Санкт-Петербургский государственный институт информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

А.В. Беляев, Д.И. Суясов, А.А. Шалыто

Компьютерная игра «Космонавт»

Версия 0.15

Проектная документация

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

Санкт-Петербург
2004

Содержание

Введение

Предлагаемая проектная документация описывает учебный пример использования SWITCH-технологии при разработке простых компьютерных игр. Логика управления игры «Космонавт» реализована в виде конечного автомата. Красивое графическое оформление игры позволяет визуально отразить текущее состояние автомата. Поэтому сама программа и ее документация могут служить наглядным примером применения автоматного программирования.

Программа иллюстрирует предложенное в работе [1] разделение состояний в программе на два типа: управляющие и вычислительные. При этом, находясь в одном из управляющих состояний, система может проходить большое число вычислительных состояний. Например, находясь в состоянии «Полет», вычисляется функция, описывающая физику полета, которая в ходе вычисления принимает большое число состояний.

Предложенный подход делает понятие «состояние» конструктивным, так как обычно [2,3] состояния на указанные типы не разделяются, а под состояниями понимаются значения ячеек памяти, число которых огромно.

Предложенный подход весьма близок к подходу, развиваемому при построении гибридных динамических систем [4].

Обоснован выбор основных вычислительных алгоритмов, использованных в игре.

Программа написана на языке *Java*. Этот язык прост в использовании и изучении. Он полностью объектно-ориентированный и платформенно-независим.

Подробная документация по каждому классу программы представлена в формате *HTML*. Она сгенерирована средствами *JavaDoc* из исходных файлов. Отметим, что эта документация не заменяет данный документ, а лишь дополняет его, что не является общепринятым. Фрагмент документации *JavaDoc* приведен в приложении 3.

1. Краткое руководство пользователя

Пример внешнего вида окна программы приведен на рис.1.

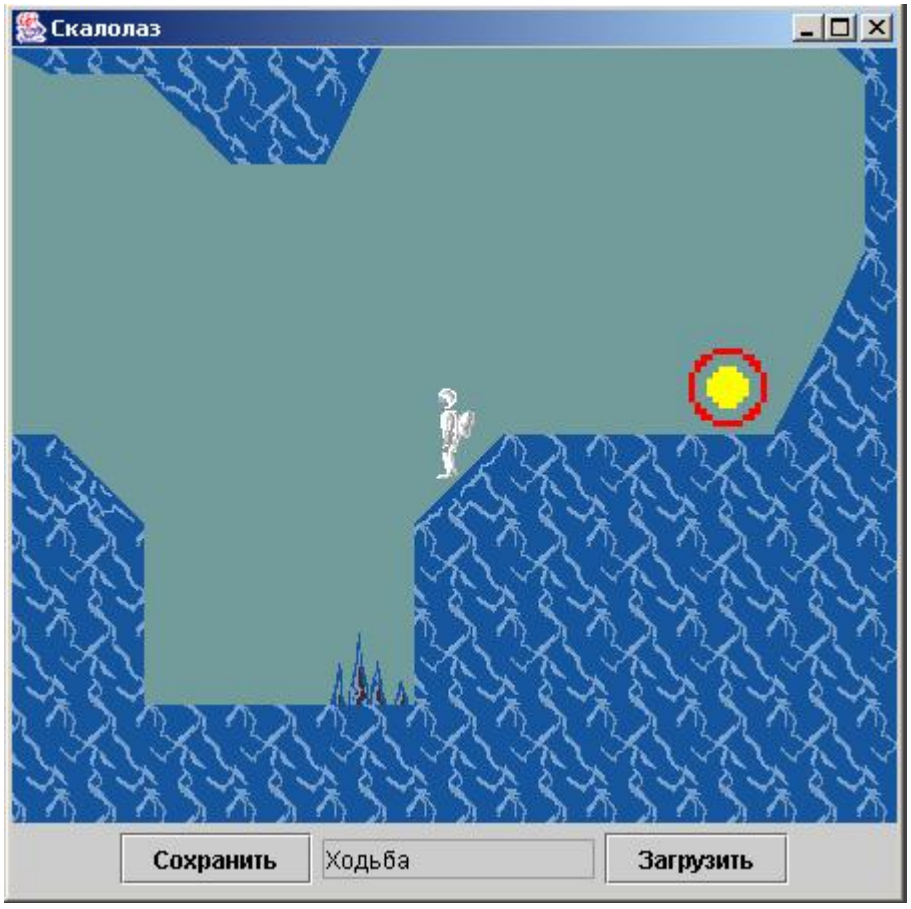


Рис. 1. Пример окна программы

В центре окна находится игровое поле, на которое осуществляется вывод графики. Внизу (слева и справа) расположены кнопки «Сохранить» и «Загрузить», позволяющие запоминать и загружать состояния игры. Между кнопками расположено текстовое поле, в которое выводится название текущего состояния автомата, управляющего игрой.

Игра состоит из одиннадцати уровней. Игрок последовательно проходит все уровни, и по завершении последнего из них игра начинается сначала.

Для прохождения уровня необходимо добраться до выхода – мигающего красно-желтого круга. Космонавт может ходить влево и вправо, забираться на наклонные поверхности, если они не очень крутые. Для обеспечения ходьбы используются кнопки «4» и «6» на цифровой клавиатуре. При этом космонавт может упасть с уступа и, если высота падения слишком велика, разбиться. После смерти уровень необходимо пройти заново. Для космонавта смертельно также его соприкосновение с водой и шипами.

Для преодоления опасных пропастей, высоких препятствий, воды и шипов в арсенале космонавта есть гарпун-веревка. Из состояния ходьбы при нажатии игроком кнопки «5»

цифровой клавиатуры космонавт переходит в состояние прицеливания. При повторном нажатии на кнопку «5» он использует гарпун. Если длины веревки достаточно, и гарпун ударится в поверхность, в которую может воткнуться, то космонавт повисает на веревке. В этом состоянии игрок может раскачивать космонавта с помощью кнопок «4» и «6» на цифровой клавиатуре. Можно укорачивать и удлинять веревку с помощью кнопок «8» и «2». Очередное нажатие кнопки «5» приводит к тому, что космонавт отпускает веревку и переходит «в свободный полет».

2. Постановка задачи

Задача, реализуемая в работе, состоит в создании описанной игры с использованием автоматного подхода.

3. Архитектура программы

В целом программу можно рассматривать как реализацию шаблона программирования MVC (Model-View-Controller) [5,6]. В проекте в качестве Модели используется класс *RockmanGame*, в качестве Представления – *RockmanView*, а Контроллером является главный класс программы - *Main*.

Класс *RockmanGame* отвечает за математическую модель игрового мира, за изменение математических данных, за их сохранение и загрузку. Внутри этого класса реализован управляющий автомат. Класс *RockmanView*, основываясь на данных математической модели, обеспечивает ее представление на экране. Главный класс программы *Main* агрегирует классы *RockmanGame* и *RockmanView*. Он является посредником между моделью, ее представлением и пользователем.

Диаграмма классов программы представлена на рис. 2.

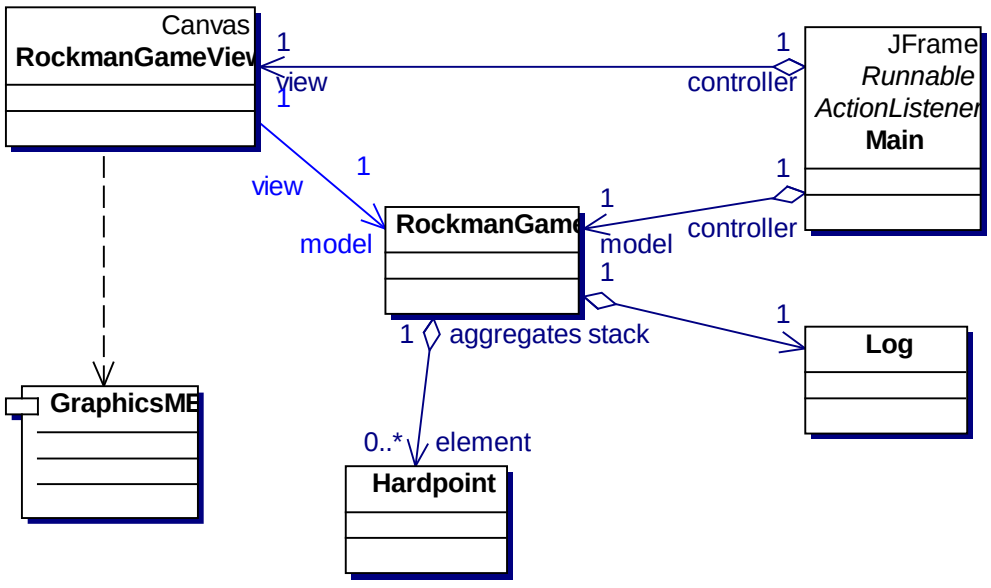


Рис. 2. Диаграмма классов

Класс *RockmanGame* содержит стек объектов класса *Hardpoint*, который представляет точку перегиба веревки. Класс *RockmanGame* агрегирует объект класса *Log*, который используется для протоколирования работы автомата. Класс *Log*, в свою очередь, содержит объект класса *FileOutputStream* для файлового вывода, и реализует несколько простых методов для ведения протокола. В терминологии шаблонов отношение классов *Log* и *FileOutputStream* называется *Facade pattern* [6].

Пример протокола, построенного с помощью класса *Log*, приведен в приложении 1.

Класс *RockmanView* наследуется от стандартного класса *JCanvas* платформы *Java*. Он отображает состояние математической модели, используя набор картинок. Для более удобного вывода спрайтов (спрайт - совокупность картинок) на экран класс *RockmanView* использует класс *GraphicsME* вместо стандартного контекста *Graphics*. Класс *GraphicsME* содержит стандартный класс *Graphics* и предоставляет более удобные для отображения спрайтов методы. Это еще один пример шаблона *Facade*.

Исходный текст программы приведен в приложении 2.

Подробная документация по каждому классу программы представлена в формате *HTML*. Она сгенерирована средствами *JavaDoc* из исходных файлов. Отметим, что эта документация не заменяет данный документ, а лишь дополняет его, что не является общепринятым. Фрагмент *JavaDoc*-документации приведен в приложении 3.

4. Автомат

Программа содержит один автомат, реализующий логику игры. Его описание и реализация выполнены на основе работы [7].

4.1. Нумерация и перечень состояний

На рис.3 – 8 приведены скриншоты, соответствующие состояниям автомата.

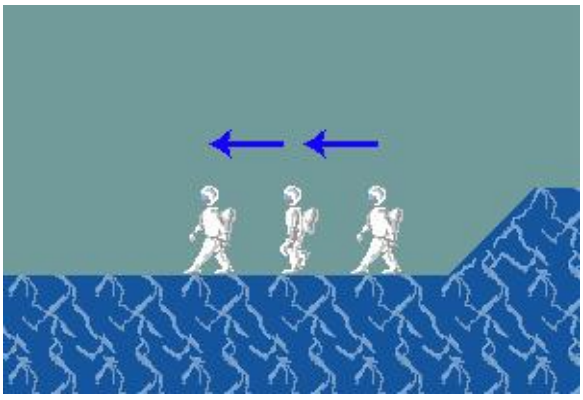


Рис. 3. Состояние 0. Ходьба

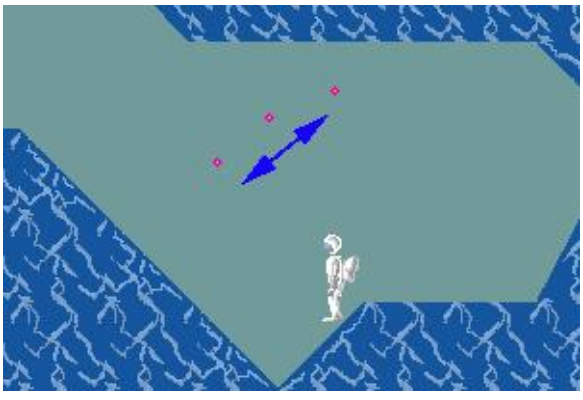


Рис. 4. Состояние 1.Прицеливание

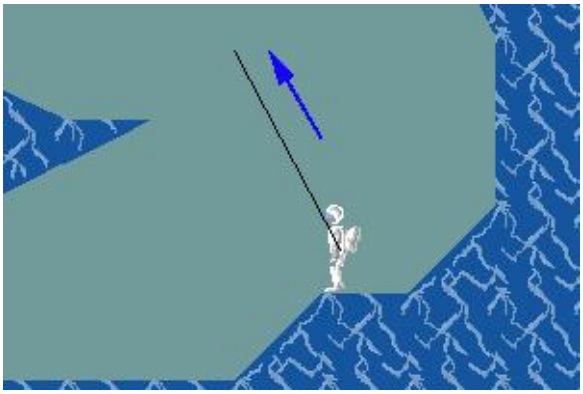


Рис. 5. Состояние 2. Стрельба

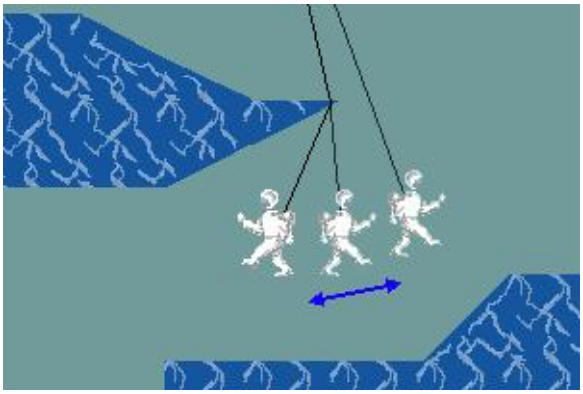


Рис. 6. Состояние 3. Качание

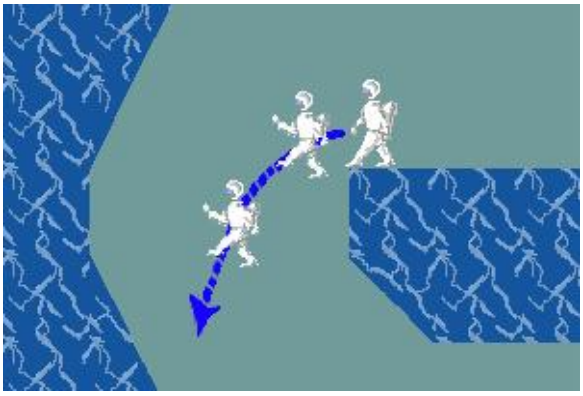


Рис. 7. Состояние 4. Полет

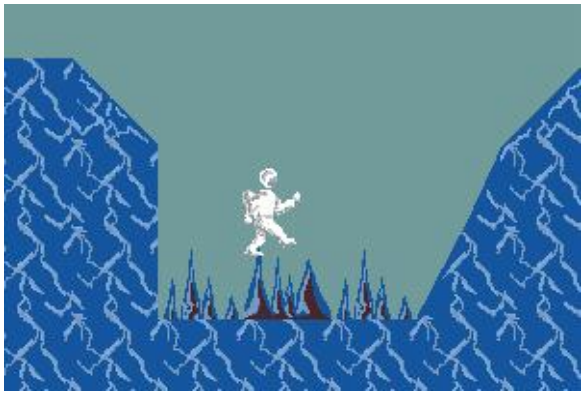
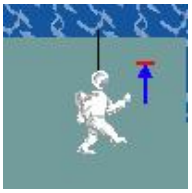


Рис. 8. Состояние 5. Смерть

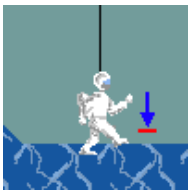
4.2. Нумерация и перечень событий

0	`EVENT_0_KEY_NONE_TIMER`	Событие от таймера
1	`EVENT_1_KEY_CENTER`	Нажатие клавиши «5»
2	`EVENT_2_KEY_LEFT`	Нажатие клавиши «4»
3	`EVENT_3_KEY_RIGHT`	Нажатие клавиши «6»
4	`EVENT_4_KEY_UP`	Нажатие клавиши «8»
5	`EVENT_5_KEY_DOWN`	Нажатие клавиши «2»

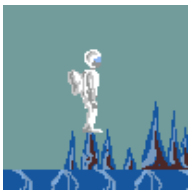
4.3. Перечень входных переменных и их описание



X5
Космонавт соприкасается головой со стеной при укорачивании веревки.



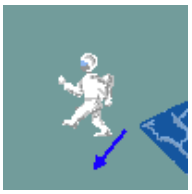
X6
Космонавт соприкасается ногами со стеной при удлинении веревки.



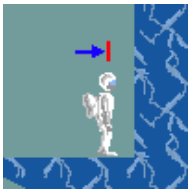
X7
Космонавт находится на смертельном участке карты.

X8
Космонавт находится на выходе.

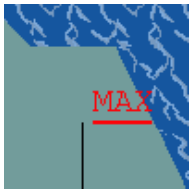
X9
Космонавт уперся в стенку слева.



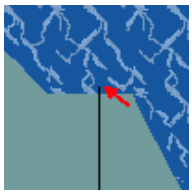
X10
Космонавт потерял опору под ногами.



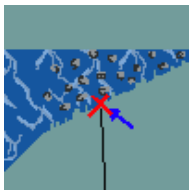
X11
Космонавт уперся в стенку справа.



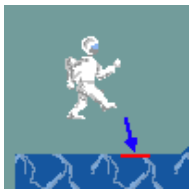
X12
Выстрелянная веревка длиннее максимальной дальности выстрела.



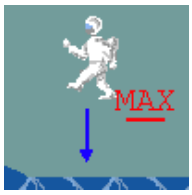
X13
Гарпун смог воткнуться в скалу.



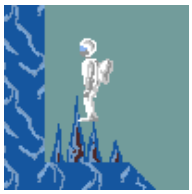
X14
Гарпун ударился о «плохую» скалу.



X15
Космонавт приземлился после падения.



X16
Текущая высота падения космонавта смертельна.



X17
Космонавт закончил мигать.

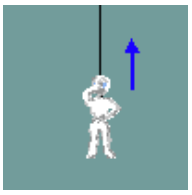
4.4. Нумерация и перечень выходных воздействий

Воздействие не описать картинкой.

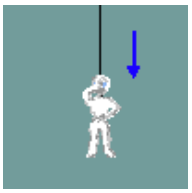
Z0
Расчет некоторых параметров раскачки при качании с нажатой кнопкой "влево".

Воздействие не описать картинкой.

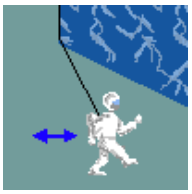
Z1
Расчет некоторых параметров раскачки при качании с нажатой кнопкой "вправо".



Z2
Космонавт подтягивается на веревке. Длина веревки уменьшается.



Z3
Космонавт отпускает веревку. Длина веревки увеличивается.



Z4
Расчет координат, скорости, ускорения космонавта при раскачке.

Воздействие не описать картинкой.

Z5
Некоторые вычисления для перехода в состояние полета из состояния качания.

Воздействие не описать картинкой.

Z6
Инициализация состояния «Смерть».

Z13
Выставляет значение направления при движении налево в зависимости от поверхности.

Z14
Определение позы космонавта при остановке в состоянии ходьбы.

Воздействие не описать картинкой.

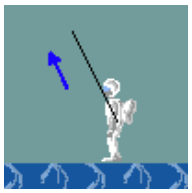
Z15
Инициализация состояния «Прицеливание».

Z16
Поворот прицела по часовой стрелки.

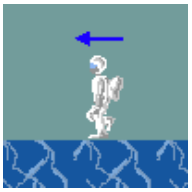
Z17
Поворот прицела против часовой стрелки

Воздействие не описать картинкой.

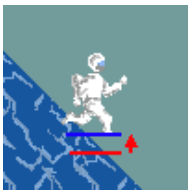
Z18
Инициализация состояния «Выстрел».



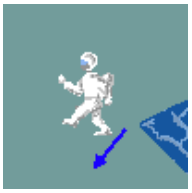
Z19
Удлиняет выстрелянную веревку и обновляет координаты конца гарпуна.



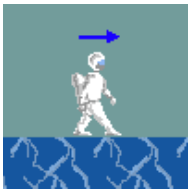
Z7
Сдвиг координат космонавта при ходьбе влево.



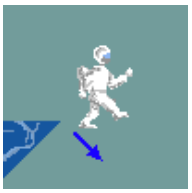
Z8
Двигает человека так, чтобы его ноги оказались на поверхности земли.



Z9
Инициализация полета после падения влево из состояния ходьбы.



Z10
Сдвиг координат космонавта при ходьбе вправо.

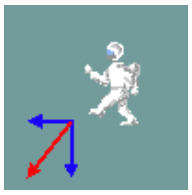


Z11
Инициализация полета после падения вправо из состояния ходьбы.

Z12
Выставляет значение направления при движении направо в зависимости от поверхности.

Воздействие не описать картинкой.

Z20
Инициализация состояния «Качание».



Z21
Физический просчет полета.

Z22
Увеличение счетчика мигания космонавта при гибели.

Воздействие не описать картинкой.

Z23
Инициализация текущего уровня игры.

Воздействие не описать картинкой.

Z24
Инициализация следующего уровня игры.

Воздействие не описать картинкой.

Z25
Рассчитывает параметры при свободном качании (без нажатых кнопок).

4.5. Схема связей

Схема связей автомата с его окружением приведена на рис. 9.

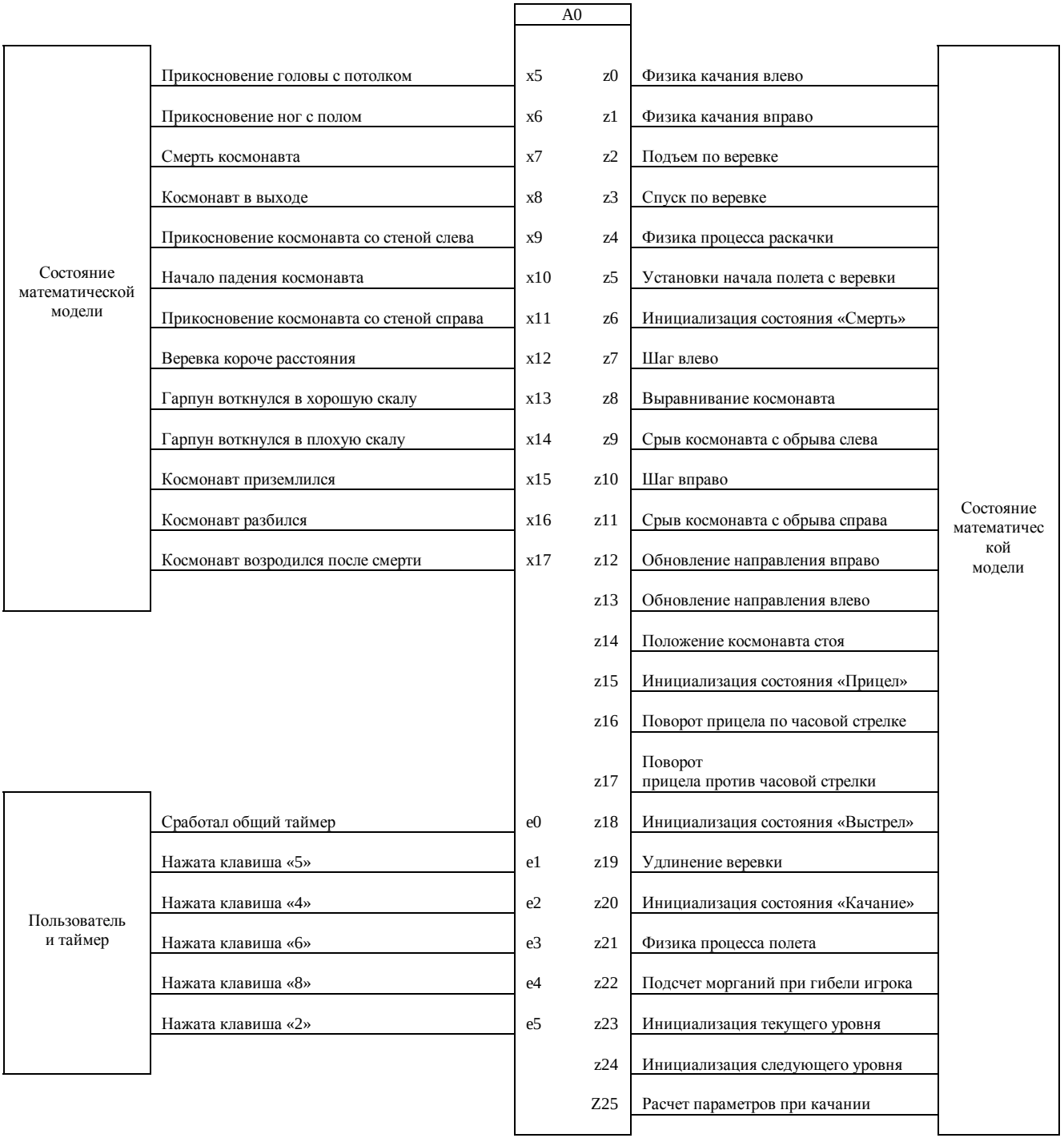


Рис. 9. Схема связей автомата

4.6. Граф переходов

На рис. 10 приведен граф переходов автомата, реализующего логику игры в централизованной форме.

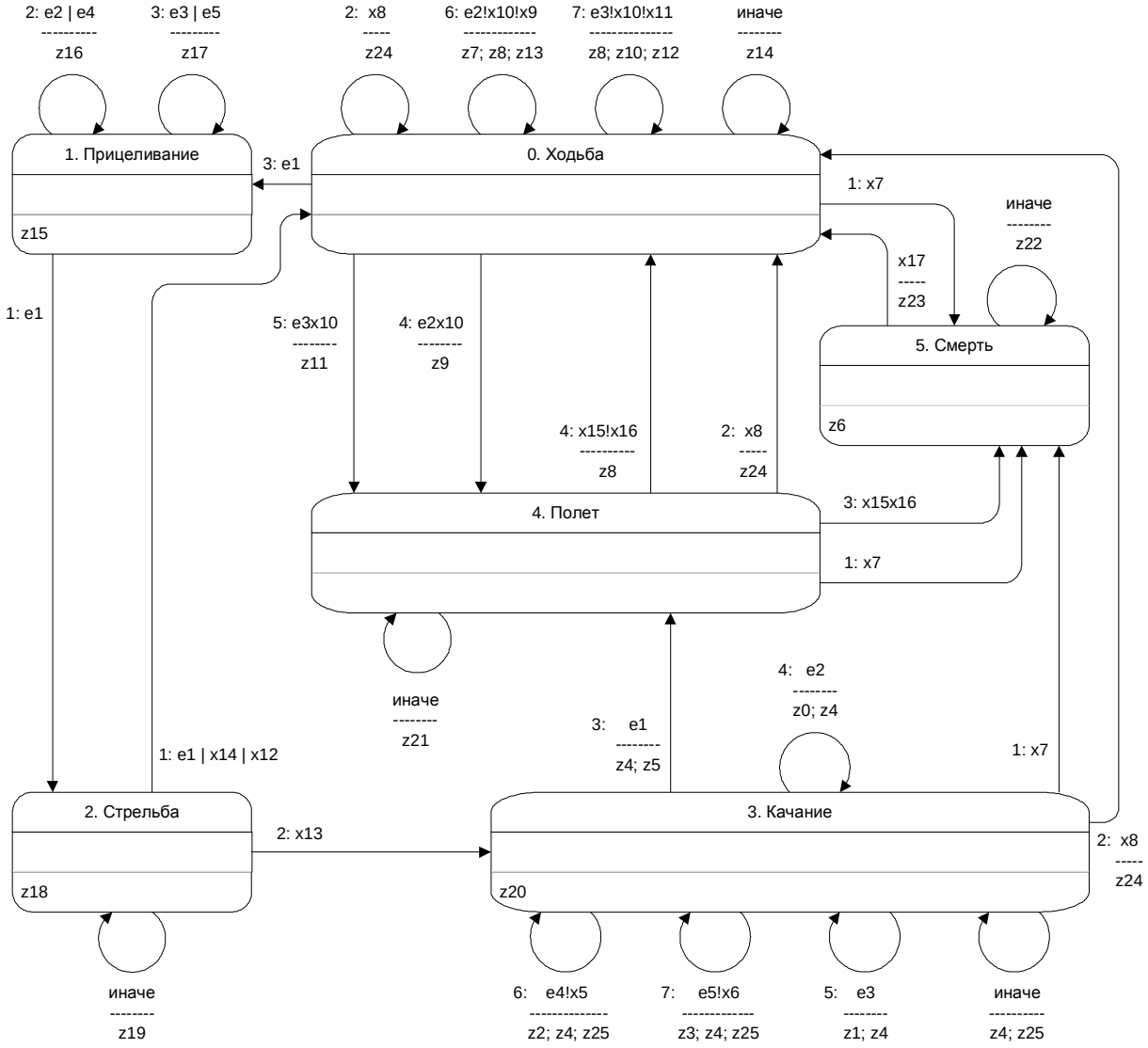


Рис. 10. Граф переходов автомата

5. Особенности вычислительной части алгоритма – физика игры

В этом разделе выполняется обоснование выбора некоторых функций входых переменных и выходных воздействий. При этом в качестве примера выбраны несколько наиболее интересных функций, таких как функции z4 (физика процесса качания), z8 («выравнивание» космонавта) и z21 (физика процесса полета).

5.1. Физика процесса полета

Рассчет параметров свободного полета в состоянии «Полет» реализован в выходном воздействии z21. Этому выходному воздействию в программе соответствует метод *z21_flightPhysics()* класса *RockmanGame*. Параметрами полета являются координаты космонавта (*fManX, fManY*) и его скорость (*fManVX, fManVY*).

При вычислении параметров полета для следующего момента времени к вертикальной скорости прибавляется приращение, обусловленное ускорением свободного падения. К координатам прибавляются приращения, зависящие от скоростей.

Важным методом класса *RockmanGame* является метод *hit(int fX, int fY)*. Для заданной точки он возвращает значение *true* при попадании точки (*fX, fY*) в скалу, и значение *false* - в противном случае.

Метод *z21_flightPhysics()* отвечает также за обработку ударов о стены пещеры при полете. С помощью метода *hit(int fX, int fY)* определяется факт удара космонавта о стену. При ударе горизонтальная скорость *fManVX* меняет знак.

5.2. «Выравнивание» космонавта

В состоянии «Ходьба» с помощью выходных воздействий z7 и z10 координатам космонавта дается горизонтальное приращение. Однако, при ходьбе, например, по наклонной поверхности, необходимо координатам давать еще и вертикальное приращение, для того чтобы космонавт не отрывался от поверхности. Для этого после выходных воздействий z7 и z10 вызывается выходное воздействие z8.

Выходному воздействию z8 в исходном коде соответствует метод *z8_stableWalk()*.
При горизонтальном смещении ноги космонавта могут оказаться либо в воздухе над поверхностью, либо под поверхностью в скале. После этого космонавта необходимо сдвинуть так, чтобы его ноги оказались на поверхности.

С заданным вертикальным шагом, выполняемым попеременно вверх и вниз, начиная от ступней космонавта, ищем поверхность. Поверхностью считаем ближайшую к ступням точку, значение функции *hit(int fX, int fY)* от которой не совпадает со значением той же функции от координат ступней космонавта. Когда точка поверхности найдена, изменяем соответствующим образом координаты космонавта.

5.3. Физика процесса качания

Рассчет координат, скорости, углового ускорения космонавта в состоянии «Качание» реализован в выходном воздействии z4. Этому выходному воздействию в исходном коде соответствует метод *z4_swingPhysics()* класса *RockmanGame*.

На рисунке показана ситуация, когда гарпун воткнулся в точку 1 (рис. 11). Затем, в результате перемещения и раскачивания космонавта, веревка перегнулась в точках 2 и 3.

Активная часть веревки – это ее часть от космонавта до точки 3 - перегиба веревки. Пока веревка «перегнута» в точке 3 при рассчете параметров качания учитывается только активная часть веревки. Участки веревки от точки 2 до точки 3 и от точки 2 до точки 1 в это время неподвижны.

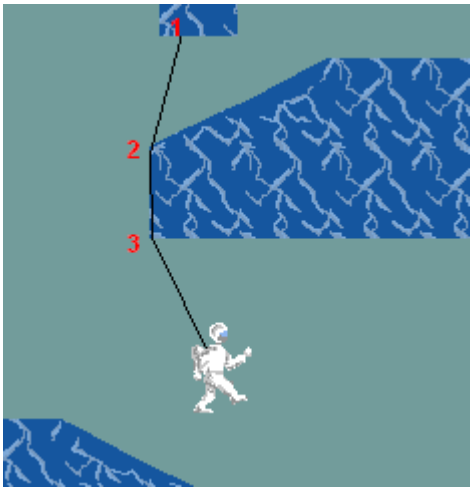


Рис. 11. Качание

Для расчета движения космонавта и активной части веревки используются следующие переменные: длина активной части веревки (*fRopeLength*), угол наклона веревки (*fAngle*), угловая скорость (*ffAngleSpeed*). Зная эти параметры и законы физики вращения, вычисляем угловое ускорение и, далее, значения этих параметров для следующего момента времени.

Отметим, что изменение этих параметров происходит не только в выходном воздействии z4. Например, изменение длины активной части веревки может быть вызвано выходным воздействием z19 – удлинение веревки.

При очередной итерации расчета параметров качания для активной части веревки проверяется факт ее соприкосновения со скалой. Для этого с заданным шагом по всей длине активной части веревки, с помощью уже рассмотренного метода *hit(int fX, int fY)*, выявляем точку ее соприкосновения со скалой. Если такая точка существует, то она будет точкой перегиба веревки. Параметры этой точки представлены классом *HardPoint*. Параметрами являются координаты точки, текущая длина активной части веревки, угловая скорость и угол наклона веревки в момент соприкосновения. Для новой точки перегиба создается объект данного класса, который заносится в стек точек перегиба. При создании новой точки перегиба очевидным образом изменяются такие параметры, как длина активной части веревки и т.д.

При каждой итерации параметры последней точки перегиба из стека сравниваются с текущими параметрами качания. При этом, если угол наклона, сохраненный в точке перегиба, совпадает с текущим углом наклона (с некоторой точностью), и текущая угловая скорость имеет разный знак с угловой скоростью, сохраненной в точке перегиба, то точка перегиба удаляется с вершины стека. При этом текущие параметры качания изменяются соответствующим образом.

Заключение

Первые версии игры были реализованы без использования автоматного программирования. На более поздних этапах был произведен рефакторинг – изменение структуры программы без изменения ее функциональности. Это перепроектирование подразумевало внедрение автоматного подхода для реализации логики игры. Рефакторинг занял значительное время, однако авторы считают, что результат стоил затраченных усилий. Простота и наглядность автоматного подхода не только упростили добавление новых возможностей в игру, но и позволили найти и исправить некоторые ошибки, скрытые в сложном коде первых версий.

Программа иллюстрирует целесообразность разделения состояний в программе на два типа: управляющие и вычислительные. При этом, находясь в одном из управляющих состояний, система может проходить большое число вычислительных состояний. Например, находясь в состоянии «Полет», вычисляется функция, описывающая физику полета, которая в ходе вычисления принимает большое число состояний.

Предложенный подход делает понятие «состояние» конструктивным, так как обычно состояния на указанные типы не разделяются, а под состояниями системы понимаются значения ячеек памяти, которых огромное количество.

Обоснован выбор основных вычислительных алгоритмов, использованных в игре.

Подробная документация по каждому классу программы представлена в формате *HTML*. Она сгенерирована средствами *JavaDoc* из исходных файлов. Отметим, что эта документация не заменяет данный документ, а лишь дополняет его, что не является общепринятым. Фрагмент документации *JavaDoc* приведен в приложении 3.

Таким образом, обоснована целесообразность использования автоматного подхода по сравнению с традиционным подходом при проектировании этой игры.

Литература

1. *Шалыто А.А., Туккель Н.И.* От тьюрингова программирования к автоматному // Мир ПК. 2002. № 2.
2. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++. М.: Бином; СПб.: Невский диалект, 1998.
3. *Лавров С.* Программирование. Математические основы, средства, теория. СПб.: БХВ, 2002.
4. *Benveniste A., Le Guernic P.* Hybrid Dinamical System Theory and the Singal Language //IEEE Trans. on Automatic Control. 1990, vol. 35, № 5.
5. *Stelting S., Maassen O.* Applied Java Patterns. NJ: Prentice Hall. 2001.
6. *Cooper J.* The Design Patterns Java Companion. NY: Addison-Wesley. 1998.
7. *Туккель Н.И., Шалыто А.А.* Система управления дизель-генератором. Программирование с явным выделением состояний. (http://is.ifmo.ru, раздел «Проекты»).

Приложение 1. Пример протокола

```
! Начало работы программы.
{ A0 начал работу в состоянии 0
% e1 - Пришло событие от таймера.
+ x7 - Космонавт находится на смертельном участке карты. Значение - ложь
+ x8 - Космонавт находится на выходе. Значение - ложь
* z14 - Определение позы космонавта при остановке в состоянии ходьбы
} A0 закончил работу в состоянии 0
{ A0 начал работу в состоянии 0
% e1 - Пришло событие от таймера.
+ x7 - Космонавт находится на смертельном участке карты. Значение - ложь
+ x8 - Космонавт находится на выходе. Значение - ложь
* z14 - Определение позы космонавта при остановке в состоянии ходьбы
} A0 закончил работу в состоянии 0
{ A0 начал работу в состоянии 0
% e1 - Нажата клавиша «5»
+ x7 - Космонавт находится на смертельном участке карты. Значение - ложь
+ x8 - Космонавт находится на выходе. Значение - ложь
* z15 - Инициализация состояния «Прицеливание»
# A0 перешел из состояния 0 в состояние 1 : Прицеливание
} A0 закончил работу в состоянии 1
{ A0 начал работу в состоянии 1
% e1 - Пришло событие от таймера.
} A0 закончил работу в состоянии 1
{ A0 начал работу в состоянии 1
% e1 - Пришло событие от таймера.
} A0 закончил работу в состоянии 1
{ A0 начал работу в состоянии 1
% e3 - Нажата клавиша «6» (вправо)
* z17 - Поворот прицела против часовой стрелки
} A0 закончил работу в состоянии 1
{ A0 начал работу в состоянии 1
% e3 - Нажата клавиша «6» (вправо)
* z17 - Поворот прицела против часовой стрелки
} A0 закончил работу в состоянии 1
{ A0 начал работу в состоянии 1
% e3 - Нажата клавиша «6» (вправо)
* z17 - Поворот прицела против часовой стрелки
} A0 закончил работу в состоянии 1
{ A0 начал работу в состоянии 1
% e1 - Пришло событие от таймера.
} A0 закончил работу в состоянии 1
{ A0 начал работу в состоянии 1
% e1 - Пришло событие от таймера.
} A0 закончил работу в состоянии 1
{ A0 начал работу в состоянии 1
% e1 - Нажата клавиша «5»
* z18 - Инициализация состояния «Стрельба»
# A0 перешел из состояния 1 в состояние 2 : Стрельба
} A0 закончил работу в состоянии 2
{ A0 начал работу в состоянии 2
% e1 - Пришло событие от таймера.
+ x14 - Гарпун ударился о «плохую» скалу. Значение - ложь
+ x12 - Выстрелянная веревка длиннее максимальной дальности выстрела. Значение -
ложь
```

```
+ x13 - Гарпун смог воткнуться в скалу. Значение - ложь
* z19 - Удлинение выстрелянной веревки и обновление координат конца гарпуна
} A0 закончил работу в состоянии 2
{ A0 начал работу в состоянии 2
% e1 - Пришло событие от таймера.
+ x14 - Гарпун ударился о «плохую» скалу. Значение - ложь
+ x12 - Выстрелянная веревка длиннее максимальной дальности выстрела. Значение -
ложь
+ x13 - Гарпун смог воткнуться в скалу. Значение - истина
* z20 - Инициализация состояния «Качание»
# A0 перешел из состояния 2 в состояние 3 : Качание
} A0 закончил работу в состоянии 3
{ A0 начал работу в состоянии 3
% e4 - Нажата клавиша «8» (вверх)
+ x7 - Космонавт находится на смертельном участке карты. Значение - ложь
+ x8 - Космонавт находится на выходе. Значение - ложь
+ x5 - Космонавт соприкасается головой со стеной при укорачивании веревки.
Значение - ложь
* z25 - Расчет параметров при свободном качании (без нажатых кнопок)
* z4 - Расчет координат, скорости, ускорения космонавта при раскачке
* z2 - Скалолаз подтягивается на веревку. Длина веревки уменьшается.
} A0 закончил работу в состоянии 3
{ A0 начал работу в состоянии 3
% e4 - Нажата клавиша «8» (вверх)
+ x7 - Космонавт находится на смертельном участке карты. Значение - ложь
+ x8 - Космонавт находится на выходе. Значение - ложь
+ x5 - Космонавт соприкасается головой со стеной при укорачивании веревки.
Значение - ложь
* z25 - Расчет параметров при свободном качании (без нажатых кнопок)
* z4 - Расчет координат, скорости, ускорения космонавта при раскачке
* z2 - Скалолаз подтягивается на веревку. Длина веревки уменьшается.
} A0 закончил работу в состоянии 3
{ A0 начал работу в состоянии 3
% e1 - Пришло событие от таймера.
+ x7 - Космонавт находится на смертельном участке карты. Значение - ложь
+ x8 - Космонавт находится на выходе. Значение - ложь
* z25 - Расчет параметров при свободном качании (без нажатых кнопок)
* z4 - Расчет координат, скорости, ускорения космонавта при раскачке
} A0 закончил работу в состоянии 3
{ A0 начал работу в состоянии 3
% e1 - Пришло событие от таймера.
+ x7 - Космонавт находится на смертельном участке карты. Значение - ложь
+ x8 - Космонавт находится на выходе. Значение - ложь
* z25 - Расчет параметров при свободном качании (без нажатых кнопок)
* z4 - Расчет координат, скорости, ускорения космонавта при раскачке
} A0 закончил работу в состоянии 3
{ A0 начал работу в состоянии 3
% e1 - Нажата клавиша «5»
+ x7 - Космонавт находится на смертельном участке карты. Значение - ложь
+ x8 - Космонавт находится на выходе. Значение - ложь
* z4 - Расчет координат, скорости, ускорения космонавта при раскачке
* z5 - Некоторые вычисления для перехода в состояние полета из состояния качания
# A0 перешел из состояния 3 в состояние 4 : Полет
} A0 закончил работу в состоянии 4
{ A0 начал работу в состоянии 4
% e1 - Пришло событие от таймера.
+ x7 - Космонавт находится на смертельном участке карты. Значение - истина
* z6 - Инициализация состояния «Смерть»
# A0 перешел из состояния 4 в состояние 5 : Смерть
```

```
} A0 закончил работу в состоянии 5
{ A0 начал работу в состоянии 5
% e1 - Пришло событие от таймера.
+ x17 - Космонавт закончил мигать. Значение - ложь
* z22 - Увеличение счетчика мигания космонавта при гибели
} A0 закончил работу в состоянии 5
{ A0 начал работу в состоянии 5
% e1 - Пришло событие от таймера.
+ x17 - Космонавт закончил мигать. Значение - истина
* z23 - Инициализация текущего уровня игры
# A0 перешел из состояния 5 в состояние 0 : Ходьба
} A0 закончил работу в состоянии 0
{ A0 начал работу в состоянии 0
% e1 - Пришло событие от таймера.
+ x7 - Космонавт находится на смертельном участке карты. Значение - ложь
+ x8 - Космонавт находится на выходе. Значение - ложь
* z14 - Определение позы космонавта при остановке в состоянии ходьбы
} A0 закончил работу в состоянии 0
{ A0 начал работу в состоянии 0
% e1 - Пришло событие от таймера.
+ x7 - Космонавт находится на смертельном участке карты. Значение - ложь
+ x8 - Космонавт находится на выходе. Значение - ложь
* z14 - Определение позы космонавта при остановке в состоянии ходьбы
} A0 закончил работу в состоянии 0
! Конец работы программы.
```

Приложение 2. Исходный код

Класс Main

```
package rockman;

import rockman.RockmanGame;
import rockman.RockmanGameView;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.io.*;

/**
 * Главный класс игры.
 */
public class Main extends JFrame implements Runnable, ActionListener {
    /**
     * Отрисовщик.
     * @clientCardinality 1
     * @directed
     * @supplierCardinality 1
     * @link aggregation
     * @clientRole controller
     * @supplierRole view*/
    private RockmanGameView painter;
```

```
    /**
     * Математическая модель.
     * @link aggregation
     * @clientRole controller
     * @supplierRole model
     * @supplierCardinality 1
     * @clientCardinality 1
     */
    private RockmanGame game;

    /**
     * Текущая нажатая кнопка.
     */
    private int curKey;

    /**
     * Кнопка сохранения игры.
     */
    private JButton save;

    /**
     * Кнопка загрузки.
     */
    private JButton load;

    /**
     * Текстовое поле для отображения текущего состояния.
     */
    private JTextField field;

    /**
     * Загружает спрайты.
     * @param dir Файл с индексами спрайтов.
     * @param file Директория с картинками.
     * @return Массив спрайтов.
     */
    public static Image[] extract(String dir, String file) {
        ArrayList al = new ArrayList(100);
        try {
            LineNumberReader lnr = new LineNumberReader(new FileReader(dir + "/" +
file));
            String s;
            while (lnr.ready()) {
                s = lnr.readLine();
                if (s != null) {
                    Image ai = (new javax.swing.ImageIcon((dir + "/" + s +
".gif").toLowerCase())).getImage();
                    al.add(ai);
                }
            }
            lnr.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        Object[] obj = al.toArray();
        Image[] ime = new Image[obj.length];
        for (int i = 0; i < ime.length; i++) {
            ime[i] = (Image) obj[i];
        }
        return ime;
    }

    /**
     * Массив с расшифровкой номеров состояний.
     */
    public static final String[] stateName = {
```

```
    "Ходьба", "Прицеливание", "Выстрел", "Качание", "Полет", "Смерть"
};

/**
 * Конструктор.
 */
public Main() {
    super("Космонавт");
    game = new RockmanGame();
    painter = new RockmanGameView(game, extract("pic/7210", "index.txt"));
    curKey = RockmanGame.EVENT_0_KEY_NONE_TIMER;
    setVisible(true);
    setSize(450, 450);
    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(painter, BorderLayout.CENTER);
    JPanel controls = new JPanel(new FlowLayout());
    getContentPane().add(controls, BorderLayout.SOUTH);
    save = new JButton("Сохранить");
    save.addActionListener(this);
    load = new JButton("Загрузить");
    load.addActionListener(this);
    field = new JTextField("", 12);
    field.setEditable(false);
    controls.add(save);
    controls.add(field);
    controls.add(load);
    File slot = new File("SLOT00");
    load.setEnabled(slot.exists());
    validate();

    KeyListener keySpy = new KeyAdapter() {
        public void keyPressed(KeyEvent e) {
            switch (e.getKeyChar()) {
                case '8':
                    curKey = RockmanGame.EVENT_4_KEY_UP;
                    break;
                case '2':
                    curKey = RockmanGame.EVENT_5_KEY_DOWN;
                    break;
                case '4':
                    curKey = RockmanGame.EVENT_2_KEY_LEFT;
                    break;
                case '6':
                    curKey = RockmanGame.EVENT_3_KEY_RIGHT;
                    break;
                case '5':
                    curKey = RockmanGame.EVENT_1_KEY_CENTER;
                    break;
                default:
                    //curKey = RockmanGame.KEY_NONE;
                    break;
            }
        }

        public void keyReleased(KeyEvent e) {
            if (curKey != RockmanGame.EVENT_1_KEY_CENTER) {
                curKey = RockmanGame.EVENT_0_KEY_NONE_TIMER;
            }
        }
    };
    painter.addKeyListener(keySpy);
    addKeyListener(keySpy);
    load.addKeyListener(keySpy);
    save.addKeyListener(keySpy);
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
```

```
                game.log.programMes("Конец работы программы.");
                game.log.close();
                System.exit(0);
            }
        });

        new Thread(this).start();
    }

    /**
     * Статический метод входа в программу.
     * @param args Входные параметры не используются.
     */
    public static void main(String[] args) {
        new Main();
    }

    /**
     * Метод работы главного потока программы.
     */
    public void run() {
        for (; ;) {
            try {
                game.A0_nextGameStep(curKey);
                if (curKey == RockmanGame.EVENT_1_KEY_CENTER) {
                    curKey = RockmanGame.EVENT_0_KEY_NONE_TIMER;
                }
                if (field.getText() != stateName[game.y0_state]) {
                    field.setText(stateName[game.y0_state]);
                }
                painter.repaint();
                Thread.sleep(50);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * Реализация интерфейса слушателя событий.
     * @param e Событие.
     */
    public void actionPerformed(ActionEvent e) {
        try {
            if (e.getSource() == load) {
                FileInputStream fis = new FileInputStream("SLOT00");
                game.loadGameState(new DataInputStream(fis));
                fis.close();
            } else if (e.getSource() == save) {
                FileOutputStream fos = new FileOutputStream("SLOT00");
                game.saveGameState(new DataOutputStream(fos));
                fos.flush();
                fos.close();
                load.setEnabled(true);
            }
        } catch (Exception ex) {
            File slot = new File("SLOT00");
            load.setEnabled(slot.exists());
            System.out.println(ex);
        }
    }
}
```

Класс Hardpoint

```
package rockman;

/**
 * Класс, представляющий точку перегиба веревки.
 */
public class Hardpoint {
    /**
     * Абсцисса точки перегиба веревки.
     * Значение с фиксированной точкой.
     */
    public int fX;

    /**
     * Ордината точки перегиба веревки.
     * Значение с фиксированной точкой.
     */
    public int fY;

    /**
     * Длина веревки до предыдущей точки перегиба.
     * Значение с фиксированной точкой.
     */
    public int fLength;

    /**
     * Угол, при котором произошел перегиб.
     * Значение с фиксированной точкой двойной точности.
     */
    public int ffAngle;

    /**
     * Положительность угловой скорости качания при установке данной точки перегиба.
     */
    public boolean angleSpeedP;

    /**
     * Конструктор.
     * @param fX Абсцисса.
     * @param fY Ордината.
     * @param fLength Длина веревки.
     * @param ffAngle Угол.
     * @param angleSpeedP Положительность угловой скорости.
     */
    public Hardpoint(int fX, int fY, int fLength, int ffAngle, boolean angleSpeedP) {
        this.fX = fX;
        this.fY = fY;
        this.fLength = fLength;
        this.ffAngle = ffAngle;
        this.angleSpeedP = angleSpeedP;
    }
}
```

Класс GraphicsME

```
package rockman;

import java.security.InvalidParameterException;
import java.awt.*;

/**
 * Модифицированый класс для графического контекста.
 */
public class GraphicsME {

    /**
     * Выравнивание картинки: центрирование по горизонтали.
     */
    public static final int HCENTER = 1;

    /**
     * Выравнивание картинки: центрирование по вертикали.
     */
    public static final int VCENTER = 2;

    /**
     * Выравнивание картинки: по левой точке.
     */
    public static final int LEFT = 4;

    /**
     * Выравнивание картинки: по правой точке.
     */
    public static final int RIGHT = 8;

    /**
     * Выравнивание картинки: по верхней точке.
     */
    public static final int TOP = 16;

    /**
     * Выравнивание картинки: по нижней точке.
     */
    public static final int BOTTOM = 32;

    /**
     * Освобождение графического контекста.
     */
    public void dispose() {
        ag.dispose();
    }

    /**
     * Стандартный графический контекст.
     */
    private java.awt.Graphics ag;

    /**
     * Конструктор.
     * @param awtGraphics Стандартный графический контекст.
     */
    public GraphicsME(java.awt.Graphics awtGraphics) {
        super();
        this.ag = awtGraphics;
    }

    /**
```

```
 * Смещение центра координат.
 * @param x Смещение по X.
 * @param y Смещение по Y.
 */
public void translate(int x, int y) {
    ag.translate(x, y);
}

/**
 * Установить цвет.
 * @param RGB Цвет в формате RGB.
 */
public void setColor(int RGB) {
    ag.setColor(new java.awt.Color(RGB));
}

/**
 * Получить абсциссу левой верхней точки активного прямоугольника.
 * @return Абсцисса левой верхней точки активного прямоугольника.
 */
public int getClipX() {
    return ag.getClipBounds().x;
}

/**
 * Получить ординату левой верхней точки активного прямоугольника.
 * @return Ордината левой верхней точки активного прямоугольника.
 */
public int getClipY() {
    return ag.getClipBounds().y;
}

/**
 * Получить ширину активного прямоугольника.
 * @return Ширина активного прямоугольника.
 */
public int getClipWidth() {
    return ag.getClipBounds().width;
}

/**
 * Получить высоту активного прямоугольника.
 * @return Высота активного прямоугольника.
 */
public int getClipHeight() {
    return ag.getClipBounds().width;
}

/**
 * Пересечь активный прямоугольник с данным.
 * @param x Абсцисса левой верхней точки.
 * @param y Ордината левой верхней точки.
 * @param width Ширина.
 * @param height Высота.
 */
public void clipRect(int x, int y, int width, int height) {
    ag.clipRect(x, y, width, height);
}

/**
 * Установить активный прямоугольник.
 * @param x Абсцисса левой верхней точки.
 * @param y Ордината левой верхней точки.
 * @param width Ширина.
 * @param height Высота.
 */
```

```
    public void setClip(int x, int y, int width, int height) {
        ag.setClip(x, y, width, height);
    }

    /**
     * Нарисовать линию.
     * @param x1 Абсцисса точки 1.
     * @param y1 Ордината точки 1.
     * @param x2 Абсцисса точки 2.
     * @param y2 Ордината точки 2.
     */
    public void drawLine(int x1, int y1, int x2, int y2) {
        ag.drawLine(x1, y1, x2, y2);
    }

    /**
     * Заполнить прямоугольник текущим цветом.
     * @param x Абсцисса левой верхней точки.
     * @param y Ординита левой верхней точки.
     * @param width Ширина.
     * @param height Высота.
     */
    public void fillRect(int x, int y, int width, int height) {
        ag.fillRect(x, y, width, height);
    }

    /**
     * Нарисовать прямоугольник текущим цветом.
     * @param x Абсцисса левой верхней точки.
     * @param y Ординита левой верхней точки.
     * @param width Ширина.
     * @param height Высота.
     */
    public void drawRect(int x, int y, int width, int height) {
        ag.drawRect(x, y, width, height);
    }

    /**
     * Нарисовать картинку.
     * @param ai Объект картинки.
     * @param x Абсцисса точки, относительно которой будет происходить выравнивание.
     * @param y Ордината точки, относительно которой будет происходить выравнивание.
     * @param anchor Выравнивание. Задается константами выравнивания.
     */
    public void drawImage(Image ai, int x, int y, int anchor) {
        int w = ai.getWidth(null);
        int h = ai.getHeight(null);
        int dx, dy;
        if (anchor == 0) {
            ag.drawImage(ai, x, y, null);
        } else {
            switch (anchor & (LEFT | RIGHT | HCENTER)) {
                case LEFT:    dx = 0;        break;
                case HCENTER: dx = -w >> 1;  break;
                case RIGHT:   dx = -w;       break;
                default: throw new InvalidParameterException("Invalid anchor parameter: "
+ anchor);
            }
            switch (anchor & (TOP | BOTTOM | VCENTER)) {
                case TOP:     dy = 0;        break;
                case VCENTER: dy = -h >> 1;  break;
                case BOTTOM:  dy = -h;       break;
                default: throw new InvalidParameterException("Invalid anchor parameter: "
+ anchor);
            }
            ag.drawImage(ai, x + dx, y + dy, null);
```

```
        }
    }
}
```

Класс RockmanGame

```
package rockman;

import java.io.*;
import java.util.*;

/**
 * Класс математической модели игры "Космонавт".
 */
public class RockmanGame {
    /**
     * Распаковывает массив байтов, используя алгоритм RLE.
     * @param inarray Запакованый входной массив.
     * @param dstlen Размер распакованного массива.
     * @return Распакованный массив.
     */
    public static final byte[] RLEdecompress(byte[] inarray, int dstlen) {
        System.gc();
        byte[] out = new byte[dstlen];
        int indx = 0;
        int outindx = 0;
        while (indx < inarray.length) {
            int val = inarray[indx++] & 0xFF;
            int counter = 1;
            int value = 0;
            if ((val & 0xC0) == 0xC0) {
                counter = val & 0x3F;
                value = inarray[indx++] & 0xFF;
            } else {
                value = val;
            }
            while (counter != 0) {
                out[outindx++] = (byte) value;
                counter--;
            }
        }
        System.gc();
        return out;
    }

    /**
     * Пакует массив байтов алгоритмом RLE.
     * @param _inarray Входной массив.
     * @return Запакованый массив.
     */
    public static final byte[] RLEcompress(byte[] _inarray) {
        System.gc();
        ByteArrayOutputStream baos = new ByteArrayOutputStream(_inarray.length);
        int inindx = 0;
        while (inindx < _inarray.length) {
            int value = _inarray[inindx++] & 0xFF;
            int count = 1;
            while (inindx < _inarray.length) {
                if ((_inarray[inindx] & 0xFF) == value) {
                    count++;
                    inindx++;
                    if (count == 63)
                        break;
                } else
                    break;
```

```
            }
            if (count > 1) {
                baos.write(count | 0xC0);
                baos.write(value);
            } else {
                if (value > 63) {
                    baos.write(0xC1);
                    baos.write(value);
                } else {
                    baos.write(value);
                }
            }
        }
        byte[] outarray = baos.toByteArray();
        baos = null;
        System.gc();
        return outarray;
    }

    /**
     * Таблица констант для вычисления синуса.
     */
    public final static int[] sineTable = {0, 25, 50, 74, 98, 121, 142, 162, 181, 198,
213, 226, 237, 245, 251, 255, 256, 255, 251, 245, 237, 226, 213, 198, 181, 162, 142, 121,
98, 74, 50, 25, 0, -25, -50, -74, -98, -121, -142, -162, -181, -198, -213, -226, -237, -
245, -251, -255, -256, -255, -251, -245, -237, -226, -213, -198, -181, -162, -142, -121,
-98, -74, -50, -25, 0};

    /**
     * Преобразует угол в эквивалентный в диапазоне от 0 до 63.
     * Например
     * вход           ыход
     *    5             5
     *   -5            59
     *   64             0
     *  129             1
     * -129            63
     * @param a Угол.
     * @return 64 нормированный угол.
     */
    public static int angle64(int a) {
        int b = Math.abs(a);
        b &= 63;  // %=64 - for positive integres these operations are equval
        return a >= 0 ? b : 64 - b;
    }

    /**
     * То же самое, что и angle64, но для 64 << 8 нормированных углов. (Фиксированная
точка).
     * @param a Входной угол.
     * @return 64 << 8 нормированный угол.
     */
    public static int angle64float(int a) {
        int b = Math.abs(a);
        b &= (64 << 8) - 1;
        return a >= 0 ? b : (64 << 8) - b;
    }

    /**
     * Подсчет синуса с умножением.
     * @param x Множитель.
     * @param index Угол.
     * @return Результат.
     */
    public static int xSine(int x, int index) {
        return (x * sineTable[index]) >> 8;
```

```
}

/**
 * Подсчет синуса.
 * @param angle Угол в формате с фиксированной точкой.
 * @return Результат в формате с фиксированной точкой.
 */
public static int sineFloatPreciese(int angle) {
    int a1 = angle >> 8;
    int a2 = a1 + 1;
    int s1 = sineTable[a1];
    int s2 = sineTable[a2];
    return s1 + ((s2 - s1) * (angle - (a1 << 8)) >> 8);
}

/**
 * Подсчет косинуса с умножением.
 * @param x Множитель.
 * @param index Угол.
 * @return Результат.
 */
public static int xCosine(int x, int index) {
    index += 16;
    return (x * sineTable[index & 63]) >> 8;
}

/**
 * Подсчет синуса.
 * @param angle Угол в формате с фиксированной точкой.
 * @return Результат в формате с фиксированной точкой.
 */
public static int cosineFloatPreciese(int angle) {
    int a1 = angle >> 8;
    int a2 = a1 + 1;
    int c1 = sineTable[(a1 + 16) & 63];
    int c2 = sineTable[(a2 + 16) & 63];
    return c1 + ((c2 - c1) * (angle - (a1 << 8)) >> 8);
}

// figures
public static final byte C = 0;
public static final byte UL = 1;
public static final byte UR = 2;
public static final byte DL = 3;
public static final byte DR = 4;
public static final byte DR1V = 5;
public static final byte DR2V = 6;
public static final byte DL1V = 7;
public static final byte DL2V = 8;
public static final byte UR1V = 9;
public static final byte UR2V = 10;
public static final byte UL1V = 11;
public static final byte UL2V = 12;
public static final byte DR1H = 13;
public static final byte DR2H = 14;
public static final byte DL1H = 15;
public static final byte DL2H = 16;
public static final byte UR1H = 17;
public static final byte UR2H = 18;
public static final byte UL1H = 19;
public static final byte UL2H = 20;
public static final byte NOP = 31;
// anim objects
public static final byte EXIT = 0;
public static final byte SPIKE = 1;
```

```
    // can stick        figure | 0
    // cannot stick     figure | 32
    // with water       figure | 64
    // animation object object | 96

    /**
     * Загружает карту.
     * @param stage Номер карты.
     */
    private void loadStage(int stage) {
        try {
            System.gc();
            DataInputStream ds = new DataInputStream(new
Object().getClass().getResourceAsStream("/res/map" + stage + ".bin"));
            int dimX = ds.readUnsignedByte();
            int dimY = ds.readUnsignedByte();
            fManX = fEyeX = fStirvingX = (ds.readUnsignedByte() << 8) + (1 << 7);
            fManY = fEyeY = fStirvingY = (ds.readUnsignedByte() << 8) + (1 << 7);
            int packedDataSize = ds.readUnsignedByte() * 256 + ds.readUnsignedByte();
            byte[] u = new byte[packedDataSize];
            ds.read(u);
            ds.close();
            u = RLEdecompress(u, dimX * dimY);
            field = null;
            field = new byte[dimY][dimX];
            int k = 0;
            for (int i = 0; i < dimY; i++) {
                for (int j = 0; j < dimX; j++) {
                    field[i][j] = u[k];
                    k++;
                }
            }
            u = null;
            System.gc();
        } catch (Exception ex) {
            System.out.println("stageLoadErr");
        }
    }

    /**
     * Количество карт (уровней) всего.
     */
    public static final int TOTAL_STAGES = 11;

    /**
     * Текущий уровень.
     */
    public int stage;

    /**
     * Состояние главного автомата.
     */
    public int y0_state;

    /**
     * Счетчик длительности состояния.
     */
    public int stateCounter;

    /**
     * Константа номерос состояния ходьбы.
     */
    public static final int STATE_0_WALK = 0;

    /**
     * Константа номера состояния прицеливания.
```

```
 */
public static final int STATE_1_AIM = 1;

/**
 * Константа номера состояния выстрела.
 */
public static final int STATE_2_SHOT = 2;

/**
 * Константа номера состояния качания.
 */
public static final int STATE_3_SWING = 3;

/**
 * Константа номера состояния полета.
 */
public static final int STATE_4_FLY = 4;

/**
 * Константа номера состояния смерти.
 */
public static final int STATE_5_BLINK = 5;

/**
 * Переменная направления спрайта космонавта.
 */
public int direction;

/**
 * Переменная направления спрайта космонавта при качании.
 */
private int swingDirState;

/**
 * Константы спрайта направления.
 */
public static final int DIR_STOP_RIGHT = 0;
public static final int DIR_STOP_LEFT = 1;
public static final int DIR_LEFT = 2;
public static final int DIR_RIGHT = 3;
public static final int DIR_UP = 4;
public static final int DIR_DOWN = 5;
public static final int DIR_LEFT_UP = 6;
public static final int DIR_LEFT_DOWN = 7;
public static final int DIR_RIGHT_UP = 8;
public static final int DIR_RIGHT_DOWN = 9;

/**
 * Константа события таймера без нажатых кнопок.
 */
public static final int EVENT_0_KEY_NONE_TIMER = 0;

/**
 * Константа события таймера с нажатой конопкой стрельбы.
 */
public static final int EVENT_1_KEY_CENTER = 1;

/**
 * Константа события таймера с нажатой кнопкой "влево".
 */
public static final int EVENT_2_KEY_LEFT = 2;

/**
 * Константа события таймера с нажатой кнопкой "вправо".
 */
public static final int EVENT_3_KEY_RIGHT = 3;
```

```
/**
 * Константа события таймера с нажатой кнопкой "вверх".
 */
public static final int EVENT_4_KEY_UP = 4;

/**
 * Константа события таймера с нажатой кнопкой "вниз".
 */
public static final int EVENT_5_KEY_DOWN = 5;

/**
 * Матрица уровня (карты).
 */
public byte[][] field;

/**
 * Ширина уровня в клетках.
 */
public int fieldCellsX;

/**
 * Высота уровня в клетках.
 */
public int fieldCellsY;

/**
 * Положение центра камеры в уровне. Абсцисса.
 * Значение с фиксированой точкой.
 */
public int fEyeX;

/**
 * Положение центра камеры в уровне. Ордината.
 * Значение с фиксированой точкой.
 */
public int fEyeY;

/**
 * Точка, к которой стремится камера. Абсцисса.
 * Значение с фиксированой точкой.
 */
public int fStirvingX;

/**
 * Точка, к которой стремится камера. Ордината.
 * Значение с фиксированой точкой.
 */
public int fStirvingY;

/**
 * Координаты центра космонавта. Абсцисса.
 * Значение с фиксированой точкой.
 */
public int fManX;

/**
 * Координаты центра космонавта. Ордината.
 * Значение с фиксированой точкой.
 */
public int fManY;

/**
 * Ордината точки начала полета. Используется для определения высоты падения.
 */
private int fBeginFlyY;
```

```
/**
 * Скорость человека по оси X.
 * Значение с фиксированой точкой.
 */
private int fManVX;

/**
 * Скорость человека по оси У.
 * Значение с фиксированой точкой.
 */
private int fManVY;

/**
 * Константа, определяющая гравитацию.
 */
private int GRAV = 2;

/**
 * Абсцисса конца гарпуна.
 * Значение с фиксированой точкой.
 */
public int fHarX;

/**
 * Ордината конца гарпуна.
 * Значение с фиксированой точкой.
 */
public int fHarY;

/**
 * Угол прицела гарпуна.
 * Значение с фиксированой точкой.
 */
private int fAngle;

/**
 * Угол при качании.
 * Значение с фиксированой точкой двойной точности.
 */
private int ffAngle;

/**
 * Угловая скорость при качании.
 * Значение с фиксированой точкой двойной точности.
 */
private int ffAngleSpeed;

/**
 * Длина активной части веревки.
 * Значение с фиксированой точкой.
 */
private int fRopeLength;

/**
 * Угол, под которым был произведен выстрел в предыдущий раз.
 * Значение с фиксированой точкой.
 */
private int fPrevAngle;

/**
 * Стек точек перегиба веревки.
 */
public Stack hard;

/**
```

```
 * Абсцисса ячейки, в которой не надо делать перегиб.
 */
private int avoidX;

/**
 * Ордината ячейки, в которой не надо делать перегиб.
 */
private int avoidY;

/**
 * Направления качание, в котором не надо делать перегиб.
 */
private boolean avoidP;

/**
 * Скорость поворота прицела.
 * Значение с фиксированой точкой.
 */
private static final int AIM_SPEED = (1 << 8) / 2;

/**
 * Скорость выстрела.
 * Значение с фиксированой точкой.
 */
private static final int SHOT_SPEED = (1 << 8) / 5;

/**
 * Скорость удлиннения / укорачивания веревки.
 * Значение с фиксированой точкой.
 */
private static final int ROPE_SPEED = (2 << 8) / 16;

/**
 * Минимальная длина веревки.
 * Значение с фиксированой точкой.
 */
private static final int ROPE_MIN_LEN = (1 << 8);

/**
 * Максимальная длина веревки.
 * Значение с фиксированой точкой.
 */
private static final int ROPE_MAX_LEN = (6 << 8) + (1 << 7);

/**
 * Полувысота космонавта.
 * Значение с фиксированой точкой.
 */
private static final int MAN_HEIGHT_2 = (1 << 8) / 2;

/**
 * Полуширина космонавта.
 * Значение с фиксированой точкой.
 */
private static final int MAN_WIDTH_2 = (1 << 8) / 2;

/**
 * Скорость ходьбы.
 * Значение с фиксированой точкой.
 */
private static final int WALK_SPEED = (1 << 8) / 16;

/**
 * Максимальная скорость падения.
 * Значение с фиксированой точкой.
 */
```

```
    private static final int MAX_FALL_SPEED = (1 << 8);

    /**
     * Угол, на который изменяется направление силы гравитации при раскачивании
     * кнопками "влево" и "вправо".
     * Значение с фиксированой точкой.
     */
    private static final int SWAY_ANGLE = (1 << 8);

    /**
     * Максимальная безопасная высота падения.
     */
    private static final int FALL_HEIGHT = 4 << 8;

    /**
     * Объект поддержки протокола.
     * @link aggregation
     * @clientCardinality 1
     * @supplierCardinality 1
     */
    public Log log;

    /**
     * Загружает игру из потока данных.
     * @param is Входной поток данных.
     * @throws IOException
     */
    public synchronized void loadGameState(DataInputStream is) throws IOException {
        stage = is.readInt();
        loadStage(stage);
        avoidP = is.readBoolean();
        avoidX = is.readInt();
        avoidY = is.readInt();
        direction = is.readInt();
        fAngle = is.readInt();
        fBeginFlyY = is.readInt();
        fEyeX = is.readInt();
        fEyeY = is.readInt();
        ffAngle = is.readInt();
        ffAngleSpeed = is.readInt();
        fHarX = is.readInt();
        fHarY = is.readInt();
        fieldCellsX = is.readInt();
        fieldCellsY = is.readInt();
        fManVX = is.readInt();
        fManVY = is.readInt();
        fManX = is.readInt();
        fManY = is.readInt();
        fPrevAngle = is.readInt();
        fRopeLength = is.readInt();
        fStirvingX = is.readInt();
        fStirvingY = is.readInt();
        y0_state = is.readInt();
        stateCounter = is.readInt();
        swingDirState = is.readInt();
        int n = is.readInt();
        hard = null;
        hard = new Stack();
        for (int i = 0; i < n; i++) {
            hard.addElement(new Hardpoint(is.readInt(), is.readInt(), is.readInt(),
is.readInt(), is.readBoolean()));
        }
    }
```

```
/**
 * Сохраняет игру в выходной поток данных.
 * @param os Выходной поток.
 * @throws IOException
 */
public synchronized void saveGameState(DataOutputStream os) throws IOException {
    os.writeInt(stage);
    os.writeBoolean(avoidP);
    os.writeInt(avoidX);
    os.writeInt(avoidY);
    os.writeInt(direction);
    os.writeInt(fAngle);
    os.writeInt(fBeginFlyY);
    os.writeInt(fEyeX);
    os.writeInt(fEyeY);
    os.writeInt(ffAngle);
    os.writeInt(ffAngleSpeed);
    os.writeInt(fHarX);
    os.writeInt(fHarY);
    os.writeInt(fieldCellsX);
    os.writeInt(fieldCellsY);
    os.writeInt(fManVX);
    os.writeInt(fManVY);
    os.writeInt(fManX);
    os.writeInt(fManY);
    os.writeInt(fPrevAngle);
    os.writeInt(fRopeLength);
    os.writeInt(fStirvingX);
    os.writeInt(fStirvingY);
    os.writeInt(y0_state);
    os.writeInt(stateCounter);
    os.writeInt(swingDirState);
    os.writeInt(hard.size());
    for (int i = 0; i < hard.size(); i++) {
        os.writeInt(((Hardpoint) hard.elementAt(i)).fX);
        os.writeInt(((Hardpoint) hard.elementAt(i)).fY);
        os.writeInt(((Hardpoint) hard.elementAt(i)).fLength);
        os.writeInt(((Hardpoint) hard.elementAt(i)).ffAngle);
        os.writeBoolean(((Hardpoint) hard.elementAt(i)).angleSpeedP);
    }
}

/**
 * Конструктор математической модели.
 */
public RockmanGame() {
    log = new Log("protocol.txt");
    log.programMes("Начало работы программы.");
    initStage(0);
    y0_state = STATE_0_WALK;
}

/**
 * Загружает карту (уровень).
 * @param stage Номер уровня.
 */
private void initStage(int stage) {
    this.stage = stage;
    hard = null;
    System.gc();
    loadStage(stage);
    hard = new Stack();
    fieldCellsY = field.length;
    fieldCellsX = field[0].length;
    direction = DIR_STOP_RIGHT;
    fPrevAngle = 16 << 8;
```

```
    }

    /**
     * Поворачивает прицел по часовой стрелке.
     */
    private void z16_aimUpdateCW() {
        log.action(16, "Поворот прицела по часовой стрелки");
        fAngle += AIM_SPEED;
        if (fAngle > 32 << 8) {
            fAngle = 32 << 8;
        }
        direction = fAngle <= 16 << 8 ? DIR_RIGHT : DIR_LEFT;
    }

    /**
     * Поворачивает прицел против часовой стрелки.
     */
    private void z17_aimUpdateCCW() {
        log.action(17, "Поворот прицела против часовой стрелки");
        fAngle -= AIM_SPEED;
        if (fAngle < 0) {
            fAngle = 0;
        }
        direction = fAngle <= 16 << 8 ? DIR_RIGHT : DIR_LEFT;
    }

    /**
     * Инициализация состояния выстрела.
     */
    private void z18_initShoot() {
        log.action(18, "Инициализация состояния «Стрельба»");
        fPrevAngle = fAngle;
        fRopeLength = 0;
        fHarX = fManX;
        fHarY = fManY;
    }

    /**
     * Инициализация состояния качания.
     */
    private void z20_initSwing() {
        log.action(20, "Инициализация состояния «Качание»");
        fAngle = angle64float(fAngle + (32 << 8));
        ffAngle = fAngle << 8;
        ffAngleSpeed = 0;
        swingDirState = direction;
    }

    /**
     * @return Ударился ли гарпун в скалу, в которую может втыкаться? (Бывает два типа
скалы,
     * в которую втыкается, и в которую не втыкается).
     */
    private boolean x13_harpooHitGood() {
        boolean ret = false;
        if (hit(fHarX, fHarY)) {
            if ((field[fHarY >> 8][fHarX >> 8] & 96) == 0 || (field[fHarY >> 8][fHarX >>
8] & 96) == 64) {
                ret = true;
            }
        }
        log.input(13, ret, "Гарпун смог воткнуться в скалу");
        return ret;
    }

    /**
```

```
     * Входное воздействие.
     * @return Ударился ли гарпун о плохую скалу или другой невтыкабельный объект?
     */
    private boolean x14_harpooHitBad() {
        boolean ret = false;
        if (hit(fHarX, fHarY)) {
            if ((field[fHarY >> 8][fHarX >> 8] & 96) == 0 || (field[fHarY >> 8][fHarX >>
8] & 96) == 64) {
            } else {
                ret = true;
            }
        }
        log.input(14, ret, "Гарпун ударился о «плохую» скалу");
        return ret;
    }

    /**
     * Входное воздействие.
     * @return Длиннее ли выстреленная веревка максимальной дальности стрельбы?
     */
    private boolean x12_ropeTooLong() {
        boolean ret = fRopeLength > ROPE_MAX_LEN;
        log.input(12, ret, "Выстрелянная веревка длиннее максимальной дальности
выстрела");
        return ret;
    }

    /**
     * Удлинняет выстреленную веревку и обновляет координаты конца гарпуна.
     */
    private void z19_shootGo() {
        log.action(19, "Удлинение выстрелянной веревки и обновление координат конца
гарпуна");
        fRopeLength += SHOT_SPEED;
        fHarX = fManX + (cosineFloatPreciese(fAngle) * fRopeLength >> 8);
        fHarY = fManY - (sineFloatPreciese(fAngle) * fRopeLength >> 8);
    }

    /**
     * Ударяется ли заданная точка в гору?
     * @param x Абсцисса (фикс.).
     * @param y Ордината (фикс.).
     * @return Да или Нет.
     */
    private boolean hit(int x, int y) {
        int fx = x & 255;
        int fy = y & 255;
        int block = field[y >> 8][x >> 8];
        switch (block & 96) {
            case 96:
                return false;
            default:
                block &= 31;
                switch (block) {
                    case NOP:
                        return false;
                    case C:
                        return true;
                    case DR1V:
                        return fy > (2 << 8) - (fx << 1);
                    case DR2V:
                        return fy > (1 << 8) - (fx << 1);
                    case DL1V:
                        return fy > (fx << 1);
                    case DL2V:
                        return fy > -(1 << 8) + (fx << 1);
```

```
                    case UR2V:
                        return fy < (fx << 1);
                    case UR1V:
                        return fy < -(1 << 8) + (fx << 1);
                    case UL2V:
                        return fy < (2 << 8) - (fx << 1);
                    case UL1V:
                        return fy < (1 << 8) - (fx << 1);
                    case UL:
                        return fy < (1 << 8) - fx;
                    case DL:
                        return fy > fx;
                    case UR:
                        return fy < fx;
                    case DR:
                        return fy > (1 << 8) - fx;
                    case DR1H:
                        return fy > (1 << 8) - (fx >> 1);
                    case DR2H:
                        return fy > (1 << 7) - (fx >> 1);
                    case DL1H:
                        return fy > (1 << 7) + (fx >> 1);
                    case DL2H:
                        return fy > (fx >> 1);
                    case UR2H:
                        return fy < (1 << 7) + (fx >> 1);
                    case UR1H:
                        return fy < (fx >> 1);
                    case UL2H:
                        return fy < (1 << 8) - (fx >> 1);
                    case UL1H:
                        return fy < (1 << 7) - (fx >> 1);
                    default:
                        return false;
                }
            }
        }

        /**
         * Некоторые вычисления для перехода в состояние «Полет» из состояния «Качание».
         */
        private void z5_swingPostclean() {
            log.action(5, "Некоторые вычисления для перехода в состояние полета из состояния
качания");
            int cfp = cosineFloatPreciese(fAngle);
            int sfp = sineFloatPreciese(fAngle);
            hard.removeAllElements();
            int v = 3 * (fRopeLength * (ffAngleSpeed >> 8) >> 8) >> 5;
            fManVX = -v * sfp >> 8;
            fManVY = -v * cfp >> 8;
            if (fManVX == 0) {
                direction = swingDirState;
            } else if (fManVX > 0) {
                direction = DIR_RIGHT;
            } else {
                direction = DIR_LEFT;
            }
            fBeginFlyY = fManY;
        }

        /**
         * Рассчитывает некоторые параметры раскачки при качании с нажатой кнопкой "влево".
         */
        private void z0_swingKeyLeft() {
            log.action(0, "Расчет некоторых параметров раскачки при качании с нажатой кнопкой
«влево»");
```

```
        direction = DIR_LEFT;
        swingDirState = DIR_LEFT;
        ffE = (sineFloatPreciese(angle64float((48 << 8) - fAngle - SWAY_ANGLE)) << 16) /
(fRopeLength * fRopeLength >> 8);
    }

    /**
     * Рассчитывает некоторые параметры раскачки при качании с нажатой кнопкой "вправо".
     */
    private void z1_swingKeyRight() {
        log.action(1, "Расчет некоторых параметров раскачки при качании с нажатой кнопкой
«вправо»");
        direction = DIR_RIGHT;
        swingDirState = DIR_RIGHT;
        ffE = (sineFloatPreciese(angle64float((48 << 8) - fAngle + SWAY_ANGLE)) << 16) /
(fRopeLength * fRopeLength >> 8);
    }

    /**
     * Рассчитывает качание при свободном качании (без нажатых кнопок).
     */
    private void z25_swingNoRightNoLeft() {
        log.action(25, "Расчет параметров при свободном качании (без нажатых кнопок)");
        ffE = (sineFloatPreciese(angle64float((48 << 8) - fAngle)) << 16) / (fRopeLength
* fRopeLength >> 8);
    }

    /**
     * Космонавт подтягивается на веревку. Длина веревки уменьшается.
     */
    private void z2_ropeMinus() {
        log.action(2, "Космонавт подтягивается на веревку. Длина веревки уменьшается.");
        fRopeLength -= ROPE_SPEED;
        if (fRopeLength < ROPE_MIN_LEN) {
            fRopeLength = ROPE_MIN_LEN;
        } else {
            direction = DIR_UP;
        }
    }

    /**
     * Инициализация состояния прицела.
     * Выставляем значение спрайта направления и начального угла прицела.
     */
    private void z15_initAim() {
        log.action(15, "Инициализация состояния «Прицеливание»");
        switch (direction) {
            case DIR_RIGHT:
            case DIR_RIGHT_DOWN:
            case DIR_STOP_RIGHT:
            case DIR_RIGHT_UP:
                fAngle = (16 << 8) - Math.abs(fPrevAngle - (16 << 8));
                direction = DIR_RIGHT;
                break;
            case DIR_LEFT:
            case DIR_LEFT_DOWN:
            case DIR_STOP_LEFT:
            case DIR_LEFT_UP:
                fAngle = (16 << 8) + Math.abs(fPrevAngle - (16 << 8));
                direction = DIR_LEFT;
                break;
            default:
                direction = -1;
                break;
        }
    }
```

```
/**
     * Космонавт отпускает веревку. Длина веревки увеличивается.
 */
private void z3_ropePlus() {
    log.action(3, "Космонавт отпускает веревку. Длина веревки увеличивается.");
    fRopeLength += ROPE_SPEED;
    if (fRopeLength > ROPE_MAX_LEN) {
        fRopeLength = ROPE_MAX_LEN;
    } else {
        direction = DIR_DOWN;
    }
}

private int ffE;

/**
 * @labelDirection forward
 * @directed
 * @link aggregation
 * @clientRole aggregates stack
 * @supplierRole element
 * @clientCardinality 1
 * @supplierCardinality 0..*
 */
private Hardpoint lnkHardpoint;

/**
 * Физика качания: раскачка, удары об стены, угловое ускорение...
 */
private void z4_swingPhysics() {
    log.action(4, "Расчет координат, скорости, ускорения космонавта при раскачке");
    int dxUD = xCosine(MAN_HEIGHT_2, angle64(fAngle - (32 << 8) >> 8));
    int dyUD = xSine(MAN_HEIGHT_2, angle64(fAngle - (32 << 8) >> 8));
    int fUpX = fManX + dxUD;
    int fUpY = fManY - dyUD;
    int fDownX = fManX - dxUD;
    int fDownY = fManY + dyUD;
    int dxPM = xCosine(MAN_WIDTH_2, angle64(fAngle - (16 << 8) >> 8));
    int dyPM = xSine(MAN_WIDTH_2, angle64(fAngle - (16 << 8) >> 8));
    int fApX = fManX - dxPM;
    int fApY = fManY + dyPM;
    int fAmX = fManX + dxPM;
    int fAmY = fManY - dyPM;
    boolean hitp = hit(fApX, fApY);
    boolean hitm = hit(fAmX, fAmY);
    boolean hitpu = hit(fApX + fUpX >> 1, fApY + fUpY >> 1);
    boolean hitpd = hit(fApX + fDownX >> 1, fApY + fDownY >> 1);
    boolean hitmu = hit(fAmX + fUpX >> 1, fAmY + fUpY >> 1);
    boolean hitmd = hit(fAmX + fDownX >> 1, fAmY + fDownY >> 1);
    if (hitp || hitpd || hitpu) {
        ffAngleSpeed = -Math.abs(ffAngleSpeed);
    } else if (hitm || hitmd || hitmu) {
        ffAngleSpeed = Math.abs(ffAngleSpeed);
    } else {
        ffAngleSpeed += ffE;
    }
    ffAngleSpeed -= (ffAngleSpeed >> 6);

    ffAngle += ffAngleSpeed;
    fAngle = angle64float(ffAngle >> 8);
    int cfp = cosineFloatPreciese(fAngle);
    int sfp = sineFloatPreciese(fAngle);
    fManX = fHarX + (cfp * fRopeLength >> 8);
    fManY = fHarY - (sfp * fRopeLength >> 8);
    boolean angleSpeedP = ffAngleSpeed > 0;
```

```
        // hardpoint removing
        if (!hard.isEmpty()) {
            Hardpoint h = (Hardpoint) hard.peek();
            if (Math.abs(h.ffAngle - ffAngle) < Math.abs(ffAngleSpeed) && h.angleSpeedP
!= angleSpeedP) {
                h = (Hardpoint) hard.pop();
                avoidX = fHarX >> 8;
                avoidY = fHarY >> 8;
                avoidP = angleSpeedP;
                fHarX = h.fX;
                fHarY = h.fY;
                ffAngleSpeed = ffAngleSpeed * fRopeLength / (fRopeLength + h.fLength);
                fRopeLength += h.fLength;
                ffAngle = h.ffAngle;
                fAngle = ffAngle >> 8;
            }
        }

        // hardpoint adding
        final int STEP = 15;
        int ffDX = cfp * STEP;
        int ffDY = -sfp * STEP;
        int count = (fRopeLength - (1 << 7)) / STEP;
        int ffX = fManX << 8;
        int ffY = fManY << 8;
        int fX;
        int fY;
        for (int i = 1; i <= count; i++) {
            fX = ffX >> 8;
            fY = ffY >> 8;
            if (hit(fX, fY) && !(fX >> 8 == avoidX && fY >> 8 == avoidY && angleSpeedP ==
avoidP)) {
                int fLenOld = Math.abs(sfp) > Math.abs(cfp) ? Math.abs((fY - fHarY << 8)
/ sfp) : Math.abs((fX - fHarX << 8) / cfp);
                Hardpoint hp = new Hardpoint(fHarX, fHarY, fLenOld, ffAngle, ffAngleSpeed
> 0);
                hard.push(hp);
                fHarX = fX;
                fHarY = fY;
                int fRest = fRopeLength - fLenOld;
                ffAngleSpeed = ffAngleSpeed * fRopeLength / fRest;
                fRopeLength = fRest;
                break;
            }
            ffX -= ffDX;
            ffY -= ffDY;
        }
        direction = swingDirState;
    }

    /**
     * Входное воздействие.
     * @return Стукается ли космонавт головой? Используется при укорачивании веревки.
     */
    private boolean x5_hitUp() {
        int dxUD = xCosine(MAN_HEIGHT_2, angle64(fAngle - (32 << 8) >> 8));
        int dyUD = xSine(MAN_HEIGHT_2, angle64(fAngle - (32 << 8) >> 8));
        int fUpX = fManX + dxUD;
        int fUpY = fManY - dyUD;
        int dxPM = xCosine(MAN_WIDTH_2, angle64(fAngle - (16 << 8) >> 8));
        int dyPM = xSine(MAN_WIDTH_2, angle64(fAngle - (16 << 8) >> 8));
        int fApX = fManX - dxPM;
        int fApY = fManY + dyPM;
        int fAmX = fManX + dxPM;
        int fAmY = fManY - dyPM;
```

```
        boolean hitu = hit(fUpX, fUpY);
        boolean hitpu = hit(fApX + fUpX >> 1, fApY + fUpY >> 1);
        boolean hitmu = hit(fAmX + fUpX >> 1, fAmY + fUpY >> 1);
        boolean ret = hitu || hitpu || hitmu;
        log.input(5, ret, "Космонавт соприкасается головой со стеной при укорачивании
веревки");
        return ret;
    }

    /**
     * Проверяет, можно ли удлиннить веревку. (Стукается ли ногами в
землю/стену/потолок).
     * @return Стукается - да, иначе - нет.
     */
    private boolean x6_hitDown() {
        int dxUD = xCosine(MAN_HEIGHT_2, angle64(fAngle - (32 << 8) >> 8));
        int dyUD = xSine(MAN_HEIGHT_2, angle64(fAngle - (32 << 8) >> 8));
        int fDownX = fManX - dxUD;
        int fDownY = fManY + dyUD;
        int dxPM = xCosine(MAN_WIDTH_2, angle64(fAngle - (16 << 8) >> 8));
        int dyPM = xSine(MAN_WIDTH_2, angle64(fAngle - (16 << 8) >> 8));
        int fApX = fManX - dxPM;
        int fApY = fManY + dyPM;
        int fAmX = fManX + dxPM;
        int fAmY = fManY - dyPM;
        boolean hitd = hit(fDownX, fDownY);
        boolean hitpd = hit(fApX + fDownX >> 1, fApY + fDownY >> 1);
        boolean hitmd = hit(fAmX + fDownX >> 1, fAmY + fDownY >> 1);
        boolean ret = hitd || hitpd || hitmd;
        log.input(6, ret, "Космонавт соприкасается ногами со стеной при удлинении
веревки");
        return ret;
    }

    /**
     * Инициирование процесса проигрыша. Установка счетчика состояния в 0 и установка
направления.
     */
    private void z6_initBlink() {
        log.action(6, "Инициализация состояния «Смерть»");
        switch (direction) {
            case DIR_RIGHT:
            case DIR_RIGHT_DOWN:
            case DIR_RIGHT_UP:
                direction = DIR_RIGHT;
                break;
            case DIR_LEFT:
            case DIR_LEFT_DOWN:
            case DIR_LEFT_UP:
                direction = DIR_LEFT;
                break;
            default:
                direction = DIR_LEFT;
                break;
        }
        stateCounter = 0;
    }

    /**
     * Входное воздействие.
     * @return Задел ли объект, который его убивает?
     */
    private boolean x7_deathObject() {
        boolean ret = false;
        int manX = fManX >> 8;
        int manY = fManY >> 8;
```

```
    int cell = field[manY][manX];
    switch (cell & 96) {
        case 64:
            ret = true;
            break;
        case 96:
            switch (cell & 31) {
                case SPIKE:
                    ret = true;
                    break;
            }
            break;
    }
    log.input(7, ret, "Космонавт находится на смертельном участке карты");
    return ret;
}

/**
 * Входное воздействие.
 * @return Задел ли выход с уровня.
 */
private boolean x8_exitObject() {
    boolean ret = false;
    int manX = fManX >> 8;
    int manY = fManY >> 8;
    int cell = field[manY][manX];
    switch (cell & 96) {
        case 96:
            // objects
            switch (cell & 31) {
                // win level point
                case EXIT:
                    ret = true;
                    break;
            }
            break;
    }
    log.input(8, ret, "Космонавт находится на выходе");
    return ret;
}

/**
 * Входное воздействие.
 * @return Превышает ли текущая высота падения максимальную?
 */
private boolean x16_fallTooHigh() {
    boolean ret = Math.abs(fBeginFlyY - fManY) > FALL_HEIGHT;
    log.input(16, ret, "Текущая высота падения космонавта смертельна");
    return ret;
}

/**
 * Физика полета: гравитацаонное ускорение, удары об стены...
 */
private void z21_flightPhysics() {
    log.action(21, "Физический просчет полета");
    fManVY += GRAV;
    if (fManVY > MAX_FALL_SPEED) {
        fManVY = MAX_FALL_SPEED;
    }
    fManX += fManVX;
    fManY += fManVY;
    int fUpY = fManY - MAN_HEIGHT_2;
    int fDownY = fManY + MAN_HEIGHT_2;
    int fLeftX = fManX - MAN_WIDTH_2;
    int fRightX = fManX + MAN_WIDTH_2;
```

```
        boolean hitu = hit(fManX, fUpY);
        //boolean hitd = hit(fManX, fDownY);
        boolean hitr = hit(fRightX, fManY);
        boolean hitl = hit(fLeftX, fManY);
        boolean hitur = hit(fRightX + fManX >> 1, fUpY + fManY >> 1);
        boolean hitul = hit(fLeftX + fManX >> 1, fUpY + fManY >> 1);
        boolean hitdr = hit(fRightX + fManX >> 1, fDownY + fManY >> 1);
        boolean hitdl = hit(fLeftX + fManX >> 1, fDownY + fManY >> 1);
        if (hitur || hitul || hitu) {
            fManVY = Math.abs(fManVY);
        }
        if (hitr || hitdr) {
            fManVX = -Math.abs(fManVX >> 1);
        }
        if (hitl || hitdl) {
            fManVX = Math.abs(fManVX >> 1);
        }
        if (fManVX != 0) {
            direction = fManVX > 0 ? DIR_RIGHT : DIR_LEFT;
        }
    }

    /**
     * Входное воздействие.
     * @return Ударился ли ногами при полете? (приземление).
     */
    private boolean x15_flightHitLegs() {
        int fDownY = fManY + MAN_HEIGHT_2;
        boolean hitd = hit(fManX, fDownY);
        log.input(15, hitd, "Космонавт приземлился после падения");
        return hitd;
    }

    /**
     * Выставляет значение направления движения, в зависимости от поверхности.
     */
    private void z12_walkRightUpdateDir() {
        log.action(12, "Выставление значения направления при движении направо в
зависимости от поверхности");
        // now these vars are mans feets
        int fManX = this.fManX;
        int fManY = this.fManY + MAN_HEIGHT_2;
        switch (field[fManY + 10 >> 8][fManX >> 8] & 31) {
            case DR:
            case DR1H:
            case DR2H:
            case DR1V:
            case DR2V:
                direction = DIR_RIGHT_UP;
                break;
            case DL:
            case DL1H:
            case DL2H:
            case DL1V:
            case DL2V:
                direction = DIR_RIGHT_DOWN;
                break;
            default:
                direction = DIR_RIGHT;
                break;
        }
    }
```

```
    /**
     * Выставляет значение направления движения, в зависимости от поверхности.
     */
    private void z13_walkLeftUpdateDir() {
        log.action(13, "Выставление значения направления при движении налево в
зависимости от поверхности");
        // now these vars are mans feets
        int fManX = this.fManX;
        int fManY = this.fManY + MAN_HEIGHT_2;
        switch (field[fManY + 10 >> 8][fManX >> 8] & 31) {
            case DL:
            case DL1H:
            case DL2H:
            case DL1V:
            case DL2V:
                direction = DIR_LEFT_UP;
                break;
            case DR:
            case DR1H:
            case DR2H:
            case DR1V:
            case DR2V:
                direction = DIR_LEFT_DOWN;
                break;
            default:
                direction = DIR_LEFT;
                //direction = -1;
                break;
        }
    }

    /**
     * Выбирает спрайт (направление) для стояния.
     */
    private void z14_walkStop() {
        log.action(14, "Определение позы космонавта при остановке в состоянии ходьбы");
        switch (direction) {
            case DIR_RIGHT:
            case DIR_RIGHT_DOWN:
            case DIR_RIGHT_UP:
                direction = DIR_STOP_RIGHT;
                break;
            case DIR_LEFT:
            case DIR_LEFT_DOWN:
            case DIR_LEFT_UP:
                direction = DIR_STOP_LEFT;
                break;
        }
    }

    /**
     * Входное воздействие.
     * @return Уперся ли в стену слева при ходьбе?
     */
    private boolean x9_walkHitLeft() {
        boolean ret = hit(fManX - MAN_WIDTH_2 + 30, fManY) || hit(fManX - MAN_WIDTH_2 +
30, fManY - MAN_HEIGHT_2);
        log.input(9, ret, "Космонавт уперся в стенку слева");
        return ret;
    }

    /**
     * Входное воздействие.
     * @return Уперся ли в стену справа при ходьбе?
     */
    private boolean x11_walkHitRight() {
```

```
        boolean ret = hit(fManX + MAN_WIDTH_2 - 30, fManY) || hit(fManX + MAN_WIDTH_2 -
30, fManY - MAN_HEIGHT_2);
        log.input(11, ret, "Космонавт уперся в стенку справа");
        return ret;
    }

    /**
     * Входное воздействие.
     * @return Началось ли падение при ходьбе? (Потерял ли опору под ногами?)
     */
    private boolean x10_walkFalling() {
        boolean ret = true;
        for (int fDY = 0; fDY < 100; fDY++) {
            if (hit(fManX, fManY + fDY + MAN_HEIGHT_2)) {
                ret = false;
            }
        }
        log.input(10, ret, "Космонавт потерял опору под ногами");
        return ret;
    }

    /**
     * Идем влево.
     */
    private void z7_walkLeft() {
        log.action(7, "Сдвиг координат космонавта при ходьбе влево");
        fManX -= WALK_SPEED;
    }

    /**
     * Идем вправо.
     */
    private void z10_walkRight() {
        log.action(10, "Сдвиг координат космонавта при ходьбе вправо");
        fManX += WALK_SPEED;
    }

    /**
     * Двигает человека так, чтобы его ноги оказались на поверхности земли.
     */
    private void z8_stableWalk() {
        log.action(8, "Двигает космонавта так, чтобы его ноги оказались на поверхности
земли");
        int fManX = this.fManX;
        int fManY = this.fManY + MAN_HEIGHT_2;
        boolean h = hit(fManX, fManY);
        for (int fDY = 0; ; fDY++) {
            if (h != hit(fManX, fManY + fDY)) {
                fManY += fDY;
                break;
            }
            if (h != hit(fManX, fManY - fDY)) {
                fManY -= fDY - 1;
                break;
            }
        }
        this.fManX = fManX;
        this.fManY = fManY - MAN_HEIGHT_2;
    }

    /**
     *  Инициализация полета после падения влево из состояния «Ходьба».
     */
    private void z9_startLeftFlight() {
        log.action(9, "Инициализация полета после падения влево из состояния «Ходьба»");
        int fManX = this.fManX;
```

```
        int fManY = this.fManY + MAN_HEIGHT_2;
        fManX += -WALK_SPEED;
        fManX += -WALK_SPEED;
        fManVY = 0;
        fManVX = -WALK_SPEED > 0 ? WALK_SPEED : -WALK_SPEED;
        direction = -WALK_SPEED > 0 ? DIR_RIGHT : DIR_LEFT;
        fBeginFlyY = this.fManY;
        this.fManX = fManX;
        this.fManY = fManY - MAN_HEIGHT_2;
    }

    /**
     * Инициализация полета после падения вправо из состояния «Ходьба».
     */
    private void z11_startRightFlight() {
        log.action(11, "Инициализация полета после падения вправо из состояния
«Ходьба»");
        int fManX = this.fManX;
        int fManY = this.fManY + MAN_HEIGHT_2;
        fManX += WALK_SPEED;
        fManX += WALK_SPEED;
        fManVY = 0;
        fManVX = WALK_SPEED > 0 ? WALK_SPEED : -WALK_SPEED;
        direction = WALK_SPEED > 0 ? DIR_RIGHT : DIR_LEFT;
        fBeginFlyY = this.fManY;
        this.fManX = fManX;
        this.fManY = fManY - MAN_HEIGHT_2;
    }

    /**
     * Входное воздействие.
     * @return Хватит мигать?
     */
    private boolean x17_blinkTimeout() {
        boolean ret = stateCounter >= 20;
        log.input(17, ret, "Космонавт закончил мигать");
        return ret;
    }

    /**
     * Инкремент счетчика состояния.
     */
    private void z22_stateCounterIncrement() {
        log.action(22, "Увеличение счетчика мигания космонавта при гибели");
        stateCounter++;
    }

    /**
     * Реинициализация уровня (карты).
     */
    private void z23_initStage() {
        log.action(23, "Инициализация текущего уровня игры");
        initStage(stage);
    }

    /**
     * Инкремент уровня и его загрузка.
     */
    private void z24_nextStage() {
        log.action(24, "Инициализация следующего уровня игры");
        stage++;
        if (stage == TOTAL_STAGES - 1) {
            stage = 0;
        }
        initStage(stage);
    }
```

```
/**
 * Метод запуска главного автомата.
 * @param e Событие.
 */
public synchronized void A0_nextGameStep(int e) {
    int y0_statePrevious = y0_state;
    log.beginAutomataWork(0, y0_state);
    switch (e) {
        case EVENT_0_KEY_NONE_TIMER: log.event(1, "Пришло событие от таймера.");
                                     break;
        case EVENT_1_KEY_CENTER: log.event(1, "Нажата клавиша «5»");
                                 break;
        case EVENT_2_KEY_LEFT: log.event(2, "Нажата клавиша «4» (влево)");
                               break;
        case EVENT_3_KEY_RIGHT: log.event(3, "Нажата клавиша «6» (вправо)");
                                break;
        case EVENT_4_KEY_UP: log.event(4, "Нажата клавиша «8» (вверх)");
                             break;
        case EVENT_5_KEY_DOWN: log.event(5, "Нажата клавиша «2» (вниз)");
                               break;
    }
    switch (y0_state) {
        case STATE_3_SWING:
            if (x7_deathObject()) {
                y0_state = STATE_5_BLINK;
            } else if (x8_exitObject()) {
                z24_nextStage();
                y0_state = STATE_0_WALK;
            } else if (e == EVENT_1_KEY_CENTER) {
                z4_swingPhysics();
                z5_swingPostclean();
                y0_state = STATE_4_FLY;
            } else if (e == EVENT_2_KEY_LEFT) {
                z0_swingKeyLeft();
                z4_swingPhysics();
            } else if (e == EVENT_3_KEY_RIGHT) {
                z1_swingKeyRight();
                z4_swingPhysics();
            } else if (e == EVENT_4_KEY_UP && !x5_hitUp()) {
                z25_swingNoRightNoLeft();
                z4_swingPhysics();
                z2_ropeMinus();
            } else if (e == EVENT_5_KEY_DOWN && !x6_hitDown()) {
                z25_swingNoRightNoLeft();
                z4_swingPhysics();
                z3_ropePlus();
            } else {
                z25_swingNoRightNoLeft();
                z4_swingPhysics();
            }
            fStirvingX = fManX;
            fStirvingY = fManY;
            break;
        case STATE_0_WALK:
            if (x7_deathObject()) {
                y0_state = STATE_5_BLINK;
            } else if (x8_exitObject()) {
                z24_nextStage();
                y0_state = STATE_0_WALK;
            } else if (e == EVENT_1_KEY_CENTER) {
                y0_state = STATE_1_AIM;
            } else if (e == EVENT_2_KEY_LEFT && x10_walkFalling()) {
                z9_startLeftFlight();
                y0_state = STATE_4_FLY;
            } else if (e == EVENT_3_KEY_RIGHT && x10_walkFalling()) {
```

```
                    z11_startRightFlight();
                    y0_state = STATE_4_FLY;
                } else if (e == EVENT_2_KEY_LEFT && !x10_walkFalling() &&
!x9_walkHitLeft()) {
                    z8_stableWalk();
                    z7_walkLeft();
                    z13_walkLeftUpdateDir();
                } else if (e == EVENT_3_KEY_RIGHT && !x10_walkFalling() &&
!x11_walkHitRight()) {
                    z8_stableWalk();
                    z10_walkRight();
                    z12_walkRightUpdateDir();
                } else  {
                    z14_walkStop();
                }
                fStirvingX = fManX;
                fStirvingY = fManY;
                break;
            case STATE_1_AIM:
                if (e == EVENT_1_KEY_CENTER) {
                    y0_state = STATE_2_SHOT;
                } else if (e == EVENT_2_KEY_LEFT || e == EVENT_4_KEY_UP) {
                    z16_aimUpdateCW();
                } else if (e == EVENT_3_KEY_RIGHT || e == EVENT_5_KEY_DOWN) {
                    z17_aimUpdateCCW();
                }
                fStirvingX = fManX + cosineFloatPreciese(fAngle) * 2;
                fStirvingY = fManY - sineFloatPreciese(fAngle) * 2;
                break;
            case STATE_2_SHOT:
                if (e == EVENT_1_KEY_CENTER || x14_harpooHitBad() || x12_ropeTooLong()) {
                    y0_state = STATE_0_WALK;
                } else if (x13_harpooHitGood()) {
                    y0_state = STATE_3_SWING;
                } else {
                    z19_shootGo();
                }
                fStirvingX = fHarX;
                fStirvingY = fHarY;
                break;
            case STATE_4_FLY:
                if (x7_deathObject()) {
                    y0_state = STATE_5_BLINK;
                } else if (x8_exitObject()) {
                    z24_nextStage();
                    y0_state = STATE_0_WALK;
                } else if (x15_flightHitLegs() && x16_fallTooHigh()) {
                    y0_state = STATE_5_BLINK;
                } else if (x15_flightHitLegs() && !x16_fallTooHigh()) {
                    z8_stableWalk();
                    y0_state = STATE_0_WALK;
                } else {
                    z21_flightPhysics();
                }
                fStirvingX = fManX;
                fStirvingY = fManY;
                break;
            case STATE_5_BLINK:
                if (x17_blinkTimeout()) {
                    z23_initStage();
                    y0_state = STATE_0_WALK;
                } else {
                    z22_stateCounterIncrement();
                }
                break;
        }
```

```
        if (y0_statePrevious != y0_state) {
            switch (y0_state) {
                case STATE_0_WALK:
                    log.switchState(0, y0_statePrevious, y0_state, "Ходьба");
                    break;
                case STATE_1_AIM:
                    z15_initAim();
                    log.switchState(0, y0_statePrevious, y0_state, "Прицеливание");
                    break;
                case STATE_2_SHOT:
                    z18_initShoot();
                    log.switchState(0, y0_statePrevious, y0_state, "Стрельба");
                    break;
                case STATE_3_SWING:
                    z20_initSwing();
                    log.switchState(0, y0_statePrevious, y0_state, "Качание");
                    break;
                case STATE_4_FLY:
                    log.switchState(0, y0_statePrevious, y0_state, "Полет");
                    break;
                case STATE_5_BLINK:
                    z6_initBlink();
                    log.switchState(0, y0_statePrevious, y0_state, "Смерть");
                    break;
            }
        }
        log.endAutomataWork(0, y0_state);
        fEyeX += (fStirvingX - fEyeX) / 5;
        fEyeY += (fStirvingY - fEyeY) / 5;
    }
}
```

Класс RockmanGameView

```
package rockman;

import rockman.*;

import java.awt.*;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;

/**
 * Класс, отображающий математическую модель в графическом виде.
 */
public class RockmanGameView extends Canvas {
    /**
     * Набор спрайтов.
     */
    private Image[] imagesArray;

    /**
     * Экранный буфер.
     */
    private Image buffer;

    /**
     * Получить картинку спрайта с заданным индексом.
     * @param n Индекс спрайта.
     * @return Картинка спрайта.
     */
    protected Image getImage(int n) {
```

```
    return imagesArray[n];
}

/**
 * Раздел объявления индексов спрайтов.
 */
private static final int IMG_FP = 0;
private static final int IMG_WIN = 1;
private static final int IMG_LOST = 2;
private static final int IMG_DEMO = 3;
private static final int IMG_NFONT = 4;
private static final int IMG_EXIT01 = 5;
private static final int IMG_EXIT02 = 6;
private static final int IMG_M_UP01 = 7;
private static final int IMG_M_UP02 = 8;
private static final int IMG_M_WALK_DOWN_L01 = 9;
private static final int IMG_M_WALK_DOWN_L02 = 10;
private static final int IMG_M_WALK_UP_L01 = 11;
private static final int IMG_M_WALK_UP_L02 = 12;
private static final int IMG_M_WALK_DOWN_R01 = 13;
private static final int IMG_M_WALK_DOWN_R02 = 14;
private static final int IMG_M_WALK_UP_R01 = 15;
private static final int IMG_M_WALK_UP_R02 = 16;
private static final int IMG_M_WALK_L01 = 17;
private static final int IMG_M_WALK_L02 = 18;
private static final int IMG_M_WALK_L03 = 19;
private static final int IMG_M_WALK_R01 = 20;
private static final int IMG_M_WALK_R02 = 21;
private static final int IMG_M_WALK_R03 = 22;
private static final int IMG_M_STAND_R = 23;
private static final int IMG_M_STAND_L = 24;
private static final int IMG_TORN01 = 25;
private static final int IMG_TORN02 = 26;
private static final int IMG_WATER01 = 27;
private static final int IMG_WATER02 = 28;
private static final int IMG_GROUND_C = 29;
private static final int IMG_GROUND_UL = 30;
private static final int IMG_GROUND_UR = 31;
private static final int IMG_GROUND_DL = 32;
private static final int IMG_GROUND_DR = 33;
private static final int IMG_GROUND_DR1V = 34;
private static final int IMG_GROUND_DR2V = 35;
private static final int IMG_GROUND_DL1V = 36;
private static final int IMG_GROUND_DL2V = 37;
private static final int IMG_GROUND_UR1V = 38;
private static final int IMG_GROUND_UR2V = 39;
private static final int IMG_GROUND_UL1V = 40;
private static final int IMG_GROUND_UL2V = 41;
private static final int IMG_GROUND_DR1H = 42;
private static final int IMG_GROUND_DR2H = 43;
private static final int IMG_GROUND_DL1H = 44;
private static final int IMG_GROUND_DL2H = 45;
private static final int IMG_GROUND_UR1H = 46;
private static final int IMG_GROUND_UR2H = 47;
private static final int IMG_GROUND_UL1H = 48;
private static final int IMG_GROUND_UL2H = 49;
private static final int IMG_ROCK_C = 50;
private static final int IMG_ROCK_UL = 51;
private static final int IMG_ROCK_UR = 52;
private static final int IMG_ROCK_DL = 53;
private static final int IMG_ROCK_DR = 54;
private static final int IMG_ROCK_DR1V = 55;
private static final int IMG_ROCK_DR2V = 56;
private static final int IMG_ROCK_DL1V = 57;
private static final int IMG_ROCK_DL2V = 58;
private static final int IMG_ROCK_UR1V = 59;
```

```
    private static final int IMG_ROCK_UR2V = 60;
    private static final int IMG_ROCK_UL1V = 61;
    private static final int IMG_ROCK_UL2V = 62;
    private static final int IMG_ROCK_DR1H = 63;
    private static final int IMG_ROCK_DR2H = 64;
    private static final int IMG_ROCK_DL1H = 65;
    private static final int IMG_ROCK_DL2H = 66;
    private static final int IMG_ROCK_UR1H = 67;
    private static final int IMG_ROCK_UR2H = 68;
    private static final int IMG_ROCK_UL1H = 69;
    private static final int IMG_ROCK_UL2H = 70;
    private static final int IMG_CROSS = 71;
    private static final int TOTAL_IMAGES_NUMBER = 72;

    /**
     * Размер ячейки в пикселях.
     */
    private static final int CELL = 45;

    /**
     * Математическая модель.
     * @clientRole view
     * @supplierRole model
     * @supplierCardinality 1
     * @clientCardinality 1
     */
    private RockmanGame game;

    /**
     * Ширина игрового поля.
     */
    private int scrWidth;

    /**
     * Высота игрового поля.
     */
    private int scrHeight;

    /**
     * Сколько игровых едениц измерения (ячеек) содержится в половине ширины игрового
поля.
     * Значение с фиксированной точкой.
     */
    private int fHalfScrX;

    /**
     * Сколько игровых едениц измерения (ячеек) содержится в половине высоты игрового
поля.
     * Значение с фиксированной точкой.
     */
    private int fHalfScrY;

    /**
     * Цвет заднего фона в формате RGB.
     */
    private int cDrawBg;

    /**
     * Цвет веревки в формате RGB.
     */
    private int cRopeColor;

    /**
     * Цвет игрового поля в формате RGB.
     */
    private int cField;
```

```
/**
 * Номер "тика". Используется для анимации спрайтов.
 */
private int tics;

/**
 * Конструктор.
 * @param game Математическая модель.
 * @param imagesArray Массив спрайтов.
 */
public RockmanGameView(RockmanGame game, Image[] imagesArray) {
    this.imagesArray = imagesArray;
    this.game = game;

    addComponentListener(new ComponentAdapter() {
        public void componentResized(ComponentEvent e) {
            scrWidth = (int) getSize().getWidth();
            scrHeight = (int) getSize().getHeight();
            fHalfScrX = (scrWidth << 8) / CELL / 2;
            fHalfScrY = (scrHeight << 8) / CELL / 2;
            buffer = createImage(scrWidth, scrHeight);
        }
    });

    cDrawBg = 0x534741;
    cField = 0x729c9b;
    cRopeColor = 0x000000;
}

/**
 * Обновляет экран. Вызывается системой.
 * @param g Графический контекст.
 */
public void update(Graphics g) {
    paint(g);
}

/**
 * Управляет прорисовкой экрана и двойной буферизацей.
 * @param g Графический контекст.
 */
public void paint(Graphics g) {
    draw(new GraphicsME(buffer.getGraphics()));
    g.drawImage(buffer, 0, 0, null);
}

/**
 * Рисует состояние математической модели на графическом контексте.
 * @param g Модифицированный графический контекст.
 */
public void draw(GraphicsME g) {
    tics++;
    // prepare background
    g.setColor(cDrawBg);
    g.fillRect(0, 0, scrWidth, scrHeight);

    // player eyes in the center of the screen
    int tx = (scrWidth >> 1) - (CELL * game.fEyeX >> 8);
    int ty = (scrHeight >> 1) - (CELL * game.fEyeY >> 8);
    g.translate(tx, ty);

    // viewable bounds of game field
    int minViewX = game.fEyeX - fHalfScrX >> 8;
    int maxViewX = game.fEyeX + fHalfScrX >> 8;
    int minViewY = game.fEyeY - fHalfScrY >> 8;
```

```
        int maxViewY = game.fEyeY + fHalfScrY >> 8;

        // wich cells of field should be drawn
        int minX = Math.max(0, minViewX);
        int maxX = Math.min(game.fieldCellsX - 1, maxViewX);
        int minY = Math.max(0, minViewY);
        int maxY = Math.min(game.fieldCellsY - 1, maxViewY);

        // drawing field inside stage bounds
        int x;
        int y = minY * CELL;
        int xBeg = minX * CELL;
        g.setColor(cField);
        // filling field backgroung inside stage bounds
        g.fillRect(xBeg, y, (maxX - minX + 1) * CELL, (maxY - minY + 1) * CELL);
        int cell;
        for (int i = minY; i <= maxY; i++) {
            x = xBeg;
            for (int j = minX; j <= maxX; j++) {
                cell = game.field[i][j];
                switch (cell & 96) {
                    case 0:
                        // can stick
                        cell = cell & 31;
                        switch (cell) {
                            case 31:
                                // nothing
                                break;
                            default:
                                g.drawImage(getImage(IMG_ROCK_C + cell), x, y, 0);
                                break;
                        }
                        break;
                    case 32:
                        // cannot stick
                        cell = cell & 31;
                        switch (cell) {
                            case 31:
                                // nothing
                                break;
                            default:
                                g.drawImage(getImage(IMG_GROUND_C + cell), x, y, 0);
                                break;
                        }
                        break;
                    case 64:
                        // with water
                        g.drawImage(getImage(IMG_WATER01 + (((tics & 8) >> 3) + i & 1)),
x, y, 0);
                        cell = cell & 31;
                        switch (cell) {
                            case 31:
                                // nothing
                                break;
                            default:
                                g.drawImage(getImage(IMG_ROCK_C + cell), x, y, 0);
                                break;
                        }
                        break;
                    case 96:
                        // animation object
                        cell = cell & 31;
                        switch (cell) {
                            // finish point
                            case RockmanGame.EXIT:
```

```
                                g.drawImage(getImage(IMG_EXIT01 + (tics & 1)), x + (CELL
>> 1), y + (CELL >> 1), GraphicsME.VCENTER | GraphicsME.HCENTER);
                                break;
                            // spike
                            case RockmanGame.SPIKE:
                                int k = (tics + j) % 5;
                                switch (k) {
                                    case 0:
                                    case 4:
                                        break;
                                    case 1:
                                    case 3:
                                        g.drawImage(getImage(IMG_TORN01), x, y, 0);
                                        break;
                                    case 2:
                                        g.drawImage(getImage(IMG_TORN02), x, y, 0);
                                        break;
                                }
                                break;
                        }
                        break;
                }
                x += CELL;
            }
            y += CELL;
        }
        int k;
        switch (game.y0_state) {
            case RockmanGame.STATE_0_WALK:
                switch (game.direction) {
                    case RockmanGame.DIR_LEFT:
                        k = SPRITE_WALK_L[(tics & 7) >> 1];
                        break;
                    case RockmanGame.DIR_RIGHT:
                        k = SPRITE_WALK_R[(tics & 7) >> 1];
                        break;
                    case RockmanGame.DIR_RIGHT_DOWN:
                        k = SPRITE_WALK_DR[(tics & 7) >> 1];
                        break;
                    case RockmanGame.DIR_RIGHT_UP:
                        k = SPRITE_WALK_UR[(tics & 7) >> 1];
                        break;
                    case RockmanGame.DIR_LEFT_DOWN:
                        k = SPRITE_WALK_DL[(tics & 7) >> 1];
                        break;
                    case RockmanGame.DIR_LEFT_UP:
                        k = SPRITE_WALK_UL[(tics & 7) >> 1];
                        break;
                    case RockmanGame.DIR_STOP_LEFT:
                        k = IMG_M_STAND_L;
                        break;
                    case RockmanGame.DIR_STOP_RIGHT:
                        k = IMG_M_STAND_R;
                        break;
                    default:
                        k = -1;
                }
                g.drawImage(getImage(k), CELL * game.fManX >> 8, CELL * game.fManY >> 8,
GraphicsME.VCENTER | GraphicsME.HCENTER);
                break;
            case RockmanGame.STATE_4_FLY:
                switch (game.direction) {
                    case RockmanGame.DIR_LEFT:
                        k = IMG_M_WALK_UP_L02;
                        break;
                    case RockmanGame.DIR_RIGHT:
```

```
                        k = IMG_M_WALK_UP_R02;
                        break;
                    default:
                        k = -1;
                        break;
                }
                g.drawImage(getImage(k), CELL * game.fManX >> 8, CELL * game.fManY >> 8,
GraphicsME.VCENTER | GraphicsME.HCENTER);
                break;
            case RockmanGame.STATE_1_AIM:
                switch (game.direction) {
                    case RockmanGame.DIR_LEFT:
                        k = IMG_M_STAND_L;
                        break;
                    case RockmanGame.DIR_RIGHT:
                        k = IMG_M_STAND_R;
                        break;
                    default:
                        k = -1;
                        break;
                }
                g.drawImage(getImage(k), CELL * game.fManX >> 8, CELL * game.fManY >> 8,
GraphicsME.VCENTER | GraphicsME.HCENTER);
                //g.setColor(cBlack);
                //g.drawLine(CELL * game.fManX >> 8, CELL * game.fManY >> 8, CELL *
game.fStirvingX >> 8, CELL * game.fStirvingY >> 8);
                g.drawImage(getImage(IMG_CROSS), CELL * game.fStirvingX >> 8, CELL *
game.fStirvingY >> 8, GraphicsME.VCENTER | GraphicsME.HCENTER);
                break;
            case RockmanGame.STATE_2_SHOT:
                switch (game.direction) {
                    case RockmanGame.DIR_LEFT:
                        k = IMG_M_STAND_L;
                        break;
                    case RockmanGame.DIR_RIGHT:
                        k = IMG_M_STAND_R;
                        break;
                    default:
                        k = -1;
                        break;
                }
                g.drawImage(getImage(k), CELL * game.fManX >> 8, CELL * game.fManY >> 8,
GraphicsME.VCENTER | GraphicsME.HCENTER);
                g.setColor(cRopeColor);
                g.drawLine(CELL * game.fManX >> 8, CELL * game.fManY >> 8, CELL *
game.fHarX >> 8, CELL * game.fHarY >> 8);
                //g.drawImage(getImage(IMG_BALL), CELL * game.fHarX >> 8, CELL *
game.fHarY >> 8, GraphicsME.VCENTER | GraphicsME.HCENTER);
                break;
            case RockmanGame.STATE_3_SWING:
                // rope
                g.setColor(cRopeColor);
                for (int i = 0; i < game.hard.size() - 1; i++) {
                    g.drawLine(
                            CELL * ((Hardpoint) game.hard.elementAt(i)).fX >> 8,
                            CELL * ((Hardpoint) game.hard.elementAt(i)).fY >> 8,
                            CELL * ((Hardpoint) game.hard.elementAt(i + 1)).fX >> 8,
                            CELL * ((Hardpoint) game.hard.elementAt(i + 1)).fY >> 8);
                }
                if (!game.hard.empty()) {
                    g.drawLine(
                            CELL * ((Hardpoint) game.hard.elementAt(game.hard.size() -
1)).fX >> 8,
                            CELL * ((Hardpoint) game.hard.elementAt(game.hard.size() -
1)).fY >> 8,
                            CELL * game.fHarX >> 8,
```

```
                            CELL * game.fHarY >> 8);
                }
                g.drawLine(CELL * game.fManX >> 8, CELL * game.fManY >> 8, CELL *
game.fHarX >> 8, CELL * game.fHarY >> 8);
                // man
                switch (game.direction) {
                    case RockmanGame.DIR_LEFT:
                        k = IMG_M_WALK_UP_L02;
                        break;
                    case RockmanGame.DIR_RIGHT:
                        k = IMG_M_WALK_UP_R02;
                        break;
                    case RockmanGame.DIR_UP:
                    case RockmanGame.DIR_DOWN:
                        k = IMG_M_UP01 + (tics & 1);
                        break;
                    default:
                        k = -1;
                        break;
                }
                g.drawImage(getImage(k), CELL * game.fManX >> 8, CELL * game.fManY >> 8,
GraphicsME.VCENTER | GraphicsME.HCENTER);
                break;
            case RockmanGame.STATE_5_BLINK:
                if ((tics & 1) == 0) {
                    switch (game.direction) {
                        case RockmanGame.DIR_LEFT:
                            k = IMG_M_STAND_L;
                            break;
                        case RockmanGame.DIR_RIGHT:
                            k = IMG_M_STAND_R;
                            break;
                        default:
                            k = -1;
                            break;
                    }
                    g.drawImage(getImage(k), CELL * game.fManX >> 8, CELL * game.fManY >>
8, GraphicsME.VCENTER | GraphicsME.HCENTER);
                }
                break;
        }
        // back translate
        g.translate(-tx, -ty);
    }

    /**
     * Индекс чередования спрайтов для ходьбы влево.
     */
    private static final int[] SPRITE_WALK_L = {IMG_M_WALK_L01, IMG_M_WALK_L02,
IMG_M_WALK_L03, IMG_M_WALK_L02};

    /**
     * Индекс чередования спрайтов для ходьбы вправо.
     */
    private static final int[] SPRITE_WALK_R = {IMG_M_WALK_R01, IMG_M_WALK_R02,
IMG_M_WALK_R03, IMG_M_WALK_R02};

    /**
     * Индекс чередования спрайтов для подхема влево.
     */
    private static final int[] SPRITE_WALK_UL = {IMG_M_WALK_UP_L01, IMG_M_WALK_L02,
IMG_M_WALK_UP_L02, IMG_M_WALK_L02};

    /**
     * Индекс чередования спрайтов  для подъема вправо.
     */
```

```
    private static final int[] SPRITE_WALK_UR = {IMG_M_WALK_UP_R01, IMG_M_WALK_R02,
IMG_M_WALK_UP_R02, IMG_M_WALK_R02};

    /**
     * Индекс чередования спрайтов для спуска влево.
     */
    private static final int[] SPRITE_WALK_DL = {IMG_M_WALK_DOWN_L01, IMG_M_WALK_L02,
IMG_M_WALK_DOWN_L02, IMG_M_WALK_L02};

    /**
     * Индекс чередования спрайтов для спуска вправо.
     */
    private static final int[] SPRITE_WALK_DR = {IMG_M_WALK_DOWN_R01, IMG_M_WALK_R02,
IMG_M_WALK_DOWN_R02, IMG_M_WALK_R02};

    /** @link dependency */
    /*# GraphicsME lnkGraphicsME; */
}
```

Класс Log

```
package rockman;

import java.io.FileWriter;
import java.io.IOException;
import java.io.BufferedWriter;
import java.io.PrintWriter;

/**
 * Класс поддержки файла протокола.
 */
public class Log {
    /**
     * Выходной поток.
     */
    private PrintWriter out;

    /**
     * Конструктор.
     * @param filename Имя файла протокола.
     */
    public Log(String filename) {
        try {
            out = new PrintWriter(new BufferedWriter(new FileWriter(filename)));
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }

    /**
     * Протоколирует выходное воздействие.
     * @param n Номер выходного воздействия.
     * @param str Название выходного воздействия.
     */
    public void action(int n, String str) {
        out.println("* z" + n + " - " + str);
        out.flush();
    }

    /**
     * Протоколирует проверку входного параметра с заданным номером и значением.
     * @param n Номер входного параметра.
     * @param result Значение входного параметра.
     * @param str Название входного параметра.
```

```
     */
    public void input(int n, boolean result, String str) {
        String rusRes;
        if (result)
            rusRes = "истина";
        else
            rusRes = "ложь";
        out.println("+ x" + n + " - " + str + ". Значение - " + rusRes);
        out.flush();
    }

    /**
     * Протоколирует событие.
     * @param n Номер события.
     * @param str Название события.
     */
    public void event(int n, String str) {
        out.println("% e" + n + " - " + str);
        out.flush();
    }

    /**
     * Протоколирует смену автоматом состояния.
     * @param a Номер автомата.
     * @param yOld Старое состояние.
     * @param yNew Новое состояние.
     * @param str Название состояния.
     */
    public void switchState(int a, int yOld, int yNew, String str) {
        out.println("# A" + a + " перешел из состояния " + yOld + " в состояние " + yNew
+ " : " + str);
        out.flush();
    }

    /**
     * Протоколирует програмные сообщения.
     * @param str Програмное сообщение.
     */
    public void programMes(String str) {
        out.println("! " + str);
        out.flush();
    }

    /**
     * Протоколирует начало работы автомата.
     * @param n Номер автомата.
     * @param state Начальное состояние автомата.
     */
    public void beginAutomataWork(int n, int state) {
        out.println("{ A" + n + " начал работу в состоянии " + state);
        out.flush();
    }

    /**
     * Протоколирует завершение работы автомата.
     * @param n Номер автомата.
     * @param state Конечное состояние автомата.
     */
    public void endAutomataWork(int n, int state) {
        out.println("} A" + n + " закончил работу в состоянии " + state);
        out.flush();
    }
```

```
    /**
     * Закрывает файл протокола.
     */
    public void close() {
        out.close();
    }
}
```

Приложение 3. Фрагмент документации *JavaDoc*

Package **Class** **Tree** **Index** **Help**

PREV CLASS NEXT CLASS | FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

---

rockman
Class Log

```
java.lang.Object
  └ rockman.Log
```

---

public class **Log**
extends java.lang.Object

Класс поддержки файла протокола.

Field Summary

private java.io.PrintWriter	**out** Выходной поток.

Constructor Summary

Log(java.lang.String filename) Конструктор.

Method Summary

void	**action**(int n, java.lang.String str) Протоколирует выходное воздействие.
void	**beginAutomataWork**(int n, int state) Протоколирует начало работы автомата.
void	**close**() Закрывает файл протокола.
void	**endAutomataWork**(int n, int state) Протоколирует завершение работы автомата.
void	**event**(int n, java.lang.String str) Протоколирует событие.
void	**input**(int n, boolean result, java.lang.String str) Протоколирует проверку входного параметра с заданным номером и значением.
void	**programMes**(java.lang.String str) Протоколирует програмные сообщения.
void	**switchState**(int a, int yOld, int yNew, java.lang.String str) Протоколирует смену автоматом состояния.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

out

```
private java.io.PrintWriter out
```

Выходной поток.

Constructor Detail

Log

```
public Log(java.lang.String filename)
```

Конструктор.

Parameters:

`filename` - Имя файла протокола.

Method Detail

action

```
public void action(int n,
                   java.lang.String str)
```

Протоколирует выходное воздействие.

Parameters:

`n` - Номер выходного воздействия.

`str` - Название выходного воздействия

beginAutomataWork

```
public void beginAutomataWork(int n,
                              int state)
```

Протоколирует начало работы автомата.

Parameters:

`n` - Номер автомата.

`state` - Начальное состояние автомата.

---

close

```
public void close()
```

Закрывает файл протокола.

---

endAutomataWork

```
public void endAutomataWork(int n,
                            int state)
```

Протоколирует завершение работы автомата.

Parameters:

`n` - Номер автомата.

`state` - Конечное состояние автомата.

---

event

```
public void event(int n,
                  java.lang.String str)
```

Протоколирует событие.

Parameters:

`n` - Номер события.

`str` - Название события

---

input

```
public void input(int n,
                  boolean result,
                  java.lang.String str)
```

Протоколирует проверку входного параметра с заданным номером и значением.

Parameters:

`n` - Номер входного параметра.

`result` - Значение входного параметра.

`str` - Название входного параметра

---

programMes

```
public void programMes(java.lang.String str)
```

Протоколирует програмные сообщения.

Parameters:

`str` - Програмное сообщение.

---

switchState

```
public void switchState(int a,
                        int yOld,
                        int yNew,
                        java.lang.String str)
```

Протоколирует смену автоматом состояния.

Parameters:
`a` - Номер автомата.
`yOld` - Старое состояние.
`yNew` - Новое состояние.
`str` - Название состояния.
