        operations.add( new_operation );

        return true;
    }


    public void achtung( BeingInterface bi, Yeti_2 me ) {
        Location achtung_loc = bi.getLocation( me );

        if( field.isGolden( achtung_loc )  && ( ( stage == Stage.STAGE_0 ) || ( stage ==
Stage.STAGE_1 ) ) ) {
            me.analyzeEnemyPower( bi );
            WarriorDetachment wg = new WarriorDetachment( achtung_loc, field, bi );
            for( Yeti_2 yeti : population ) {
                if( ( yeti.being_kind.getCurrent() == BeingKind.CurrentType.EXPLORER )
                     && ( bi.distance( achtung_loc, bi.getLocation( yeti ) ) < 30 ) ) {
                    if( initial_explorers.contains( yeti ) ) {
                        initial_explorers.remove( yeti );
                        total_initial_explorers--;
                    }
                    wg.addNewWarrior( yeti );
                }
            }
            if( wg.getWarriorList().size() > 0 ) {
```

```
                detachments.add( wg );
                wg.forceAttack( true );
            }

        }
    }

    public Location moveTo( BeingInterface bi, Yeti_2 yeti, Location loc ) {

        List<Location> all_loc = bi.getReachableLocations( yeti );

        int len = all_loc.size();

//        bi.log( (Being)yeti , "!!!! " + all_loc.toString() );


        Location sorted_loc[] = all_loc.toArray( new Location[ len ] );

        LocationComparator  comparator = new LocationComparator( bi, loc );

        Arrays.sort( sorted_loc, comparator );

        // need to improve !!!!!!!!!!!!!!!1

//        bi.log( (Being)yeti , "x=" + String.valueOf( bi.getLocation( yeti ).getX() ) +
//                              " y=" + String.valueOf( bi.getLocation( yeti ).getY() +
//                              " x_1=" + String.valueOf( sorted_loc[0].getX() ) ) +
//                              " y_1=" + String.valueOf( sorted_loc[0].getY() ) );

        List<Location> candidate_list = new ArrayList();
        candidate_list.add( sorted_loc[0] );

        for( int i = 1; i < len; i++ ) {
            if( bi.distance( sorted_loc[i - 1], loc ) == bi.distance( sorted_loc[i], loc ) ) {
                candidate_list.add( sorted_loc[i] );
            } else {
                break;
            }
        }

        int candate_len = candidate_list.size();

        double decar_len = 900000000.0;

        Location candidate = sorted_loc[0];

        for( int i = 0; i < candate_len; i++ ) {
            Location cand = candidate_list.get(i);
//            bi.log( (Being)yeti , " cand_x=" + String.valueOf( cand.getX() ) +
//                                  " cand_y=" + String.valueOf( cand.getY() ) );
            double decar_len2 = ( cand.getX() - loc.getX() ) * ( cand.getX() - loc.getX() )
                              + ( cand.getY() - loc.getY() ) * ( cand.getY() - loc.getY() );

            if( decar_len2 < decar_len ) {
                decar_len = decar_len2;
                candidate = cand;
            }
        }

//       bi.log( (Being)yeti , " move_x=" + String.valueOf( candidate.getX() ) +
//                             " move_y=" + String.valueOf( candidate.getY() ) );


        return candidate;

    }

    private Location followTask( BeingInterface bi, Yeti_2 yeti ) {

        if( ! yeti.getTask().tasksLeft() ) {
            initial_explorers.remove( yeti );
```

```
            return null;
        }

        if( yeti.getTask().peekWP() == bi.getLocation( yeti ) ) {
            yeti.getTask().popWP();
        }

        if( ! yeti.getTask().tasksLeft() ) {
            initial_explorers.remove( yeti );
            return null;
        }

//        bi.log( (Being)yeti , "x=" + String.valueOf( bi.getLocation( yeti ).getX() ) +
//                                  " y=" + String.valueOf( bi.getLocation( yeti ).getY() ) );
//
//
//        bi.log( (Being)yeti , "wp_x=" + String.valueOf( yeti.getTask().peekWP().getX() ) +
//                                  " wp_y=" +
String.valueOf( yeti.getTask().peekWP().getY() ) );

        return moveTo( bi, yeti, yeti.getTask().peekWP() );
    }

    public Location whereToGo( BeingInterface bi, Yeti_2 yeti ) {

        Location res;

        if( initial_explorers.contains( yeti ) ) {
//            bi.log( yeti, "following the task" );
            // TODO: move according to task
            res = followTask( bi, yeti );
            if( res != null ) {
                return res;
            }
        }

        // move to best reachable non-explored

        res = field.getReachableNonExplored( bi.getLocation( yeti ), bi, yeti );

        if( res != null ) {
//            bi.log( yeti, "moving to reachable non-explored" );
            return res;
        }


        // move to nearest non-explored

        res = field.getNearestNoneExplored( bi.getLocation( yeti ), bi, yeti );

        if( res != null ) {
//            bi.log( yeti, "moving to nearest non-explored" );
            return moveTo( bi, yeti, res );
        }

        // need to be modified

        return null;

//        List<Location> all_loc = bi.getReachableLocations( yeti );
////        bi.log( yeti, "moving to random place" );
//        // nothing good around, go to random place
//        int idx = Util.rnd(all_loc.size());
//        return all_loc.get(idx);
    }

}
```

## *4. Файл* `Task.java. Задания для "исследователей".`

```
/*
 * Task.java
 *
 * Created on 15 Январь 2007 г., 15:07
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package universum.beings.Yeties;

/**
 *
 * @author kalugin
 */

import universum.bi.*;
import universum.util.*;
import java.util.List;
import java.util.ArrayList;
import java.util.Stack;

public class Task {

    private Stack<Location> waypoints;

    /** Creates a new instance of Task */
    public Task() {
        waypoints = new Stack();
    }

    public synchronized void pushWP( Location loc ) {
        waypoints.push( loc );
    }

    public synchronized void popWP() {
        waypoints.pop();
    }

    public synchronized Location peekWP() {
        return waypoints.peek();
    }

    public synchronized boolean tasksLeft() {
        return ! waypoints.empty();
    }

}
```

## *5. Файл* `Field.java. Игровое поле.`

```
/*
 * Field.java
 *
 * Created on 16 Январь 2007 г., 9:33
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package universum.beings.Yeties;


import universum.bi.*;
import universum.util.*;
import java.util.List;
import java.util.ArrayList;
```

```
import universum.beings.Yeties.*;


/**
 *
 * @author kalugin
 */
public class Field {

    public boolean is_explored[][];

    public boolean is_source[][];

    public boolean is_golden[][];

    public boolean is_medium[][];

    public boolean is_simple[][];

    public int guards_count[][];

    private List<Yeti_2> guards[][];

    public float energy[][];

    private boolean occupied[][];
    MyPointInfo enemies[][];

    public int MAX_GOLDEN_GUARDS;
    public int MAX_MEDIUM_GUARDS;
    public int MAX_SIMPLE_GUARDS;

    private List<Location> golden;

    private List<Location> medium;

    private List<Location> simple_source;

//    private List<Location> totaly_guarded;

    private int max_x,max_y;

    private static boolean all_explored = false;
    private static int fields_explored = 0;


    /** Creates a new instance of Field */
    public Field( int x, int y ) {
        this.max_x = x;
        this.max_y = y;

        is_explored = new boolean[x][y];
        is_source = new boolean[x][y];
        is_golden = new boolean[x][y];
        is_medium = new boolean[x][y];
        is_simple = new boolean[x][y];
        energy = new float[x][y];

        occupied = new boolean[x][y];

        enemies = new MyPointInfo[x][y];

        guards_count = new int[x][y];
        guards = new ArrayList[x][y];

        golden = new ArrayList();
        medium = new ArrayList();
        simple_source = new ArrayList();
  //      totaly_guarded = new ArrayList();
```

```
        for( int i = 0; i < x; i++ ) {
            for( int j = 0; j < y; j++ ) {
                is_explored[i][j] = false;
                is_source[i][j] = false;
                is_golden[i][j] = false;
                is_medium[i][j] = false;
                is_simple[i][j] = false;
                energy[i][j] = 0;
                guards_count[i][j] = 0;
                occupied[i][j] = false;
                guards[i][j] = null;
                enemies[i][j] = null;
            }
        }
    }

    public synchronized void setOccupied( Location loc ) {
        occupied[ loc.getX() ][ loc.getY() ] = true;
    }

    public synchronized void clearOccupied( Location loc ) {
        occupied[ loc.getX() ][ loc.getY() ] = false;

//        System.out.println( "field : removing occupied flag from" + loc.toString() );
    }

    public synchronized void setMaxGoldedGuard( int gg ) {
        MAX_GOLDEN_GUARDS = gg;
    }

    public synchronized void setMaxMediumGuard( int mg ) {
        MAX_MEDIUM_GUARDS = mg;
    }


    public synchronized void setMaxSimpleGuard( int sg ) {
        MAX_SIMPLE_GUARDS = sg;
    }


//    public Location addGuard( BeingInterface bi, Being me ) {
//        Location loc = null;
//
//        loc = getNearestGolden( bi, me );
//
//        while ( loc != null ) {
//            List<Location> excluding = new ArrayList();
//            if( guards[ loc.getX() ][ loc.getY() ] < MAX_GOLDEN_GUARDS ) {
//                guards[ loc.getX() ][ loc.getY() ]++;
//                return loc;
//            }
//            excluding.add( loc );
//            loc = getNearestGolden( bi, me, excluding );
//        }
//
//        loc = getNearestSource( bi, me );
//        while ( loc != null ) {
//            List<Location> excluding = new ArrayList();
//            if( guards[ loc.getX() ][ loc.getY() ] < MAX_SIMPLE_GUARDS ) {
//                guards[ loc.getX() ][ loc.getY() ]++;
//                return loc;
//            }
//            excluding.add( loc );
//            loc = getNearestSource( bi, me, excluding );
//        }
//
//        return null;
//    }

    public Boolean isGolden( Location loc ) {
        return is_golden[ loc.getX() ][ loc.getY() ];
```

```
    }
    public Boolean isMedium( Location loc ) {
        return is_medium[ loc.getX() ][ loc.getY() ];
    }

    public Boolean isSimple( Location loc ) {
        return is_simple[ loc.getX() ][ loc.getY() ];
    }

    public Location selectAssemblyPoint( Location loc, BeingInterface bi ) {
        int rad = 2;
        int x0 = loc.getX();
        int y0 = loc.getY();
        int x,y;

        List <Location> list = new ArrayList();

        for( int i = -rad; i <=rad; i++ ) {
            for( int j = -rad; j <=rad; j++ ) {
                x = ( max_x + x0 + i ) % max_x;
                y = ( max_y + y0 + j ) % max_y;

                if( ( ! occupied[x][y] ) && ( ! is_source[x][y] ) && ( bi.distance( loc,
bi.createLocation( x, y ) ) == rad ) ) {
                    list.add( bi.createLocation( x, y ) );
                }
            }
        }

        if( list.size() > 0 ) {
            int idx = Util.rnd( list.size() );
            return list.get( idx );
        }

        for( int i = -rad; i <=rad; i++ ) {
            for( int j = -rad; j <=rad; j++ ) {
                x = ( max_x + x0 + i ) % max_x;
                y = ( max_y + y0 + j ) % max_y;

                if( ( ! occupied[x][y] ) ) {
                    return bi.createLocation( x, y );
                }
            }
        }

        return null;

    }


    public synchronized Location addGoldenGuard( BeingInterface bi, Being me ) {
        return addGuard( bi, me, this.golden, MAX_GOLDEN_GUARDS );
    }

    public synchronized Location addMediumGuard( BeingInterface bi, Being me ) {
        return addGuard( bi, me, this.medium, MAX_MEDIUM_GUARDS );
    }

    public synchronized Location addSimpleGuard( BeingInterface bi, Being me ) {
        return addGuard( bi, me, this.simple_source, MAX_SIMPLE_GUARDS );
    }



    public synchronized Location addGuard( BeingInterface bi, Being me, List<Location> where,
int max ) {
        List <Location> except = new ArrayList();
        Location my_loc = bi.getLocation( me );
        for( Location l : where ) {
            if( ( guards_count[ l.getX() ][ l.getY() ] >= max ) ||
                ( occupied[ l.getX() ][ l.getY() ] ) ) {
```

```
                except.add( l );
            }
        }

        Location loc = getNearest( my_loc, where, except, bi );

        if( loc != null ) {
            guards_count[ loc.getX() ][ loc.getY() ]++;
            return loc;
        }

        return null;


//        Location loc = null;
//        loc = getNearest( bi.getLocation( me ), where, totaly_guarded, bi );
//
//        while ( loc != null ) {
//
//            if( ( guards[ loc.getX() ][ loc.getY() ] < max ) &&
//                ( !occupied[ loc.getX() ][ loc.getY() ] ) ) {
//                guards[ loc.getX() ][ loc.getY() ]++;
////                System.out.println( "adding guard to " + loc.toString() );
////                System.out.println( "   addGuard " + loc.toString()  + " curr=" +
String.valueOf( guards[ loc.getX() ][ loc.getY() ] ) );
//                return loc;
//            } else {
//
////                System.out.println( "lazha wyshla" + loc.toString() + " occ=" +
String.valueOf( occupied[ loc.getX() ][ loc.getY() ] ) + " gurargs=" +
String.valueOf( guards[ loc.getX() ][ loc.getY() ] ) );
//            }
//            totaly_guarded.add( loc );
//            loc = getNearest( bi.getLocation( me ), where, totaly_guarded, bi );
//        }
//
//        return null;
    }

    public boolean enoughGoldFound() {
        int count = 0;
        for( Location loc : golden ) {
            if( ! occupied[ loc.getX() ][ loc.getY() ] ) {
                count++;
            }
        }
        return ( count >= 3 );
    }

    public boolean enoughExplored() {
        return ( (fields_explored / ( max_x * max_y )) > 0.95 );
    }

    public synchronized void setEnemyPowerInfo( BeingInterface bi, Yeti_2 me, Location loc, int
total_mass, List<Integer> enemy_list ) {
        int x = loc.getX();
        int y = loc.getY();

        enemies[x][y] = new MyPointInfo();
        for( Integer enemy : enemy_list ) {
            enemies[x][y].add( enemy, bi.getMass( me, enemy ) );
        }
//        enemies[ loc.getX() ][ loc.getY() ] = enemy_list;
    }

    public float getTotalEnemyMass( Location loc ) {
        return enemies[ loc.getX() ][ loc.getY() ].getEnemyTotalMass();
    }

    public List<Integer> getEnemyIdList( Location loc ) {
        return enemies[ loc.getX() ][ loc.getY() ].getEnemyList();
```

```
    }

    public float getEnemyMass( Location loc, Integer enemy_id ) {
        return enemies[ loc.getX() ][ loc.getY() ].getEnemyMass( enemy_id );
    }


    public synchronized void delGuard( BeingInterface bi, Being me, Location guarded ) {
        guards_count[ guarded.getX() ][ guarded.getY() ]--;

//        System.out.println( "delGuard " + guarded.toString() + " curr=" +
String.valueOf( guards[ guarded.getX() ][ guarded.getY() ] ) );
    }

    public synchronized int getGuardsNumber( Location guarded ) {
        return guards_count[ guarded.getX() ][ guarded.getY() ];
    }

    public boolean atGolden( BeingInterface bi, Entity me ) {
        Location loc = bi.getLocation( me );
        return golden.contains( loc );
//        return is_golden[ loc.getX() ][ loc.getY() ];
    }

    public boolean atMedium( BeingInterface bi, Entity me ) {
        Location loc = bi.getLocation( me );
        return medium.contains( loc );
//        return is_medium[ loc.getX() ][ loc.getY() ];
    }


    public List<Location> getGolden() {
        return golden;
    }

    public List<Location> getSimpleSource() {
        return simple_source;
    }

    public void setGuardsCountList( int new_guards[][] ) {
        this.guards_count = new_guards;
    }

    public void setGuardsList( List<Yeti_2> list[][] ) {
        guards = list;
    }

    public List<Yeti_2> getGuards( Location loc ) {
        return guards[ loc.getX() ][ loc.getY() ];
    }



    public synchronized void analyzePoints( List<PointInfo> list, Being me ) {
        for( PointInfo pi : list ) {
            Location loc = pi.getLocation();
            int x = loc.getX();
            int y = loc.getY();
            energy[x][y] = pi.getCount( me );

            if( energy[x][y] >= 1.0 ) {
                energy[x][y] -= 1;
            }

            if( is_explored[x][y] ) {
                continue;
            }

//            System.out.println( loc.toString() + " explored, total=" +
String.valueOf( fields_explored ) );
```

```
            is_explored[x][y] = true;
            fields_explored++;

            if( fields_explored == max_x * max_y ) {
                all_explored = true;
            }

            float growth_rate = pi.getGrowthRate( me );
            float max_energy = pi.getMaxCount( me );

            if( growth_rate > 0 ) {
                is_source[x][y] = true;
                if( growth_rate >= ( Constants.BORN_BONUS_GROWTH * 0.8 ) ) {
                    is_golden[x][y] = true;
                    golden.add( pi.getLocation() );
                    System.out.println( "golden found!" + "x =" +
                            String.valueOf( x ) + " y=" + String.valueOf( y ) +
                            " now golden = " + String.valueOf( this.golden.size() +
                            " growth_rate=" + String.valueOf( growth_rate )) );
                } else if( growth_rate >= ( Constants.GROW_REGULAR * 2 ) ) {
                    is_medium[x][y] = true;
                    medium.add( pi.getLocation() );
                    System.out.println( "medium found!" + "x =" +
                            String.valueOf( x ) + " y=" + String.valueOf( y ) +
                            " now medium = " + String.valueOf( this.medium.size() +
                            " growth_rate=" + String.valueOf( growth_rate )) );
                } else {
                    is_simple[x][y] = true;
                    simple_source.add( pi.getLocation() );
                }
            }

        }
    }

    private synchronized int howManyNonExploredNear( int x0, int y0 ) {
        int len = 1;
        int x, y;
        int res = 0;
        for( int i = -len; i <= len; i++  ) {
            for( int j = -len; j <= len; j++ ) {
                x = ( max_x + x0 + i ) % max_x;
                y = ( max_y + y0 + j ) % max_y;
                if( ! is_explored[x][y] ) {
                    res++;
                }
            }
        }
        return res;
    }

    public synchronized Location getReachableNonExplored( Location loc, BeingInterface bi,
Yeti_2 yeti ) {
        List <Location> all[] = new ArrayList[10];
        int len0 = ( int )( yeti.S );

        for( int i = 0; i < 10; i++ ) {
            all[i] = new ArrayList();
        }

        if( all_explored ) {
            return null;
        }

        int x0 = loc.getX();
        int y0 = loc.getY();

        for( int len = len0; len > 0; len-- ) {
            for( int j = -len; j <= len; j++ ) {
                for( int i = -len; i <=len; i++ ) {
                    int x = ( max_x + x0 + i ) % max_x;
```

```
                    int y = ( max_y + y0 + j ) % max_y;
//                    System.out.println( "non_explored = " +  howManyNonExploredNear( x,
y ) );
                    all[ howManyNonExploredNear( x, y ) ].add( bi.createLocation( x, y ) );
                }
            }
        }

        for( int i = 9; i > 0; i-- ) {
            if( all[i].size() > 0 ) {
                int idx = Util.rnd( all[i].size() );
                return all[i].get( idx );
            }
        }

        return null;
    }

    public synchronized Location getNearestNoneExplored( Location loc, BeingInterface bi,
Yeti_2 yeti ) {

        List <Location> all[] = new ArrayList[9];
        int len0 = ( int )( yeti.S );

        if( all_explored ) {
            return null;
        }

        int x0 = loc.getX();
        int y0 = loc.getY();

        for( int len = len0 + 1; len <= ( max_x / 2) + 1; len++ ) {
            for( int j = -len; j <= len; j++ ) {

                int x = ( max_x + x0 + len ) % max_x;
                int y = ( max_y + y0 + j ) % max_y;

                if( ! is_explored[ x ][ y ] )
                    return bi.createLocation( x, y );

                x = ( max_x + x0 - len ) % max_x;
                y = ( max_y + y0 + j ) % max_y;
                if( ! is_explored[ x ][ y ] )
                    return bi.createLocation( x, y );

                x = ( max_x + x0 + j ) % max_x;
                y = ( max_y + y0 + len ) % max_y;
                if( ! is_explored[ x ][ y ] )
                    return bi.createLocation( x, y );

                x = ( max_x + x0 + j ) % max_x;
                y = ( max_y + y0 - len ) % max_y;
                if( ! is_explored[ x ][ y ] )
                    return bi.createLocation( x, y );

            }
        }

        all_explored = true;

        return null;
    }


    public Location getNearestSimpleSource( BeingInterface bi, Entity me ) {
        return getNearest( bi.getLocation( me ), simple_source, null, bi );
    }

    public Location getNearestSimpleSource( BeingInterface bi, Entity me, List<Location>
excluding ) {
```

```
        return getNearest( bi.getLocation( me ), simple_source, excluding, bi );
    }

    public Location getNearestGolden( BeingInterface bi, Entity me ) {
        return getNearest( bi.getLocation( me ), golden, null, bi );
    }

    public Location getNearestGolden( BeingInterface bi, Entity me, List<Location> excluding )
{
        return getNearest( bi.getLocation( me ), golden, excluding, bi );
    }

    public Location getNearestMedium( BeingInterface bi, Entity me ) {
        return getNearest( bi.getLocation( me ), medium, null, bi );
    }

    public Location getNearestMedium( BeingInterface bi, Entity me, List<Location> excluding )
{
        return getNearest( bi.getLocation( me ), medium, excluding, bi );
    }



    public synchronized Location getNearest( Location my_loc, List<Location> from,
List<Location> excluding, BeingInterface bi ) {
        List<Location> list = new ArrayList();

        for( Location elem : from ) {
            list.add( elem );
        }

        if( excluding != null ) {
            list.removeAll( excluding );
        }
        Location curr_loc = null;
        float curr_distance = 10000;

        for( Location loc: list ) {
            float distance = curr_distance;

            if( my_loc != null ) {
                distance = bi.distance( my_loc, loc );
            }


            if( distance < curr_distance ) {
                curr_distance = distance;
                curr_loc = loc;
            }
        }
        return curr_loc;
    }

}
```

## *6. Файл* LocationComparator.java. Метрика игрового поля.

```
/*
 * LocationComparator.java
 *
 * Created on 15 Январь 2007 г., 17:17
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package universum.beings.Yeties;
```

```
import java.util.Comparator;

import universum.bi.*;
import universum.util.*;

/**
 *
 * @author kalugin
 */
public class LocationComparator implements Comparator<Location> {

    private BeingInterface bi;
    private Location loc0;

    /** Creates a new instance of LocationComparator */
    public LocationComparator( BeingInterface bi, Location loc0 ) {
        this.bi = bi;
        this.loc0 = loc0;
    }

    public int compare( Location loc1, Location loc2 ) {
        return (int)( bi.distance( loc0, loc1 ) - bi.distance( loc0, loc2 ) );
    }

}
```

## *7. Файл* `BeingKind.java`. Информация о текущем статусе (стражник, исследователь, воин) существа.

```
/*
 * BeingKind.java
 *
 * Created on 17 Январь 2007 г., 9:33
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package universum.beings.Yeties;

import universum.bi.*;
import universum.util.*;

/**
 *
 * @author kalugin
 */
public class BeingKind {

    public enum CurrentType {
        INITIAL,
        EXPLORER,
        WARRIOR,
        GUARDIAN
    };

    private CurrentType current = CurrentType.INITIAL;

    // explorer data
    private Task task;

    // guardian data
    private Location to_guard;

    // warrior data
    private Location target_point;
    private Location assembly_point;
    private boolean attack_flag = false;
```

```
    private boolean victory_flag = false;

    /** Creates a new instance of BeingKind */
    public BeingKind(  ) {
        current = CurrentType.INITIAL;
    }

    public BeingKind( CurrentType ct ) {
        current = ct;
    }

    public CurrentType getCurrent() {
        return current;
    }

    public void switchTo( CurrentType new_type ) {
        current = new_type;
    }

    public void setAttackFlag( boolean af ) {
        this.attack_flag = af;
    }

    public boolean getAttackFlag() {
        return this.attack_flag;
    }

    public void setVictoryFlag( boolean flag ) {
        victory_flag = flag;
    }

    public boolean getVictoryFlag() {
        return victory_flag;
    }

    public void setTask( Task task_ ) {
        this.task = task_;
    }

    public Task getTask() {
        return this.task;
    }

    public void setGuardObject( Location to_guard_ ) {
        this.to_guard = to_guard_;
    }

    public Location getGuardObject() {
        return this.to_guard;
    }

    public void setTargetPoint( Location tp ) {
        target_point = tp;
    }

    public Location getTargetPoint() {
        return target_point;
    }

    public void setAssemplyPoint( Location ap ) {
        assembly_point = ap;
    }

    public Location getAssemplyPoint() {
        return assembly_point;
    }

}
```

## *8. Файл* Operation.java. Класс для обеспечения эффективности синхронной атаки.

```
/*
 * Operation.java
 *
 * Created on 25 Январь 2007 г., 19:26
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package universum.beings.Yeties;

/**
 *
 * @author kalugin
 */
public class Operation {

    public enum OperationKind {
        GIVE_ENERGY,
        ATTACK
    }

    private Integer animal;
    private float value;

    private OperationKind operation_kind;

    /** Creates a new instance of Operation */
    public Operation( OperationKind op ) {
        operation_kind = op;
    }

    public void attackOperation( Integer victim, float damage ) {
        this.animal = victim;
        this.value = damage;
    }

    public void energyGrantOperation( Integer friend, float energy ) {
        this.animal = friend;
        this.value = energy;
    }

    public float getValue() {
        return value;
    }

    public Integer getAnimal() {
        return animal;
    }

    public OperationKind getOperationKind() {
        return operation_kind;
    }
}
```

## *9. Файл* MyPointInfo.java. Вспомогательная информация о точках на игровой карте.

```
/*
 * MyPointInfo.java
 *
 * Created on 26 Январь 2007 г., 10:46
 *
```

```
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package universum.beings.Yeties;


import universum.bi.*;
import universum.util.*;
import java.util.List;
import java.util.ArrayList;

/**
 *
 * @author kalugin
 */
public class MyPointInfo {

    private class EnemyInfo {
        public Integer id;
        public float mass;
        public EnemyInfo( Integer id, float mass ) {
            this.id = id;
            this.mass = mass;
        }
    }

    List<EnemyInfo> enemies;
    float total_mass;

    /** Creates a new instance of MyPointInfo */
    public MyPointInfo() {
        enemies = new ArrayList();
        total_mass = 0;
    }

    public synchronized void add( Integer id, float mass ) {
        EnemyInfo enemy = new EnemyInfo( id, mass );
        enemies.add( enemy );
        total_mass += mass;
    }

    public float getEnemyMass( Integer id ) {
        for( EnemyInfo info : enemies ) {
            if( info.id == id ) {
                return info.mass;
            }
        }
        return 0;
    }

    public float getEnemyTotalMass() {
        return total_mass;
    }


    public List<Integer> getEnemyList() {
        List<Integer> list = new ArrayList();

        for( EnemyInfo info : enemies ) {
            list.add( info.id );
        }
        return list;
    }

}
```

## *10. Файл* `WarriorDetachment.java`. Боевой отряд.

```
/*
 * WarriorDetachment.java
 *
 * Created on 18 Январь 2007 г., 15:20
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package universum.beings.Yeties;



import universum.bi.*;
import universum.util.*;
import java.util.List;
import java.util.ArrayList;

/**
 *
 * @author kalugin
 */
public class WarriorDetachment {


    public enum Status {
        INIT,
        NEED_MORE_POWER,
        PREPARE,
        SCOUTING,
        REQUEST_MORE_POWER,
        ATTACKING,
        WIN,
        LOSE
    }

    public enum Events {
        e0_enough_power,
        e1_everyone_assembled,
        e2_no_chance,
        e3_power_granted,
        e4_no_one_left,
        e5_victory,
        e6_not_scouted,
        no_events
    }


    private Status status;
    private Events event;

    private List<Yeti_2> detachment;

    private Yeti_2 scout;

    private Location assembly_point;
    public Location attack_point;

    private Field field;

    private int scout_turn;

    private int start_turn;

    private final int MAX_ASSEMBLY_TURNS = 50;

    private final float ATTACK_COEF_MASS = 2f;
```

```
    private final float ATTACK_COEF_ENERGY = 1.5f;

    private int enemy_power;

    private boolean force_attack;

    /** Creates a new instance of WarriorDetachment */
    public WarriorDetachment( Location attack, Field fld, BeingInterface bi ) {
        detachment = new ArrayList();
//        this.assembly_point = assembly;
        this.attack_point = attack;
        this.field = fld;
        this.assembly_point = field.selectAssemblyPoint( attack, bi );
        this.status = Status.INIT;
        this.event = Events.no_events;
        this.enemy_power = (int)field.getTotalEnemyMass( attack );
        this.start_turn = bi.getTurnsCount();
        this.scout_turn = 0;
        this.force_attack = false;
        this.enemy_power = 0;
    }

    public List<Yeti_2> getWarriorList() {
        return detachment;
    }

    public synchronized void nextTurn( BeingInterface bi ) {

        if( detachment.size() == 0 ) {
            System.out.println( "removing zombie detachment from state " + status.toString() +
attack_point.toString() );
            status = Status.LOSE;
        }

        // Finite-State Machine. Автомат "Боевой отряд"

        switch( status ) {
            case INIT:
                status = Status.NEED_MORE_POWER;
                break;

            case NEED_MORE_POWER:

                z0_check_my_power( bi );
                if( event == Events.e0_enough_power ) {
                    status = Status.PREPARE;
                }
                break;

            case PREPARE:
                z1_check_assembled( bi );
                if( event == Events.e1_everyone_assembled ) {
                    z2_send_scout( bi );
                    status = Status.SCOUTING;
                }
                break;

            case SCOUTING:
                z3_check_enemy_forces( bi );
                if( event == Events.e6_not_scouted ) {
                    break;
                }
                if( event == Events.e2_no_chance ) {
//                      status = Status.REQUEST_MORE_POWER;       !!!!!!!!!!!!!!!
//                      z4_start_attack( bi );
                    status = Status.NEED_MORE_POWER;
                    break;
                }
                if( event == Events.e0_enough_power ) {
                    status = Status.ATTACKING;
                    z4_start_attack( bi );
```

```
                    break;
                }
                break;

            case REQUEST_MORE_POWER:
                break;

            case ATTACKING:
//                System.out.println( "Attacking!!!!" + attack_point.toString() );
                z5_battle_control( bi );
                if( event == Events.e5_victory ) {
                    status = Status.WIN;
                    System.out.println( "We win!!!!" );
                    break;
                }
                if( event == Events.e4_no_one_left ) {
                    status = Status.LOSE;
                    System.out.println( "We loose!!!!" );
                    break;
                }
                break;

            case WIN:
                break;

            case LOSE:
                break;
        }
//        System.out.println( status.toString() + attack_point.toString() + " my_size=" +
detachment.size() );
//        if( detachment.size() == 0 ) {
//            System.out.println( "I'm zombie!!!" + status.toString() + attack_point.toString()
+ " my_size=" + detachment.size() );
//        } else {
//            bi.log( detachment.get(0), status.toString() + attack_point.toString() + "
my_size=" + detachment.size() );
//        }
    }

    public synchronized Status getCurrentStatus() {
        return status;
    }

    public void forceAttack( boolean flag ) {
        this.force_attack = flag;
    }

    public synchronized void z0_check_my_power( BeingInterface bi ) {
        float my_total_mass = 0;
        float my_total_energy = 0;
        for( Yeti_2 yeti : detachment ) {
            my_total_mass += yeti.M;
            my_total_energy += bi.getEnergy( yeti );
        }

        if( ( my_total_energy > ATTACK_COEF_ENERGY * enemy_power ) ||
                ( my_total_mass > ATTACK_COEF_MASS * enemy_power ) ||
              ( bi.getTurnsCount() > start_turn + MAX_ASSEMBLY_TURNS ) ||
                force_attack ) {
            event = Events.e0_enough_power;
        }
    }

    public synchronized void z1_check_assembled( BeingInterface bi ) {

        if ( bi.getTurnsCount() > start_turn + MAX_ASSEMBLY_TURNS ) {
            event = Events.e1_everyone_assembled;
            return;
        }

        for( Yeti_2 yeti : detachment ) {
```

```
        if( bi.getLocation( yeti ) != assembly_point ) {
            return;
        }
    }
    event = Events.e1_everyone_assembled;
}

public synchronized void z2_send_scout( BeingInterface bi ) {
    if( detachment.size() == 0 ) {
        event = Events.e2_no_chance;
        return;
    }

    Yeti_2 curr_scout = detachment.get( 0 );

    for( Yeti_2 yeti : detachment ) {
        if( yeti.M < curr_scout.M ) {
            curr_scout = yeti;
        }
    }

    curr_scout.getBeingKind().setAttackFlag( true );
    scout = curr_scout;

    scout_turn = bi.getTurnsCount();
}

public synchronized void z3_check_enemy_forces( BeingInterface bi ) {

    if( bi.getTurnsCount() < scout_turn + 2 ) {
        event = Events.e6_not_scouted;
        return;
    }

    enemy_power = (int) field.getTotalEnemyMass( attack_point );

    float my_total_mass = 0;
    float my_total_energy = 0;

    for( Yeti_2 yeti : detachment ) {
        my_total_mass += yeti.M;
        my_total_energy += bi.getEnergy( yeti );
    }

    if( ( my_total_energy > enemy_power * 0.4 ) || ( my_total_mass > enemy_power ) ||
         ( bi.getTurnsCount() > start_turn + MAX_ASSEMBLY_TURNS ) || force_attack ) {
        event = Events.e0_enough_power;
    } else {
        event = Events.e2_no_chance;
    }

}

public synchronized void z4_start_attack( BeingInterface bi ) {
    for( Yeti_2 yeti : detachment ) {
        yeti.getBeingKind().setAttackFlag( true );
    }
}

public synchronized void z5_battle_control( BeingInterface bi ) {
    if( detachment.size() == 0 ) {
        event = Events.e4_no_one_left;
        return;
    }

    for( Yeti_2 yeti : detachment ) {
        if( yeti.getBeingKind().getVictoryFlag() ) {
            event = Events.e5_victory;
            yeti.getBeingKind().setVictoryFlag( false );
            return;
        }
```

```
        }
    }


    public synchronized void addNewWarrior( Yeti_2 yeti ) {
        BeingKind bk = new BeingKind();

        bk.switchTo( BeingKind.CurrentType.WARRIOR );

        bk.setAssemplyPoint( this.assembly_point );
        bk.setTargetPoint( this.attack_point );

        yeti.upgradeBeingKind( bk );

        detachment.add( yeti );
    }

    public synchronized void deleteWarrior( Yeti_2 yeti ) {
        detachment.remove( yeti );
    }


    public synchronized boolean isMyWarrior( Yeti_2 yeti ) {
        if( detachment.contains( yeti ) ) {
            return true;
        }
        return false;
    }

}
```

# Приложение 2. Лог работы программы

```
Yeti_2[1813026547]: EXPLORER[117,4]: переход из состояния НАЧАЛО в ДЕЛЕНИЕ
Yeti_2[1813026547]: EXPLORER[117,4]: событие e1, остаёмся в состоянии ДЕЛЕНИЕ
Yeti_2[1813026547]: EXPLORER[117,4]: действие z0
Yeti_2[1557076875]: GUARDIAN[117,4]: переход из состояния НАЧАЛО в ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[1813026547]: EXPLORER[117,4]: [else], переход из состояния ДЕЛЕНИЕ в ПОИСК ПИЩИ
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[1557076875]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[1813026547]: EXPLORER[117,4]: событие e0, переход из состояния ПОИСК ПИЩИ в ЕДА
Yeti_2[1813026547]: EXPLORER[117,4]: переход из состояния ЕДА в ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: действие z3
Yeti_2[1813026547]: EXPLORER[117,4]: действие z3
Yeti_2[1813026547]: EXPLORER[117,4]: [else], переход из состояния ДЕЛЕНИЕ в ПОИСК ПИЩИ
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[1813026547]: EXPLORER[117,4]: событие e0, переход из состояния ПОИСК ПИЩИ в ЕДА
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[1813026547]: EXPLORER[117,4]: переход из состояния ЕДА в ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[1813026547]: EXPLORER[117,4]: действие z3
Yeti_2[1557076875]: GUARDIAN[117,4]: действие z3
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e0, остаёмся в состоянии ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: действие z0
Yeti_2[1813026547]: EXPLORER[117,4]: [else], переход из состояния ДЕЛЕНИЕ в ПОИСК ПИЩИ
Yeti_2[1813026547]: EXPLORER[117,4]: событие e0, переход из состояния ПОИСК ПИЩИ в ЕДА
Yeti_2[1813026547]: EXPLORER[117,4]: переход из состояния ЕДА в ДЕЛЕНИЕ
Yeti_2[1813026547]: EXPLORER[117,4]: действие z3
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[-1236818306]: GUARDIAN[117,4]: переход из состояния НАЧАЛО в ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[1557076875]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: действие z3
```

```
Yeti_2[-1236818306]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[1813026547]: EXPLORER[117,4]: [else], переход из состояния ДЕЛЕНИЕ в ПОИСК ПИЩИ
Yeti_2[-1236818306]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[1813026547]: EXPLORER[117,4]: событие e0, переход из состояния ПОИСК ПИЩИ в ЕДА
Yeti_2[-1236818306]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[1813026547]: EXPLORER[117,4]: переход из состояния ЕДА в ДЕЛЕНИЕ
Yeti_2[-1236818306]: GUARDIAN[117,4]: действие z3
Yeti_2[1813026547]: EXPLORER[117,4]: действие z3
Yeti_2[-1236818306]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[1813026547]: EXPLORER[117,4]: [else], переход из состояния ДЕЛЕНИЕ в ПОИСК ПИЩИ
Yeti_2[1813026547]: EXPLORER[117,4]: событие e0, переход из состояния ПОИСК ПИЩИ в ЕДА
Yeti_2[1813026547]: EXPLORER[117,4]: переход из состояния ЕДА в ДЕЛЕНИЕ
Yeti_2[-1236818306]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[1813026547]: EXPLORER[117,4]: действие z3
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[-1236818306]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[-1236818306]: GUARDIAN[117,4]: действие z3
Yeti_2[1557076875]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: действие z3
Yeti_2[1813026547]: EXPLORER[117,4]: [else], переход из состояния ДЕЛЕНИЕ в ПОИСК ПИЩИ
Yeti_2[1813026547]: EXPLORER[117,4]: событие e0, переход из состояния ПОИСК ПИЩИ в ЕДА
Yeti_2[1813026547]: EXPLORER[117,4]: переход из состояния ЕДА в ДЕЛЕНИЕ
Yeti_2[1813026547]: EXPLORER[117,4]: действие z3
Yeti_2[-1236818306]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[-1236818306]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[-1236818306]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[-1236818306]: GUARDIAN[117,4]: действие z3
Yeti_2[1557076875]: GUARDIAN[117,4]: действие z3
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[1557076875]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: действие z3
Yeti_2[-1236818306]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[1813026547]: EXPLORER[117,4]: событие e1, остаёмся в состоянии ДЕЛЕНИЕ
Yeti_2[1813026547]: EXPLORER[117,4]: действие z0
Yeti_2[-1236818306]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[-1236818306]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[-1236818306]: GUARDIAN[117,4]: действие z3
Yeti_2[1104831225]: EXPLORER[117,4]: переход из состояния НАЧАЛО в ДЕЛЕНИЕ
Yeti_2[-1236818306]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[-1236818306]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[-1236818306]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[-1236818306]: GUARDIAN[117,4]: действие z3
Yeti_2[1813026547]: EXPLORER[117,4]: [else], переход из состояния ДЕЛЕНИЕ в ПОИСК ПИЩИ
Yeti_2[1813026547]: EXPLORER[117,4]: событие e0, переход из состояния ПОИСК ПИЩИ в ЕДА
Yeti_2[1813026547]: EXPLORER[117,4]: переход из состояния ЕДА в ДЕЛЕНИЕ
Yeti_2[1813026547]: EXPLORER[117,4]: действие z3
Yeti_2[1104831225]: EXPLORER[117,4]: [else], переход из состояния ДЕЛЕНИЕ в ПОИСК ПИЩИ
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[1104831225]: EXPLORER[117,4]: событие e0, переход из состояния ПОИСК ПИЩИ в ЕДА
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[1104831225]: EXPLORER[117,4]: переход из состояния ЕДА в ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[1104831225]: EXPLORER[117,4]: действие z3
Yeti_2[1557076875]: GUARDIAN[117,4]: действие z3
Yeti_2[1104831225]: EXPLORER[117,4]: [else], переход из состояния ДЕЛЕНИЕ в ПОИСК ПИЩИ
Yeti_2[1104831225]: EXPLORER[117,4]: событие e0, переход из состояния ПОИСК ПИЩИ в ЕДА
Yeti_2[1813026547]: EXPLORER[117,4]: [else], переход из состояния ДЕЛЕНИЕ в ПОИСК ПИЩИ
```

```
Yeti_2[1813026547]: EXPLORER[117,4]: событие e0, переход из состояния ПОИСК ПИЩИ в ЕДА
Yeti_2[1813026547]: EXPLORER[117,4]: переход из состояния ЕДА в ДЕЛЕНИЕ
Yeti_2[1813026547]: EXPLORER[117,4]: действие z3
Yeti_2[-1236818306]: GUARDIAN[117,4]: событие e0, остаёмся в состоянии ДЕЛЕНИЕ
Yeti_2[-1236818306]: GUARDIAN[117,4]: действие z0
Yeti_2[1104831225]: EXPLORER[117,4]: переход из состояния ЕДА в ДЕЛЕНИЕ
Yeti_2[1104831225]: EXPLORER[117,4]: действие z3
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e1, переход из состояния ДЕЛЕНИЕ в ОХРАНА
ИСТОЧНИКА
Yeti_2[1557076875]: GUARDIAN[117,4]: событие e2, переход из состояния ОХРАНА ИСТОЧНИКА в ЕДА
Yeti_2[1557076875]: GUARDIAN[117,4]: переход из состояния ЕДА В ДЕЛЕНИЕ
Yeti_2[1557076875]: GUARDIAN[117,4]: действие z3
```