Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Кафедра “Компьютерные технологии”

М.И. Гаврилов, М.Д. Медвинский, А.А. Шалыто

Система управления объектами в играх типа “Стратегия”

Программирование с явным выделением состояний

Проектная документация

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

Санкт-Петербург
2004

Содержание

. . . доделывать, переделывать,
начинать с начала придется
нам еще не раз. . .

В.И. Ленин

1 Введение

В данной работе исследуется эффективность применения технологии автоматного программирования [1] для организации процесса управления пользователем большим числом объектов для известного класса игр типа “Стратегия” [2]. Применение автоматного программирования позволило существенно упростить как процесс разработки и отладки, так и понимание принципа построения используемых моделей. Отметим, что авторы не ставили своей целью разработать “искусственный интеллект” компьютера для управления игрой и ограничились лишь созданием системы управления объектами, предназначенной для человека.

1.1 Что такое игры типа “Стратегия”?

“Стратегия” - это игра, рассчитанная на нескольких игроков, в которой каждый игрок имеет возможность управлять целой инфраструктурой. За некоторых игроков может играть компьютер. Под руководством игрока находятся, например, ресурсы, рабочие, фабрики и воины различных типов. Все эти объекты в игре находятся примерно в следующих отношениях:

- рабочие могут добывать ресурсы, расходуя время;
- рабочие могут строить здания, расходуя ресурсы и время;
- в различных зданиях можно производить рабочих, производить различных воинов, совершенствовать воинов, изучать технологии для производства новых “видов” воинов, расходуя время и ресурсы;
- некоторые здания предназначены для обороны, они могут уничтожать воинов врага, расходуя время;
- воины могут разрушать здания врага и убивать его воинов (а могут и свои), расходуя время;
- некоторые более продвинутые воины могут накладывать, так называемые, “заклинания” на воинов врага (например, замедление), что ухудшает их боевые характеристики. На это расходуется некоторая энергия воина, накапливающаяся со временем.

Таким образом, имея в начале игры всего несколько рабочих и несколько зданий, можно развить достаточно сильную инфрастуктуру, построить большую армию и уничтожить врага. Можно, конечно, и самому оказаться уничтоженным. Таким образом, цель игры - развиться, построить армию быстрее, чем враг, и уничтожить его. Уделяя внимание скорости развития, следует помнить о тактике ведения боя и умении побеждать не количеством, а качеством. В подобного рода играх это очень актуально, поскольку разнообразие воинов и технологий дает иногда возможность небольшой, но

“умной” армией, разгромить большую, но “тупую” армию. Поясним сказанное примером.

В большинстве игр боевые единицы можно разделить на несколько типов по способу передвижения, например, на летающих и на наземных. Летающие единицы могут перемещаться беспрепятственно и при этом, как правило, с большей скоростью. Наземные единицы - только по земле, и поэтому могут встретить препятствия. Каждая боевая единица (независимо от способа передвижения) обладает такими боевыми характеристиками, как способность или неспособность атаковать наземные и воздушные единицы врага. Как правило, если боевая единица обладает возможностью атаковать лишь какой-то один тип войск, то это компенсируется, например, большей силой повреждения по сравнению с более универсальными единицами.

При атаке врага, следует учитывать наличие у него различных видов войск, и поэтому следует проводить атаку, используя боевые единицы, умеющие атаковать как по земле, так и по воздуху. Если, допустим, игрок решил провести атаку мощными наземными войсками, умеющими атаковать только по земле (надеясь на отсутствие у врага авиации), то его войска могут серьезно пострадать, поскольку небольшая армия врага из нескольких летающих единиц может разгромить все его дорогостоящее войско. Если, наоборот, враг строит защиту в основном против авиации, то, обладая сведениями разведки о его действиях, можно небольшой сухопутной армией нанести мощный удар.

Это лишь самый простой пример из практики. В некоторых играх боевые единицы классифицируются по гораздо большему числу характеристик. При этом сражения могут стать очень интересными для изучения.

1.2 Пример игры типа “Стратегия”

Для пояснения принципов игр типа “Стратегия” рассмотрим прототип создаваемой игры - игру “Starcraft” (”Звездное мастерство”), созданную компанией ”Blizzard” [3]. Это игра была выпущена в 1998 г. и, несмотря на свой возраст, до сих пор является почти классической среди игр подобного жанра. Посмотрим, как внешне выглядит эта игра (рис. 1).

Все объекты игры размещаются на, так называемой, “карте”. Она содержит информацию о том, какие участки земли пригодны для строительства, какие пригодны для передвижения наземных объектов и т. п. В левом нижнем углу (область 9) можно видеть схематичное изображение всей карты.

На карте отражены следующие объекты:

- 1 - главное здание, предназначенное для производства рабочих;
- 2 - рабочий, добывающий ресурс “минералы”;
- 3 - “минералы”;
- 4 - здание, предназначенное для добычи другого вида ресурсов - “газа”;

- 5 - здание, обеспечивающее возможность производства некоторого числа рабочих и воинов в игре;
- 6 - здание, предназначенное для производства некоторых наземных воинов;
- 7 - здание, предназначенное для производства воздушных боевых единиц;
- 8 - воздушная боевая единица “самолет”.

Рис. 1. Внешний вид игры “StarCraft”

Обратим внимание на то, что в данный момент времени объект ”самолет” выделен игроком (область 8). Это означает, что игрок может получить подробную информацию о состоянии объекта. Непосредственно под самолетом в виде процентных линеек изображаются такие характеристики, как “здоровье” (или ”целостность”, “жизненная сила”) объекта и количество энергии. Используя область 10, можно получить более подробную информацию - на ней представлено более крупное изображение объекта, его название, количество убитых вражеских единиц и различные технические характеристики (сила атаки по воздуху, по земле и сила брони).

Наиболее важной является область 11. Именно с помощью панели управления, размещенной в этой области, игроком осуществляется управление объектом.

Управление объектами в игре происходит с помощью приказов, которые периодически объект получает от игрока. В большинстве случаев сам объект не может изменить значение выполняемого им приказа.

Почти для всех объектов игры общими являются следующие четыре приказа: “Стоп”, “Двигаться”, “Атаковать” и “Держать местность”. Автомат, моделирующий поведение объекта в игре, создан на основании лишь некоторого набора правил, полученных из практики, поскольку авторам не известен автомат, использовавшийся при написании данной игры, и неизвестно даже, использовались ли автоматы при ее создании. Данный набор правил приведен в разд. 6.1 в описании автомата “Юнит[1]”(A0).

Команды передаются объекту посредством нажатия одной из кнопок на панели управления. Используя такой способ, можно проводить довольно сложные боевые операции с большим числом боевых единиц.

В задачу работы входит создание упрощенной игры, состоящей из панели управления, боевых единиц и других объектов, с использованием автоматного подхода.

2 Автоматный подход

2.1 Внешний вид проекта

Созданная модель игры сильно упрощена по сравнению с игрой “StarCraft”. Пользователю предлагается попробовать управлять объектами сразу двух враждующих команд. Внешне разработанная игра выглядит следующим образом (рис. 2).

На экране можно наблюдать простую “карту”, на которой расположены самолеты и радары. Самолеты являются боевыми единицами, и, поэтому, они могут передвигаться и атаковать. Радары в некотором смысле “картонные”, они ничего не могут делать, правда, и уничтожить их нельзя. Радары и самолеты разных команд различаются цветами - красным и синим.

Внизу расположена панель управления с четырьмя кнопками и индикатором состояния, показывающим характеристики “Здоровье” и “Повреждение” управляемого объекта.

[1] В данной работе русским аналогом английского слова “unit” можно считать словосочетание “боевая единица” так же, как и в русской версии игры “StarCraft”.

Рис. 2. Внешний вид игры

2.2 Особенности объектно-ориентированной модели

2.2.1 Выбор объектно-ориентированного языка

Для разработки игры был выбран всесторонне использующийся в настоящее время объектно-ориентированный язык *Java* [4]. Главная сложность при программировании на объектно-ориентированном языке - правильно продумать объектную модель задачи. При этом следует определить, какие объекты следует использовать в программе и в каких отношениях эти объекты должны находиться.

Проблема использования ”объектно-ориентированных возможностей” языков программирования ранее уже затрагивалась в работах по автоматному программированию, например в работе [5]. Используемая в настоящей работе модель за счет указанных возможностей может быть достаточно легко распространена на более общие случаи, что является большим преимуществом такого подхода.

Поясним это. Сложность построения модели для данной игры связана с большим числом используемых в игре объектов и наличием явных связей

между ними. В игре “StarCraft”, выбранной в качестве прототипа, число объектов и связей еще больше. Поскольку при создании упрощенной игры авторы всегда должны предусматривать возможность ее развития, то объектная модель также должна создаваться в расчете на дальнейшее расширение.

2.2.2 Общие требования к объектам

Перечислим основные положения, которыми руководствовались авторы при создании объектно-ориентированной модели игры.

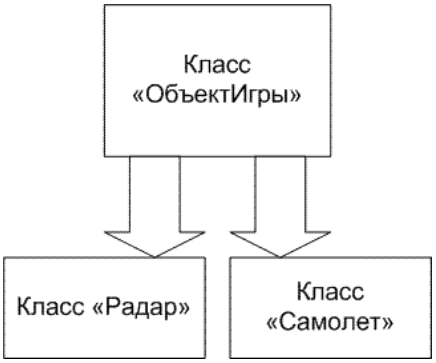

Рис. 3. Фрагмент схемы наследования для классов “радар” и “самолет”

Рис. 4. Фрагмент схемы наследования для усовершенствованной игры

1. Радары и самолеты являются однотипными объектами. Поясним это. Эти объекты имеют много общих характеристик (например, "Здоровье", "Область видимости"и т.д.), имеют способность перерисовываться на карте

и могут выступать в качестве цели для атаки. Поэтому логично было бы создать общего предка “объект игры” для классов, реализующих объекты “радар” и “самолет” (рис. 3).

На рис.4 показано, как мог бы усложниться фрагмент схемы наследования при усовершенствовании игры.

2. Классы, реализующие автоматы, которые управляют различными объектами, имеют похожую структуру. Поэтому логично по аналогии с работой [6] реализовать абстрактный базовый класс “Автомат”, для которого конкретные автоматы будут являться наследниками. При этом каждый автомат реализуется отдельным от управляемого им объекта классом (рис.5).

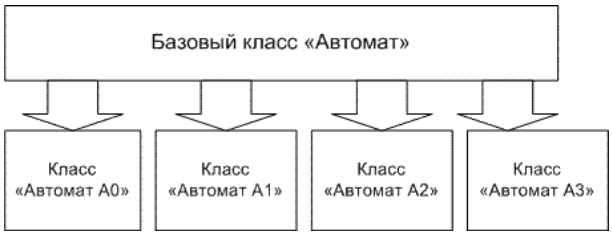

Рис. 5. Фрагмент схемы наследования для классов, реализующих автоматы

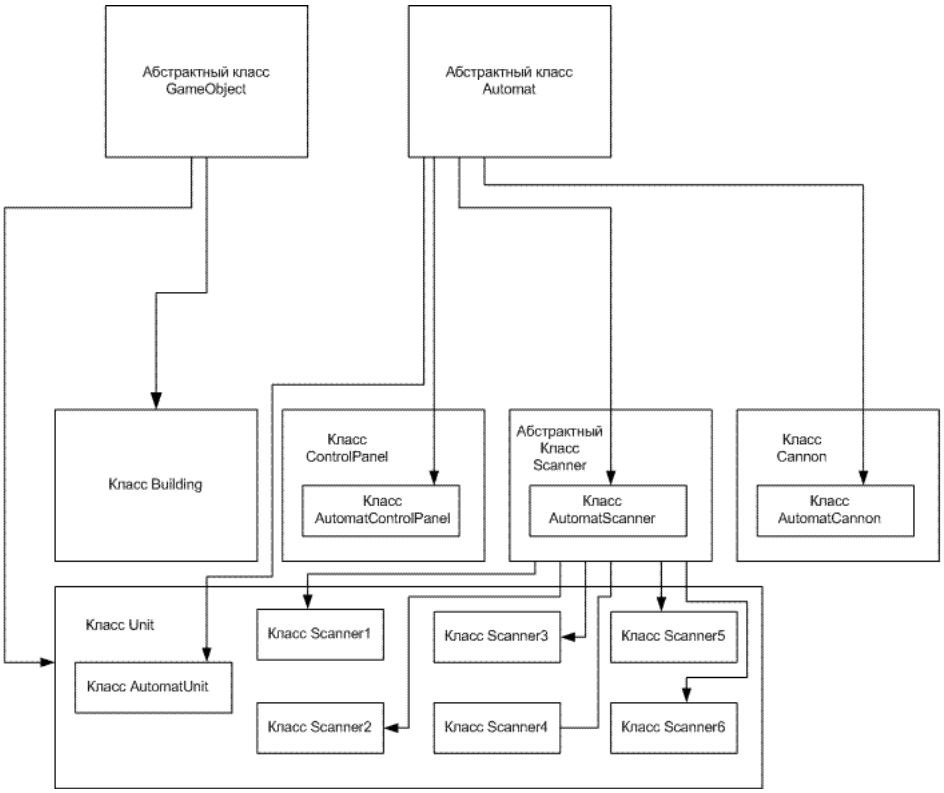

Рис. 6. Схема наследования для игры

3. Класс, реализующий автомат, обращается к полям класса, реализующего управляемый им объект. Это вызвано тем, что переходы автомата

из состояния в состояние так или иначе зависят от физического состояния объекта. Поэтому класс, реализующий автомат, логично описать внутри класса, реализующего объект, и, тем самым, дать ему свободный доступ к полям самого объекта.

2.2.3 Структура классов

Схема наследования на рис.6 объединяет схемы, приведенные на рис. 3, 5. Наличие стрелки между классами означает наследование. Если изображение некоторого класса помещено внутрь изображения другого, то это означает, что первый класс описан внутри второго класса, и тем самым имеет доступ ко всем его полям. Как следует из этой схемы, четыре класса программы соответствуют объектам, управляемым автоматами. Подробнее эти классы и их взаимодействие между собой описаны ниже.

На рис.7 изображена диаграмма связей классов, показывающая, какие классы обрабатывают события, поступающие от пользователя, какие отвечают за перерисовку и т.п.

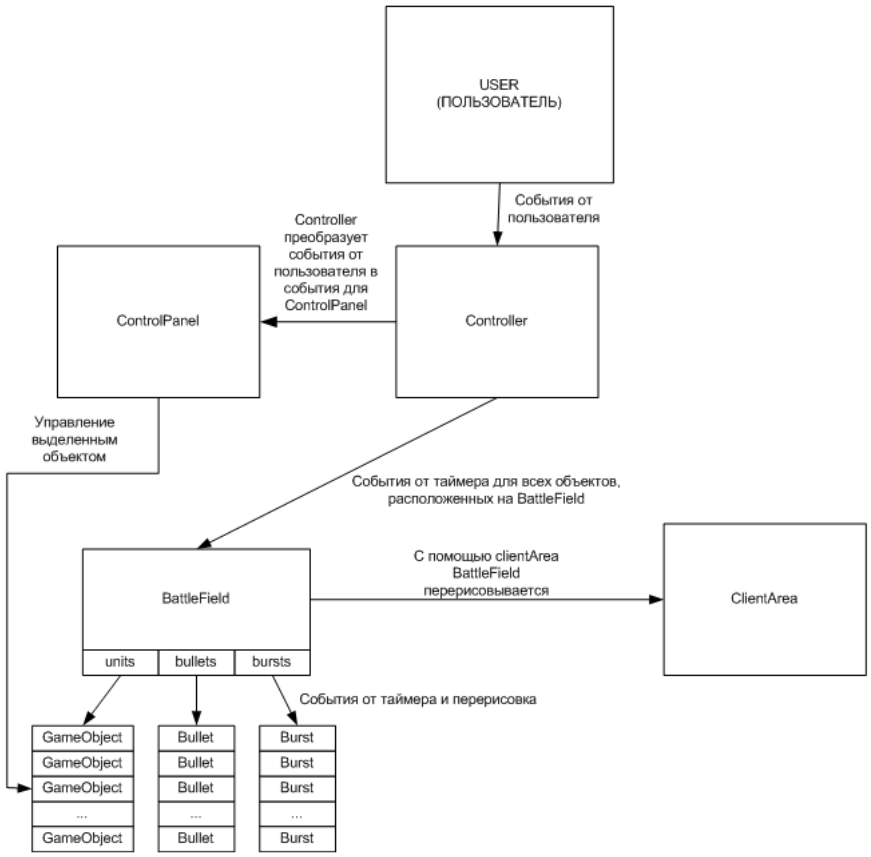

Рис. 7. Диаграмма связей классов

Из диаграммы связей классов следует, что все события, поступающие от

пользователя (нажатия клавиш и кнопок мыши), обрабатывает отдельный объект `Controller`.

Этот объект распределяет события между объектами `ControlPanel` (панель управления) и `BattleField` (поле боя). Объект `BattleField` содержит в себе векторы, состоящие из объектов `GameObject` (объект игры), `Bullet` (снаряд) и `Burst` (взрыв).

Перерисовка объектов происходит с помощью объекта `ClientArea` (клиентская область).

Объект `ControlPanel` управляет текущим выделенным объектом `GameObject`.

2.3 Схема связей автоматов

В программе управление объектами реализовано с помощью четырех типов автоматов, управляющих сканерами, самолетами, пушками и панелью управления. При этом каждый автомат управляет только одним объектом. В качестве примера приведем схему связей автоматов для управления тремя самолетами с помощью двух сканеров (рис. 8). В реальности самолетов может быть сколько угодно, а сканеров - шесть.

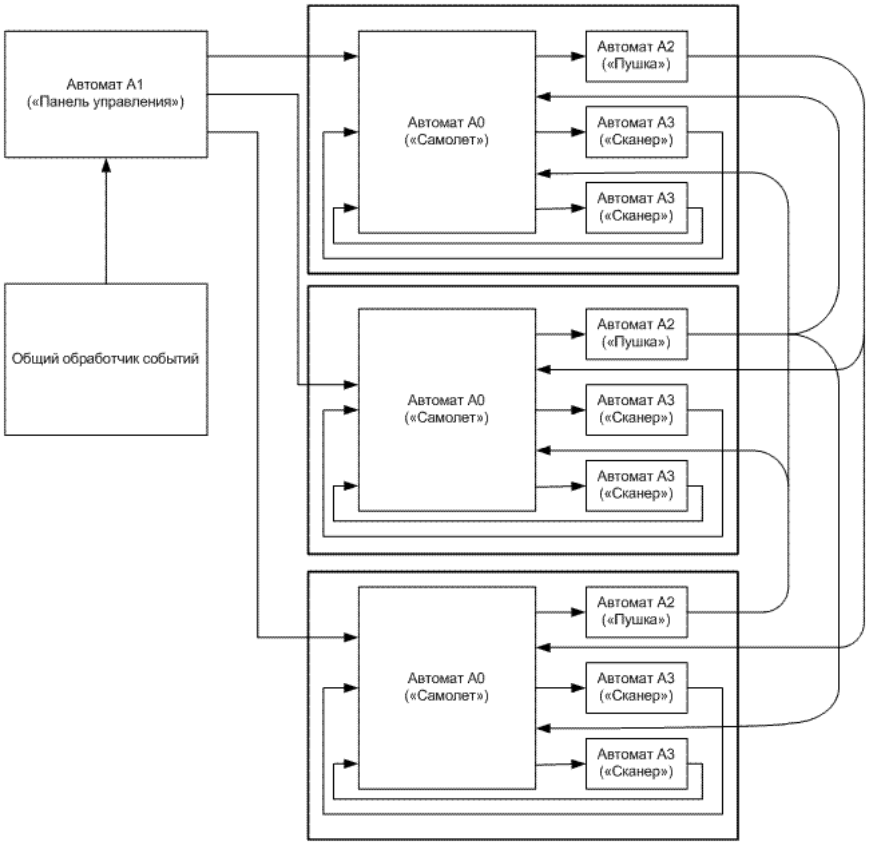

Рис. 8. Схема связей автоматов

Набор автоматов, объединенных в группу контуром, соответствует набору физических устройств, расположенных на борту одного самолета.

Как показано на схеме на рис. 6, классы, реализующие автоматы, “запрятаны” внутрь классов, реализующих соответствующие им объекты. Поэтому, передача события от одного автомата другому происходит следующим образом. Положим, автомату A нужно послать событие автомату B, управляющего объектом, который реализуется классом C. Для всех классов, кроме класса C, класс, реализующий автомат B, недоступен. В этом случае в классе C должен быть определен некоторый неавтоматный метод с ”говорящим” названием (например, orderAttack() - приказАтаковать()), который непосредственно передает нужное событие автомату B.

Автомату A необходимо только вызвать этот метод.

Такой способ реализации взаимодействия автоматов делает программу более читаемой и простой для понимания.

Ниже приведено подробное описание самых важных классов и автоматов игры.

Автоматы разрабатывались не в том порядке, в котором следует их описывать, поэтому не стоит удивляться тому, что, например, описание автомата A3 изложено раньше описания автомата A0.

3 Класс “Automat”

3.1 Словесное описание

Абстракный класс “Automat” является базовым для всех автоматов, управляющих объектами игры, и, поэтому, содержит в себе все их общие поля и методы.

3.2 Структурная схема класса

На рис. 9 приведена структурная схема класса “Automat”.

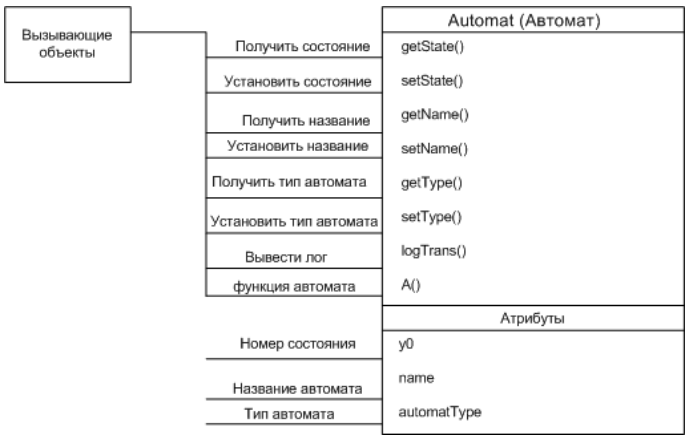

Рис. 9. Схема класса “Automat”

Схемы всех классов, унаследованных от класса “Automat”, отличаются лишь наличием функций-входов xi() и функций-выходов zi(). Поэтому далее не будем приводить схемы классов, реализующих автоматы.

4 Классы “Scanner” и “AutomatScanner”

4.1 Словесное описание

Класс “Scanner” представляет собой объект, управляемый автоматом “Сканер” (A3), который имеет всего два состояния - “Включен” и “Выключен”. Задача этого устройства — постоянно сканировать местность или самолет, которому оно принадлежит, на наличие какого-либо свойства и посылать сообщения самолету в зависимости от результата сканирования.

Для нормального функционирования самолета в игре существует шесть видов сканеров, каждый из которых выполняет отдельную задачу. Реализации разнотипных сканеров различаются лишь неавтоматным методом

`run()`, который собственно и сканирует местность или самолет на наличие какого-либо свойства. Поэтому автоматы, реализующие разные сканеры, одинаковы.

4.2 Структурная схема класса “Scanner”

Структурная схема класса “Scanner” приведена на рис.10.

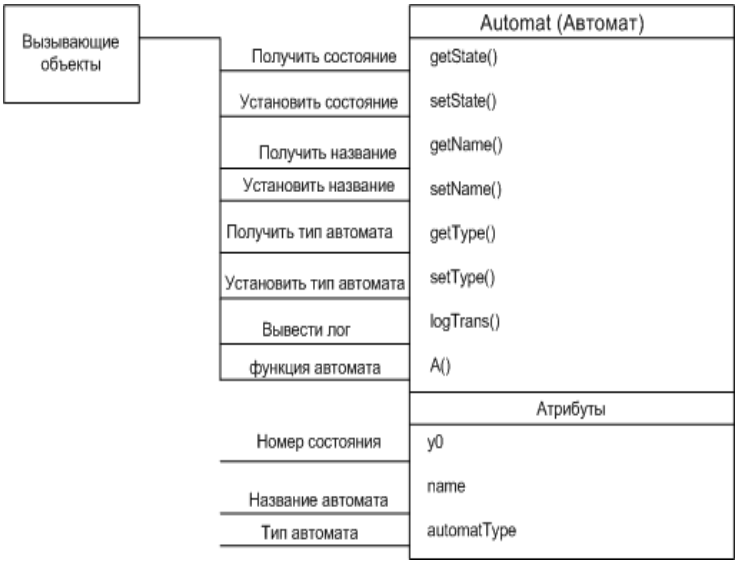

Рис. 10. Схема класса “Scanner”

4.3 Автомат “Сканер” (A3)

4.3.1 Состояния автомата

Здесь и далее в разделах “Состояния автомата” приведены названия состояний автоматов на русском и английском языках. Причиной этому является то, что изображения графов переходов используют русскоязычные названия, а протоколирование автоматов в программе (см. раздел “Протоколирование”) ведется на английском языке.

В следующей таблице приведены русские и английские названия состояний автомата “Сканер” (A3):

№	*Русское название*	*Английское название*
0	Выключен	Switched off
1	Включен	Switched on

4.3.2 Нумерация и описание событий (e)

Для сканера существует всего два события, которые поступают от самолета, являющегося хозяином этого сканера. Перечислим эти события:

e1 - включить сканер. Событие поступает от автомата “Unit” (A0);

e2 - выключить сканер. Событие поступает от автомата “Unit” (A0).

4.3.3 Нумерация и описание выходных воздействий (z)

z1() - произвести сканирование местности или самолета на наличие какого-либо свойства. Это выходное воздействие вызывает неавтоматный метод `run()`.

Описание событий, посылаемых различными сканерами своему самолету, приведено в разд. 6.1 в описании класса “Unit” .

4.3.4 Схема связей автомата “Сканер” (A3)

На рис.11 показана схема связей автомата “Сканер” (A3).

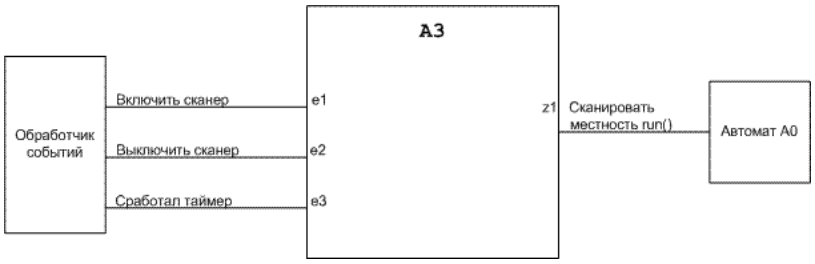

Рис. 11. Схема связей автомата “Сканер” (A3)

4.3.5 Граф переходов автомата “Сканер” (A3)

На рис.12 показан граф переходов автомата “Сканер” (A3).

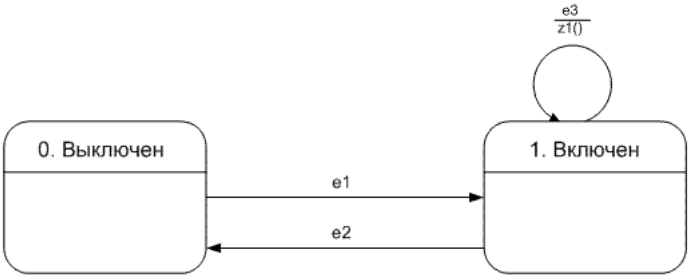

Рис. 12. Граф переходов автомата “Сканер” (A3)

5 Классы “Cannon” и “AutomatCannon”

5.1 Словесное описание

Класс “Cannon” (Пушка) предназначен для реализации модели пушки для самолета.

Основным и довольно естественным требованием к модели является невозможность непрерывного огня — способность пушки стрелять только через некоторые промежутки времени. Таким образом, существует некоторое число, являющееся характеристикой самолета, определяющее минимальный промежуток времени между двумя выстрелами пушки.

Для выполнения этого требования класс “Cannon” содержит таймер, который посылает пушке события “Сработал таймер”. Приказ о начале или прекращении стрельбы поступает к пушке от самолета, которому она принадлежит.

5.2 Структурная схема класса “Cannon”

Структурная схема класса “Cannon” приведена на рис.13.

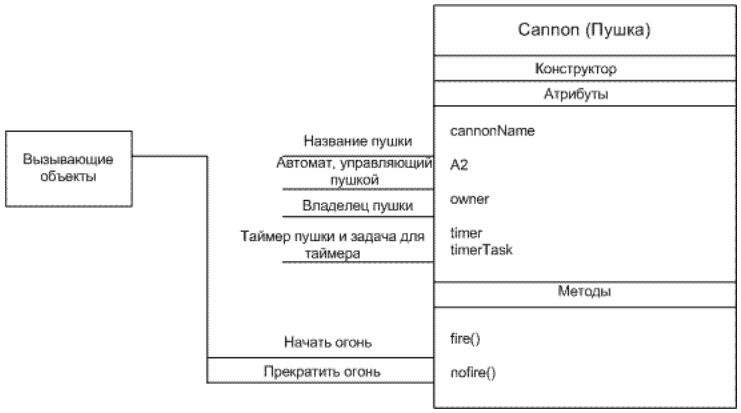

Рис. 13. Схема класса “Cannon”

5.3 Автомат “Пушка” (A2)

5.3.1 Состояния автомата

В следующей таблице приведены русские и английские названия состояний автомата “Пушка” (A2):

№	*Русское название*	*Английское название*
0	Не стреляет. Таймер выключен	Not shooting. Timer is swiched off
1	Стреляет. Таймер включен	Shooting. Timer is swiched on
2	Не стреляет. Таймер включен	Not shooting. Timer is swiched on

5.3.2 Нумерация и описание событий (e)

e1 - включение пушки (начать стрельбу). Событие поступает от автомата “Unit” (A0);

e2 - выключение пушки (прекратить стрельбу). Событие поступает от автомата “Unit” (A0);

e3 - сработал таймер. Событие поступает от таймера.

5.3.3 Нумерация и описание выходных воздействий (z)

z1() - включение таймера. После включения таймер начинает посылать события;

z2() - выпустить ракету по текущей цели самолета;

z3() - выключение таймера. После этого таймер перестает посылать события.

Рис. 14. Схема связей автомата “Пушка” (A2)

5.3.4 Схема связей автомата

На рис.14 показана схема связей автомата “Пушка” (A2).

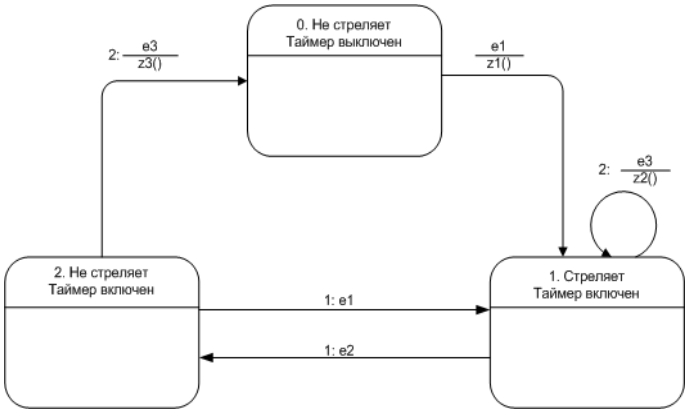

Рис. 15. Граф переходов автомата “Пушка” (A2)

5.3.5 Граф переходов

На рис.15 показан граф переходов автомата “Пушка” (A2).

6 Классы “Unit” и “AutomatUnit”

6.1 Словесное описание

Классы “Unit” (Юнит) и “AutomatUnit” реализуют в игре самолет и автомат, которым он управляется.

Выше было отмечено, что в каждый момент времени самолет выполняет какую-либо директиву. Состояние самолета (в физическом смысле этого слова) напрямую зависит от выполняемой им директивы. Изменение директивы происходит либо по команде игрока, либо в одном из случаев, встречающихся в следующих четырех правилах, которые описывают поведение самолета.

Для определения правил понадобятся два понятия:

- “область видимости” — область на карте, в пределах которой самолет ”видит”;
- “область атаки” — область на карте, в пределах которой самолет может атаковать.

Итак, правила:

1. По команде “Стоп” самолет останавливается. Однако, если в его области видимости появляется вражеская боевая единица, то самолет автоматически меняет выполняемый им приказ на “Атаковать” и начинает атаковать новую цель. Если цель уходит из поля атаки самолет, то начинается погоня за целью. Поэтому команда “Стоп” не гарантирует того, что самолет останется неподвижным;
2. По команде “Двигаться” самолет начинает передвигаться в указанную точку. При достижении ее он автоматически меняет свой приказ на “Стоп”;
3. По команде “Атаковать” самолет начинает атаковать текущую цель. Этот процесс продолжается до тех пор, пока либо сам самолет не умрет, либо не умрет атакуемая цель. В случае гибели атакуемой цели самолет автоматически меняет свой приказ на “Стоп”;
4. По команде “Держать местность” самолет останавливается и до поступления нового приказа от игрока с места больше не сдвигается. При этом самолет атакует все вражеские цели, находящиеся в его поле атаки.

Анализируя перечисленные правила, можно сделать вывод о том, что самолет изменяет свою директиву сам только в случае поступления события от внешней среды. Для того, чтобы правильно организовать процесс

получения этих событий, авторы решили создать автоматы–сканеры. Всего сканеров шесть. Каждый сканер проверяет окружающую местность на наличие какого-либо свойства.

- Сканер 1 проверяет, прибыл ли самолет на место, в случае, если он выполнял директиву “Двигаться”. Если сканер обнаруживает, что самолет действительно прибыл в назначенный пункт, то он посылает сообщение ”Сканер 1: прибыли на место!”. Таким образом, описанное в правиле 2 “самопроизвольное” изменение директивы на “Стоп” становится уже не “самопроизвольным”, а закономерным и происходит при поступлении соответствующего события от сканера 1. При выполнении других директив самолет не нуждается в информации от этого сканера. Поэтому в состояниях автомата, соответствующих директивам, отличным от директивы “Двигаться”, сканер 1 следует выключить.

- Сканер 2 проверяет, есть ли какая-нибудь вражеская цель в поле видимости. В случае нахождения цели сканер посылает событие “Сканер 2: нашел цель в поле видимости!” самолету и изменяет поле `target` самолета на новую цель. Этот сканер позволяет осуществить, например, изменение директивы, описанное в правиле 1.

- Сканер 3 проверяет, не умерла ли текущая цель. В случае ее смерти сканер посылает самолету событие “Сканер 3: цель умерла!”

- Сканер 4 проверяет, находится ли текущая цель в поле атаки. При этом осуществляется проверка, есть ли возможность стрелять по текущей цели, или следует ее догонять. Сканер может посылать два события: ”Сканер 4: цель находится в поле атаки!” и “Сканер 4: цель ушла из поля атаки!”.

- Сканер 5 проверяет, есть ли какая-нибудь вражеская цель в поле атаки. Этот сканер существенно отличается от сканера 2 — он находит цели, которые можно атаковать, не сдвигаясь с места, что соответствует директиве “Держать местность”. Этот сканер так же, как и сканер 2, посылает самолету сообщение ”Сканер 5: найдена цель в поле атаки!” и изменяет поле `target` самолета.

- Сканер 6 проверяет, не умер ли сам самолет — не превосходит ли суммарное повреждение величину характеристики “здоровье”. В случае смерти посылается сообщение “Сканер 6: самолет умер :(”.

Для каждого из состояний автомата определено, какие сканеры включены, а какие выключены.

6.2 Структурная схема класса “Unit”

Структурная схема класа “Unit” приведена на рис. 16.

6.3 Алгоритм движения юнита

Для того, чтобы движение юнита было плавным (т.е. без резких поворотов), после приказа о начале движения юнит начинает двигаться по одной из дуг, соединяющих ценрт масс юнита и точку назначения. Эта дуга расчитывается таким образом, чтобы касательная к ней в центре масс юнита совпадала с направлением его движения. Все необходимые для этого расчеты производятся в методе `move()` класса Unit.

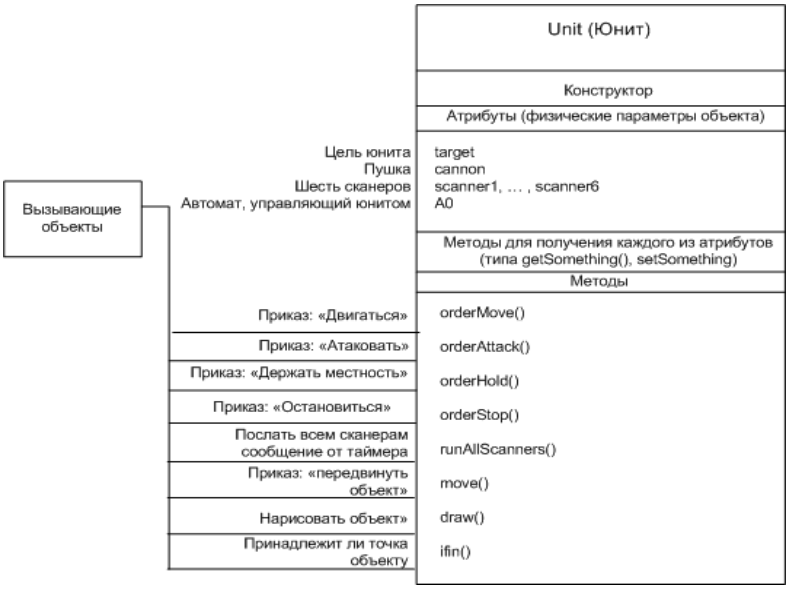

Рис. 16. Схема класса “Unit”

6.4 Автомат “Юнит” (A0)

6.4.1 Состояния автомата

Предлагаемая авторами модель автомата имеет семь состояний, смысл которых понятен из их названий. При этом каждое состояние соответствует паре “элементов” — директиве, выполняемой самолетом, и его физическому состоянию (покой, движение, стрельба). По номеру состояния в любой момент времени можно определить действие, которое следует выполнять самолету.

В следующей таблице приведены русские и английские названия состояний автомата “Юнит” (A0):

№	*Русское название*	*Английское название*
0	Стоп	Stop
1	Движение к месту назначения	Movement to the destination

2	Держать местность (никого не атакует)	Hold position (without attack)
3	Держать местность (атака)	Hold position (with attack)
4	Атака	Attack
5	Движение к цели для атаки	Pursuit of target
6	Умер	Dead

6.4.2 Нумерация и описание событий (e)

e1 – “Приказ: Двигаться”. Событие поступает от автомата “Панель управления” (A1). Событие имеет атрибут — точка, являющаяся пунктом назначения.

e2 – “Приказ: Атаковать”. Событие поступает от автомата “Панель управления” (A1). Событие имеет атрибут — объект, который следует атаковать.

e3 – “Приказ: Держать местность”. Событие поступает от автомата “Панель управления” (A1).

e4 – “Приказ: Стоп”. Событие поступает от автомата “Панель управления” (A1).

e6 – “Сканер 1: прибыли на место!”. Событие поступает от автомата “Сканер 1” (A3).

e7 – “Сканер 2: найдена цель в поле видимости!”. Событие поступает от автомата “Сканер 2” (A3) и имеет атрибут — найденная цель.

e8 – “Сканер 3: цель умерла!”. Событие поступает от автомата “Сканер 3” (A3).

e9 – “Сканер 4: текущая цель в поле атаки!”. Событие поступает от автомата “Сканер 4” (A3).

e10 – “Сканер 4: текущая цель ушла из поля атаки!”. Событие поступает от автомата “Сканер 4” (A3).

e11 – “Сканер 5: найдена цель в поле атаки!”. Событие поступает от автомата “Сканер 5” (A3) и имеет атрибут - найденная цель.

e12 – “Сканер 6: объект “Самолет” умер :(”. Событие поступает от автомата “Сканер 6” (A3).

6.4.3 Нумерация и описание выходных воздействий (z)

Выходные воздействия автомата “Юнит” (A0) действуют только на сканеры и на пушку, включая или выключая их. Порядок включения и выключения автоматов показан на схеме связей.

6.4.4 Схема связей автомата

На рис. 17 показана схема связей автомата “Юнит” (A0).

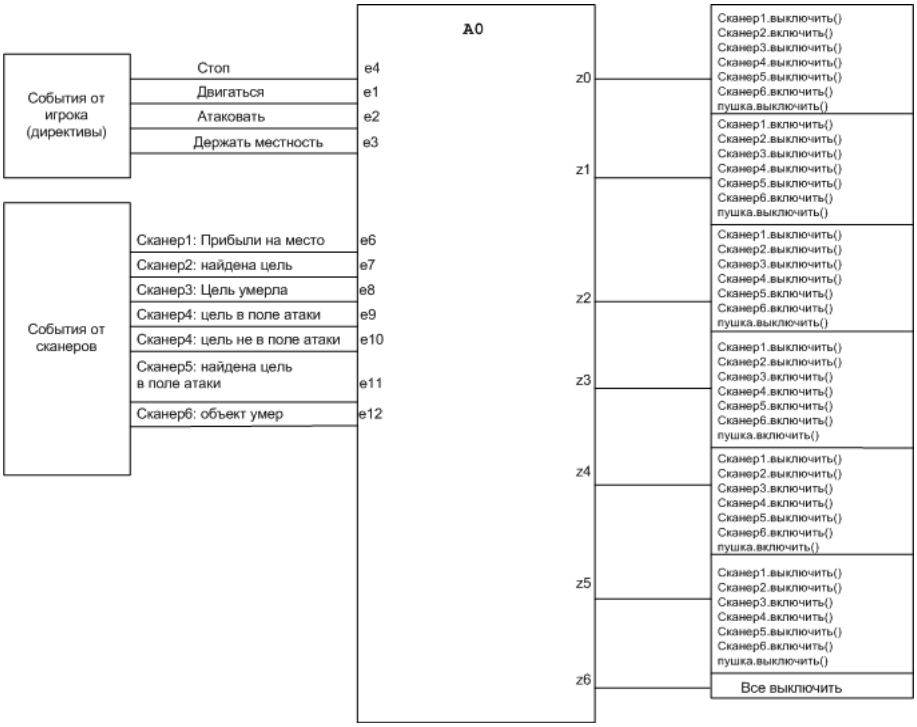

Рис. 17. Схема связей автомата “Юнит” (A0)

6.4.5 Граф переходов автомата

На рис. 18 показан граф переходов автомата “Юнит” (A0).

7 Классы “ControlPanel” и “AutomatControlPanel”

7.1 Словесное описание

Классы “ControlPanel” (Панель управления) и “AutomatControlPanel” предназначены для реализации панели управления самолетом и автомата, которым эта панель управляется. Панель состоит из четырех кнопок и индикатора состояния самолета (рис. 2).

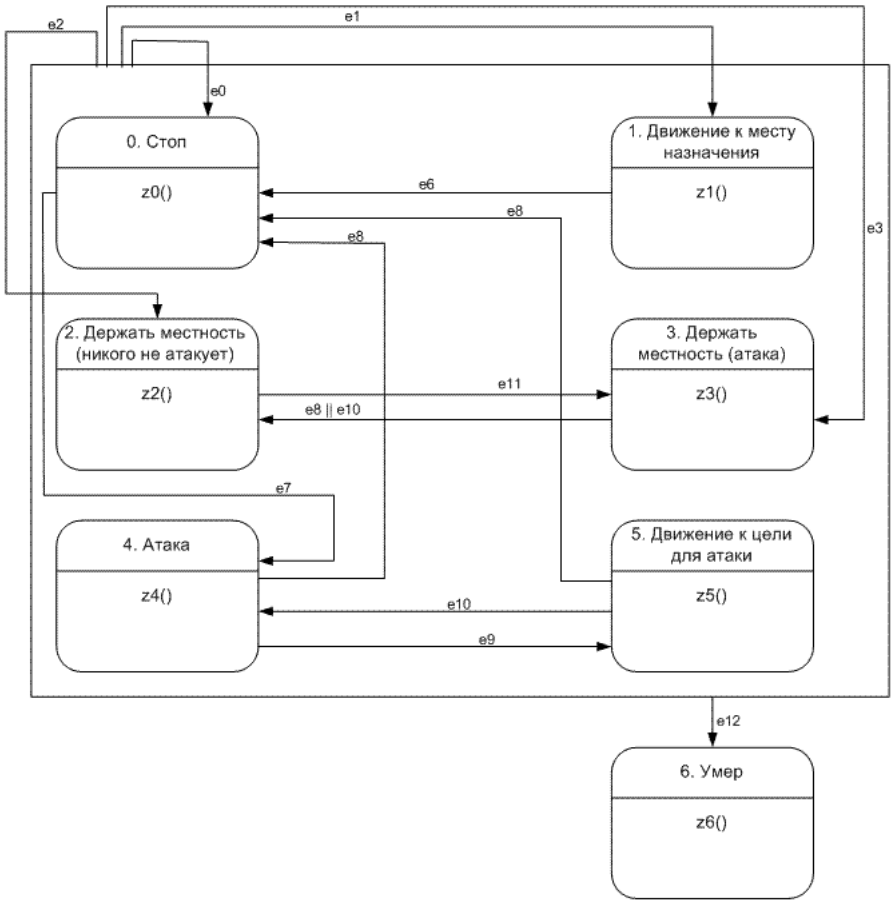

Рис. 18. Граф переходов автомата “Юнит” (A0)

К автомату, реализующему панель управления, предъявляются следующие требования:

- автомат должен реагировать на события, поступающие от игрока, нажимающего на кнопки;
- автомат должен реагировать на изменения состояния самолета.

Про первую группу событий следует сказать, что управление самолетом посредством такой панели подчиняется следующим правилам, по которым устроена панель игры–прототипа:

1. Игрок может переключать управление на любой объект, щелкая по нему левой кнопкой мыши;
2. Игрок может давать приказ “Стоп” выделенному объекту, нажимая соответственно кнопку “Стоп” на панели управления;

3. Игрок может давать приказ “Держать местность” выделенному объекту, нажимая соответственно кнопку “Держать местность” на панели управления;

4. Игрок может давать приказ “Атаковать” выделенному объекту двумя способами. Игрок может нажать кнопку “Атаковать”, а после этого “нажать левой кнопкой мыши”. При этом в процессе ожидания нажатия кнопки мыши на цель, кнопки панели управления должны быть недоступными. Также игрок может отказаться от своего решения, нажав вместо кнопки мыши клавишу ESC. При этом панель должна вернуться в исходное состояние. Игрок также может отдать приказ, просто нажав правой кнопкой мыши на цель, что быстрее и проще.

5. Аналогичным образом игрок может давать приказ “Двигаться” выделенному объекту двумя способами. Игрок может нажать кнопку “Двигаться”, а после этого “нажать левой кнопкой мыши” на точку. При этом в процессе ожидания нажатия кнопки мыши, кнопки панели управления должны быть недоступными. Также игрок может отказаться от своего решения, нажав вместо кнопки мыши клавишу ESC. При этом панель должна вернуться в исходное состояние. Игрок также может отдать приказ, просто нажав правой кнопкой мыши на точку.

Вторая группа событий содержит всего два события: изменение состояния объекта и его смерть.

7.2 Структурная схема класса “ControlPanel”

Структурная схема класса “ControlPanel” приведена на рис. 19.

7.3 Автомат “Панель управления” (A1)

7.3.1 Состояния автомата

Автомат “Панель управления” (A1) имеет семь состояний. Первые шесть состояний (0–5) соответствуют ситуации, при которой выбран объект управления. Смысл этих состояний понятен из названий. Состояние шесть соответствует ситуации, при которой объект управления не выбран.

В следующей таблице приведены русские и английские названия состояний автомата “Панель управления” (A1):

№	*Русское название*	*Английское название*
0	Стоп	Stop
1	Атака	Attack
2	Держать местность	Hold position
3	Двигаться	Move
4	Ожидание выбора цели для атаки	Awaiting of choosing a target to attack

5	Ожидание выбора точки для передвижения	Awaiting of choosing a destination
6	Неактивна	Inactive

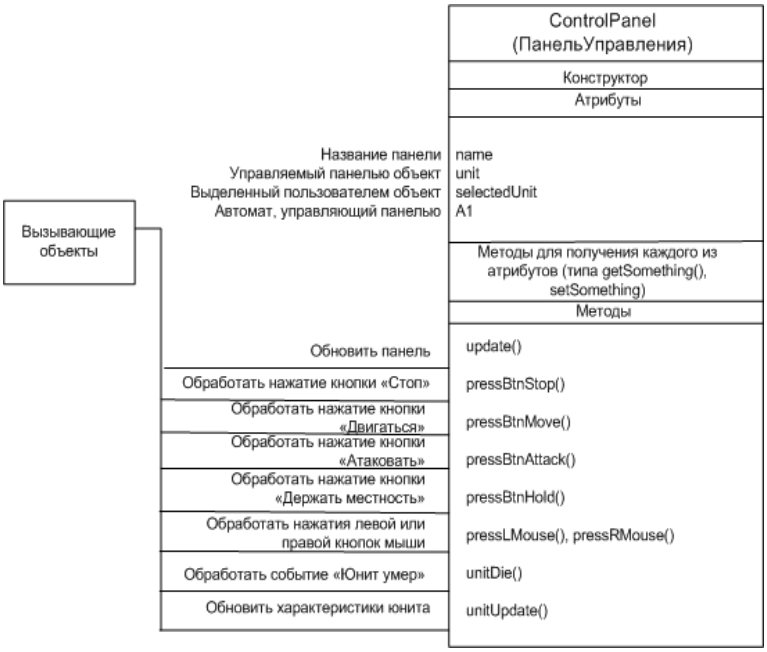

Рис. 19. Схема класса “ControlPanel”

7.3.2 Нумерация и описание событий (e)

e1 - нажатие левой кнопкой на объект. Событие поступает от игрока.

e2 - нажатие левой кнопкой на точку. Событие поступает от игрока.

e3 - нажатие на кнопку “Стоп”. Событие поступает от игрока.

e4 - нажатие на кнопку “Двигаться”. Событие поступает от игрока.

e5 - нажатие на кнопку “Атака”. Событие поступает от игрока.

e6 - нажатие на кнопку “Держать местность”. Событие поступает от игрока.

e7 - нажатие на клавишу ESC. Событие поступает от игрока.

e8 - нажатие правой кнопкой на объект. Событие поступает от игрока.

e9 - нажатие правой кнопкой на точку. Событие поступает от игрока.

e10 - смерть управляемого объекта. Событие поступает от общего обработчика событий в случае смерти управляемого самолета.

e11 - изменение характеристик объекта. Событие поступает от общего обработчика событий в случае уменьшения здоровья самолета или изменения исполняемой им директивы.

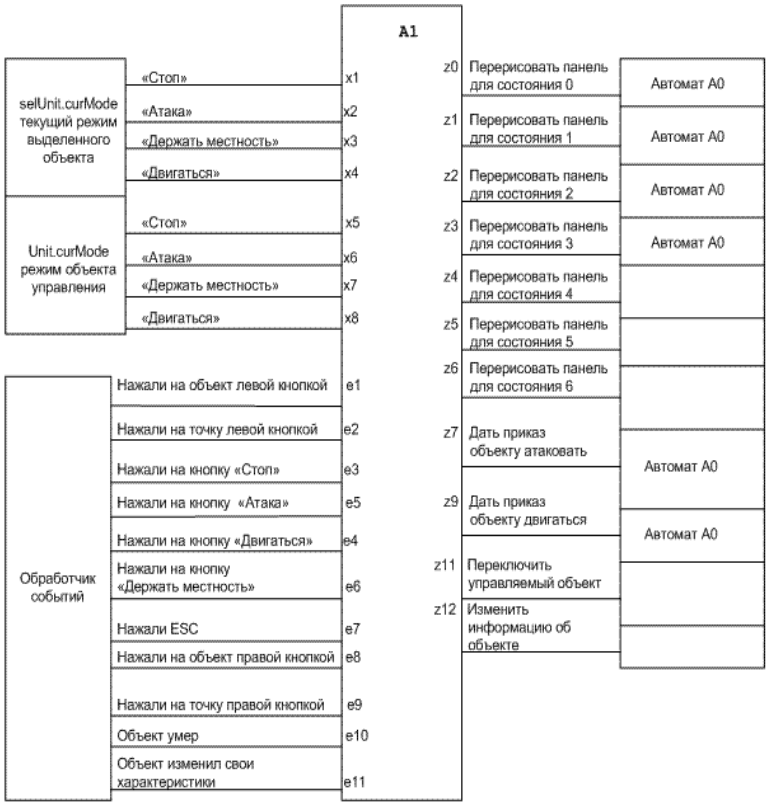

Рис. 20. Схема связей автомата “Панель управления” (A1)

7.3.3 Нумерация и описание выходных воздействий (z)

Смысл выходных воздействий понятен из названий, приведенных на схеме связей автомата.

7.3.4 Схема связей автомата

На рис. 20 показана схема связей автомата “Панель управления” (A1).

7.3.5 Граф переходов автомата

На рис. 21 показан граф переходов автомата “Панель управления” (A1).

8 Описание остальных классов

В качестве описания остальных классов предлагается автоматически сгенерированная *Java* документация. Документация расположена в подкаталоге `DOC` основного каталога программы.

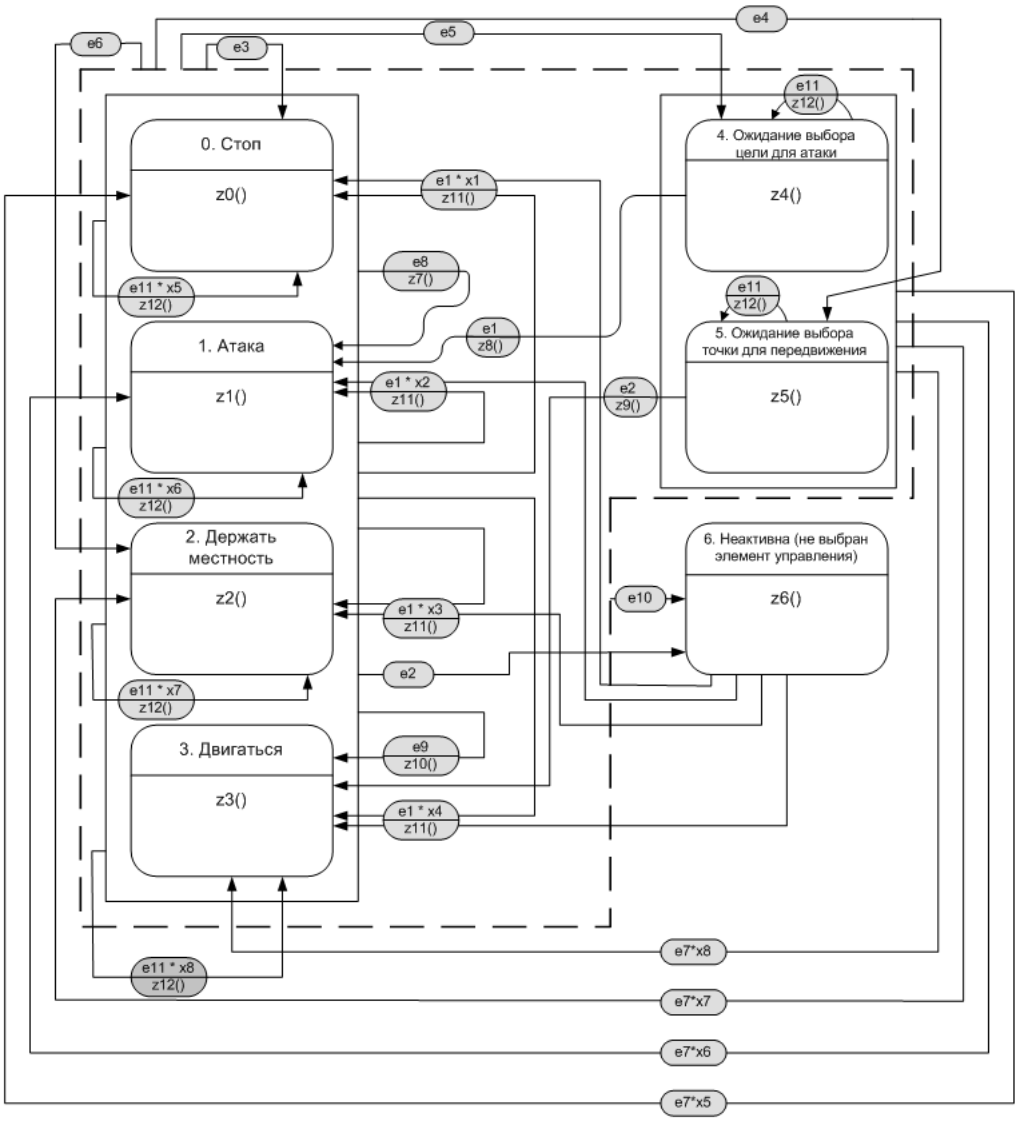

Рис. 21. Граф переходов автомата “Панель управления” (A1)

9 Протоколирование

Для слежения за всеми переходами автоматов в течение работы программы ведется протоколирование. В файл `log.txt` записывается информация об изменениях состояний автоматов. Записи в протоколе разбиваются на три типа.

Первый тип – записи о запуске автомата (содержат символ ‘‘L’’):

```
Tue Jun 01 00:46:11 MSD 2004| L
```

```
A2 <Unit #1 cannon>: launched in state 0
("Not shooting. Timer is swiched off")
```

В начале указывается время и дата перехода, далее указывается тип автомата (в данном случае A2), имя автомата и номер состояния, в котором этот автомат запущен. Таким образом, приведенную строку протокола следует трактовать так:

```
Во вторник, 01 июня 2004, в 0 часов 46 минут
11 секунд автомат <Unit #1 cannon> типа A2
запущен в состоянии 0 ("Не стреляет. Таймер выключен")
```

Второй тип – записи о переходе (содержат символ ‘‘T’’):

```
Tue Feb 10 00:32:21 MSK 2004| T
A0 <Unit #3>: | T A0 <Unit #2>:
has changed state from 0 ("Stop")
to 1 ("Movement to the destination") | event: 1
```

В начале также указывается время и дата перехода, далее указывается тип автомата (в данном случае A0), имя автомата, номера состояний, между которыми был осуществлен переход, и номер события, получив которое автомат осуществил данный переход. Приведенную строку протокола следует понимать так:

```
Во вторник, 10 февраля 2004, в 0 часов 32 минуты
21 секунду автомат Unit #3 типа A0 перешел из
состояния 0 ("Стоп") в состояние 1 ("Движение
к месту назначения"), получив событие 2
```

Третий тип – записи о выполнении автоматом выходных воздействий (содержат символ ‘‘Z’’):

```
Thu Feb 12 21:17:27 MSK 2004| Z
A3 <Scanner #2 (Unit #1)>: z1 |
```

Такую строку следует понимать следующим образом:

```
В четверг, 12 февраля 2004, в 21 час 17 минут
27 секунд автомат Scanner #2 (Unit #1) типа A3
выполнил выходное воздействие z1
```

Пример протокола приведен в приложении 1.

Логирование автоматов осуществляется с помощью методов `logTrans`, `logLaunch` и `logZ` класса “Automat”. Кроме того, в программе существует специальный класс “Log”, содержащий релизации методов вывода строк в файл для ведения лога.

10 Некоторые пояснения по запуску программы

Все используемые программой классы и ресурсы собраны в один jar-архив `ssc.jar`.
Для запуска программы неоходимо:
– иметь установленную виртуальную машину Java;
– в файле `run.bat` указать правильный путь к местонахождению виртуальной машины Java.
Для сборки программы использовалась версия Java 1.5.0, поэтому при использовании более низкой версии Java корректность работы не гарантирована.

11 Заключение

При написании работы авторы ставили следующую задачу – реализовать упрощенную версию популярной игры [3]. В работе [7] отмечается, что программирование игр во многом является "показателем развития программирования вообще". Поэтому данная работа служит своего рода "проверкой на прочность" технологии автоматного программирования. Успешная реализация поставленной задачи свидетельствует о том, что такую проверку технология выдержала.

12 Список используемых источников

1. Сайт кафедры “Информационные системы” СПбГУ ИТМО
   http://is.ifmo.ru/

2. Русскоязычный сайт, посвященный разработке игр
   http://www.gamedev.ru/

3. Сайт компании “Blizzard”, посвященный игре “StarCraft”
   http://www.blizzard.com/broodwar/

4. Сайт компании “Sun Microsystems”, посвященный языку Java
   http://java.sun.com/

5. *Кузнецов Д.В., Шалыто А.А.* Система управления танком для игры “Robocode”. Вариант 2 //
   http://is.ifmo.ru/?i0=projects\&i1=robocode2

6. *Пенев В.П., Степаненков В.В., Сучкоусов Е.А., Шалыто А.А.* Компьютерная игра “Automatic Bomber”//
   http://is.ifmo.ru/?i0=projects\&i1=bomber

7. *Туккель Н.И., Шалыто А.А.* Программирование с явным выделением состояний // Мир ПК. 2001. №8, 9.
   http://is.ifmo.ru/?i0=works\&i1=mirk

13 Приложения

13.1 Приложение 1. Пример протокола

Ниже приведен пример *неполного* протокола программы для иллюстрации процесса сражения двух юнитов. В случае, когда некоторые выходные воздействия выполняются слишком часто (например, по событиям от таймера), полный протокол оказывается слишком большим, поскольку содержит слишком много записей о выполнении таких выходных воздействий. Такие записи явно являются излишними, поскольку не несут никакой полезной информации. Поэтому они исключены из протокола.

Пример самого протокола расположен в файле `example.log`. Ниже приведен текст протокола из этого файла с комментариями. `Таким шрифтом` показаны непосредственно строки протокола, а обычным - комментарии.

```
Tue Jun 01 00:46:11 MSD 2004| L A2 <Unit #1 cannon>: launched in state 0 ("Not
shooting. Timer is swiched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #1 (Unit #1)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #2 (Unit #1)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #3 (Unit #1)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #4 (Unit #1)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #5 (Unit #1)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #6 (Unit #1)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| T A3 <Scanner #2 (Unit #1)>: has changed state
from 0 ("Switched off") to 1 ("Switched on") | event: 1

Tue Jun 01 00:46:11 MSD 2004| T A3 <Scanner #6 (Unit #1)>: has changed state
from 0 ("Switched off") to 1 ("Switched on") | event: 1

Tue Jun 01 00:46:11 MSD 2004| L A0 <Unit #1>: launched in state 0 ("Stop")
```

Запущен автомат “юнит 1”, запущены автомат “пушка юнита 1” и все автоматы-сканеры юнита 1. Кроме того, два сканера включились (перешли в состояние “включен”, получив соответствующие события от автомата “юнит 1”). Таким образом, юнит 1 готов к выполнению боевых задач.

```
Tue Jun 01 00:46:11 MSD 2004| L A2 <Unit #2 cannon>: launched in state 0 ("Not
shooting. Timer is swiched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #1 (Unit #2)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #2 (Unit #2)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #3 (Unit #2)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #4 (Unit #2)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #5 (Unit #2)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| L A3 <Scanner #6 (Unit #2)>: launched in state
0 ("Switched off")

Tue Jun 01 00:46:11 MSD 2004| T A3 <Scanner #2 (Unit #2)>: has changed state
from 0 ("Switched off") to 1 ("Switched on") | event: 1

Tue Jun 01 00:46:11 MSD 2004| T A3 <Scanner #6 (Unit #2)>: has changed state
from 0 ("Switched off") to 1 ("Switched on") | event: 1

Tue Jun 01 00:46:11 MSD 2004| L A0 <Unit #2>: launched in state 0 ("Stop")
```

То же самое для автомата “юнит 2”.

```
Tue Jun 01 00:46:12 MSD 2004| L A1 <ControlPanel>: launched in state 0 ("Stop")
```

Запущен автомат “панель управления” в состоянии 0 (“Стоп”)

```
Tue Jun 01 00:46:15 MSD 2004| T A1 <ControlPanel>: has changed state from 0
("Stop") to 3 ("Move") | event: 9

Tue Jun 01 00:46:15 MSD 2004| Z A1 <ControlPanel>: z10 | Give an order "Move"to
the controled unit

Tue Jun 01 00:46:15 MSD 2004| Z A1 <ControlPanel>: z3 | Redraw panel in accordance
with state 3 ("Move")

Tue Jun 01 00:46:15 MSD 2004| T A0 <Unit #1>: has changed state from 0 ("Stop")
```

```
to 1 ("Movement to the destination") | event: 1

Tue Jun 01 00:46:15 MSD 2004| Z A0 <Unit #1>: z1 | Switch scanners in accordance
with state 1 ("Moving to the destination")

Tue Jun 01 00:46:15 MSD 2004| T A3 <Scanner #1 (Unit #1)>: has changed state
from 0 ("Switched off") to 1 ("Switched on") | event: 1

Tue Jun 01 00:46:15 MSD 2004| T A3 <Scanner #2 (Unit #1)>: has changed state
from 1 ("Switched on") to 0 ("Switched off") | event: 2
```

Процесс получения юнитом 1 приказа “Двигаться”: сначала автомат “Панель управления” получил событие 9 (“нажатие правой кнопкой мыши на точку”) от пользователя. Далее этот автомат изменил свое состояние и послал соответствующее событие автомату “юнит 1”. “Юнит 1” изменил свое состояние и переключил свои сканеры.

```
Tue Jun 01 00:46:17 MSD 2004| T A0 <Unit #4>: has changed state from 0 ("Stop")
to 5 ("Pursuit of target") | event: 7

Tue Jun 01 00:46:17 MSD 2004| Z A0 <Unit #4>: z5 | Switch scanners in accordance
with state 5 ("Pursuit of targer")

Tue Jun 01 00:46:17 MSD 2004| T A3 <Scanner #2 (Unit #4)>: has changed state
from 1 ("Switched on") to 0 ("Switched off") | event: 2

Tue Jun 01 00:46:17 MSD 2004| T A3 <Scanner #3 (Unit #4)>: has changed state
from 0 ("Switched off") to 1 ("Switched on") | event: 1

Tue Jun 01 00:46:17 MSD 2004| T A3 <Scanner #4 (Unit #4)>: has changed state
from 0 ("Switched off") to 1 ("Switched on") | event: 1
```

Юнит 4 заметил приближающийся Юнит 1 и поэтому перешел в состояние 5 “Погоня за целью”, получив событие 7 (“Сканер 2: Найдена цель в поле видимости!”) от одного из своих сканеров. Далее юнит 4 переключил свои сканеры.

```
Tue Jun 01 00:46:20 MSD 2004| T A0 <Unit #4>: has changed state from 5 ("Pursuit
of target") to 4 ("Attack") | event: 9

Tue Jun 01 00:46:20 MSD 2004| Z A0 <Unit #4>: z4 | Switch scanners in accordance
with state 4 ("Attack")

Tue Jun 01 00:46:20 MSD 2004| T A3 <Scanner #2 (Unit #4)>: has changed state
from 1 ("Switched on") to 0 ("Switched off") | event: 2
```

```
Tue Jun 01 00:46:20 MSD 2004| T A2 <Unit #4 cannon>: has changed state from
0 ("Not shooting. Timer is swiched off") to 1 ("Shooting. Timer is swiched
on") | event: 1

Tue Jun 01 00:46:20 MSD 2004| Z A2 <Unit #4 cannon>: z1 | Switch on timer
```

Юнит 4 догнал Юнит 1 и перешел в состояние 4 “атака”, переключив свои сканеры и включив пушку. Автомат “пушка юнита 4” перешел в состояние “Стреляет. Таймер включен” и включил свой таймер, выполнив выходное воздействие z1.

```
Tue Jun 01 00:46:22 MSD 2004| Z A2 <Unit #4 cannon>: z2 | Launch a rocket
```

Пушка юнита 4 выпустила снаряд, выполнив выходное воздействие z2.

13.2 Приложение 2. Листинги модулей

Здесь приведены листинги модулей, реализующих автоматы и объекты, управляемые автоматами.

13.2.1 Листинг модуля Automat.java

```
/**
 * <code>Automat</code> -
 * базовый класс для любого автомата. Содержит
    объявление необходимых методов.
 * @author Medvinskiy M.
 */
public abstract class Automat{

    /**
    * Номер состояния автомата
    */
    private int y0;

    /**
    * Названия состояний автомата
    */
    private String[] states;

    /**
    * Количество состояний
    */
    private int statesCount;

    /**
    * Устанавливает количество состояний автомата.
    * @param count количество состояний
    */
    public void setStatesCount(int count) {
        statesCount = count;
        states = new String[statesCount];
    };

    /**
    * Получить название состояния.
    * Возвращает название состояния с номером <code>
       index</code>
    */
    public String getStateName(int index) {
        if (!(index >= statesCount)) return states[
           index];
        return "";
    };
```

```
/**
* Устанавливает новое название для состояния
* автомата с указанным номером.
* @param index номер состояния
* @param name название состояния
*/
public void setStateName(int index, String name) {
    states[index] = name;
};

/**
* Возвращает номер состояния автомата.
*/
protected int getState() {return y0;};

/**
* Устанавливает новое состояния автомата.
* @param y номер состояния
*/
protected void setState(int y) {y0 = y;};

/**
* Имя автомата
*/
private String name;

/**
* Получить имя автомата
*/
protected String getName() {return name;};

/**
* Установить имя автомата
* @param n имя автомата
*/
protected void setName(String n) {name = n;};

/**
* Тип автомата (A0, A1, A2 и.т.д...)
*/
private String automatType;

/**
* Получить тип автомата
*/
protected String getType() {return automatType;};

/**
* Установить тип автомата
* @param t название типа
```

```
*/
protected void setType(String t) {automatType = t
    ;};

/**
* Вести лог или нет
*/
private boolean isLogged;

/**
* Показывает, ведется лог или нет
*/
protected boolean getLogged() {return isLogged;};

/**
* Устанавливает, вести ли лог
*/
protected void setLogged(boolean b) {isLogged = b
    ;};

/**
* Логирование переходов автомата.
* Выводит лог о том, что автомат перешел
* в состояние <code>to</code> из состояния
* <code>from</code>, получив событие <code>e</code
    >
* @param to новое состояние
* @param from исходное состояние
* @param e переданное событие
*/
protected void logTrans(int to, int from, int e) {
    if (!isLogged) return;

    java.util.Date time = new java.util.Date();
    time = java.util.Calendar.getInstance(java.
       util.Locale.getDefault()).getTime();
    Log.writeln(time.toString() + "| T " +
       automatType +
      " <" + name + ">: has changed state from
          " + from + " (\""+ getStateName(from) +
      "\") to " + to + " (\"" + getStateName(to)
          + "\") | event: " + e);
};

/**
* Логирование выходных воздействий автомата.
* Выводит лог о том, что автомат выполнил
   процедуру
* zi()
* @param z номер выходного воздействия
```

```
    * @param comment описание воздействия
    */
    protected void logZ(int z, String comment) {
        if (!isLogged) return;

        java.util.Date time = new java.util.Date();
        time = java.util.Calendar.getInstance(java.
           util.Locale.getDefault()).getTime();
        Log.writeln(time.toString() + "| Z " +
           automatType +
          " <" + name + ">: z" + z + " | " + comment);
    };

    /**
    * Логирование запуска автомата.
    * Выводит лог о том, что автомат запущен в
       состоянии state
    * @param state состояние
    */
    protected void logLaunch(int state) {
        if (!isLogged) return;

        java.util.Date time = new java.util.Date();
        time = java.util.Calendar.getInstance(java.
           util.Locale.getDefault()).getTime();
        Log.writeln(time.toString() + "| L " +
           automatType +
          " <" + name + ">: launched in state " +
             state + " (\"" + getStateName(state)
              + "\")");
    };

    /**
    * Функция автомата. Должна осуществлять
    * переход автомата, получившего событие <code>e</
       code>,
    * в другое состояние
    * @param e передаваемое событие
    */
    public abstract void A(int e);
};
```

13.2.2 Листинг модуля Scanner.java

```
/**
 * <code>Scanner</code> - класс, являющийся базовым
    для всех сканеров
 */

public abstract class Scanner{
```

```
protected class AutomatScanner extends Automat{

    public AutomatScanner() {
        setLogged(true);
        setStatesCount(2);
        setStateName(0, "Switched off");
        setStateName(1, "Switched on");

        setType("A3");
    };

    /**
    * Функция автомата
    */
    public void A(int e) {
        int y0 = getState();
        int yold = y0;
        switch (y0) {
            case 0: if (e == 1) { y0 = 1; };
                    break;

            case 1: if (e == 2) { y0 = 0; };
                    if (e == 3) { y0 = 1; z1(); };
                    break;

            default: break;
        };

        if (y0 != yold) {
            //Прибыли в новое состояние y0.
            logTrans(y0, yold, e);
        };

        setState(y0);
    };

    /**
    * Выходная функция z1()
    */
    private void z1() {
        run();
    };

};

/**
* Юнит, которому принадлежит сканер
*/
private Unit owner;
```

```
    /**
    * Возвращает значение юнита-владелеца
    */
    protected Unit getOwner() {return owner;};

    /**
    * устанавливает нового юнита-владелеца
    */
    protected void setOwner(Unit o) {owner = o;};

    /**
    * Функция проверяет, включен ли сканер
    */
    protected boolean onOff() {
        return (A3.getState() == 1);
    };

    /**
    * Автомат, управляющий сканером
    */
    protected AutomatScanner A3;

    /**
    * Управление автоматом сканера: Передать событие "
       включить сканер" или "выключить сканер"
    */
    public void switchScanner(boolean b) {
        if (b) A3.A(1); else A3.A(2);
    };

    /**
    * Управление автоматом сканера: Передать событие "
       сработал таймер"
    */
    public void work() {
        A3.A(3);
    };

    /**
    * Процедура сканирует местность на наличие какого-
       либо признака,
    * должна быть переопределена для каждого
       отдельного сканера
    */
    public abstract void run();

    public void logLaunch() {
        A3.logLaunch(A3.getState());
    }
};
```

13.2.3 Листинг модуля Cannon.java

```
import java.util.*;

public class Cannon{

    class AutomatCannon extends Automat{

        public AutomatCannon() {
            setLogged(true);
            setStatesCount(3);
            setStateName(0, "Not shooting. Timer is
                swiched off");
            setStateName(1, "Shooting. Timer is
                swiched on");
            setStateName(2, "Not shooting. Timer is
                swiched on");
            setState(0);
            setType("A2");
            setName(getCannonName());
            logLaunch(getState());
        };


        //включить таймер
        public void z1() {
            timerTask = new MyTimerTask();
            timer.schedule(timerTask, 10, 2000);
            logZ(1, "Switch on timer");
        };

        //выпустить ракету
        public void z2() {
            owner.getField().bullets.add(new Bullet(
                owner));
            logZ(2, "Launch a rocket");
        };

        //выключить таймер
        public void z3() {
            timerTask.cancel();
            logZ(3, "Switch off timer");
        };

        public void A(int e) {
            int y0 = getState();
            int yold = y0;
            switch (y0) {
            case 0: if (e == 1) { z1(); y0 = 1; };
                    break;
```

```
            case 1: if (e == 2) { y0 = 2; };
                    if (e == 3) { z2(); y0 = 1; };
                    break;

            case 2: if (e == 1) { y0 = 1; };
                    if (e == 3) { z3(); y0 = 0; };
                    break;

            default: break;
            };

            if (y0 != yold) logTrans(y0, yold, e);

            setState(y0);
        };


};

private String cannonName;

/**
* Возвращает имя пушки
*/
public String getCannonName() {return cannonName
    ;};

private AutomatCannon A2;

private java.util.Timer timer = new java.util.
    Timer();
private class MyTimerTask extends java.util.
    TimerTask {
     public void run() {
         //послать сообщение Таймер.сработал!
         A2.A(3);
     };
};
private MyTimerTask timerTask = new MyTimerTask();

/**
* Владелец пушки
*/
private Unit owner;

/**
* Возвращает владельца пушки
*/
public Unit getOwner() {return owner;};
```

```
    /**
    * Конструктор. Создает новую пушку для указанного
       юнита
    * @param o владелец пушки
    */
    public Cannon(Unit o) {
        owner = o;
        cannonName = o.getName() + " cannon";
        A2 = new AutomatCannon();
    };

    /**
    * Передать событие "начать огонь"
    */
    public void fire() {
        A2.A(1);
    }

    /**
    * Передать событие "прекратить огонь"
    */
    public void nofire() {
        A2.A(2);
    }

};
```

13.2.4 Листинг модуля Unit.java

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;



/**
* Сканер 1 - проверка, не прибыли ли на место в случае
    передвижения
*/
class Scanner1 extends Scanner{
    public Scanner1(Unit o) {
        A3 = new AutomatScanner();
        A3.setName("Scanner #1 (" + o.getName() + ")")
           ;
        switchScanner(false); //сканер выключен
        setOwner(o);
        logLaunch();
    };
```

```
    public void run(){
        Unit o = getOwner();
        if (Math.abs(o.getX() - o.getDestinationX()) +
            Math.abs(o.getY() - o.getDestinationY())
                <= 10) {
                    o.getA0().A(6); //передать
                        событие "Сканер1.Приехали!"
        };
    };
};

/**
* Сканер 2 - проверка, есть ли какая-нибудь цель в
   поле видимости?
*/
class Scanner2 extends Scanner{
    public Scanner2(Unit o) {
        A3 = new AutomatScanner();
        A3.setName("Scanner #2 (" + o.getName() + ")")
            ;
        switchScanner(false); //сканер выключен
        setOwner(o);
        logLaunch();
    };

    public void run(){
        Unit o = getOwner();
        double r;
        double xx;
        double yy;
        GameObject AnotherUnit;
        for (int i = 0; i < o.getField().units.size()
            ; i++) {
            AnotherUnit = (GameObject)o.getField().
                units.elementAt(i);
            if ((AnotherUnit != null)&&(getOwner() !=
                AnotherUnit)&&(o.getTeam() !=
                AnotherUnit.getTeam())) {
                xx = AnotherUnit.getX();
                yy = AnotherUnit.getY();
                r = Math.sqrt((o.getX() - xx) * (o.
                    getX() - xx) + (o.getY() - yy) * (o
                    .getY() - yy));
                if (r <= o.getSightRange()) {
                    o.setTarget(AnotherUnit);
                    o.getA0().A(7); //передать событие
                        "Сканер2.Нашел цель!"
                    return;
                };
            }
```

```
        };
    };
};

/**
* Сканер 3 - проверка, не умерла ли текущая цель?
*/
class Scanner3 extends Scanner{
    public Scanner3(Unit o) {
        A3 = new AutomatScanner();
        A3.setName("Scanner #3 (" + o.getName() + ")")
            ;
        switchScanner(false); //сканер выключен
        setOwner(o);
        logLaunch();
    };

    public void run(){
        Unit o = getOwner();
        if (!o.getTarget().isAlive()) o.getA0().A(8)
            ; //передать событие "Сканер3.цель умерла!"
    };
};

/**
 * Сканер 4 - проверка, на ходится ли текущая цель в
    поле атаки?
 */
class Scanner4 extends Scanner{
    public Scanner4(Unit o) {
        A3 = new AutomatScanner();
        A3.setName("Scanner #4 (" + o.getName() + ")")
            ;
        switchScanner(false); //сканер выключен
        setOwner(o);
        logLaunch();
    };

    public void run(){
        Unit o = getOwner();
        double xx = o.getTarget().getX();
        double yy = o.getTarget().getY();
        double r = Math.sqrt((o.getX() - xx) * (o.getX
            () - xx) + (o.getY() - yy) * (o.getY() - yy
            ));
        if (r <= o.getAttackRange()) {
            o.getA0().A(9); //передать событие "
                Сканер4.цель в поле атаки!"
        }else{
```

```
            o.getA0().A(10); //передать событие "
               Сканер4.цель не в поле атаки, нужно
               догонять!"
        }
    };
};

/**
* Сканер 5 - проверка, есть ли какая-нибудь цель в
   поле атаки?
*/
class Scanner5 extends Scanner{
    public Scanner5(Unit o) {
        A3 = new AutomatScanner();
        A3.setName("Scanner #5 (" + o.getName() + ")")
            ;
        switchScanner(false); //сканер выключен
        setOwner(o);
        logLaunch();
    };

    public void run(){
        Unit o = getOwner();
        double r;
        double xx;
        double yy;
        for (int i = 0; i < o.getField().units.size()
            ; i++) {
            GameObject unit = (GameObject)o.getField()
               .units.elementAt(i);
            if ((unit.isAlive())&&(o != unit)&&(o.
               getTeam() != unit.getTeam())) {
                xx = unit.getX();
                yy = unit.getY();
                r = Math.sqrt((o.getX() - xx) * (o.
                   getX() - xx) + (o.getY() - yy) * (o
                   .getY() - yy));
                if (r <= o.getAttackRange()) {
                    o.setTarget(unit);
                    o.getA0().A(11); //передать
                       событие "Сканер2.Нашел цель в
                       поле атаки!"
                    return;
                };
            };
        };
    };
};

/**
```

```
* Сканер 6 - проверка, не умерли ли мы?
*/
class Scanner6 extends Scanner{
    public Scanner6(Unit o) {
        A3 = new AutomatScanner();
        A3.setName("Scanner #6 (" + o.getName() + ")")
            ;
        switchScanner(false); //сканер выключен
        setOwner(o);
        logLaunch();
    };

    public void run(){
        Unit o = getOwner();
        if (o.getHealth() <= 0) {
            o.setAlive(false);
            o.getA0().A(12);
            return;
        };
    };
};


public class Unit extends GameObject{


    class AutomatUnit extends Automat{

        private Unit owner;
        public Unit getOwner() {return owner;};

        public AutomatUnit(Unit o) {
            setLogged(true);
            setStatesCount(7);
            setStateName(0, "Stop");
            setStateName(1, "Movement to the
                destination");
            setStateName(2, "Hold position (without
                attack)");
            setStateName(3, "Hold position (with
                attack)");
            setStateName(4, "Attack");
            setStateName(5, "Pursuit of target");
            setStateName(6, "Dead");
            owner = o;
            setName(o.getName());
            setState(0);
            setType("A0");
            logLaunch(getState());
        };
```

```
/**
* функция автомата
*/
public void A(int e) {
    int y0 = getState();
    int yold = y0;
    switch (y0) {
                case 0: if (e == 1) { y0
                    = 1; };
                        if (e == 2) { y0
                            = 5; };
                        if (e == 3) { y0
                            = 2; };
                        if (e == 4) { y0
                            = 0; };

                        if (e == 7) { y0
                            = 5; };
                        if (e == 12) { y0
                            = 6; };
                        break;

                case 1: if (e == 1) { y0
                    = 1; };
                        if (e == 2) { y0
                            = 5; };
                        if (e == 3) { y0
                            = 2; };
                        if (e == 4) { y0
                            = 0; };

                        if (e == 6) { y0
                            = 0; };
                        if (e == 12) { y0
                            = 6; };
                        break;

                case 2: if (e == 1) { y0
                    = 1; };
                        if (e == 2) { y0
                            = 5; };
                        if (e == 3) { y0
                            = 2; };
                        if (e == 4) { y0
                            = 0; };

                        if (e == 11) { y0
                            = 3; };
```

```
            if (e == 12) { y0
                = 6; };
            break;

        case 3: if (e == 1) { y0
            = 1; };
            if (e == 2) { y0
                = 5; };
            if (e == 3) { y0
                = 2; };
            if (e == 4) { y0
                = 0; };

            if ((e == 8)||(e
                == 10)) { y0
                = 2; };
            if (e == 12) { y0
                = 6; };
            break;

        case 4: if (e == 1) { y0
            = 1; };
            if (e == 2) { y0
                = 5; };
            if (e == 3) { y0
                = 2; };
            if (e == 4) { y0
                = 0; };

            if (e == 8) { y0
                = 0; };
            if (e == 10) { y0
                = 5; };
            if (e == 12) { y0
                = 6; };
            break;

        case 5: if (e == 1) { y0
            = 1; };
            if (e == 2) { y0
                = 5; };
            if (e == 3) { y0
                = 2; };
            if (e == 4) { y0
                = 0; };

            if (e == 9) { y0
                = 4; };
            if (e == 8) { y0
                = 0; };
```

```
                        if (e == 12) { y0
                            = 6; };
                        break;
                case 6: break;

                default: break;
        };

        if (y0 != yold) {
                        //Прибыли в новое
                            состояние y0.
                            Теперь включим или
                            выключим некоторые
                            сканеры
            switch (y0) {
                        case 0: z0();
                                break;
                        case 1: z1();
                                break;
                        case 2: z2();
                                break;
                        case 3: z3();
                                break;
                        case 4: z4();
                                break;
                        case 5: z5();
                                break;
                        case 6: z6();
                                break;
            };

                        //            field.
                            btnpanel.unitUpdate
                            (); //проапдейтить
                            панель управления
            logTrans(y0, yold, e);

        };

        setState(y0);
};

private void z0() {
        setCurMode(MODE_STOP);
        scanner1.switchScanner(false);
        scanner2.switchScanner(true);
        scanner3.switchScanner(false);
        scanner4.switchScanner(false);
        scanner5.switchScanner(false);
        scanner6.switchScanner(true);
```

```
        cannon.nofire();
        logZ(0, "Switch scanners in accordance
           with state 0 (\"Stop\")");
    };

    private void z1() {
        setCurMode(MODE_MOVE);
        scanner1.switchScanner(true);
        scanner2.switchScanner(false);
        scanner3.switchScanner(false);
        scanner4.switchScanner(false);
        scanner5.switchScanner(false);
        scanner6.switchScanner(true);
        cannon.nofire();
        logZ(1, "Switch scanners in accordance
           with state 1 (\"Moving to the
           destination\") ");
    };

    private void z2() {
        setCurMode(MODE_HOLD);
        scanner1.switchScanner(false);
        scanner2.switchScanner(false);
        scanner3.switchScanner(false);
        scanner4.switchScanner(false);
        scanner5.switchScanner(true);
        scanner6.switchScanner(true);
        cannon.nofire();
        logZ(2, "Switch scanners in accordance
           with state 2 (\"Hold position without
           attack\")");
    };

    private void z3() {
        setCurMode(MODE_HOLD);
        scanner1.switchScanner(false);
        scanner2.switchScanner(false);
        scanner3.switchScanner(true);
        scanner4.switchScanner(true);
        scanner5.switchScanner(false);
        scanner6.switchScanner(true);
        cannon.fire();
        logZ(3, "Switch scanners in accordance
           with state 3 (\"Hold position with
           attack\")");
    };

    private void z4() {
        setCurMode(MODE_ATTACK);
        scanner1.switchScanner(false);
```

```
            scanner2.switchScanner(false);
            scanner3.switchScanner(true);
            scanner4.switchScanner(true);
            scanner5.switchScanner(false);
            scanner6.switchScanner(true);
            cannon.fire();
            logZ(4, "Switch scanners in accordance
               with state 4 (\"Attack\")");
        };

        private void z5() {
            setCurMode(MODE_ATTACK);
            scanner1.switchScanner(false);
            scanner2.switchScanner(false);
            scanner3.switchScanner(true);
            scanner4.switchScanner(true);
            scanner5.switchScanner(false);
            scanner6.switchScanner(true);
            cannon.nofire();
            logZ(5, "Switch scanners in accordance
               with state 5 (\"Pursuit of targer\")");
        };

        private void z6() {
            setCurMode(NO_MODE);
            scanner1.switchScanner(false);
            scanner2.switchScanner(false);
            scanner3.switchScanner(false);
            scanner4.switchScanner(false);
            scanner5.switchScanner(false);
            scanner6.switchScanner(false);
            cannon.nofire();
            logZ(6, "Switch scanners in accordance
               with state 6 (\"Dead\")");
        };

    };

    /***
    * Цель объекта
    */
    private GameObject target;

    /**
    * Возвращает текущую цель юнита
    */
    protected GameObject getTarget() {return target;};

    /**
    * Устанавливает новую цель юнита
```

```
* @param t новая цель
*/
protected void setTarget(GameObject t) {target = t
    ;};

private Scanner1 scanner1;
private Scanner2 scanner2;
private Scanner3 scanner3;
private Scanner4 scanner4;
private Scanner5 scanner5;
private Scanner6 scanner6;

private Cannon cannon;

/**
* Автомат, управляющий сканером
*/
private AutomatUnit A0;

/**
* Получить автомат, управляющий сканером
*/
public AutomatUnit getA0() {return A0;};

/**
* Отдать приказ "Двигаться"
*/
public void orderMove() {
    A0.A(1);
};

/**
* Отдать приказ "Атаковать"
*/
public void orderAttack() {
    A0.A(2);
};

/**
* Отдать приказ "Держать местность"
*/
public void orderHold() {
    A0.A(3);
};

/**
* Отдать приказ "Стоять"
*/
public void orderStop() {
    A0.A(4);
```

```
};


/**
* Константа для невыбранного приказа
*/
public final static int NO_MODE = 0;  // Нет
    приказа
/**
* Константа для приказа "стоп"
*/
public final static int MODE_STOP = 1;  // Приказ
    : Стоим
/**
* Константа для приказа "двигаться"
*/
public final static int MODE_MOVE = 2;  // Приказ
    : Двигаемся
/**
* Константа для приказа "атаковать"
*/
public final static int MODE_ATTACK = 3;// Приказ
    : Атакуем
/**
* Константа для приказа "держать местность"
*/
public final static int MODE_HOLD = 4;  // Приказ
    : Держим позицию

/**
* Конструктор. Создает новый юнит на указанном
    поле боя
* @param a поле боя
* @param n имя юнита
* @param t номер команды
* @param xx x координата
* @param yy y координата
*/
public Unit(BattleField a, String n, int t, double
     xx, double yy) {
     setControlled(true);
     img = ImageMap.getImage("gif/unit" + t + ".gif
         ");
     setTeam(t);

     setCurMode(MODE_STOP);
     setAlive(true);

     setX(xx);
     setY(yy);
```

```
        setDirect(1, 0);
        deselect();
        setField(a);
        setSightRange(150);
        setAttackRange(120);
        setHealth(100);
        setName(n);
        cannon = new Cannon(this);

        scanner1 = new Scanner1(this);
        scanner2 = new Scanner2(this);
        scanner3 = new Scanner3(this);
        scanner4 = new Scanner4(this);
        scanner5 = new Scanner5(this);
        scanner6 = new Scanner6(this);

        scanner1.switchScanner(false);
        scanner2.switchScanner(true);
        scanner3.switchScanner(false);
        scanner4.switchScanner(false);
        scanner5.switchScanner(false);
        scanner6.switchScanner(true);
        cannon.nofire();

        setCenter(-img.getWidth(null) / 2, -img.
            getHeight(null) / 2);
        setOvalRect(-img.getWidth(null) / 2 - 5, -10,
            img.getWidth(null) + 10, 30);
        setHealthRect(-img.getWidth(null) / 2, -img.
            getHeight(null) / 2 - 10, img.getWidth(null
            ), 5);

        A0 = new AutomatUnit(this);
    };

    /**
    * Сообщение всем сканерам: "сработал таймер"
    */
    public void runAllScanners() {
        scanner1.work();
        scanner2.work();
        scanner3.work();
        scanner4.work();
        scanner5.work();
        scanner6.work();
    };

    /**
    * Передвигает юнит
    */
```

```
public void move() {

    double sx, sy, sz;
    double A1, B1, C1, A2, B2, C2, A3, B3, C3, D1
       , D2, D3;
    double xA, yA, zA;
    double xC, yC, zC;
    double delta = 4;

    int dx = 0;
    int dy = 0;

    switch (getCurMode()) {
            case MODE_ATTACK:
                    dx = target.getIntX();
                    dy = target.getIntY();
                    break;

            case MODE_MOVE:
                    dx = getDestinationX();
                    dy = getDestinationY();
                    break;
    };


    if ((Math.abs(getX() - dx) < delta)&&(Math.abs
       (getY() - dy) < delta)) return;
    Vector2 v1 = new Vector2(getX() - dx, getY()
        - dy, 0);

    int n = 5;
    Vector2 v2 = new Vector2(getDirectX(),
       getDirectY(), 0);

    double xT = dx;
    double yT = dy;
    double zT = 0;

    if (v1.Angle(v2) > Math.PI * 0.1) {
        xT = (dx * 1 + getX() * n) / (n + 1);
        yT = (dy * 1 + getY() * n) / (n + 1);
        zT = 0;
    };

    sx = (getX() + xT) * 0.5;
    sy = (getY() + yT) * 0.5;

    xA = xT - getX();
    yA = yT - getY();
    zA = zT - 0;
```

```
A1 = xA;
B1 = yA;
C1 = 0;
D1 = sx * xA + sy * yA;

A2 = getDirectX();
B2 = getDirectY();
C2 = 0;
D2 = getX() * getDirectX() + getY() *
   getDirectY();

Vector2 vect = new Vector2(xA, yA, 0);

Vector2 dir = new Vector2(getDirectX(),
   getDirectY(), 0);
Vector2 ortv = new Vector2();
ortv.Ort(vect, dir);

A3 = ortv.x;
B3 = ortv.y;
C3 = ortv.z;
D3 = getX() * ortv.x + getY() * ortv.y + 0 *
   ortv.z;

if ((ortv.x == 0)&&(ortv.y == 0)&&(ortv.z
    == 0)) {
    double l = getDirectX() * getDirectX() +
       getDirectY() * getDirectY() + 0 * 0;
    l = Math.sqrt(l);
    setDirectX(getDirectX() / l);
    setDirectY(getDirectY() / l);

    //расстояние до цели до изменения
       положения
    double distDo = (dx - getX())*(dx - getX()
       ) + (dy - getY())*(dy - getY());
    setX(getX() + getDirectX() * 2);
    setY(getY() + getDirectY() * 2);

    //расстояние до цели после изменения
       положения
    double distPosle = (dx - getX())*(dx -
       getX()) + (dy - getY())*(dy - getY());
    if (distDo < distPosle) {
        setY(getY() - 2 * Math.random());
        setX(getX() - 2 * Math.random());
    };

    return;
```

```
        };
        double dett = Vector2.det3(A1, B1, C1, A2, B2
            , C2, A3, B3, C3);
        double detx = Vector2.det3(D1, B1, C1, D2, B2
            , C2, D3, B3, C3);
        double dety = Vector2.det3(A1, D1, C1, A2, D2
            , C2, A3, D3, C3);
        double detz = Vector2.det3(A1, B1, D1, A2, B2
            , D2, A3, B3, D3);

        double x0 = detx/dett;
        double y0 = dety/dett;
        double z0 = detz/dett;

        xC = x0;
        yC = y0;
        zC = z0;

        double qradius = (x0 - getX())*(x0 - getX())
             + (y0 - getY())*(y0 - getY()) + (z0 - 0)*(
            z0 - 0);

        double k = 2;
        double l = getDirectX() * getDirectX() +
            getDirectY() * getDirectY() + 0 * 0;

        l = Math.sqrt(l);
        setDirectX(getDirectX() / l);
        setDirectY(getDirectY() / l);

        setX(getX() + getDirectX() * k);
        setY(getY() + getDirectY() * k);

        double xAcceler = ((x0 - getX())) * k /
            qradius;
        double yAcceler = ((y0 - getY())) * k /
            qradius;

        setDirectX(getDirectX() + xAcceler);
        setDirectY(getDirectY() + yAcceler);

    };

    /**
    * Рисует объект
    * @param g графический контекст
    */
    public void draw(Graphics g) {
        Color old = g.getColor();
```

```
        Graphics2D g2 = (Graphics2D)g;

        g2.setColor(Color.green);

        AffineTransform aT = g2.getTransform();


        double d = Math.sqrt(getDirectX() * getDirectX
            () + getDirectY() * getDirectY());
        double angle = Vector2.acossin(getDirectY()/d
            , getDirectX()/d);

        AffineTransform rotate = AffineTransform.
            getRotateInstance(angle, getIntX(), getIntY
            ());

        g2.transform(rotate);
        g2.drawImage(img, getIntX() - img.getWidth(
            null) / 2, getIntY() - img.getHeight(null)
             / 2 , null);

        g2.setTransform(aT);
        g2.setColor(old);
    };

    /**
    * Метод проверяет, принадлежит ли точка (xx, yy)
       объекту
    * @param x x координата
    * @param y y координата
    */
    public boolean ifin(int x, int y) {
        return ((x - getX())*(x - getX()) + (y - getY
            ())*(y - getY()) < 200);
    };
};
```

13.2.5 Листинг модуля ControlPanel.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import java.util.*;

/**
* Класс для работы с панелью управления
*/
public class ControlPanel extends JPanel implements
    ActionListener{
```

```
class AutomatControlPanel extends Automat{

    public ControlPanel owner;

    public AutomatControlPanel(ControlPanel o) {
        setLogged(true);
        setStatesCount(7);
        setStateName(0, "Stop");
        setStateName(1, "Attack");
        setStateName(2, "Hold position");
        setStateName(3, "Move");
        setStateName(4, "Awaiting of choosing a
            target to attack");
        setStateName(5, "Awaiting of choosing a
            destination");
        setStateName(6, "Inactive");
        owner = o;
        setName(o.name);
        setState(0);
        setType("A1");
        logLaunch(getState());
        A(0);
    };

    private boolean x1() {
        return (selectedUnit.getCurMode() ==
            MODE_STOP);
    };

    private boolean x2() {
        return (selectedUnit.getCurMode() ==
            MODE_ATTACK);
    };

    private boolean x3() {
        return (selectedUnit.getCurMode() ==
            MODE_HOLD);
    };

    private boolean x4() {
        return (selectedUnit.getCurMode() ==
            MODE_MOVE);
    };

    private boolean x5() {
        return (unit.getCurMode() == MODE_STOP);
    };

    private boolean x6() {
        return (unit.getCurMode() == MODE_ATTACK);
```

```
};

private boolean x7() {
    return (unit.getCurMode() == MODE_HOLD);
};

private boolean x8() {
    return (unit.getCurMode() == MODE_MOVE);
};

public synchronized void A(int e) {
    int y0 = getState();
    int yold = y0;
    switch (y0) {
    case 0: if ((e == 1)&& x1()) {
        y0 = 0; z11();
            };

        if ((e == 1)&& x2()) {
            y0 = 1; z11();
        };

        if ((e == 1)&& x3()) {
            y0 = 2; z11();
        };

        if ((e == 1)&& x4()) {
            y0 = 3; z11();
        };

        if (e == 2) { y0 = 6; };
        if (e == 3) { y0 = 0; };
        if (e == 4) { y0 = 5; };
        if (e == 5) { y0 = 4; };
        if (e == 6) { y0 = 2; };
        if (e == 8) { y0 = 1; z7(); };
        if (e == 9) { y0 = 3; z10(); };
        if (e == 10) { y0 = 6; };
        if ((e == 11)&& x5()) {
            y0 = 0; z12();
        };

        if ((e == 11)&& x6()) {
            y0 = 1; z12();
        };

        if ((e == 11)&& x7()) {
            y0 = 2; z12();
        };
```

```
        if ((e == 11)&& x8()) {
            y0 = 3; z12();
        };
        break;

    case 1: if ((e == 1)&& x1()) {
        y0 = 0; z11();
            };

        if ((e == 1)&& x2()) {
            y0 = 1; z11();
        };

        if ((e == 1)&& x3()) {
            y0 = 2; z11();
        };

        if ((e == 1)&& x4()) {
            y0 = 3; z11();
        };

        if (e == 2) { y0 = 6; };
        if (e == 3) { y0 = 0; };
        if (e == 4) { y0 = 5; };
        if (e == 5) { y0 = 4; };
        if (e == 6) { y0 = 2; };
        if (e == 8) { y0 = 1; z7(); };
        if (e == 9) { y0 = 3; z10(); };
        if (e == 10) { y0 = 6; };

        if ((e == 11)&& x5()) {
            y0 = 0; z12();
        };

        if ((e == 11)&& x6()) {
            y0 = 1; z12();
        };

        if ((e == 11)&& x7()) {
            y0 = 2; z12();
        };

        if ((e == 11)&& x8()) {
            y0 = 3; z12();
        };
        break;

    case 2: if ((e == 1)&& x1()) {
        y0 = 0; z11();
            };
```

```
        if ((e == 1)&& x2()) {
            y0 = 1; z11();
        };

        if ((e == 1)&& x3()) {
            y0 = 2; z11();
        };

        if ((e == 1)&& x4()) {
            y0 = 3; z11();
        };

        if (e == 2) { y0 = 6; };
        if (e == 3) { y0 = 0; };
        if (e == 4) { y0 = 5; };
        if (e == 5) { y0 = 4; };
        if (e == 6) { y0 = 2; };
        if (e == 8) { y0 = 1; z7(); };
        if (e == 9) { y0 = 3; z10(); };
        if (e == 10) { y0 = 6; };
        if ((e == 11)&& x5()) {
            y0 = 0; z12();
        };

        if ((e == 11)&& x6()) {
            y0 = 1; z12();
        };

        if ((e == 11)&& x7()) {
            y0 = 2; z12();
        };

        if ((e == 11)&& x8()) {
            y0 = 3; z12();
        };
        break;

    case 3: if ((e == 1)&& x1()) {
        y0 = 0; z11();
            };

        if ((e == 1)&& x2()) {
            y0 = 1; z11();
        };

        if ((e == 1)&& x3()) {
            y0 = 2; z11();
        };
```

```
        if ((e == 1)&& x4()) {
            y0 = 3; z11();
        };

        if (e == 2) { y0 = 6; };
        if (e == 3) { y0 = 0; };
        if (e == 4) { y0 = 5; };
        if (e == 5) { y0 = 4; };
        if (e == 6) { y0 = 2; };
        if (e == 8) { y0 = 1; z7(); };
        if (e == 9) { y0 = 3; z10(); };
        if (e == 10) { y0 = 6; };
        if ((e == 11)&& x5()) {
            y0 = 0; z12();
        };

        if ((e == 11)&& x6()) {
            y0 = 1; z12();
        };

        if ((e == 11)&& x7()) {
            y0 = 2; z12();
        };

        if ((e == 11)&& x8()) {
            y0 = 3; z12();
        };
        break;

    case 4: if (e == 1) { y0 = 1; z8(); };

        if (e == 3) { y0 = 0; };
        if (e == 4) { y0 = 5; };
        if (e == 5) { y0 = 4; };
        if (e == 6) { y0 = 2; };

        if ((e == 7)&& x5()) { y0 = 0; };
        if ((e == 7)&& x8()) { y0 = 3; };
        if ((e == 7)&& x7()) { y0 = 2; };
        if ((e == 7)&& x6()) { y0 = 1; };

        if (e == 10) { y0 = 6; };
        if (e == 11) { y0 = 4; z12(); }
        break;

    case 5: if (e == 2) { y0 = 3; z9(); };

        if (e == 3) { y0 = 0; };
        if (e == 4) { y0 = 5; };
        if (e == 5) { y0 = 4; };
```

```
        if (e == 6) { y0 = 2; };

        if ((e == 7)&& x5()) { y0 = 0; };
        if ((e == 7)&& x8()) { y0 = 3; };
        if ((e == 7)&& x7()) { y0 = 2; };
        if ((e == 7)&& x6()) { y0 = 1; };

        if (e == 10) { y0 = 6; };
        if (e == 11) { y0 = 5; z12(); }
        break;

    case 6: if ((e == 1)&& x1()) {
        y0 = 0; z11();
            };

        if ((e == 1)&& x2()) {
            y0 = 1; z11();
        };

        if ((e == 1)&& x3()) {
            y0 = 2; z11();
        };

        if ((e == 1)&& x4()) {
            y0 = 3; z11();
        };

        break;

    };

    if (y0 != yold) {
        logTrans(y0, yold, e);
        switch (y0) {
        case 0: z0();
            break;
        case 1: z1();
            break;
        case 2: z2();
            break;
        case 3: z3();
            break;
        case 4: z4();
            break;
        case 5: z5();
            break;
        case 6: z6();
            break;
        };
```

```
        };

        setState(y0);
    };

    private void z0() {
        order = MODE_STOP;
        if (unit instanceof Unit) {
            Unit u = (Unit)unit;
            u.orderStop();
            curMode = MODE_STOP;
        };
        check();
        logZ(0, "");
    };

    private void z1() {
        order = MODE_ATTACK;
        if (unit instanceof Unit) {
            Unit u = (Unit)unit;
            u.orderAttack();
            curMode = MODE_ATTACK;
        }
        check();
        logZ(1, "");
    };

    private void z2() {
        order = MODE_HOLD;
        if (unit instanceof Unit) {
            Unit u = (Unit)unit;
            u.orderHold();
            curMode = MODE_HOLD;
        }
        check();
        logZ(2, "");
    };

    private void z3() {
        order = MODE_MOVE;
        if (unit instanceof Unit) {
            Unit u = (Unit)unit;
            u.orderMove();
            curMode = MODE_MOVE;
        };
        check();
        logZ(3, "Redraw panel in accordance with
           state 3 (\"Move\") ");
    };
```

```
private void z4() {
    order = MODE_ATTACK;
    check();
    logZ(4, "");
};

private void z5() {
    order = MODE_MOVE;
    check();
    logZ(5, "");
};

private void z6() {
    unit = null;
    order = NO_MODE;
    update();
    check();
    logZ(6, "");
};

private void z7() {
    if (unit instanceof Unit) {
        Unit u = (Unit)unit;
        u.setTarget(selectedUnit);
        u.orderAttack();
    }
    logZ(7, "");
};

private void z8() {
    if (unit instanceof Unit) {
        Unit u = (Unit)unit;
        u.setTarget(selectedUnit);
        u.orderAttack();
    }
    logZ(8, "");
};

private void z9() {
    if (unit instanceof Unit) {
        Unit u = (Unit)unit;
        u.setDestination(mouseX, mouseY);
        u.orderMove();
    };
    logZ(9, "");
};

private void z10() {
    if (unit instanceof Unit) {
        Unit u = (Unit)unit;
```

```
                u.setDestination(mouseX, mouseY);
                u.orderMove();
            };
            logZ(10, "Give an order \"Move\" to the
                controled unit");
        };

        private void z11() {
            unit = selectedUnit;
            update();
            //logZ(11, "");
        };

        private void z12() {
            //unit = selectedUnit;
            update();
            //logZ(12, "");
        };

    };

    class UnitState extends JPanel {
        protected JLabel unitNameLabel = new JLabel
            ("", SwingConstants.CENTER);
        protected JProgressBar unitHealthLabel = new
            JProgressBar(0, 100);
        protected JLabel unitDamageLabel = new JLabel
            ("", SwingConstants.CENTER);
        public UnitState() {
            setLayout(new GridLayout(3, 1));
            add(unitNameLabel);
            add(unitHealthLabel);
            add(unitDamageLabel);
            set("", 0, 0);
        }
        public void set(String name, int health, int
            damage) {
            unitNameLabel.setText(name);
            unitHealthLabel.setValue(health);
            if (health >= 0)
                unitHealthLabel.setVisible(true);
            else
                unitHealthLabel.setVisible(false);
            if (damage >= 0)
                unitDamageLabel.setText("Damage: " +
                    damage);
            else
                unitDamageLabel.setText("");
        }
```

```
}

private JPanel orderPanel;
private JButton move;
private JButton attack;
private JButton stop;
private JButton hold;

private UnitState unitState;

private JLabel tooltipLabel = new JLabel("",
    SwingConstants.CENTER);

/**
* Константа для невыбранного приказа
*/
public final static int NO_MODE = 0;     //

/**
* Константа для приказа "стоп"
*/
public final static int MODE_STOP = 1;  // Приказ
    : Стоим
/**
* Константа для приказа "двигаться"
*/
public final static int MODE_MOVE = 2;  // Приказ
    : Двигаемся
/**
* Константа для приказа "атаковать"
*/
public final static int MODE_ATTACK = 3;// Приказ
    : Атакуем
/**
* Константа для приказа "держать местность"
*/
public final static int MODE_HOLD = 4;  // Приказ
    : Держим позицию

private String name;

private AutomatControlPanel A1;

/**
* Конструктор
*/
public ControlPanel() {
     super();
```

```
        move = new JButton(ImageMap.getImageIcon("
            buttons/move.gif"));
        attack = new JButton(ImageMap.getImageIcon("
            buttons/attack.gif"));
        stop = new JButton(ImageMap.getImageIcon("
            buttons/stop.gif"));
        hold = new JButton(ImageMap.getImageIcon("
            buttons/hold.gif"));
        unitState = new UnitState();

        addActionListener(this);

        setLayout(new GridLayout(1, 4));

        add(unitState);

        add(new JPanel());
        add(tooltipLabel);

        orderPanel = new JPanel();
        orderPanel.setLayout(new GridLayout(2, 2));
        orderPanel.add(move);
        orderPanel.add(attack);
        orderPanel.add(stop);
        orderPanel.add(hold);
        add(orderPanel);


        name = "ControlPanel";
        A1 = new AutomatControlPanel(this);
    };

    private void addActionListener(ActionListener
        listener) {
        move.addActionListener(listener);
        attack.addActionListener(listener);
        stop.addActionListener(listener);
        hold.addActionListener(listener);
    }

    /**
    * Добавляет к панели нового слушателя типа
        KeyListener
    * @param listener слушатель
    */
    public void addKeyListener(KeyListener listener) {
        super.addKeyListener(listener);
        move.addKeyListener(listener);
        attack.addKeyListener(listener);
        stop.addKeyListener(listener);
```

```
        hold.addKeyListener(listener);
        orderPanel.addKeyListener(listener);
    }

    /**
    * объект, управляемый панелью
    */
    private GameObject unit;

    /**
    * Возвращет объект, который управляется панелью в
       данный момент
    */
    public GameObject getUnit() {return unit;};

    /**
    * Устанавливает новый объект, управляемый панелью
    * @param gameObject объект, управляемый панелью
    */
    public void setUnit(GameObject gameObject) {unit
        = gameObject;};

    private GameObject selectedUnit; //Объект, на
       который была нажата мышь

    /**
    * Возвращает объект, выделенный пользователем
    */
    public GameObject getSelectedUnit() {return
       selectedUnit;};

    /**
    * Устанавливает новый объект, выделенный
       пользователем
    */
    public void setSelectedUnit(GameObject u) {
       selectedUnit = u;};


    private int mouseX, mouseY; //Точка, куда была
       нажата мышь

    /*автоматные методы и переменные*/
    private int order; //внутренний флаг, указывающий
       на приказ, указанный на панели

    private int curMode; //внутренний флаг,
       показывающий приказ, выполняемый объектом

    /**
```

```
* Возвращает текущую форму курсора мыши
*/
public Cursor getPreferredClientAreaCursor() {
    if (A1.getState() == 4) return ImageMap.
       getCursor("cursors/attack.gif", new Point
       (16, 16), "move cursor");
    if (A1.getState() == 5) return ImageMap.
       getCursor("cursors/move.gif", new Point
       (16, 16), "move cursor");
    return ImageMap.getCursor("cursors/select.gif
       ", new Point(5, 6), "usual cursor");
}

/**
* Обновляет панель
*/
public void update() {
    if (unit != null) {
        if (unit.isAlive()) {
            curMode = unit.getCurMode();
            unitState.set(unit.getName(), unit.
               getHealth(), 10);
        } else {
            unit = null;
            A1.A(10);
        }
    }else{
        curMode = NO_MODE;
        unitState.set("", -1, -1);
    };
    check();
};

private void check() {
    if ((unit == null) || (!unit.isControlled()))
        {
        orderPanel.setVisible(false);
        tooltipLabel.setVisible(false);
        return;
    }
    if (A1.getState() == 5) {
        tooltipLabel.setText("Select destination
           place");
        tooltipLabel.setVisible(true);
        orderPanel.setVisible(false);
        return;
    }
    if (A1.getState() == 4) {
        tooltipLabel.setText("Select target for
           attack");
```

```
            tooltipLabel.setVisible(true);
            orderPanel.setVisible(false);
            return;
        }
        tooltipLabel.setVisible(false);
        orderPanel.setVisible(true);
        move.setIcon(ImageMap.getImageIcon("buttons/
            move.gif"));
        stop.setIcon(ImageMap.getImageIcon("buttons/
            stop.gif"));
        attack.setIcon(ImageMap.getImageIcon("buttons/
            attack.gif"));
        hold.setIcon(ImageMap.getImageIcon("buttons/
            hold.gif"));
        if (selectedUnit != null) {
            //Максим! Здесь было неправильно
            switch (unit.getCurMode()) {
            case MODE_MOVE : move.setIcon(ImageMap.
                getImageIcon("buttons/move_c.gif"));
                break;
            case MODE_ATTACK : attack.setIcon(ImageMap
                .getImageIcon("buttons/attack_c.gif"));
                break;
            case MODE_STOP : stop.setIcon(ImageMap.
                getImageIcon("buttons/stop_c.gif"));
                break;
            case MODE_HOLD : hold.setIcon(ImageMap.
                getImageIcon("buttons/hold_c.gif"));
                break;
            };
        };
};

/*Неавтоматные методы*/
/**
* Передает событие "нажата кнопка стоп"
*/
public void pressBtnStop() {
    A1.A(3);
};

/**
* Передает событие "нажата кнопка двигаться"
*/
public void pressBtnMove() {
    A1.A(4);
};

/**
* Передает событие "нажата кнопка атаковать"
```

```
*/
public void pressBtnAttack() {
    A1.A(5);
};

/**
* Передает событие "нажата кнопка держать
   местность"
*/
public void pressBtnHold() {
    A1.A(6);
};

/**
* Передает событие "нажата левая кнопка мыши"
* @param o объект, на который нажата кнопка
*/
public void pressLMouse(GameObject o) {
    selectedUnit = o;
    A1.A(1);
};

/**
* Передает событие "нажата левая кнопка мыши"
* @param x x координата точки, на которую нажата
   мышка
* @param y y координата точки, на которую нажата
   мышка
*/
public void pressLMouse(int x, int y) {
    mouseX = x;
    mouseY = y;
    A1.A(2);
};

/**
* Передает событие "нажата правая кнопка мыши"
* @param o объект, на который нажата кнопка
*/
public void pressRMouse(GameObject o) {
    selectedUnit = o;
    A1.A(8);
};

/**
* Передает событие "нажата правая кнопка мыши"
* @param x x координата точки, на которую нажата
   мышка
* @param y y координата точки, на которую нажата
   мышка
```

```
    */
    public void pressRMouse(int x, int y) {
        mouseX = x;
        mouseY = y;
        A1.A(9);
    };

    /**
    * Передает событие "нажата кнопка ESC"
    */
    public void pressESC() {
        A1.A(7);
    };

    /**
    * Передает событие "управляемый юнит умер"
    */
    public void unitDie() {
        A1.A(10);
    };

    /**
    * Передает событие "юнит изменил свои
       характеристики"
    */
    public void unitUpdate() {
        if (unit != null) A1.A(11);
    };

    /**
    * Обрабатывает события от ActionListener
    */
    public void actionPerformed(ActionEvent e)     {
        Object source = e.getSource();
        if (source == stop) {
            pressBtnStop();
        } else if (source == move) {
            pressBtnMove();
        } else if (source == hold) {
            pressBtnHold();
        } else if (source == attack) {
            pressBtnAttack();
        };
    };
}
```

Здесь приведены листинги остальных модулей.

13.2.6 Листинг модуля BattleField.java

```
import java.awt.*;
import java.util.*;

/**
 * Класс, описывающий поле боя, содержащее такие
 * объекты, как здания (<code>Building</code>),
 * юниты (<code>Unit</code>) и местность (<code>
    Landscape</code>)
 * @author Gavrilov M.
 * @version 0.1 beta
 */

class BattleField {
     /**
      * Ландшафт
      */
     public Landscape landscape = new Landscape();

     /**
     * Центр поля боя
     */
     public Point viewCenter = new Point();

     /**
      * Вектор объектов GameObject, расположенных на
         данном поле боя
      */
     public Vector units = new Vector();

     /**
      * Вектор объектов Bullet на данном поле боя
      */
     public Vector bullets = new Vector();

     /**
      * Вектор объектов Burst на данном поле боя
      */
     public Vector bursts = new Vector();

     /**
      * Конструктор. Добавляет шесть зданий и шесть
         юнитов с
      * фиксированными координатами
      */
     public BattleField() {
         units.add(new Building(this, "Building
             #1", 1, 50, 10));
         units.add(new Building(this, "Building
             #2", 1, 50, 110));
```

```
        units.add(new Building(this, "Building
            #3", 1, 50, 210));
        units.add(new Building(this, "Building
            #4", 2, 400, 10));
        units.add(new Building(this, "Building
            #5", 2, 400, 110));
        units.add(new Building(this, "Building
            #6", 2, 400, 210));

        units.add(new Unit(this, "Unit
            #1", 1, 100, 10));
        units.add(new Unit(this, "Unit
            #2", 1, 100, 110));
        units.add(new Unit(this, "Unit
            #3", 1, 100, 210));

        units.add(new Unit(this, "Unit
            #4", 2, 300, 10));
        units.add(new Unit(this, "Unit
            #5", 2, 300, 110));
        units.add(new Unit(this, "Unit
            #6", 2, 300, 210));
    }

    public void screenToAbsolute(Point x) {
       x.x -= viewCenter.x;
       x.y -= viewCenter.y;
    }

    public void processTimer() {
        int i;
        for (i = 0; i < units.size(); i++) {
            if (units.elementAt(i) instanceof Unit) {
                Unit unit = (Unit)units.elementAt(i);
                if ((unit != null)&&(unit.isAlive()))
                    {
                    unit.runAllScanners();
                    switch (unit.getA0().getState())
                        {
                        case 0: break;

                        case 1: unit.move();
                                break;

                        case 5: unit.move();
                                break;

                        default: break;
                    };
                };
```

```
            }
        };

        for (i = 0; i < bullets.size(); i++) {
            Bullet bullet = (Bullet)bullets.elementAt
                (i);
                bullet.move();
            }
    }

    public void checkDeaded() {
        int i;
        for (i = 0; i < units.size(); i++) {
            GameObject go = (GameObject)units.
                elementAt(i);
            if ((go != null) && (!(go.isAlive()))) {
                units.remove(go);
            };
        };

        for (i = 0; i < bullets.size(); i++) {
            Bullet bullet = (Bullet)bullets.elementAt
                (i);
            if (!bullet.isAlive()) bullets.remove(
                bullet);
        };

        for (i = 0; i < bursts.size(); i++) {
            Burst burst = (Burst)bursts.elementAt(i);
            if (!burst.isAlive()) bursts.remove(burst
                );
        };

    }

    public void paint(Graphics g, int width, int
        height) {
        landscape.paint(g, viewCenter.x, viewCenter.y
            , width, height);
        g.translate(viewCenter.x, viewCenter.y);
        int i;
        for (i = 0; i < units.size(); i++) ((
            GameObject)units.elementAt(i)).paint(g);

        for (i = 0; i < bullets.size(); i++) ((Bullet
            )bullets.elementAt(i)).paint(g);

        for (i = 0; i < bursts.size(); i++)  ((Burst)
            bursts.elementAt(i)).paint(g);
    }
```

```
}
```

13.2.7 Листинг модуля Building.java

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;


/**
* Класс для работы со зданиями (радарами)
*/
public class Building extends GameObject{

    /**
    * Нанести повреждение текущему зданию
    * @param d величина повреждения
    */
    public void damage(int d) {

    };

    /**
    * Конструктор, создающий новое здание на заданном
       поле боя
    * @param a поле боя
    * @param name название здания
    * @param team номер команды
    * @param x x координата
    * @param y y координата
    */
    public Building(BattleField a, String name, int
       team, double x, double y) {
        setControlled(false);
        img = ImageMap.getImage("gif/turret" + team
            + ".gif");
        setAlive(true);

        setX(x);
        setY(y);
        setDirect(1, 0);
        deselect();
        setField(a);
        setSightRange(150);
        setAttackRange(0);
        setHealth(100);
        setName(name);
        setCurMode(Unit.MODE_STOP);
        setTeam(team);
```

```
        setCenter(-img.getWidth(null) / 2, -img.
            getHeight(null) / 2);
        setOvalRect(-img.getWidth(null) / 2, img.
            getHeight(null) / 2 - 15, img.getWidth(null
            ), 20);
        setHealthRect(-img.getWidth(null) / 2, -img.
            getHeight(null) / 2 - 10, img.getWidth(null
            ), 5);
    };

    /**
    * Движение здания (ничего не делает)
    */
    public void move() {

    };

    /**
    * метод проверяет, принадлежит ли точка (x, y)
       зданию
    * @param x x координата
    * @param y y координата
    */
    public boolean ifin(int x, int y) {
        return ((x - getX())*(x - getX()) + (y - getY
            ())*(y - getY()) < 400);
    };
};
```

13.2.8 Листинг модуля Bullet.java

```
import java.awt.*;

public class Bullet{

    private Unit owner;
    private GameObject target;

    private int y0;

    private int getState() {return y0; };

    //координаты объекта
    private double x;
    private double y;

    /**
    * Возвращает x координату
    */
    public int getX() {return (int)x;};
```

```
/**
* Возвращает y координату
*/
public int getY() {return (int)y;};

//вектор направления объекта
private double directX;
private double directY;

/**
* Устанавливает вектор движения снаряда
* @param x x составляющая
* @param y y составляющая
*/
public void setDirect(double x, double y) {
    directX = x;
    directY = y;
};

/**
* Конструктор. Создает новый снаряд, выпущенный
   указанным юнитом
* @param o юнит, выпустивший снаряд
*/
public Bullet(Unit o) {
    owner = o;
    target = owner.getTarget();
    x = o.getX();
    y = o.getY();
    y0 = 1; //состояние жизни, т.е. атаки
};

/**
* Пустой конструктор. Создает взорвавшийся снаряд
*/
public Bullet() {
    y0 = 0;
};

/**
* Передвигает снаряд на один шаг к цели
*/
public void move() {
    if ((x - target.getX()) * (x - target.getX())
      + (y - target.getY()) * (y - target.getY())
          > 10) {

        double directX = target.getX() - x;
        double directY = target.getY() - y;
```

```
            double d = Math.sqrt(directX * directX +
                directY * directY);

            x = x + 6 * directX/d;
            y = y + 6 * directY/d;

        }else{
            y0 = 0; //сдох
            target.damage(30);
//            owner.field.btnpanel.unitUpdate(); //
    проапдейтить панель управления
            owner.getField().bullets.remove(this);
            owner.getField().bursts.add(new Burst(
                target));
        };
    };

    /**
    * Рисует снаряд
    * @param g графический контекст
    */
    public void paint(Graphics g) {
        Color old = g.getColor();
        g.setColor(Color.yellow);

        double directX = target.getX() - x;
        double directY = target.getY() - y;

        double d = Math.sqrt(directX * directX +
            directY * directY);
        double l = 5;
        g.drawLine((int)(x - l/d * directX), (int)(y
            - l/d * directY), (int)(x + l/d * directX)
            , (int)(y + l/d * directY));

        g.setColor(old);
    };

    /**
    * Возвращает, жив ли снаряд
    */
    public boolean isAlive() {
        return (y0 == 1);
    };

};
```

13.2.9 Листинг модуля Burst.java

```
import java.awt.*;
import java.util.*;

    /**
    * Класс, описывающий взрыв юнита или здания
    */
public class Burst{

    /**
    * Класс, описывающий осколок от взрыва юнита или
       здания
    */
    private class Flash{
        public Vector2 vec;
        public Burst owner;
        public Color color;
        public Flash(Burst o) {
            owner = o;
            vec = new Vector2();
            double l = 1 + Math.random();
            double angle = Math.random() * 2 * Math.PI
               ;
            vec.x = 2 * l * Math.sin(angle);
            vec.y = l * Math.cos(angle);
            vec.z = 0;

            double k = Math.random();
            if (k < 0.5) color = Color.red;
            if (k >= 0.5) color = Color.yellow;
        };
    };

    private GameObject burstUnit;
    private double burstDeg;

    private boolean alive;

    /**
    * Возвращет, в процессе ли данный взрыв, или уже
       нет
    */
    public boolean isAlive() {return alive;};

    private Vector flashes = new Vector();

    /**
    * Создает новый взрыв объекта
    * @param o взрываемый объект
    */
    public Burst(GameObject o) {
```

```
        burstUnit = o;
        burstDeg = 0;
        alive = true;
        for(int i = 0; i < 40; i++) {
            flashes.add(new Flash(this));
        };
    };

    /**
    * Рисует взрыв
    * @param g графический контекст
    */
    public void paint(Graphics g) {
        Color old = g.getColor();
        g.setColor(Color.yellow);

        burstDeg = Math.sqrt(burstDeg * burstDeg + 20)
            ;
        if (burstDeg > 40) alive = false;
        int dx, dy;
        int x = (int)burstUnit.getX();
        int y = (int)burstUnit.getY();
        int r = (int)(30 / (burstDeg + 1));
        for (int i = 0; i < flashes.size(); i++) {
            Flash flash = (Flash)flashes.elementAt(i);
            dx = (int)(burstDeg * flash.vec.x);
            dy = (int)(burstDeg * flash.vec.y);
            g.setColor(flash.color);
            g.fillOval(dx + x - r, dy + y - r, 2*r, 2*
                r);
        };

        g.setColor(old);
    };
};
```

13.2.10 Листинг модуля ClientArea.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class ClientArea extends Panel {
         public Image buffer;
         private Graphics bg;

         public SwitchStarCraft owner;

         public ClientArea(SwitchStarCraft o) {
             super();
```

```
            setBackground(Color.black);
            owner = o;
            buffer = createImage(getSize().width,
               getSize().height);
            addComponentListener(new ComponentAdapter
               () {
               public void componentResized(
                  ComponentEvent e) {
                                    if ((getSize()
                                       .width
                                        >= 0) && (
                                       getSize().
                                       height
                                        >= 0))
                              buffer = createImage(
                                 getSize().width,
                                 getSize().height);
               }
            });
        }
        public void paint(Graphics g) {
               if (buffer == null)
                        buffer = createImage(getSize()
                           .width, getSize().height);
               Graphics bg = buffer.getGraphics();
               buffer.flush();
               bg.clearRect(0, 0, getSize().width,
                  getSize().height);
               owner.paintClient(bg);
               g.drawImage(buffer, 0, 0, null);
               bg.dispose();
        }

        public final void update(Graphics g) {
            paint(g);
        }

    };
```

13.2.11 Листинг модуля Controller.java

```
import java.awt.*;
import java.awt.event.*;

/**
 * Класс, обрабатывающий внешние события от
    пользователя,
 * и делающий соответствующие операции над BattleField
    .
```

```
 * Выстпает в качестве посредника между элементами
    управления
 * и реакцией игровой обстановки на них.
 * @author Gavrilov M.
 */
public class Controller
        implements MouseMotionListener, MouseListener
           , KeyListener
{
        /**
         * Контроллируемое поле
         */
        protected BattleField field;

        /**
         * Панель с кнопками управления
         */
        ControlPanel control;

        /**
         * Область вывода
         */
        ClientArea clientArea;

        ControlPanel btnpanel;

        /**
         * конструктор контроллера
         * @param field контроллируемое поле
         */
        public Controller(BattleField field,
           ControlPanel control, ClientArea clientArea
           ) {
                this.field = field;
                this.control = control;
                this.clientArea = clientArea;
        clientArea.addMouseMotionListener(this);
        clientArea.addMouseListener(this);

        clientArea.addKeyListener(this);
        control.addKeyListener(this);

        btnpanel = control;
        control.setUnit(null);
        }

        /**
         * Получить связанное с этим контроллером поле
            боя
         * @return поле боя
```

```
 */
public BattleField getBattleField() {
        return field;
}

/**
 * Получить понель управления
 * @return панель управления
 */
 public ControlPanel getControlPanel() {
        return control;
 }

/**
 * Получить область вывода
 * @return область вывода
 */
 public ClientArea getClientArea() {
        return clientArea;
 }

/**
 * Нажатие левой кнопки мыши в экранных
    координатах (x, y)
 * @param x x координата
 * @param y y координата
 */
public void mouseLPressed(int x, int y) {
        Point p = new Point(x, y);
        field.screenToAbsolute(p);
        int mouseX = p.x;
        int mouseY = p.y;
for (int i = 0; i < field.units.size(); i++) {
    GameObject AnotherUnit = (GameObject)field
       .units.elementAt(i);
    if (AnotherUnit != null)
    if ((AnotherUnit.ifin(mouseX, mouseY))&&(
       AnotherUnit != btnpanel.getUnit())) {
        btnpanel.pressRMouse(AnotherUnit);
        return;
    };
}
if (btnpanel.getUnit() != null) btnpanel.
   getUnit().deselect();
btnpanel.pressRMouse(mouseX, mouseY);
if (btnpanel.getUnit() != null) btnpanel.
   getUnit().select();
}

/**
```

```
 * Нажатие правой кнопки мыши в экранных
    координатах (x, y)
 * @param x x координата
 * @param y y координата
 */
public void mouseRPressed(int x, int y) {
        Point p = new Point(x, y);
        field.screenToAbsolute(p);
        int mouseX = p.x;
        int mouseY = p.y;
boolean flag = true; //флаг используется для
   того, чтобы определить, был выбран юнит или
    точка
for (int i = 0; i < field.units.size(); i++) {
    GameObject AnotherUnit = (GameObject)field
       .units.elementAt(i);
    if (AnotherUnit != null)
    if (AnotherUnit.ifin(mouseX, mouseY)) {
        if (btnpanel.getUnit() != null) {
            btnpanel.getUnit().deselect();
        }
        btnpanel.pressLMouse(AnotherUnit);
        if (btnpanel.getUnit() != null) {
            btnpanel.getUnit().select();
        }
        flag = false;
    };
};
if (flag) {
    if (btnpanel.getUnit() != null) btnpanel.
       getUnit().deselect();
    btnpanel.pressLMouse(mouseX, mouseY);
    if (btnpanel.getUnit() != null) btnpanel.
       getUnit().select();
};
}

/**
 * Событие от таймера
 */
public void processTimer() {
        field.checkDeaded();
        field.processTimer();
        clientArea.repaint();
        btnpanel.unitUpdate();
        control.update();
}

/**
 * Запрос на перерисовку
```

```
     * @param g графический контест
     */
    public void processPaint(Graphics g) {
    clientArea.setCursor(control.
       getPreferredClientAreaCursor());
            field.paint(g, clientArea.getWidth(),
               clientArea.getHeight());
    }


public void mouseDragged(MouseEvent e) {
};

public void mouseMoved(MouseEvent e) {
};

public void mouseClicked(MouseEvent e) {};

public void mouseEntered(MouseEvent e) {}

public void mouseExited(MouseEvent e) {}

public void mousePressed(MouseEvent e) {
    if ((e.getModifiers() & MouseEvent.
       BUTTON1_MASK) == 0)
            mouseLPressed(e.getX(), e.getY());
    else
            mouseRPressed(e.getX(), e.getY());
}

public void mouseReleased(MouseEvent e) {}


public void keyTyped(KeyEvent e) {
}

public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_ESCAPE) {
        btnpanel.pressESC();
    } else if (e.getKeyCode() == KeyEvent.VK_DOWN)
        {
            field.viewCenter.y -= 20;
    } else if (e.getKeyCode() == KeyEvent.VK_UP) {
            field.viewCenter.y += 20;
    } else if (e.getKeyCode() == KeyEvent.VK_LEFT)
        {
            field.viewCenter.x += 20;
    } else if (e.getKeyCode() == KeyEvent.VK_RIGHT
       ) {
            field.viewCenter.x -= 20;
```

```
        }
    }
    public void keyReleased(KeyEvent e) {}
}
```

13.2.12 Листинг модуля GameObject.java

```
import java.awt.*;

/**
* <code>GameObject</code>
* Базовый класс для любого объекта в игре, являющегося
    юнитом или зданием
*/
public abstract class GameObject{

    /**
    * Овал, показывающий на то, что объект выделен
    */
    private Rectangle ovalRect = new Rectangle();
    protected void setOvalRect(int x, int y, int width
        , int height) {
        ovalRect.x = x;
        ovalRect.y = y;
        ovalRect.width = width;
        ovalRect.height = height;
    };


    /**
    * Прямоугольник, на котором показывается здоровье
       объекта в случае, если объект выделен
    */
    private Rectangle healthRect = new Rectangle();
    protected void setHealthRect(int x, int y, int
       width, int height) {
        healthRect.x = x;
        healthRect.y = y;
        healthRect.width = width;
        healthRect.height = height;
    };

    /**
    * Центр объекта
    */
    private Point center = new Point();

    /**
    * Установить центр объекта
    */
```

```
protected void setCenter(int x, int y) {
    center.x = x;
    center.y = y;
}

/**
* Название объекта
*/
private String name;

/**
* Получить название объекта
*/
protected String getName() {
    return name;
};

/**
* Установить название объекта
*/
protected void setName(String n) {
    name = n;
};

/**
* Поле боя, на котором расположен объект
*/
private BattleField field;

protected BattleField getField() {return field;};
protected void setField(BattleField f) {field = f
    ;};

/**
* Номер команды, которой принадлежит данный объект
*/
private int team;

/**
* Получить номер команды
*/
protected int getTeam() {
    return team;
};

/**
* Установить номер команды
*/
protected void setTeam(int t) {
    team = t;
```

```
};

/**
* физическая координата объекта по оси x
*/
private double x;

/**
* физическая координата объекта по оси y
*/
private double y;


protected double getX() {return x;};
protected double getY() {return y;};

protected void setX(double xx) {x = xx;};
protected void setY(double yy) {y = yy;};

/**
* радиус области видимости объекта
*/
private int sightRange;

protected int getSightRange() {
    return sightRange;
};

protected void setSightRange(int s) {
    sightRange = s;
};

/**
* радиус области атаки объекта
*/
private int attackRange;

/**
* получить радиус области атаки объекта
*/
protected int getAttackRange() {
    return attackRange;
};

protected void setAttackRange(int r) {
    attackRange = r;
};

/**
* Здоровье (величина жизненной силы объекта)
```

```
*/
private int health;

protected int getHealth() {return health; };

protected void setHealth(int h) {health = h; };

/**
* может ли пользователь управлять объектом?
*/
private boolean controlled;

protected boolean isControlled() {
    return controlled;
}

protected void setControlled(boolean c) {
    controlled = c;
}

protected void damage(int d) {
    health = health - d;
};

/**
* Получить целую часть координаты x;
*/
protected int getIntX() {return (int)x;};

/**
* Получить целую часть координаты y;
*/
protected int getIntY() {return (int)y;};

//вектор направления объекта
private double directX;

private double directY;

protected double getDirectX() {return directX;};
protected double getDirectY() {return directY;};

protected void setDirectX(double dx) {directX = dx
    ;};
protected void setDirectY(double dy) {directY = dy
    ;};

protected void setDirect(double x, double y) {
    directX = x;
    directY = y;
```

```
};

/**
* Координата x пункта назначения объекта
*/
private int destinationX;

/**
* Координата y пункта назначения объекта
*/
private int destinationY;

protected int getDestinationX() {return
    destinationX;};
protected int getDestinationY() {return
    destinationY;};

protected void setDestinationX(int d) {
    destinationX = d;};
protected void setDestinationY(int d) {
    destinationY = d;};

protected void setDestination(int x, int y) {
     destinationX = x;
     destinationY = y;
};

private boolean selected;
protected boolean ifSelected() {return selected
    ; };
protected void select() {selected = true; };
protected void deselect() {selected = false; };

protected Image img;

/**
* Живой объект или нет
*/
private boolean alive;

protected boolean isAlive() {return alive;};

protected void setAlive(boolean a) {alive = a;};

/**
* Текущий режим (исполняемый приказ)
*/
private int curMode;

/**
```

```
* Получить текущий режим
*/
protected int getCurMode() {return curMode; };

/**
* Установить новый текущий режим
*/
protected void setCurMode(int c) {curMode = c; };

/**
* ОСуществить движение объекта к его текущей цели
*/
public abstract void move();

/**
* Нарисовать объект и нарисовать зеленый овальчик
   , если объект выделен.
* Этот метод использует draw, который должэен быть
    переопределен для каждого объекта
*/
public void paint(Graphics g) {
    Color old = g.getColor();
    Graphics2D g2 = (Graphics2D)g;
    if (selected) {
        g2.setColor(Color.green);
        g2.drawOval((int)(x + ovalRect.x), (int)(y
            + ovalRect.y), (int)(ovalRect.width)
           , (int)(ovalRect.height));
    }

    draw(g);

    if (selected) {
        g2.setColor((health < 33) ? Color.red : ((
           health < 66) ? Color.yellow : Color.
           green));
        g2.fillRect((int)x + healthRect.x, (int)y
            + healthRect.y, healthRect.width *
           health / 100, healthRect.height);
        g2.setColor(Color.gray);
        g2.drawRect((int)x + healthRect.x, (int)y
            + healthRect.y, healthRect.width,
           healthRect.height);
    }
    g2.setColor(old);
}

/**
* Нарисовать объект (для большинства объектов этот
    метод переопределяется)
```

```
    */
    public void draw(Graphics g) {
        g.drawImage(img, (int)x + center.x, (int)y +
            center.y, null);
    }

    /**
    * Проверяет, принадлежит ли точка (xx, yy) объекту
    */
    public abstract boolean ifin(int xx, int yy);

};
```

13.2.13 Листинг модуля ImageMap.java

```
import java.awt.*;
import javax.swing.*;
import java.util.*;


abstract public class ImageMap {
    protected static Map imageMap = new HashMap(); //
        хэш картинок
    protected static Map cursorMap = new HashMap()
        ; // хэш курсоров

    protected static class CursorInfo {
        public String path;
        public Point center;
        public String name;
        public CursorInfo(String tpath, Point tcenter
            , String tname) {
            path = tpath;
            center = tcenter;
            name = tname;
        }
    };

    /**
     * Берем ImageIcon с путем path
     */
    public static ImageIcon getImageIcon(String path)
        {
                if (imageMap.containsKey(path))
                        return (ImageIcon)imageMap.get
                            (path);
                ImageIcon img = new ImageIcon(path);
                imageMap.put(path, img);
                return img;
    }
```

```
    /**
     * Берем Image с путем path
     */
    public static Image getImage(String path) {
        return getImageIcon(path).getImage();
    }


        public static Cursor getCursor(String path,
           Point center, String name) {
                CursorInfo ci = new CursorInfo(path,
                   center, name);
                if (cursorMap.containsKey(ci))
                    return (Cursor)cursorMap.get(ci);
                Cursor cursor =
                    Toolkit.getDefaultToolkit().
                       createCustomCursor(
                                getImage(path),
                                center,
                                name);
                cursorMap.put(ci, cursor);
                return cursor;
        }

}
```

13.2.14 Листинг модуля LandScape.java

```
import java.awt.*;
import java.awt.image.*;

/**
 * Класс предназначен для хранения, отображения
 * и поддержки манипулирования картой местности
 * @author Gavrilov M.
 * @version 0.1 beta
 */

public class Landscape {
        /**
         *
         */
        protected Image texture;
        protected int oldX = Integer.MAX_VALUE;
        protected int oldY = Integer.MAX_VALUE;
        protected int oldWidth = Integer.MAX_VALUE;
        protected int oldHeight = Integer.MAX_VALUE;
        protected BufferedImage cash = null;
```

```
        public Landscape() {
                texture = ImageMap.getImage("gif/
                   landscape.gif");
        }

        public void paint(Graphics g, int x, int y,
           int width, int height) {
                if ((width == oldWidth) && (height ==
                   oldHeight) && (x == oldX) && (y ==
                   oldY)) {
                        g.drawImage(cash, 0, 0, null);
                        return;
                }
                if ((oldWidth != width) || (oldHeight
                    != height)) {
                        cash = new BufferedImage(width
                           , height, BufferedImage.
                           TYPE_INT_RGB);
                        oldWidth = width; oldHeight =
                           height;
                }
                oldX = x; oldY = y;
                Graphics gg = cash.getGraphics();
                int tx = x & 31;
                int ty = y & 31;
                for (int i = -1; i <= width / 32; i++)
                        for (int j = -1; j <= height
                            / 32; j++)
                                gg.drawImage(texture,
                                   tx + 32 * i - x, ty
                                    + 32 * j - y, null
                                   );
                g.drawImage(cash, 0, 0, null);
        }
}
```

13.2.15 Листинг модуля Log.java

```
import java.io.*;
/**
 * Класс поддержки ведения логфайла
 * @author Gavrilov M.
 * @version 0.1 beta
 */
public class Log {
     public static FileOutputStream fileOutputStream;

     public synchronized static void write(String l) {
         try{
             for (int i = 0; i < l.length(); i++) {
```

```
                fileOutputStream.write((byte)(l.
                    charAt(i)));
            };
        }catch (IOException e) { };
    };

    public synchronized static void writeln(String l)
        {
        try{
            for (int i = 0; i < l.length(); i++) {
                fileOutputStream.write((byte)(l.
                    charAt(i)));
            };
            fileOutputStream.write(13);
        }catch (IOException e) { };
    };

    static {
        try{
           fileOutputStream = new FileOutputStream(
              new File("log.txt"));
        }catch (FileNotFoundException e) { };
    }
}
```

13.2.16 Листинг модуля SwitchStarCraft.java

```
import java.awt.*;

/**
* <code>GameObject</code>
* Базовый класс для любого объекта в игре, являющегося
    юнитом или зданием
*/
public abstract class GameObject{

    /**
    * Овал, показывающий на то, что объект выделен
    */
    private Rectangle ovalRect = new Rectangle();
    protected void setOvalRect(int x, int y, int width
        , int height) {
        ovalRect.x = x;
        ovalRect.y = y;
        ovalRect.width = width;
        ovalRect.height = height;
    };


    /**
```

```
* Прямоугольник, на котором показывается здоровье
    объекта в случае, если объект выделен
*/
private Rectangle healthRect = new Rectangle();
protected void setHealthRect(int x, int y, int
    width, int height) {
     healthRect.x = x;
     healthRect.y = y;
     healthRect.width = width;
     healthRect.height = height;
};

/**
* Центр объекта
*/
private Point center = new Point();

/**
* Установить центр объекта
*/
protected void setCenter(int x, int y) {
     center.x = x;
     center.y = y;
}

/**
* Название объекта
*/
private String name;

/**
* Получить название объекта
*/
protected String getName() {
     return name;
};

/**
* Установить название объекта
*/
protected void setName(String n) {
     name = n;
};

/**
* Поле боя, на котором расположен объект
*/
private BattleField field;

protected BattleField getField() {return field;};
```

```
protected void setField(BattleField f) {field = f
    ;};

/**
* Номер команды, которой принадлежит данный объект
*/
private int team;

/**
* Получить номер команды
*/
protected int getTeam() {
    return team;
};

/**
* Установить номер команды
*/
protected void setTeam(int t) {
    team = t;
};

/**
* физическая координата объекта по оси x
*/
private double x;

/**
* физическая координата объекта по оси y
*/
private double y;


protected double getX() {return x;};
protected double getY() {return y;};

protected void setX(double xx) {x = xx;};
protected void setY(double yy) {y = yy;};

/**
* радиус области видимости объекта
*/
private int sightRange;

protected int getSightRange() {
    return sightRange;
};

protected void setSightRange(int s) {
    sightRange = s;
```

```
};

/**
* радиус области атаки объекта
*/
private int attackRange;

/**
* получить радиус области атаки объекта
*/
protected int getAttackRange() {
    return attackRange;
};

protected void setAttackRange(int r) {
    attackRange = r;
};

/**
* Здоровье (величина жизненной силы объекта)
*/
private int health;

protected int getHealth() {return health; };

protected void setHealth(int h) {health = h; };

/**
* может ли пользователь управлять объектом?
*/
private boolean controlled;

protected boolean isControlled() {
    return controlled;
}

protected void setControlled(boolean c) {
    controlled = c;
}

protected void damage(int d) {
    health = health - d;
};

/**
* Получить целую часть координаты x;
*/
protected int getIntX() {return (int)x;};

/**
```

```
* Получить целую часть координаты y;
*/
protected int getIntY() {return (int)y;};

//вектор направления объекта
private double directX;

private double directY;

protected double getDirectX() {return directX;};
protected double getDirectY() {return directY;};

protected void setDirectX(double dx) {directX = dx
    ;};
protected void setDirectY(double dy) {directY = dy
    ;};

protected void setDirect(double x, double y) {
     directX = x;
     directY = y;
};

/**
* Координата x пункта назначения объекта
*/
private int destinationX;

/**
* Координата y пункта назначения объекта
*/
private int destinationY;

protected int getDestinationX() {return
    destinationX;};
protected int getDestinationY() {return
    destinationY;};

protected void setDestinationX(int d) {
    destinationX = d;};
protected void setDestinationY(int d) {
    destinationY = d;};

protected void setDestination(int x, int y) {
     destinationX = x;
     destinationY = y;
};

private boolean selected;
protected boolean ifSelected() {return selected
    ; };
```

```
protected void select() {selected = true; };
protected void deselect() {selected = false; };

protected Image img;

/**
* Живой объект или нет
*/
private boolean alive;

protected boolean isAlive() {return alive;};

protected void setAlive(boolean a) {alive = a;};

/**
* Текущий режим (исполняемый приказ)
*/
private int curMode;

/**
* Получить текущий режим
*/
protected int getCurMode() {return curMode; };

/**
* Установить новый текущий режим
*/
protected void setCurMode(int c) {curMode = c; };

/**
* ОСуществить движение объекта к его текущей цели
*/
public abstract void move();

/**
* Нарисовать объект и нарисовать зеленый овальчик
   , если объект выделен.
* Этот метод использует draw, который должэен быть
    переопределен для каждого объекта
*/
public void paint(Graphics g) {
    Color old = g.getColor();
    Graphics2D g2 = (Graphics2D)g;
    if (selected) {
        g2.setColor(Color.green);
        g2.drawOval((int)(x + ovalRect.x), (int)(y
            + ovalRect.y), (int)(ovalRect.width)
           , (int)(ovalRect.height));
    }
```

```
        draw(g);

        if (selected) {
            g2.setColor((health < 33) ? Color.red : ((
                health < 66) ? Color.yellow : Color.
                green));
            g2.fillRect((int)x + healthRect.x, (int)y
                + healthRect.y, healthRect.width *
                health / 100, healthRect.height);
            g2.setColor(Color.gray);
            g2.drawRect((int)x + healthRect.x, (int)y
                + healthRect.y, healthRect.width,
                healthRect.height);
        }
        g2.setColor(old);
    }

    /**
    * Нарисовать объект (для большинства объектов этот
        метод переопределяется)
    */
    public void draw(Graphics g) {
        g.drawImage(img, (int)x + center.x, (int)y +
            center.y, null);
    }

    /**
    * Проверяет, принадлежит ли точка (xx, yy) объекту
    */
    public abstract boolean ifin(int xx, int yy);

};
```

13.2.17 Листинг модуля Vector2.java

```
/**
* Класс, описывающий трехмерный вектор
*/
public class Vector2{
    /**
    * x координата
    */
    public double x;

    /**
    * y координата
    */
    public double y;

    /**
```

```
* z координата
*/
public double z;

/**
* Конструктор. Создает вектор с нулевыми
   координатами
*/
public Vector2() {
    x = 0;
    y = 0;
    z = 0;
};

/**
* Конструктор. Создает вектор с указанными
   координатами
* @param x x координата
* @param y y координата
* @param z z координата
*/
public Vector2(double x, double y, double z) {
    this.x = x;
    this.y = y;
    this.z = z;
};

/**
* Вычисляет угол между текущим вектором и вектором
    v2
*/
public double Angle(Vector2 v2) {
    double v3x = x - v2.x;
    double v3y = y - v2.y;
    double v3z = z - v2.z;
    double l1 = Math.sqrt(x * x + y * y + z * z);
    double l2 = Math.sqrt(v2.x * v2.x + v2.y * v2.
       y + v2.z * v2.z);
    double l3 = (v3x * v3x + v3y * v3y + v3z * v3z
       );
    double d = (l3 - l1*l1 - l2*l2)/(2*l1*l2);
    return Math.acos(d);
};

/**
* Вычисляет определитель матрицы ((A, B),(C, D))
*/
public static double det2(double A, double B,
   double C, double D) {
    return A*D - B*C;
```

```
}

/**
* Вычисляет определитель матрицы 3 на 3
*/
public static double det3(double A1, double A2,
    double A3,
                      double B1, double B2, double
                         B3,
                      double C1, double C2, double
                         C3) {
                      return (A1*B2*C3 + B1*C2*A3 +
                         C1*A2*B3 - C1*B2*A3 - B1*A2
                         *C3 - A1*C2*B3);
}

/**
* Вычисляет общий орт к векторам v1 и v2 и
    записывает его в текущий вектор
*/
public void Ort(Vector2 v1, Vector2 v2) {
double xx = 0;
    double yy = 0;
    double zz = 0;
    double D = det2(v1.y, v1.z, v2.y, v2.z);

    double A;
    double B;

    if (D != 0) {
        xx = 1;
        A = det2(-xx * v1.x, v1.z, -xx * v2.x, v2.
            z);
        B = det2(v1.y, -xx * v1.x, v2.y, -xx * v2.
            x);
        yy = A / D;
        zz = B / D;
    }else{
        D = det2(v1.z, v1.x, v2.z, v2.x);
    if (D != 0) {
        yy = 1;
                A = det2(-yy * v1.y, v1.x, -yy *
                    v2.y, v2.x);
            B = det2(v1.z, -yy * v1.y, v2.z, -yy
                * v2.y);
        zz = A / D;
                xx = B / D;
        }else{
            D = det2(v1.x, v1.y, v2.x, v2.y);
                if (D != 0) {
```

```
                zz = 1;
                    A = det2(-zz * v1.z, v1.y, -zz *
                        v2.z, v2.y);
                    B = det2(v1.x, -zz * v1.z, v2.x, -
                        zz * v2.z);
                xx = A / D;
                yy = B / D;
                }else{
                    xx = 0;
                    yy = 0;
                    zz = 0;
                };
            };
        };
        x = xx;
        y = yy;
        z = zz;

    };


    /**
    * Возвращает угол по указанному синусу и косинусу
    */
    static double acossin(double xsin, double xcos) {
        double ang = Math.acos(xcos);
        if (xsin < 0) {ang = 2 * Math.PI - ang; };
        return ang;
    }

    /**
    * Выводит на экран координаты вектора
    */
    public void print() {
        System.out.println(x + " " + y + " " + z);
    };

};
```