Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Факультет информационных технологий и программирования

Кафедра компьютерных технологий

И. С. Гунич, А. В. Иринев, А. А. Шалыто

Автоматный подход к моделированию эволюции микроорганизмов с использованием генетических алгоритмов

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru/

Санкт-Петербург
2007

ВВЕДЕНИЕ 3

1. ПОСТАНОВКА ЗАДАЧИ 4

1.1 Набор генов 4

1.2 Энергия и количество сделанных шагов 4

1.3 Координаты 5

2. АВТОМАТНЫЙ ПОДХОД 6

3. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ 7

3.1 Модель ScreenModel 7
3.1.1. Диаграмма связей 7
3.1.2. Поставщик событий *ScreenEventProvider* 7
3.1.3. Объект управления *ControlPanel* 8
3.1.4. Описание автомата 9

3.2 Модель BugModel 10
3.2.1. Диаграммы связей 10
3.2.2. Поставщик событий *EntityEventProvider* 10
3.2.3. Объект управления *Bug* 12
3.2.4. Описание автомата 14

4. ОБЩИЙ ИНТЕРФЕЙС ПРИЛОЖЕНИЯ 15

ЗАКЛЮЧЕНИЕ 18

ЛИТЕРАТУРА 20

ПРИЛОЖЕНИЕ 1. СТРУКТУРА КЛАССОВ 21

ПРИЛОЖЕНИЕ 2. ИСХОДНЫЙ КОД 16

Введение

Ключевую роль в эволюционной теории играет естественный отбор. Его суть состоит в том, что наиболее приспособленные особи лучше выживают и приносят больше потомства, а благодаря генетическому наследованию [1] часть потомков не только сохраняет высокую приспособленность родителей, но будет иметь и новые свойства. Если эти свойства оказываются полезными, то с большой вероятностью они перейдут в следующее поколение. Таким образом, происходит накопление полезных качеств и постепенное повышение приспособленности биологического вида в целом.

Цель данного проекта – моделирование абстрактного микромира, населенного жизненными формами с простейшими правилами поведения, эволюция которых достигается использованием генетического алгоритма и автоматного подхода [2]. Идея самого алгоритма взята из оригинальной статьи А. К. Дьюдни [3].

Для моделирования поведения микроорганизмов используется автоматный подход. Эта технология поддерживается инструментальным средством *UniMod* [4,5]. Технология автоматного программирования называется также *SWITCH*-технологией [6].

SWITCH-технология определяет для каждого автомата два типа диаграмм (схему связей и граф переходов) и их операционную семантику. При наличии нескольких автоматов также строится схема их взаимодействия. *SWITCH*-технология задает свою нотацию диаграмм.

В инструментальном средстве *UniMod* сохранен автоматный подход, однако для построения диаграмм используется *UML*-нотация. Применяя нотацию диаграмм классов языка *UML*, строятся схемы связей автоматов, каждая из которых определяет интерфейс этих автоматов. При этом графы переходов, описывающие поведение автоматов, строятся с помощью нотации диаграммы состояний *UML*.

1. Постановка задачи

В данном проекте биологической системой является колония микроорганизмов, размещенных на поле размером 50x50. Каждая бактерия имеет набор характерных параметров: хромосома, состоящая из восьми генов, количество энергии, число сделанных к данному моменту шагов и координаты на двумерном поле.

Опишем подробнее каждый из перечисленных параметров.

1.1. Набор генов

Каждый микроорганизм имеет набор из восьми генов, отвечающих за направления его движения (вверх, вверх-вправо, вправо, вниз-вправо, вниз, вниз-влево, влево, вверх-влево). Чем больше значение гена для данного направления у конкретной бактерии, тем больше вероятность того, что она будет двигаться в этом направлении. Вероятностная модель распределения генов подобрана таким образом, что сумма значений генов у каждой бактерии равна двадцати четырем. Другими словами, максимально возможное значение какого-либо гена равно этой величине. При этом бактерия может двигаться только в направлении, соответствующему данному гену. Состояние, в котором бактерия с равной вероятностью выберет любое из восьми направлений движения, отвечает набору генов, каждый из которых равен трем. Бактерии питаются органической едой, которая непрерывно оседает на поле, подчиняясь некоторой закономерности, примеры которой приведены ниже. Бактерия за каждую съеденную единицу еды получает восемь единиц энергии. Поэтому характер движения играет главную роль в поставленной задаче, так как от этого зависит сколько проживет бактерия, и сможет ли она дожить до момента размножения, а, следовательно, произвести потомство.

1.2. Энергия и количество сделанных шагов

Данные параметры используются для определения возможности деления. Бактерия может делиться только в том случае, когда каждый из этих параметров достигает своего порога (в рассматриваемой задаче значение порога энергии принято равным восьмидесяти единицам, а количество сделанных шагов – двадцати). Порог энергии используется для того, чтобы бактерия имела некий период созревания. Это исключает возможность деления только что появившейся бактерии. При делении каждый из двух потомков получает половину энергии родителя. Если значение энергии для данной особи опускается до нуля, то бактерия погибает. Если значение энергии достигает порога, при котором возможно деление, но бактерия еще не может делиться, так как не достигнут «период созревания», то счетчик энергии будет увеличиваться только до определенного предела (значение энергии принято равным ста единицам). Данное ограничение введено для того, чтобы не было слиш-

ком «энергичных» микроорганизмов, потомство которых уже сразу после деления родителя имеет достаточное для деления количество энергии.

1.3. Координаты

Координаты представляют собой два целочисленных значения, соответствующих расположению бактерии на двумерном поле. Также на этом поле находится еда, расположение которой можно изменять настройкой конфигурационных параметров. Еда, например, может располагаться, так что одна четверть поля полностью заполнена едой или так, что она размещена в виде продольная полоски, проходящей через центр поля. Границы поля будем считать склеенными. Например, если бактерия выходит за верхнюю границу поля, то она появляется снизу.

2. Автоматный подход

Как отмечалось выше, для моделирования поведения микроорганизмов используется автоматный подход.

Предлагаемый процесс моделирования системы состоит в следующем:

- на основе анализа предметной области разрабатывается концептуальная модель системы, определяющая сущности и отношения между ними;
- в отличие от традиционных для объектно-ориентированного программирования подходов, из числа сущностей выделяются источники событий, объекты управления и автоматы. Источники событий активны – они по собственной инициативе воздействуют на автоматы. Объекты управления пассивны – они выполняют действия по команде от автоматов. Объекты управления также формируют значения входных переменных для автоматов. Каждый автомат активизируется источниками событий и на основании значений входных переменных и текущего состояния воздействует на объекты управления, переходя в новое состояние;
- используя нотацию диаграмм классов, строится схема связей автомата, задающая его интерфейс. На этой схеме, как в теории автоматического управления, слева отображаются источники событий, в центре — автоматы, а справа — объекты управления. Источники событий с помощью *UML*-ассоциаций связываются с автоматами, события которым они поставляют. Автоматы, в свою очередь, связываются с объектами, которыми они управляют;
- каждый объект управления содержит два типа методов, реализующих входные переменные и выходные воздействия;
- для каждого автомата с помощью нотации диаграммы состояний строится граф переходов, в котором дуги могут быть помечены событием, булевой формулой из входных переменных и формируемыми на переходах выходными воздействиями. В вершинах могут указываться выходные воздействия и имена вложенных автоматов;
- события, входные переменные и выходные воздействия являются методами соответствующего объекта управления, которые реализуются вручную на целевом языке программирования. Для реализации этих методов могут использоваться вспомогательные классы, не указанные на схемах связей;
- упрощение понимания смысла символических обозначений обеспечивается за счет всплывающих подсказок, появляющихся при наведении курсора на соответствующий символ на графе переходов.

Несложная автоматная модель, используемая в нашем проекте, позволяет легко описывать достаточно сложное поведение системы. Инструментальное средство *UniMod*, в свою очередь, позволяет наиболее эффективно реализовать данную автоматную модель.

3. Проектирование и реализация

3.1. Модель ScreenModel

UniMod-модель *ScreenModel* реализует управление графическим интерфейсом пользователя (GUI).

3.1.1. Диаграмма связей

В отличие от *SWITCH*-технологии, при использовании инструментального средства *UniMod* схема связей является не картинкой, а диаграммой классов *UML*, изображенной нетрадиционным путем, поскольку ее ориентация не сверху вниз, а слева направо.

На рис.1 изображена схема связей для поставщика событий и управляемого объекта (ниже перечисленные объекты будут рассмотрены более детально).

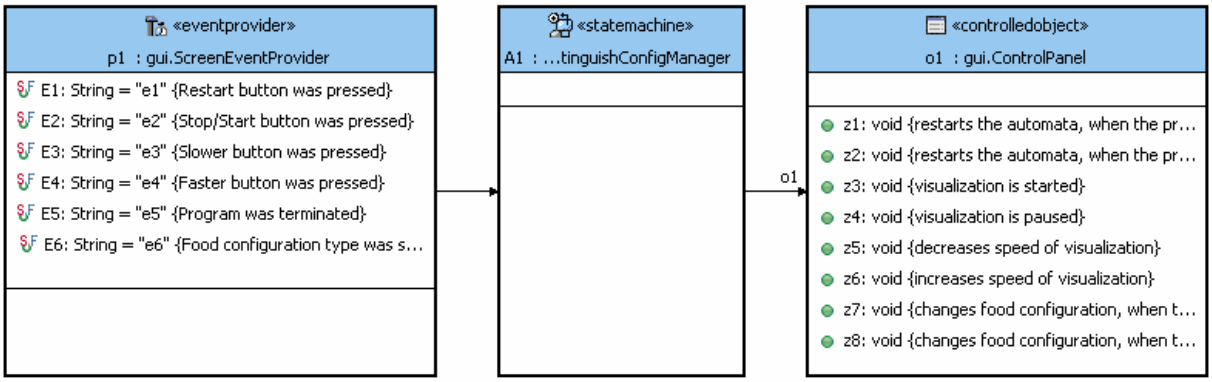

Рис. 1. Диаграмма связей *ScreenModel*

3.1.2. Поставщик событий *ScreenEventProvider*

Этот поставщик используется для передачи событий воздействия пользователя описываемому автомату, управляющему интерфейсом данной программы (выбор каких либо пунктов меню и нажатия кнопок). Описание событий данного объекта представлено в табл. 1.

Таблица 1. События поставщика событий *ScreenEventProvider*

№	Событие	Описание	Комментарий
1.	e1	Нажата кнопка «Restart»	_
2.	e2	Нажата кнопка «Stop/Start»	Событие приостановки/ возобновления визуализации
3.	e3	Нажата кнопка «<<»	Событие уменьшения скорости визуализации

4.	e4	Нажата кнопка «>>»	Событие увеличение скорости визуализации
5.	e5	Закрытие окна	Событие завершения моделирования
6.	e6	Выбрана модель расположения еды на поле	–

3.1.3. Объект управления *ControlPanel*

Объект *ControlPanel* содержит выходные воздействия, связанные с изменением внешнего вида головного окна приложения. Выходные воздействия объекта управления представлены в табл. 2.

Таблица 2. Выходные воздействия объекта управления *ControlPanel*

№	Выходное воздействие	Описание	Комментарий
1.	z1	Перезапустить автомат	Автомат находится в Состоянии «Визуализация»
2.	z2	Перезапустить автомат	Автомат находится в состоянии «Визуализация приостановлена»
3.	z3	Приостановить визуализацию	Устанавливается надпись на кнопке «Start/Stop» в «Start»
4.	z4	Возобновить визуализацию	Устанавливается надпись на кнопке «Start/Stop» в «Stop»
5.	z5	Уменьшить скорость визуализации	Шаг уменьшения и увеличения скорости, задан в статическом классе `Config`
6.	z6	Уменьшить скорость визуализации	См. комментарий к п. 5
7.	z7	Изменение расположения еды на поле и перезапуск автомата	Автомат находится в состоянии «Визуализация»
8.	z8	Изменение расположения еды на поле и перезапуск автомата	Автомат находится в состоянии «Визуализация приостановлена»

3.1.4. Описание автомата

В данном разделе рассмотрен автомат, описанный при помощи нотации диаграмм состояний *UML* (рис. 2).

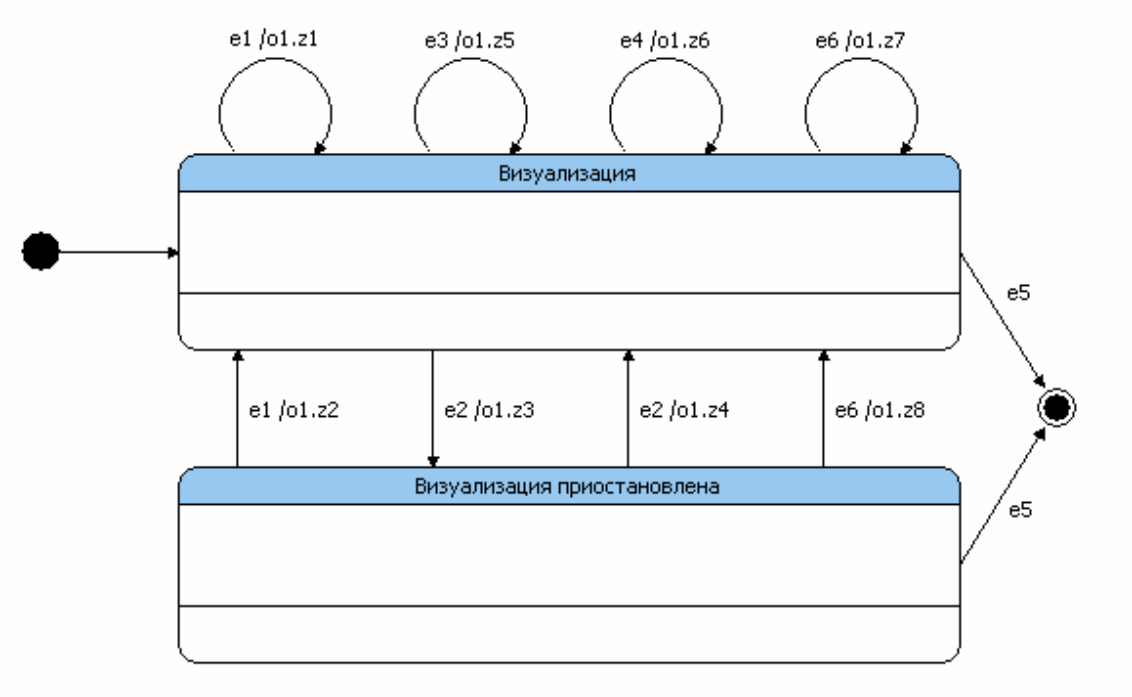

Рис. 2. Диаграмма состояний автомата управляющего GUI (*A1*)

Начальное состояние обозначено черным кругом, а конечное – двойным кругом с закрашенной серединой. Состояния обозначены прямоугольниками со скругленными углами, а переходы стрелками, рядом с которыми написано условие перехода. Общая форма условия перехода:

e#[логическое выражение]/список выходных воздействий

(вместо символа # стоит некоторое число).

Логическое выражение может состоять из:

- входных переменных в нотации *o#.x#* (здесь, как и выше, # означает любую цифру);
- логические операции *&&* (и), || (или), *!* (не);
- операции сравнения > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), == (равно), *!=* (не равно);
- скобок *(,)*.

Допустим, автомат находится в каком-либо состоянии. Если поступило событие, то для всех переходов по этому событию проверяется логическое условие (если оно присутствует). Если найден переход, для которого логическое условие истинно, то сначала выполняются все выходные воздействия, указанные на переходе символа /, а затем – выходные воздействия, предусмотренные для исполнения в момент входа в состояние (список таких воз-

действий приводится в нижней части прямоугольника-состояния после слов «enter /»). В том случае, такой переход отсутствует, автомат остается в прежнем состоянии. При проектировании графа переходов с помощью средства *UniMod* выполняется автоматическая проверка условий на полноту и непротиворечивость.

Как упоминалось ранее, данный автомат отвечает за управление пользовательским интерфейсом. Моделирование поведения микроорганизмов продолжается только в случае, когда автомат находится в состоянии *«Визуализация»*. Переход из состояния «*Визуализация*» в состояние «*Визуализация приостановлена*» происходит по событию *e2*. Более подробное описание каждого события и каждого выходного воздействия приведено в разд. 3.1.2, 3.1.3.

3.2. Модель BugModel

В данном разделе будут описаны поставщик событий, объект управления и автомат, моделирующий поведения единичной бактерии *UniMod*-модели ***BugModel***.

3.2.1. Диаграммы связей

В данной версии программы поведение всех возможных бактерий описывается одним автоматом. Возможно, в следующих версиях будут представлены другие обитатели микромира, жизненный цикл и поведение которых будет моделироваться другими автоматами. Схема связей для поставщика событий и управляемого объекта данной модели изображена на рис. 3.

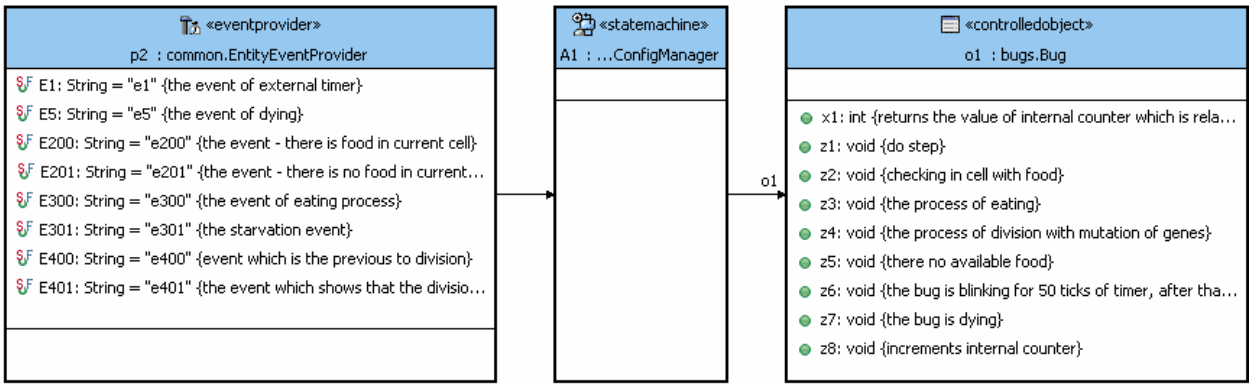

Рис. 3. Диаграмма связей ***BugModel***

3.2.2. Поставщик событий *EntityEventProvider*

Данный поставщик содержит события (табл. 3), связанные с жизненным циклом бактерии, генерируемые в зависимости от состояния самой бактерии (количество энергии, шагов и т.д.) и от конфигурации еды на поле.

Таблица 3. События поставщика событий *EntityEventProvider*

№	Событие	Описание	Комментарий
1.	e1	Событие, генерируемое внешним таймером через заданный интервал времени, при возникновении которого увеличивается «возраст» бактерии (число сделанных шагов)	Внешний таймер используется для синхронизации выполнения автоматов со временем. Интервал между тиками таймера устанавливается в статическом классе `Config` (при инициализации он равен одной секунде)
2.	e5	Событие генерируется, когда энергия бактерии равна нулю	Бактерия переходит в состояние «*Dying*»
3.	e200	Событие генерируется, если в ячейке, в которую в данный момент переместилась бактерия, есть еда	Бактерия переходит в состояние «*Decision about eating*»
4.	e201	Событие генерируется, если в ячейке, в которую в данный момент переместилась бактерия, еды нет	Бактерия переходит в состояние «*Do not eat*»
5.	e300	Событие приема пищи. Оно генерируется, если бактерия находится в состоянии «Decision about eating» и значение счетчика энергии не превосходит величины `ENERGY_EATING_LIMIT`, описанной (в нашем случае эта величина равна 100 единиц энергии)	Данное ограничение введено для того, чтобы не было слишком «энергичных» микроорганизмов, потомство которых уже сразу после деления родителя имеет достаточное для их собственного деления количество энергии (значение описанной величины установлено в `Config`)
6.	e301	Событие перехода в состояние «голодания». Оно генерируется, если бактерия находится в состоянии «Decision about	См. комментарий к п. 5

		eating» и значение счетчика энергии превосходит величины ENERGY_EATING_LIMIT	
7.	e400	Событие деления бактерии	Оно генерируется, если значение энергии бактерии достигает определенного порога, при условии что она также достигла «периода созревания». Энергия, достаточная для деления, и «период созревания» (число пройденных шагов) определяются значениями величин DIVISION_STEP_VALUE и DIVISION_ENERGY_VALUE, описанных в классе Config
8.	e401	Событие генерируемое, в случае, если деление не возможно	Событие генерируется, если не выполняется хотя бы одно из условий описанных в п. 7

3.2.3. Объект управления *Bug*

Объект управления представляет собой бактерию, управление которой осуществляется в зависимости от состояния, в котором она находится. Он содержит входные и выходные действия, связанные с управлением энергией бактерии, числом сделанных шагов и делением бактерии.

Таблица 4. Выходные воздействия объекта управления *Bug*

№	Выходное воздействие	Описание	Комментарий
1.	z1	Бактерия делает шаг	Бактерия находится в состоянии «*Moving*». Возраст бактерии увеличивается на единицу, а количество энергии уменьшается на единицу. В зависимости от того, находится в данной ячейке еда или нет, генерируется одно событие *e200* или *e201*

2.	z2	Проверка состояния бактерии при нахождении в клетке с едой	Бактерия находится в состоянии «*Decision about eating*» и в зависимости от соотношения значения счетчика энергии и величины `ENERGY_EATING_LIMIT` генерируются события *e300* или *e301*
3.	z3	Процесс приема пищи	Бактерия находится в состоянии «*Eating*». Энергия бактерии увеличивается на величину `FOOD_ENERGY_VALUE`, описанную в конфигурационном классе `Config`. Также, в зависимости от соотношения счетчика энергии и счетчика шагов с значениями `DIVI-SION_ENERGY_VALUE` и `DI-VISION_STEP_VALUE` соответственно, генерируются события *e400* или *e401*
4.	z4	Деление бактерии, сопровождающееся мутацией генов	Бактерия делится пополам. В зависимости от значения величины `PRBABILITY_OF _MUTATION`, появляется либо два обычных потомка, каждый из которых наследует набор генов своего родителя с небольшими мутациями (один из генов родительского набора уменьшается на единицу, а другой увеличивается на ту же величину), либо один из потомков становиться «полным мутантом» – набор генов такой бактерии не зависит от набора генов родителя. Также при делении энергия родительской особи делится пополам между потомками

5.	z5	Бактерия не принимает пищу	Бактерия находится в состоянии «*Do not eat*». В зависимости значения счетчика генерируются события *e5*, *e400* или *e401*
6.	z6	Запуск счетчика `counterForDying-Mode`	Используется для подсчета количества тиков внешнего таймера
7.	z8	Бактерия затухает в течение 50 тиков внешнего таймера	Бактерия находится в состоянии «*Dying*»
8.	z7	Бактерия умирает	Удаляется из коллекции `Enti-ties` объекта `EntitiesMan-ager`

Входное воздействие только одно – *x1*. Оно возвращает значение внутреннего счетчика.

3.2.4. Описание автомата

Большая часть данного раздела была изложены при описании генерируемых событий и объекта управления, поэтому достаточно перечислить все состояния автомата, моделирующего жизненный цикл бактерии, с небольшими комментариями (табл. 5). Диаграмма состояний представлена на рис. 4.

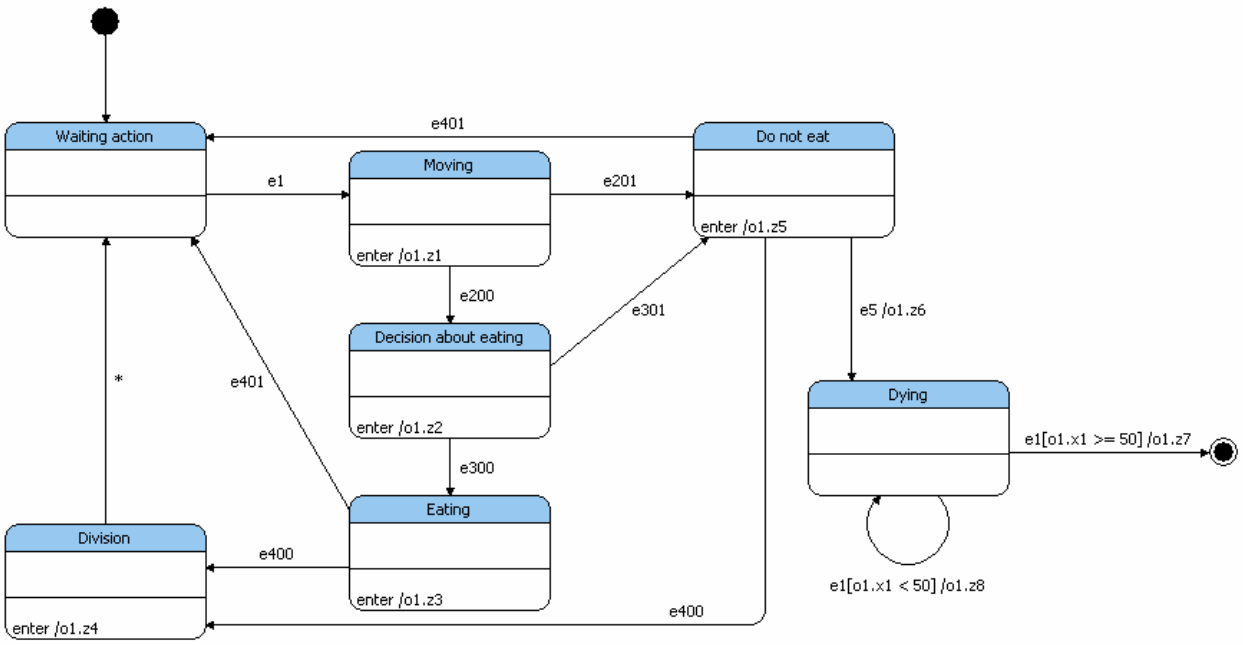

Рис. 4. Диаграмма состояний автомата жизненного цикла бактерии

Описание самой нотации диаграмм состояний *UML* было сделано в разд.1.4. Поэтому сразу перейдем к описанию состояний данного автомата.

Таблица 5. Описание состояний автомата жизненного цикла бактерии

№	Состояние	Описание
1.	«Waiting action»	Состояние ожидания, в котором оказывается бактерия после завершения жизненного цикла, связанного с одним тика таймера (если только она не перешла в состояние «Dying»)
2.	«Moving»	Состояние, в которое бактерия переходит по тику таймера
3.	«Decision about eating»	Состояние, в котором оказывается бактерия, если в клетке есть еда
4.	«Eating»	Состояние, в котором оказывается бактерия, если в клетке есть еда и счетчик энергии не превышает порог `ENERGY_EATING_LIMIT`
5.	«Do not eat»	Состояние, в котором оказывается бактерия, если в клетке нет еды, или же в клетке есть еда, но счетчик энергии превышает порог `ENERGY_EATING_LIMIT`
6.	«Division»	Состояние, в котором оказывается бактерия, если она достигла «периода созревания» (количество шагов больше или равно величине `DIVISION_STEP_VALUE`) и счетчик энергии >= `DIVISION_ENERGY_VALUE`
7.	«Dying»	Состояние, в котором оказывается бактерии, если счетчик энергии равен 0

4. Общий интерфейс приложения

Интерфейс приложения показан на рис. 5.

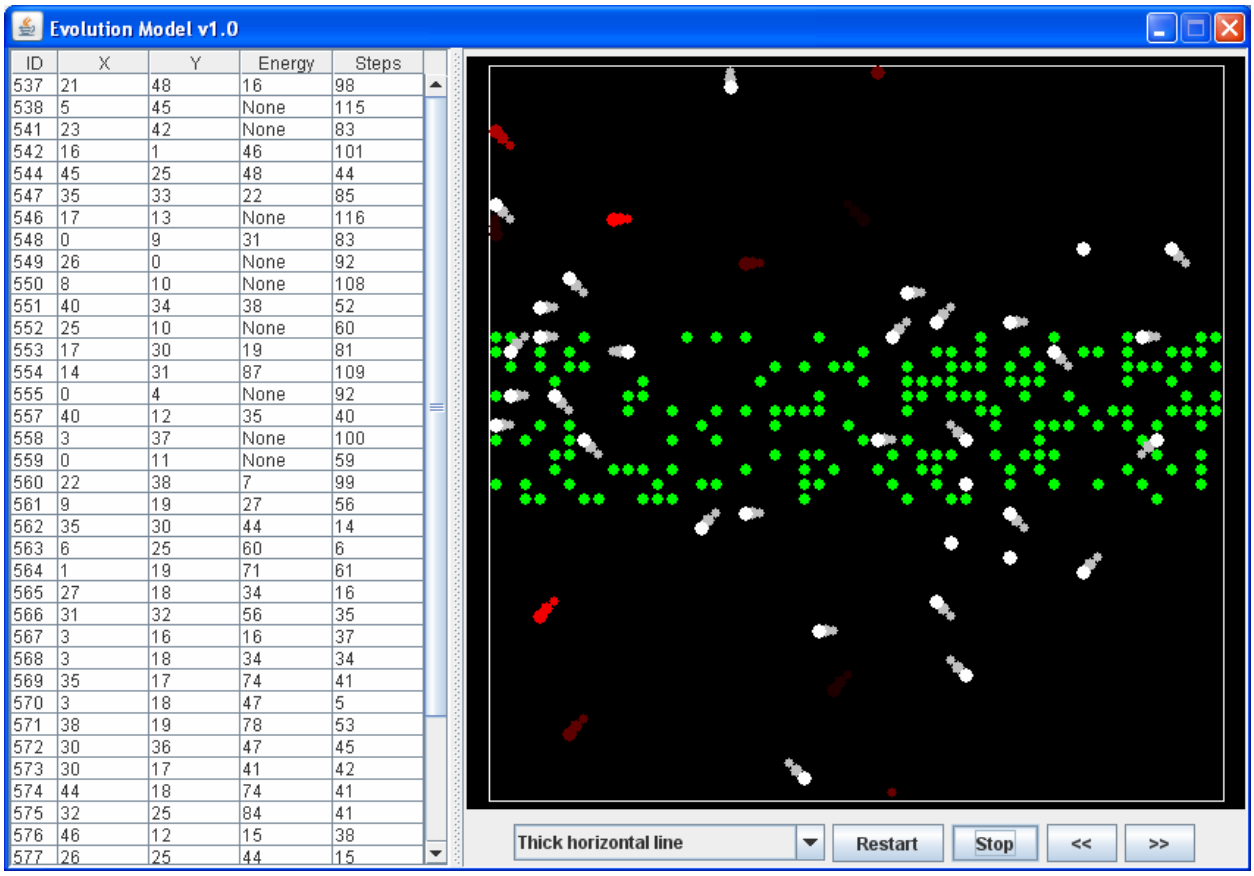

Рис. 5. Интерфейс приложения

Справа находится игровое поле, слева – информационная таблица с текущим состоянием популяции. Внизу под игровым полем расположена панель управления визуализацией алгоритма. Опишем каждый из этих элементов подробнее.

На игровом поле зеленым отмечена еда, белым – бактерии, принимающие участие в эволюционном отборе.

В таблице слева представлена информация обо всех бактериях, находящихся на игровом поле (ID – порядковый номер, X и Y – координаты, Energy – количество энергии, Steps – количество сделанных шагов). Поле «Energy» может принимать либо неотрицательные значения, либо значение «None». Последнее означает, что энергия бактерии достигла нулевой отметки и данная особь будет удалена с игрового поля – такие бактерии перед удалением окрашиваются в красный цвет.

Панель управления состоит из четырех кнопок. Кнопка «Restart» обеспечивает перезапуск визуализации, «Stop» – за остановку, «<<» – увеличение скорости и «>>» – уменьшение скорости визуализации. Также на панели находится выпадающий список, с помощью которого можно задавать располо-

жение еды на игровом поле. На данный момент существует шесть конфигураций: *"Thick horizontal line"* (горизонтальная линия, проходящая через центр экрана, рис. 5), *"Middle-thick vertical line"* (вертикальная линия), *"Thin cross-shaped configuration"* (горизонтальная и вертикальная линии, пересекающиеся в центре экрана), *"Two parallel thin horizontal lines"* (две параллельные горизонтальные линии), *"Thin horizontal line"* (тонкая горизонтальная линия), *"Rectangle of death"* (прямоугольный участок еды в центре игрового поля).

Заключение

В процессе работы программы можно визуально наблюдать изменение поведения микроорганизмов. Например, на рис. 5 имеет место хаотическое движение бактерий – такой вид движения характерен для начального этапа работы алгоритма, когда все особи имеют случайный набор генов и, следовательно, движутся в случайных направлениях.

Генетический отбор обеспечивает «осмысленный» вид движения. На экране еда располагается в виде горизонтальной полоски, проходящей через центр экрана, и большинство бактерий двигаются вдоль этой полоске в справа налево (рис. 6).

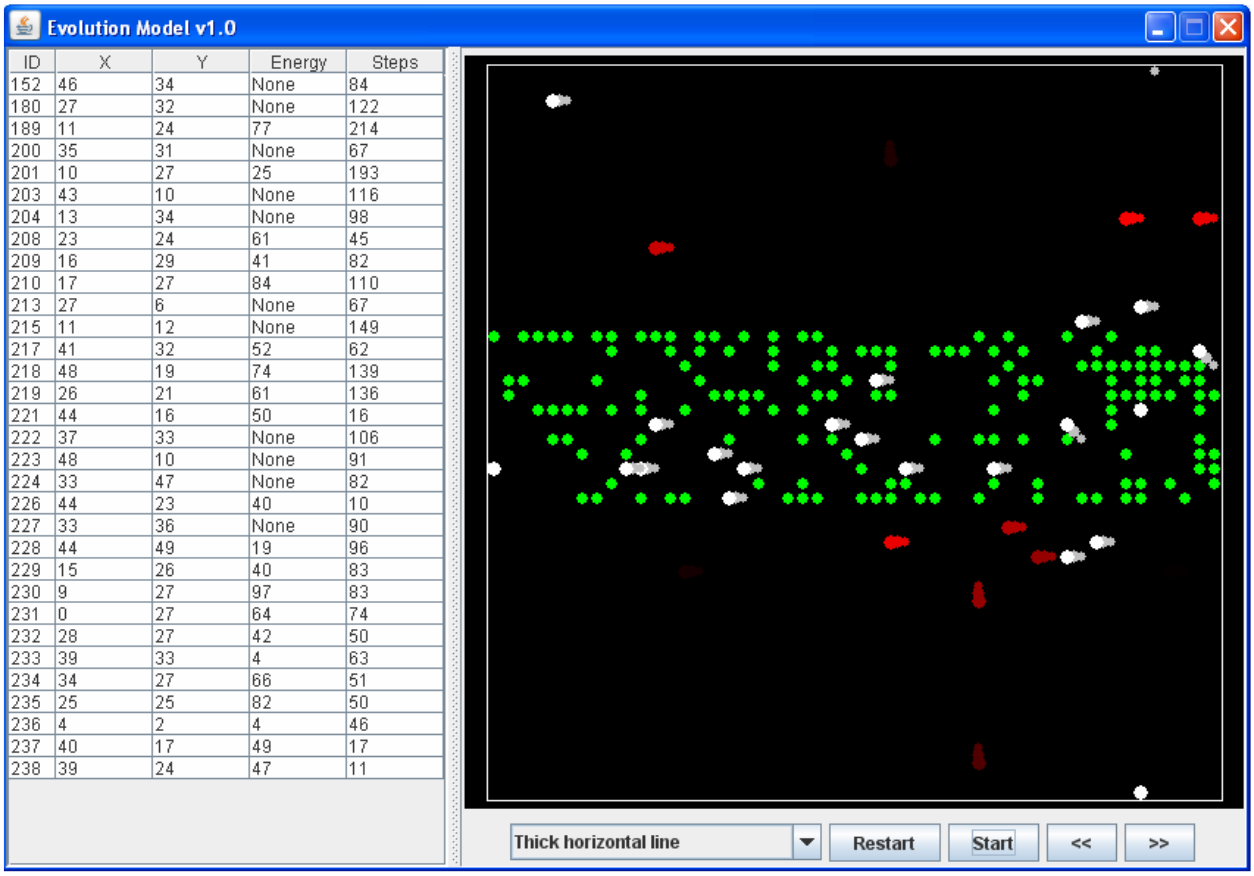

Рис. 6. Бактерии приспособились к окружающей среде

Такой вид движения один из самых рациональных для данной конфигурации – двигаясь либо направо, либо налево по полосе, бактерия будет получать максимальное количество еды. Отклонение в «мертвую зону» (нижнюю и верхнюю части игрового поля) грозит гибелью, так как каждый ход приводит к потере энергии, а её пополнение возможно только в центральной полосе, заполненной едой.

Таким образом, можно наблюдать, что бактерии подстраиваются под окружающую среду. При этом поведение микроорганизмов изменяется в зависимости от расположения еды на поле.

Читатель может использовать (программа опубликована на сайте http://is.ifmo.ru/ в разделах «Генетические алгоритмы» и «UniMod-проекты») другие конфигурации расположения еды, которые позволяет выбрать программа. В результате можно убедиться, что бактерии, за счет использования генетического алгоритма, в конечном счете, выбирают одно из оптимальных решений.

Литература

1. *Spears W. M.* Evolutionary Algorithms. The Role of Mutation and Recombination. Berlin: Springer. 2000.
2. *Шалыто А. А.* Технология автоматного программирования // Труды первой Всероссийской научной конференции «Методы и средства обработки информации» М.: МГУ. 2003. http://is.ifmo.ru/works/tech_aut_prog
3. *Дьюдни А.К.* Моделирование эволюции: «букашки» учатся охоте на бактерий // В мире науки. 1989. № 7.
4. *Гуров В., Мазин М., Нарвский А., Шалыто А.* UML. SWITCH-технология. Eclipse // Информационно-управляющие системы. 2004. № 6. http://is.ifmo.ru/works/uml-switch-eclipse
5. *UniMod project*. http://unimod.sourceforge.net
6. *Шалыто А. А.* SWITCH-технологии. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/1

Приложение 1. Структура классов

bugs

`Bug.java` – наследуется от абстрактного класса `Entity`, инкапсулирует поведение бактерии, задаваемое автоматом *bugs/A1*

`A1.xml` – автомат, задающий модель поведения бактерии

common

`Config.java` – класс, содержащий все конфигурационные параметры приложения (размеры игрового поля, параметры, отвечающие за распределение генов, инициализационные и общие параметры игровой модели)

`EntitiesManager.java` – класс, предоставляющий потокобезопасный доступ к коллекции всех организмов, участвующих в эволюционном отборе; блокировки потоков достигаются за счет использования класса `ReentrantReadWriteLock` из стандартной библиотеки *Java* (реализован в виде синглтона)

`Entity.java` – абстрактный класс, описывает общий интерфейс для всех организмов игровой среды, также содержит объект класса `ModelEngine` и метод `InitAutomata` для запуска автомата, описывающего жизненный цикл организма

`EntityEventProvider.java` – поставщик событий, общих для всех организмов системы

`EventWrappers.java` – вспомогательный класс для поставщика событий `EntityEventProvider`

`Field.java` – инкапсулирует игровое поле и предоставляет интерфейс для его управлением (реализован в виде синглтона)

gui

ControlPanel.java – предоставляет интерфейс для управления работой программы, которое реализовано головным автоматом *gui/A1.xml*

EntityTableModel.java – вспомогательный класс для отображения информационной таблицы, показывающей текущие данные о всех организмах в системе

FieldPanel.java – панель, отображающая игровое поле

MainFrame.java – главный gui-компонент, инициализирующий отрисовку приложения

ScreenEventProvider.java – поставщик событий головного автомата

A1.xml – головной автомат, запускающий и управляющий работой программы

Приложение 2. Исходный код

MainTest.java

```
import gui.MainFrame;

/**
 * Entry point of the application.
 *
 * @author Goonich Ivan, Irinev Anton
 */
public class MainTest {
    private static void createAndShowGUI() {
        new MainFrame();
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

Bug.java

```
package bugs;

import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.context.StateMachineContext;
import common.Entity;
import common.EntitiesManager;
import common.Field;
import common.Config;
import common.EventWrappers;

/**
 * Represents behavour of the bug entity.
 *
 * @author Goonich Ivan, Irinev Anton
 */
public class Bug extends Entity implements ControlledObject {

private void initialize() {
     id = nextId;
     ++nextId;
     InitAutomata("src/bugs/A1.xml", this);
}

/**
 * Initializes bug with default values.
```

```
 */
public Bug() {
     super();
     initialize();
}

/**
 * Initializes with parent bug.
 *
 * @param parent
 *            parent bug
 */
public Bug(Bug parent) {
     super(parent);
     initialize();
}

/**
 * @unimod.action.descr do step
 */
public void z1(StateMachineContext context) {
     doStep();
     Field field = Field.getInstance();
     if (field.isFoodInCell(x, y)) {
          EventWrappers.generateFoodInCellEvent(engine);
     } else {
          EventWrappers.generateNoFoodInCellEvent(engine);
     }
}

/**
 * @unimod.action.descr checking in cell with food
 */
public void z2(StateMachineContext context) {
     if (energy >= Config.ENERGY_EATING_LIMIT) {
          EventWrappers.generateStarvationEvent(engine);
     } else {
          EventWrappers.generateEatingEvent(engine);
     }
}

/**
 * @unimod.action.descr the process of eating
 */
public void z3(StateMachineContext context) {
     Field field = Field.getInstance();
     energy += Config.FOOD_ENERGY_VALUE;
     field.removeFoodInCell(x, y);

     if (energy >= Config.DIVISION_ENERGY_VALUE
               && stepsCount >= Config.DIVISION_STEP_VALUE) {
          EventWrappers.generatePredivisionEvent(engine);
     } else {
```

```
            EventWrappers.generateNoDivisionEvent(engine);
        }
    }

    /**
     * @unimod.action.descr the process of division with mutation of genes
     */
    public void z4(StateMachineContext context) {
        if (rnd.nextDouble() <= Config.PROBABILITY_OF_MUTATION) {
            // process mutation
            // child don't inherit any gene from his parent
            EntitiesManager.getInstance().addEntity(new Bug());
        } else {
            EntitiesManager.getInstance().addEntity(new Bug(this));
        }

        energy /= 2;
        stepsCount = 0;
        changeGenes();
    }

    /**
     * @unimod.action.descr there no available food
     */
    public void z5(StateMachineContext context) {
        if (energy <= 0) {
            EventWrappers.generateDyingEvent(engine);
        } else {
            if (energy >= Config.DIVISION_ENERGY_VALUE
                    && stepsCount >= Config.DIVISION_STEP_VALUE) {
                EventWrappers.generatePredivisionEvent(engine);
            } else {
                EventWrappers.generateNoDivisionEvent(engine);
            }
        }
    }

    /**
     * @unimod.action.descr the bug is blinking for 50 ticks of timer,
     *                      after that he dies
     *
     */
    public void z6(StateMachineContext context) {
        isDying = true;
        counterForDyingMode = 0;
    }

    /**
     * @unimod.action.descr the bug is dying
     */
    public void z7(StateMachineContext context) {
        isDead = true;
    }
```

```
/**
 * @unimod.action.descr increments internal counter
 */
public void z8(StateMachineContext context) {
     ++counterForDyingMode;
}

/**
 * @unimod.action.descr returns the value of internal counter which is
 *                      related with visualization of dying process
 */
public int x1(StateMachineContext c) {
     return counterForDyingMode;
}
}
```

Config.java

```
package common;

/**
 * Store of config parameters.
 *
 * @author Goonich Ivan, Irinev Anton
 */
public class Config {

/**
 * Field height.
 */
public static final int FIELD_HEIGHT = 50;

/**
 * Field width.
 */
public static final int FIELD_WIDTH = 50;

/**
 * Size of a cell on the field.
 */
public static final int CELL_SIZE = 10;

/**
 * Length of genes set.
 */
public static final int GENES_COUNT = 8;

/**
 * Initial dose of food which was placed in the field.
 */
public static final int INITIAL_FOOD_DOSE = 500;
```

```
/**
 * Periodic dose of food which was placed in the field.
 */
public static final int REGULAR_FOOD_DOSE = 20;

/**
 * Sum of all direction's probabilities in genes set.
 */
public static final int MAX_DIRECTION_PROBABILITY = 24;

/**
 * Timer delay for one entity's step.
 */
public static final int ENTITY_STEP_DELAY = 80;

/**
 * Default energy value.
 */
public static final int INITIAL_ENERGY_VALUE = 50;

/**
 * Number of steps which allows division.
 */
public static final int DIVISION_STEP_VALUE = 20;

/**
 * Amount of energy which allows division.
 */
public static final int DIVISION_ENERGY_VALUE =
                              INITIAL_ENERGY_VALUE * 2;

/**
 * Timer delay for one food generating.
 */
public static final int FOOD_GENERATING_DELAY = 500;

/**
 * Amount of energy for one unit of food.
 */
public static final int FOOD_ENERGY_VALUE = 8;

/**
 * If current amount of energy more than or equal to this limit
 * entity must leave out a food.
 */
public static final int ENERGY_EATING_LIMIT = 100;

/**
 * Initial entities count.
 */
public static final int INITIAL_ENTITIES_COUNT = 15;

/**
```

```
 * Mutation probability.
 */
public static final double PROBABILITY_OF_MUTATION = 0.08;
}
```

EntitiesManager.java

```
package common;

import java.util.ArrayList;
import java.util.concurrent.locks.ReentrantReadWriteLock;
import java.util.concurrent.locks.Lock;

/**
 * Represents collection of Entity objects.
 *
 * @author Irinev Anton
 */
public class EntitiesManager {

// singlton's instance
private static EntitiesManager manager = new EntitiesManager();

// common entities collection
private ArrayList<Entity> entities = new ArrayList<Entity>();
    private ReentrantReadWriteLock rwl;
    private Lock readLock;
    private Lock writeLock;

private EntitiesManager() {

    rwl = new ReentrantReadWriteLock();
    readLock = rwl.readLock();
    writeLock = rwl.writeLock();
}

/**
 * Gives an access to the singlton's instance.
 *
 * @return singlton's instance
 */
public static EntitiesManager getInstance() {
     return manager;
}

/**
 * Add entity to common entities collection.
 *
 * @param entity
 *            entity that will be added
 */
public void addEntity(Entity entity) {
     writeLock.lock();
```

```
        try
        {
            entities.add(entity);
        }
        finally
        {
              writeLock.unlock();
        }
    }

    /**
     * Remove entity from common entities collection.
     *
     * @param entity
     *            entity that will be removed
     */
    public void removeEntity(Entity entity) {
        writeLock.lock();
        try
        {
            entities.remove(entity);
        }
        finally
        {
            writeLock.unlock();
        }
    }

    /**
     * Handles tick of external timer.
     */
    public void handleExternalTimerEvent() {
        ArrayList<Entity> copyEntities = getEntities();

        if (copyEntities.size() == 0) {
            return;
        }

        for (Entity entity : copyEntities) {
            entity.generateNextStepEvent();
        }
    }

    /**
     * Returns the element at the specified position.
     * @param i
     * @return entity
     */
    public Entity getEntity(int i) {
       Entity res;
        readLock.lock();
        try
        {
```

```
            res = entities.get(i);
        }
        finally
        {
            readLock.unlock();
        }

        return res;
    }

    /**
     * Returns collection of entity.
     * @return entities
     */
    public ArrayList<Entity> getEntities() {
        ArrayList<Entity> copyEntities
            = new ArrayList<Entity>(entities.size());
        readLock.lock();
        try
        {
            copyEntities.addAll(entities);
        }
        finally
        {
               readLock.unlock();
        }

        return copyEntities;
    }

    /**
     * Returns the number of entities.
     * @return count
     */
    public int getEntitiesCount() {
        return entities.size();
    }

    /**
     * Clears the collection of entities.
     */
    public void clear() {
        writeLock.lock();
        try
        {
            entities.clear();
        }
        finally
        {
            writeLock.unlock();
        }
    }
}
```

Entity.java

```
package common;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Random;

import org.apache.commons.logging.LogFactory;

import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.adapter.standalone.Run;
import com.evelopers.unimod.core.stateworks.Model;
import com.evelopers.unimod.debug.ExceptionHandlerImpl;
import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.ControlledObjectsMap;
import com.evelopers.unimod.runtime.EventProcessorListener;
import com.evelopers.unimod.runtime.ExceptionHandler;
import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.interpretation.InterpretationHelper;
import com.evelopers.unimod.runtime.logger.SimpleLogger;
import com.evelopers.unimod.transform.TransformException;
import com.evelopers.unimod.transform.xml.XMLToModel;

/**
 * Represents abstract entity class.
 *
 * @author Goonich Ivan, Irinev Anton
 */
public abstract class Entity {
// coordinates
protected int x;

protected int y;

protected int oldX;

protected int oldY;

// individual parameters
protected int stepsCount;
protected int counterForDyingMode;

protected int energy;

protected boolean isDead = false;
protected boolean isDying = false;

protected int[] genes = new int[Config.GENES_COUNT];
```

```
protected static int nextId = 1;

protected int id;

// represents the movement direction
protected static int[] xMove = new int[Config.GENES_COUNT];

protected static int[] yMove = new int[Config.GENES_COUNT];

// automata managing members
protected ModelEngine engine;

protected static Random rnd = new Random();

static {
     xMove[0] = 0;
     yMove[0] = -1;
     xMove[1] = 1;
     yMove[1] = -1;
     xMove[2] = 1;
     yMove[2] = 0;
     xMove[3] = 1;
     yMove[3] = 1;
     xMove[4] = 0;
     yMove[4] = 1;
     xMove[5] = -1;
     yMove[5] = 1;
     xMove[6] = -1;
     yMove[6] = 0;
     xMove[7] = -1;
     yMove[7] = -1;
}

/**
 * Initializes with default values.
 */
public Entity() {
     stepsCount = 0;
     energy = Config.INITIAL_ENERGY_VALUE;

     x = rnd.nextInt(Config.FIELD_WIDTH);
     y = rnd.nextInt(Config.FIELD_HEIGHT);

     oldX = x;
     oldY = y;

     setRandomGenes();
}

/**
 * Initializes with parent entity.
 *
 * @param parent
```

```
     *            parent entity
     */
    public Entity(Entity parent) {
         stepsCount = 0;
         energy = parent.energy - parent.energy / 2;

         x = parent.x;
         y = parent.y;

         oldX = x;
         oldY = y;

         for (int i = 0; i < genes.length; ++i) {
              genes[i] = parent.genes[i];
         }
         changeGenes();
    }

    /**
     * Returns x coordinate.
     *
     * @return x
     */
    public int getX() {
         return x;
    }

    /**
     * Returns y coordinate.
     *
     * @return y
     */
    public int getY() {
         return y;
    }

    /**
     * Returns old x coordinate.
     *
     * @return x
     */
    public int getOldX() {
         return oldX;
    }

    /**
     * Returns old y coordinate.
     *
     * @return y
     */
    public int getOldY() {
         return oldY;
    }
```

```
/**
 * Returns id.
 *
 * @return id
 */
public int getID() {
     return id;
}

/**
 * Returns energy.
 *
 * @return energy
 */
public int getEnergy() {
     return energy;
}

/**
 * Returns count of steps.
 *
 * @return stepsCount
 */
public int getStepsCount() {
     return stepsCount;
}

/**
 * Returns true if entity is dying, false elsewise.
 *
 * @return isDying
 */
public boolean isDying() {
     return isDying;
}

/**
 * Returns dying step counter.
 * @return counterForDyingMode
 */
public int getDyingStepCounter() {
     return counterForDyingMode;
}

/**
 * Makes small random change of genes set.
 */
protected void changeGenes() {
     int numOfZeroElements = 0;
     for (int i = 0; i < genes.length; ++i) {
          if (genes[i] == 0)
```

```
                ++numOfZeroElements;
        }

        if (numOfZeroElements == 0) {
                int i = rnd.nextInt(genes.length);
                int j = rnd.nextInt(genes.length - 1);
                if (j >= i)
                        ++j;

                ++genes[i];
                --genes[j];
        } else {
                int i = rnd.nextInt(genes.length);

                if (genes[i] == 0)
                        --numOfZeroElements;
                ++genes[i];

                // random index in non zero elements subsequence
                int nonZeroInd = rnd.nextInt(
                        genes.length - numOfZeroElements - 1);
                for (int k = 0, j = -1; k < genes.length; ++k) {
                        if (genes[k] > 0) {
                                ++j;
                                if (j == nonZeroInd) {
                                        if (k == i)
                                                ++nonZeroInd;
                                        else
                                                --genes[k];
                                }
                        }
                }
        }
}

/**
 * Returns next direction.
 *
 * @return direction value
 */
protected int getDirection() {
        int k = rnd.nextInt(Config.MAX_DIRECTION_PROBABILITY) + 1;
        int i = 0;
        int sum = 0;
        while (sum < k) {
                sum += genes[i];
                ++i;
        }

        return i;
}
```

```
/**
 * Process next step.
 */
protected void doStep() {
     ++stepsCount;
     --energy;

     oldX = x;
     oldY = y;

     int dir = getDirection();
     x = (x + xMove[dir]) % Config.FIELD_WIDTH;
     y = (y + yMove[dir]) % Config.FIELD_HEIGHT;
     if (x < 0)
          x += Config.FIELD_WIDTH;
     if (y < 0)
          y += Config.FIELD_HEIGHT;
}

/**
 * Initializes genes array with random set.
 */
private void setRandomGenes() {
     int[] tmp = getDistribution();

     for (int i = 1; i < tmp.length; ++i) {
          genes[i - 1] = tmp[i] - tmp[i - 1];
     }
}

private static void sort(int[] arr) {
     for (int i = 0; i < arr.length - 1; ++i) {
          for (int j = i + 1; j < arr.length; ++j) {
               if (arr[i] > arr[j]) {
                    int k = arr[i];
                    arr[i] = arr[j];
                    arr[j] = k;
               }
          }
     }
}

private int[] getDistribution() {
     int[] res = new int[genes.length + 1];
     res[0] = 0;
     res[1] = Config.MAX_DIRECTION_PROBABILITY;

     for (int i = 2; i < res.length; ++i) {
          res[i] = rnd.nextInt(
                         Config.MAX_DIRECTION_PROBABILITY + 1);        }
```

```
        sort(res);

        return res;
    }

    /**
     * Creates automata model for the device.
     *
     * @param xmlmodel
     *            The path to the file of device xml-description
     * @param bug
     *            The device controlled object
     */
    protected void InitAutomata(String xmlmodel, ControlledObject bug) {
        /* load model from xml schema */
        Model model;
        try {
            InputStream is = new FileInputStream(xmlmodel);
            if (is == null) {
                System.out.println("is == null");
                System.exit(0);
            }
            model = XMLToModel.loadAndCompile(is);
        } catch (IOException e) {
            System.out.println("[ERROR]   Can't load xml: " + xmlmodel);
            throw new RuntimeException();
        } catch (TransformException e) {
            System.err.println(e.getMessage());
            throw new RuntimeException();
        }

        /* create runtime engine */
        InterpretationHelper helper = InterpretationHelper.getInstance();
        try {
            engine = helper.createBuildInModelEngine(
                        model, new BugsMap(bug), true);
        } catch (CommonException e) {
            System.err.println(e.getMessage());
            throw new RuntimeException();
        }

        EventProcessorListener logger = new SimpleLogger(LogFactory
                .getLog(Run.class));
        engine.getEventProcessor().addEventProcessorListener(logger);
        ExceptionHandler eh = new ExceptionHandlerImpl();
        engine.getEventProcessor().addExceptionHandler(eh);
    }

    private static class BugsMap implements ControlledObjectsMap {
        private ControlledObject co;

        public BugsMap(ControlledObject co) {
            this.co = co;
```

```
    }

    public ControlledObject getControlledObject(String coName) {
        return co;
    }
}

/**
 * Generates event for tick of external timer.
 */
public void generateNextStepEvent() {
    if (isDead) {
        EntitiesManager.getInstance().removeEntity(this);
    } else {
        EventWrappers.generateTimerEvent(engine);
    }
}
}
```

EntityEventProvider.java

```
package common;

import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ModelEngine;

/**
 * General event provider.
 *
 * @author Goonich Ivan, Irinev Anton
 */
public class EntityEventProvider implements EventProvider {

/**
 * @unimod.event.descr the event of external timer
 */
public static final String E1 = "e1";

/**
 * @unimod.event.descr the event - there is food in current cell
 */
public static final String E200 = "e200";

/**
 * @unimod.event.descr the event - there is no food in current cell
 */
public static final String E201 = "e201";

/**
 * @unimod.event.descr the event of eating process
 */
public static final String E300 = "e300";
```

```
/**
 * @unimod.event.descr the starvation event
 */
public static final String E301 = "e301";

/**
 * @unimod.event.descr event which is the previous to division
 */
public static final String E400 = "e400";

/**
 * @unimod.event.descr the event which shows
 *                     that the division is absent
 */
public static final String E401 = "e401";

/**
 * @unimod.event.descr the event of dying
 */
public static final String E5 = "e5";


public void dispose() {}

public void init(ModelEngine engine) throws CommonException {}
}
```

EventWrappers.java

```
package common;

import com.evelopers.unimod.core.stateworks.Event;
import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
import com.evelopers.unimod.runtime.ModelEngine;

public class EventWrappers {
/**
 * Generates the event of timer.
 * @param engine
 */
public static void generateTimerEvent(ModelEngine engine){
     engine.getEventManager().handle(new Event(EntityEventProvider.E1),
               StateMachineContextImpl.create());
}

/**
 * Generates the event - there is food in current cell.
 * @param engine
 */
public static void generateFoodInCellEvent(ModelEngine engine){
     engine.getEventManager().handle(
```

```
                new Event(EntityEventProvider.E200),
                StateMachineContextImpl.create()
     );
}

/**
 * Generates the event - there is no food in current cell.
 * @param engine
 */
public static void generateNoFoodInCellEvent(ModelEngine engine){
     engine.getEventManager().handle(
                new Event(EntityEventProvider.E201),
                StateMachineContextImpl.create());
}

/**
 * Generates the event of eating process.
 * @param engine
 */
public static void generateEatingEvent(ModelEngine engine){
     engine.getEventManager().handle(
                new Event(EntityEventProvider.E300),
                StateMachineContextImpl.create());
}

/**
 * Generates the starvation event.
 * @param engine
 */
public static void generateStarvationEvent(ModelEngine engine){
     engine.getEventManager().handle(
                new Event(EntityEventProvider.E301),
                StateMachineContextImpl.create());
}

/**
 * Generates event which is the previous to division.
 * @param engine
 */
public static void generatePredivisionEvent(ModelEngine engine){
     engine.getEventManager().handle(
                new Event(EntityEventProvider.E400),
                StateMachineContextImpl.create());
}

/**
 * Generates the event which shows that the division is absent.
 * @param engine
 */
public static void generateNoDivisionEvent(ModelEngine engine){
     engine.getEventManager().handle(
                new Event(EntityEventProvider.E401),
                StateMachineContextImpl.create());
```

```
}

/**
 * Generates the event of dying.
 * @param engine
 */
public static void generateDyingEvent(ModelEngine engine){
     engine.getEventManager().handle(
               new Event(EntityEventProvider.E5),
               StateMachineContextImpl.create());
}

}
```

Field.java

```
package common;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

import javax.swing.Timer;


/**
 * Represents field where entities are located.
 *
 * @author Goonich Ivan, Irinev Anton
 */
public class Field {
// singlton's instance
private static Field field = new Field();

// represents food location
private boolean[][] food
          = new boolean[Config.FIELD_HEIGHT][Config.FIELD_WIDTH];

private Random rnd = new Random();
private Timer timer = null;
private int configType = 1;

private ActionListener taskPerformer = new ActionListener() {
     public void actionPerformed(ActionEvent evt) {
          giveFeed();
     }
};
private Field() {
     createFoodDose(Config.INITIAL_FOOD_DOSE);
     timer = new Timer(Config.FOOD_GENERATING_DELAY, taskPerformer);
     timer.start();
}
```

```
public void timerStart() {
     if (!timer.isRunning()) {
          timer.start();
     }
}

public void timerStop() {
     if (timer.isRunning()) {
          timer.stop();
     }
}

public void setTimerDelay(int delay) {
     timer.setDelay(delay);
}

public void setFoodConfigType(int configType) {
     this.configType = configType;
}

/**
 * Gives an access to the singlton's instance.
 *
 * @return singlton's instance
 */
public static Field getInstance() {
     return field;
}

/**
 * Refreshes food on the field.
 */
public void restart() {
     for (int i = 0; i < Config.FIELD_HEIGHT; ++i) {
          for (int j = 0; j < Config.FIELD_WIDTH; ++j) {
               food[i][j] = false;
          }
     }

     createFoodDose(Config.INITIAL_FOOD_DOSE);
}

private void giveFeed() {
     createFoodDose(Config.REGULAR_FOOD_DOSE);
}

private void createFoodDose(int dose) {
     int i, j;

     switch (configType) {
          case 1: {
               for (int k = 0; k < dose; ++k) {
                    i = rnd.nextInt(Config.FIELD_HEIGHT);
```

```
                j = rnd.nextInt(Config.FIELD_WIDTH / 4) + 18;
                food[i][j] = true;
            }
            break;
        }
        case 2: {
            for (int k = 0; k < dose; ++k) {
                i = rnd.nextInt(Config.FIELD_HEIGHT / 5) + 22;
                j = rnd.nextInt(Config.FIELD_WIDTH);
                food[i][j] = true;
            }
            break;
        }
        case 3: {
            for (int k = 0; k < dose; ++k) {
                i = rnd.nextInt(Config.FIELD_HEIGHT);
                j = rnd.nextInt(Config.FIELD_WIDTH / 8) + 25;
                food[i][j] = true;
            }

            for (int k = 0; k < dose; ++k) {
                i = rnd.nextInt(Config.FIELD_HEIGHT / 8) + 25;
                j = rnd.nextInt(Config.FIELD_WIDTH);
                food[i][j] = true;
            }
            break;
        }
        case 4: {
            for (int k = 0; k < dose; ++k) {
                i = rnd.nextInt(Config.FIELD_HEIGHT);
                j = rnd.nextInt(Config.FIELD_WIDTH / 8) + 7;
                food[i][j] = true;
            }

            for (int k = 0; k < dose; ++k) {
                i = rnd.nextInt(Config.FIELD_HEIGHT);
                j = rnd.nextInt(Config.FIELD_WIDTH / 8) + 42;
                food[i][j] = true;
            }
            break;
        }
        case 5: {
            for (int k = 0; k < dose; ++k) {
                i = rnd.nextInt(Config.FIELD_HEIGHT);
                j = rnd.nextInt(Config.FIELD_WIDTH / 7) + 23;
                food[i][j] = true;
            }
            break;
        }
        case 6: {
            for (int k = 0; k < dose; ++k) {
                i = rnd.nextInt(Config.FIELD_HEIGHT / 3) + 19;
                j = rnd.nextInt(Config.FIELD_WIDTH / 3) + 19;
```

```
                    food[i][j] = true;
                }
                break;

            }
        }
}


/**
 * Checks if there is food in given cell.
 *
 * @param i
 *            x coordinate
 * @param j
 *            y coordinate
 * @return true if cell conatains food, false elsewise
 */
public boolean isFoodInCell(int i, int j) {
     return food[i][j];
}

/**
 * Removes food from given cell.
 *
 * @param i
 *            x coordinate
 * @param j
 *            y coordinate
 */
public void removeFoodInCell(int i, int j) {
     food[i][j] = false;
}
}
```

ControlPanel.java

```
package gui;

import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

import javax.swing.JComboBox;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JTextField;

import org.apache.commons.logging.LogFactory;
```

```
import bugs.Bug;

import common.Config;
import common.Field;
import common.EntitiesManager;

import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.adapter.standalone.Run;
import com.evelopers.unimod.core.stateworks.Event;
import com.evelopers.unimod.core.stateworks.Model;
import com.evelopers.unimod.debug.ExceptionHandlerImpl;
import com.evelopers.unimod.runtime.ControlledObject;
import com.evelopers.unimod.runtime.ControlledObjectsMap;
import com.evelopers.unimod.runtime.EventProcessorListener;
import com.evelopers.unimod.runtime.ExceptionHandler;
import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.context.StateMachineContext;
import com.evelopers.unimod.runtime.context.StateMachineContextImpl;
import com.evelopers.unimod.runtime.interpretation.InterpretationHelper;
import com.evelopers.unimod.runtime.logger.SimpleLogger;
import com.evelopers.unimod.transform.TransformException;
import com.evelopers.unimod.transform.xml.XMLToModel;

public class ControlPanel extends JPanel
implements ActionListener, ControlledObject
{

private static final long serialVersionUID = 1L;

private MainFrame frame;
private JButton btnRestart = new JButton("Restart");
private JButton btnStopPlay = new JButton("Stop");
private JButton btnSlowly = new JButton("<<");
private JButton btnFaster = new JButton(">>");

private int entitiesTimerDelay = Config.ENTITY_STEP_DELAY;
private int fieldTimerDelay = Config.FOOD_GENERATING_DELAY;

private String items[] = {
     "Thick horizontal line",
     "Middle-thick vertical line",
     "Thin cross-shaped configuration",
     "Two parallel thin horizontal lines",
     "Thin horizontal line",
     "Rectangle of death"
};

private JComboBox cmbBox = new JComboBox(items);

private static ModelEngine engine = null;

private int foodType = 1;
```

```
public ControlPanel(MainFrame frame) {
     this.frame = frame;
     Dimension size =
         new Dimension(Config.CELL_SIZE * Config.FIELD_WIDTH, 40);
     this.setSize(size);
     this.setPreferredSize(size);
     this.setDoubleBuffered(true);

     this.setLayout(new FlowLayout());
     this.add(cmbBox);
     this.add(btnRestart);
     this.add(btnStopPlay);
     this.add(btnSlowly);
     this.add(btnFaster);

     cmbBox.setActionCommand("done");
     btnRestart.setActionCommand("restart");
     btnStopPlay.setActionCommand("stop_play");
     btnSlowly.setActionCommand("slower");
     btnFaster.setActionCommand("faster");

     cmbBox.addActionListener(this);
     btnRestart.addActionListener(this);
     btnStopPlay.addActionListener(this);
     btnSlowly.addActionListener(this);
     btnFaster.addActionListener(this);

     this.InitAutomata("src/gui/A1.xml", this);
}


private void InitAutomata(String xmlmodel, ControlledObject panel) {
     /* load model from xml schema */
     Model model;
     try {
          InputStream is = new FileInputStream(xmlmodel);
          if (is == null) {
               System.out.println("is == null");
               System.exit(0);
          }
          model = XMLToModel.loadAndCompile(is);
     } catch (IOException e) {
          System.out.println("[ERROR]   Can't load xml: " + xmlmodel);
          throw new RuntimeException();
     } catch (TransformException e) {
          System.err.println(e.getMessage());
          throw new RuntimeException();
     }

     /* create runtime engine */
     InterpretationHelper helper = InterpretationHelper.getInstance();
     try {
```

```
            engine = helper.createBuildInModelEngine(
                            model,
                            new ControlPanelMap(panel),
                            true
            );
        } catch (CommonException e) {
            System.err.println(e.getMessage());
            throw new RuntimeException();
        }

        EventProcessorListener logger =
              new SimpleLogger(LogFactory.getLog(Run.class));
        engine.getEventProcessor().addEventProcessorListener(logger);
        ExceptionHandler eh = new ExceptionHandlerImpl();
        engine.getEventProcessor().addExceptionHandler(eh);

}

private static class ControlPanelMap implements ControlledObjectsMap {
        private ControlledObject co;

        public ControlPanelMap(ControlledObject co) {
            this.co = co;
        }

        public ControlledObject getControlledObject(String coName) {
            return co;
        }
}

    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

            if ("restart".equals(command)) {
                  ControlPanel.generateRestartEvent();
            } else
            if ("stop_play".equals(command)) {
                  ControlPanel.generateStopStartEvent();
            } else
            if ("slower".equals(command)) {
                  ControlPanel.generateSlowerEvent();
            } else
            if ("faster".equals(command)) {
                  ControlPanel.generateFasterEvent();
            } else
            if ("done".equals(command)) {
                  foodType = cmbBox.getSelectedIndex() + 1;
                  ControlPanel.generateDoneEvent();
            }
        }

/**
 * @unimod.action.descr restarts the automata, when the process
```

```
 * is working
 */
public void z1(StateMachineContext context) {
     frame.timerStop();
     Field.getInstance().timerStop();

     EntitiesManager.getInstance().clear();

     for (int i = 0; i < Config.INITIAL_ENTITIES_COUNT; ++i) {
           EntitiesManager.getInstance().addEntity(new Bug());
     }
     Field.getInstance().restart();

     frame.timerStart();
     Field.getInstance().timerStart();
}

/**
 * @unimod.action.descr restarts the automata,
 *                      when the process is paused
 */
public void z2(StateMachineContext context) {
     frame.timerStop();
     Field.getInstance().timerStop();

     EntitiesManager.getInstance().clear();

     for (int i = 0; i < Config.INITIAL_ENTITIES_COUNT; ++i) {
          EntitiesManager.getInstance().addEntity(new Bug());
     }
     Field.getInstance().restart();

     btnStopPlay.setText("Stop");

     frame.timerStart();
     Field.getInstance().timerStart();
}

/**
 * @unimod.action.descr visualization is paused
 */
public void z3(StateMachineContext context) {
     btnStopPlay.setText("Start");
     frame.timerStop();
     Field.getInstance().timerStop();
}

/**
 * @unimod.action.descr visualization is started
 */
public void z4(StateMachineContext context) {
     btnStopPlay.setText("Stop");
     frame.timerStart();
```

```
        Field.getInstance().timerStart();
    }

    /**
     * @unimod.action.descr decreases speed of visualization
     */
    public void z5(StateMachineContext context) {
        entitiesTimerDelay += (int)(Config.ENTITY_STEP_DELAY * 0.2);
        frame.setTimerDelay(entitiesTimerDelay);

        fieldTimerDelay += (int)(Config.FOOD_GENERATING_DELAY * 0.2);
        Field.getInstance().setTimerDelay(fieldTimerDelay);
    }

    /**
     * @unimod.action.descr increases speed of visualization
     */
    public void z6(StateMachineContext context) {
        entitiesTimerDelay -= (int)(Config.ENTITY_STEP_DELAY * 0.2);
        frame.setTimerDelay(entitiesTimerDelay);

        fieldTimerDelay -= (int)(Config.FOOD_GENERATING_DELAY * 0.2);
        Field.getInstance().setTimerDelay(fieldTimerDelay);
    }

    /**
     * @unimod.action.descr changes food configuration, when the process is
    working
     */
    public void z7(StateMachineContext context) {
        frame.timerStop();
        Field.getInstance().timerStop();

        EntitiesManager.getInstance().clear();

        for (int i = 0; i < Config.INITIAL_ENTITIES_COUNT; ++i) {
            EntitiesManager.getInstance().addEntity(new Bug());
        }
        Field.getInstance().setFoodConfigType(foodType);
        Field.getInstance().restart();

        frame.timerStart();
        Field.getInstance().timerStart();
    }


    /**
     * @unimod.action.descr changes food configuration, when the process is
        * paused
     */
    public void z8(StateMachineContext context) {
        frame.timerStop();
        Field.getInstance().timerStop();
```

```
        EntitiesManager.getInstance().clear();

        for (int i = 0; i < Config.INITIAL_ENTITIES_COUNT; ++i) {
              EntitiesManager.getInstance().addEntity(new Bug());
        }
        Field.getInstance().setFoodConfigType(foodType);
        Field.getInstance().restart();

        btnStopPlay.setText("Stop");

        frame.timerStart();
        Field.getInstance().timerStart();
    }


    /**
     * Generates restart event.
     * @param engine
     */
    public static void generateRestartEvent(){
        engine.getEventManager().handle(
                  new Event(ScreenEventProvider.E1),
                  StateMachineContextImpl.create());
    }

    /**
     * Generates event for stopping or starting visualization.
     * @param engine
     */
    public static void generateStopStartEvent(){
        engine.getEventManager().handle(
                  new Event(ScreenEventProvider.E2),
                  StateMachineContextImpl.create());
    }

    /**
     * Generates event which makes visualization slower.
     * @param engine
     */
    public static void generateSlowerEvent(){
        engine.getEventManager().handle(
                  new Event(ScreenEventProvider.E3),
                  StateMachineContextImpl.create());
    }

    /**
     * Generates event which makes visualization faster.
     * @param engine
     */
    public static void generateFasterEvent(){
        engine.getEventManager().handle(
                  new Event(ScreenEventProvider.E4),
```

```
                StateMachineContextImpl.create());
}

/**
 * Generates event which changes the food configuraion.
 * @param engine
 */
public static void generateDoneEvent(){
     engine.getEventManager().handle(
                new Event(ScreenEventProvider.E6),
                StateMachineContextImpl.create());
}

}
```

EntityTableModel.java

```
package gui;

import javax.swing.table.AbstractTableModel;

import common.EntitiesManager;
import common.Entity;

/**
 * Represents TableModel for entities information table.
 *
 * @author Goonich Ivan, Irinev Anton
 */
public class EntityTableModel extends AbstractTableModel {
private static final long serialVersionUID = 1L;

private static final String[] name = new String[] {
     "ID", "X", "Y", "Energy", "Steps"};

public String getColumnName(int columnIndex) {
     return name[columnIndex];
}

public int getRowCount() {
     return EntitiesManager.getInstance().getEntitiesCount();
}

public int getColumnCount() {
     return name.length;
}

public Object getValueAt(int rowIndex, int columnIndex) {
     Entity entity = EntitiesManager.getInstance().getEntity(rowIndex);
```

```
        switch (columnIndex) {
        case 0:
             return entity.getID();
        case 1:
             return entity.getX();
        case 2:
             return entity.getY();
        case 3:
             if (entity.isDying()) {
                  return "None";
             } else {
                  return entity.getEnergy();
             }
        case 4:
             return entity.getStepsCount();
        default:
             return null;
        }
}
}
```

FieldPanel.java

```
package gui;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Graphics2D;
import javax.swing.JPanel;
import java.util.ArrayList;

import common.EntitiesManager;
import common.Entity;
import common.Config;
import common.Field;

/**
 * Represents panel which contains game field.
 *
 * @author Goonich Ivan, Irinev Anton
 */
public class FieldPanel extends JPanel {
private static final long serialVersionUID = 1L;
private final int shiftX = 15;
private final int shiftY = 6;

public FieldPanel() {
     Dimension size = new Dimension(
          Config.CELL_SIZE * Config.FIELD_WIDTH + shiftX*2,
          Config.CELL_SIZE * Config.FIELD_HEIGHT + shiftY*2
     );
```

```
        this.setSize(size);
        this.setPreferredSize(size);
        this.setDoubleBuffered(true);
        this.setBackground(Color.BLACK);
    }

    public void drawCircle(double x, double y, double r, Graphics2D g) {
        g.fillArc((int)(x - r) + shiftX, (int)(y - r) + shiftY,
                  (int) (r * 2), (int) (r * 2), 0 , 360);
    }

    public void drawFood(Graphics2D g) {
        g.setColor(Color.GREEN);
        for (int i = 0; i < Config.FIELD_HEIGHT; ++i) {
            for (int j = 0; j < Config.FIELD_WIDTH; ++j) {
                if (Field.getInstance().isFoodInCell(i, j)) {
                    drawCircle((i + 0.5) * Config.CELL_SIZE, (j + 0.5)
                            * Config.CELL_SIZE, Config.CELL_SIZE / 2.5, g);
                }
            }
        }
    }

    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D graphics2D = (Graphics2D) g;

        drawFood(graphics2D);
        ArrayList<Entity> entities
            = EntitiesManager.getInstance().getEntities();

        graphics2D.setColor(Color.WHITE);
        g.drawLine(
            0 + shiftX,
            0 + shiftY,
            Config.FIELD_WIDTH * Config.CELL_SIZE + shiftX, 0 + shifty
        );
        g.drawLine(
            Config.FIELD_WIDTH * Config.CELL_SIZE + shiftX,
            0 + shiftY,
            Config.FIELD_WIDTH * Config.CELL_SIZE + shiftX,
            Config.FIELD_HEIGHT * Config.CELL_SIZE + shifty
        );
        g.drawLine(
            Config.FIELD_WIDTH * Config.CELL_SIZE + shiftX,
            Config.FIELD_HEIGHT * Config.CELL_SIZE + shiftY,
            0 + shiftX,
            Config.FIELD_HEIGHT * Config.CELL_SIZE + shifty
        );
        g.drawLine(
            0 + shiftX,
            Config.FIELD_HEIGHT * Config.CELL_SIZE + shiftY,
            0 + shiftX,
```

```
                0 + shifty
        );

        for (Entity entity : entities) {
                drawEntity(entity, graphics2D);
        }
}

public void drawEntity(Entity entity, Graphics2D g) {
        double screenX = (entity.getX() + 0.5) * Config.CELL_SIZE;
        double screenY = (entity.getY() + 0.5) * Config.CELL_SIZE;
        double screenOldX = (entity.getOldX() + 0.5) * Config.CELL_SIZE;
        double screenOldY = (entity.getOldY() + 0.5) * Config.CELL_SIZE;
        double midX = (screenX + screenOldX) / 2;
        double midY = (screenY + screenOldY) / 2;

        if (entity.isDying()) {
                int red = 255 - entity.getDyingStepCounter() * 5;
                if (red < 0) red = 0;
                g.setColor(new Color(red, 0, 0));

                drawCircle(screenOldX, screenOldY, Config.CELL_SIZE / 2.7, g);

                if (Math.abs(entity.getX() - entity.getOldX()) <= 1
                        && Math.abs(entity.getY() - entity.getOldY()) <= 1) {
                        drawCircle(midX, midY, Config.CELL_SIZE / 2.1, g);
                }
        } else {
                g.setColor(Color.LIGHT_GRAY);
                drawCircle(screenOldX, screenOldY, Config.CELL_SIZE / 2.7, g);

                // checking if entity is out of border of the field
                if (Math.abs(entity.getX() - entity.getOldX()) <= 1
                        && Math.abs(entity.getY() - entity.getOldY()) <= 1) {
                        drawCircle(midX, midY, Config.CELL_SIZE / 2.1, g);
                }

                g.setColor(Color.WHITE);
        }
        //draw head
        drawCircle(screenX, screenY, Config.CELL_SIZE / 2, g);
}
}
```

MainFrame.java

```
package gui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.FlowLayout;
```

```
import java.awt.Insets;
import java.awt.Toolkit;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTable;
import javax.swing.Timer;

import bugs.Bug;

import common.Config;
import common.EntitiesManager;

/**
 * Represents main frame of the program.
 *
 * @author Goonich Ivan, Irinev Anton
 */
public class MainFrame extends JFrame {

private static final long serialVersionUID = 1L;

public final FieldPanel fieldPanel;
private ControlPanel controlPanel;
private JPanel rightPanel;

private JSplitPane splitPanel;
private JPanel infoPanel;
private JTable infoTable;
//private final MainFrame frame = this;

private Timer timer = null;

private ActionListener taskPerformer = new ActionListener() {
     public void actionPerformed(ActionEvent evt) {
          fieldPanel.repaint();
          infoTable.revalidate();
               infoPanel.repaint();
          EntitiesManager.getInstance().handleExternalTimerEvent();
     }
};

private void start() {
        for (int i = 0; i < Config.INITIAL_ENTITIES_COUNT; ++i) {
          EntitiesManager.getInstance().addEntity(new Bug());
     }
```

```
        this.setVisible(true);

     timer = new Timer(Config.ENTITY_STEP_DELAY, taskPerformer);
     timer.start();
}

public void timerStart() {
     if (!timer.isRunning()) {
          timer.start();
     }
}

public void timerStop() {
     if (timer.isRunning()) {
          timer.stop();
     }
}

public void setTimerDelay(int delay) {
     timer.setDelay(delay);
}

public MainFrame() {
     super("Evolution Model v1.0");

    fieldPanel = new FieldPanel();

     controlPanel = new ControlPanel(this);

    rightPanel = new JPanel();
    rightPanel.setLayout(new FlowLayout());
    rightPanel.add(fieldPanel);
    rightPanel.add(controlPanel);

     infoTable = new JTable(new EntityTableModel());
     infoTable.getColumnModel().getColumn(0).setMaxWidth(50);
     infoTable.getColumnModel().getColumn(0).setMinWidth(30);
     infoTable.getColumnModel().getColumn(1).setPreferredWidth(500);
     infoTable.getColumnModel().getColumn(2).setPreferredWidth(500);
     infoTable.getColumnModel().getColumn(3).setPreferredWidth(500);
     infoTable.getColumnModel().getColumn(4).setPreferredWidth(500);

     infoTable.getTableHeader().setReorderingAllowed(false);
     infoTable.setColumnSelectionAllowed(false);
     infoTable.setSelectionMode(0);
     infoTable.setRowHeight(15);

     GridBagLayout gbl = new GridBagLayout();
     gbl.columnWeights = new double[] {1.0};
     gbl.rowWeights = new double[] {1.0};
     infoPanel = new JPanel(gbl);
     infoPanel.add(
          new JScrollPane(infoTable),
```

```
            new GridBagConstraints(0, 0, 1, 1, 1, 1,
                GridBagConstraints.CENTER,
                GridBagConstraints.BOTH,
                new Insets(0, 0, 0, 0), 0, 0)
        );

        splitPanel = new JSplitPane(
            JSplitPane.HORIZONTAL_SPLIT,
            infoPanel,
            rightPanel
        );
        splitPanel.setOneTouchExpandable(false);
        splitPanel.setDividerLocation(300);
        splitPanel.setEnabled(false);

        Toolkit tKit = Toolkit.getDefaultToolkit();
        Dimension size = new Dimension(
            Config.CELL_SIZE * Config.FIELD_WIDTH + 350,
            Config.CELL_SIZE * Config.FIELD_HEIGHT + 50 + 40
        );
        Dimension screen_size = tKit.getScreenSize();
        this.getContentPane().add(splitPanel, BorderLayout.CENTER);
        this.setSize(size);
        this.setLocation(
            (screen_size.height - size.height) / 2,
            (screen_size.width - size.width) / 2
        );
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        start();
    }
}
```

ScreenEventProvaider.java

```
package gui;

import com.evelopers.common.exception.CommonException;
import com.evelopers.unimod.core.stateworks.Event;
import com.evelopers.unimod.runtime.EventProvider;
import com.evelopers.unimod.runtime.ModelEngine;
import com.evelopers.unimod.runtime.context.StateMachineContextImpl;

public class ScreenEventProvider implements EventProvider {
/**
```

```
 * @unimod.event.descr Restart button was pressed
 */
public static final String E1 = "e1";

/**
 * @unimod.event.descr Stop/Start button was pressed
 */
public static final String E2 = "e2";

/**
 * @unimod.event.descr Slower button was pressed
 */
public static final String E3 = "e3";

/**
 * @unimod.event.descr Faster button was pressed
 */
public static final String E4 = "e4";

/**
 * @unimod.event.descr Program was terminated
 */
public static final String E5 = "e5";

/**
 * @unimod.event.descr Food configuration type was selected
 */
public static final String E6 = "e6";

public void init(ModelEngine engine) throws CommonException {
}

public void dispose() {
}
}
```