Статья опубликована в журнале «Компьютерные инструменты в образовании». 2004.

№ 4, с. 75-84.

А. В. Беляев, Д. И. Суясов, А. А. Шалыто

Компьютерная игра «Космонавт».

Проектирование и реализация

В настоящее время наблюдается бум в области создания компьютерных игр. Однако, несмотря на наличие отдельных работ [1], раскрывающих секреты их создания, эта область остается тайной за семью печатями. В настоящей работе на примере создания сравнительно простой игры излагается метод ее проектирования и реализации.

Введение

В настоящей работе описывается использование автоматного подхода [2] на примере создания простой компьютерной игры. 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 подхода.

Программа иллюстрирует предложенное в работе [2] 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 [3,4] состояния на указанные типы не разделяются, а под состояниями понимаются значения ячеек памяти, число которых огромно.

Предложенное решение весьма близко к подходу, развиваемому при построении гибридных динамических систем [5].

В работе 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.

1. Постановка задачи

Цель настоящей работы написать игру «Космонавт». Космонавт находится в пещере на неизвестной планете. Он должен найти выход из нее, который обозначим мигающим желто-красным шаром. При этом пещера имеет сложный рельеф, включающий возвышенности, пропасти, опасные для жизни космонавта шипы и водоемы. Космонавт может ходить влево и вправо, забираться на наклонные поверхности, если они не очень крутые. Он имеет гарпун с веревкой, который может быть воткнут в некоторые из поверхностей пещеры. Другие поверхности слишком твердые и гарпун в них не втыкается. Закрепление гарпуна на поверхности обеспечивает возможность подтягивания космонавта по веревке, прикрепленной к гарпуну.

При этом космонавт может погибнуть при соприкосновении с шипами и водой, а также при падении с большой высоты. Смерть обозначается миганием космонавта. После смерти уровень необходимо попытаться пройти заново.

В игре должна быть обеспечена реалистичность перемещений космонавта.

Должна быть предусмотрена возможность сохранения состояния игры и загрузки этого состояния. Сохранение и последующая загрузка бывают необходимы, для того, чтобы в случае смерти не начинать уровень заново, а попытаться преодолеть препятствие вновь. Кроме того, эти функции удобны для обеспечения перерыва в процессе игры и дальнейшего ее продолжения.

Игра имеет 11 уровней, которые отличаются сложностью ландшафта. Переход на следующий уровень осуществляется автоматически при достижении выхода из пещеры. Игрок последовательно проходит все уровни, и по завершении последнего из них игра начинается сначала.

2. Интерфейс пользователя

Пример внешнего вида окна программы при прохождении некоторого уровня приведен на рис.1.

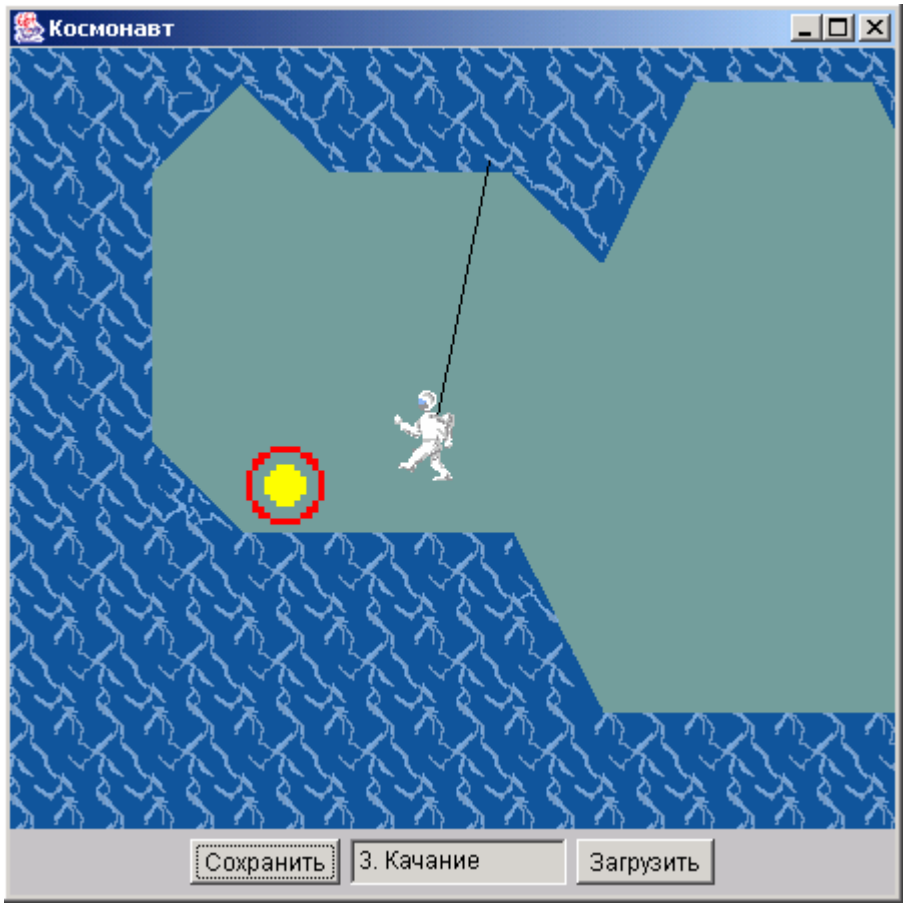


Рис. 1. Пример окна программы

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 номер и название текущего состояния автомата, управляющего игрой.

В начале игры необходимо убедиться, что индикатор «Num Lock» клавиатуры включен. Для обеспечения ходьбы используются кнопки «4←» и «6→» на цифровой клавиатуре. Переход из состояния ходьбы в состояние прицеливания должна быть нажата кнопка «5» цифровой клавиатуры. При повторном нажатии на эту кнопку космонавт выстрелит 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 «2↓» и «8↑». Очередное нажатие кнопки «5» приводит к тому, что космонавт отпускает веревку и переходит «в свободный полет».

3. Архитектура программы

Программа реализована на основе паттерна проектирования *MVC (Model – View – Controller)* [5, 6]. В проекте в качестве *Модели* используется класс `RockmanGame`, в качестве *Представления* – `RockmanGameView`, 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 `RockmanGameView`, основываясь на данных математической модели, обеспечивает ее представление на экране. Главный класс программы `Main` агрегирует классы `RockmanGame` и `RockmanGameView`. Он является посредником между моделью, ее представлением и пользователем.

Диаграмма классов программы представлена на рис. 2.

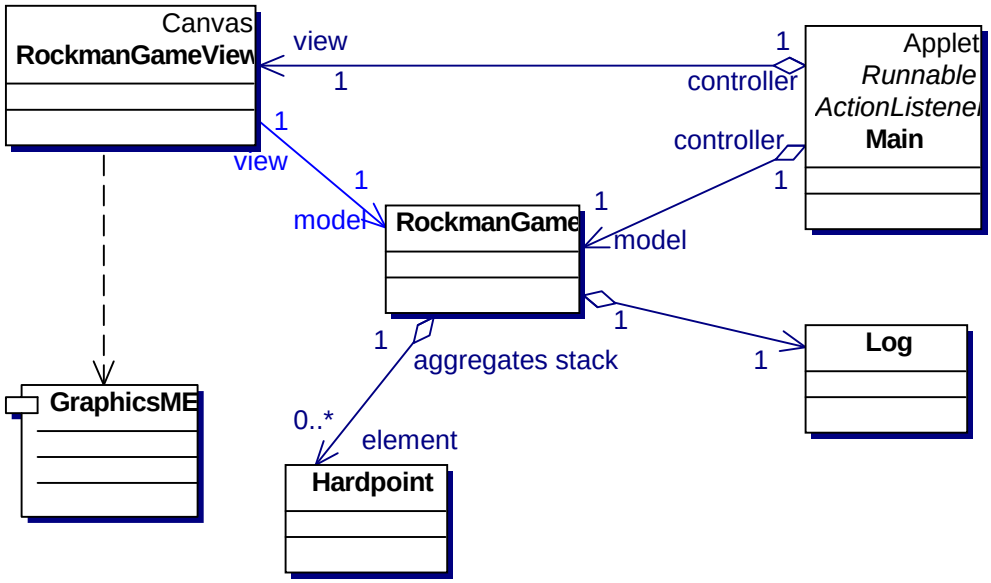


Рис. 2. Диаграмма классов

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` (принадлежащего стандартной библиотеке) для файлового вывода протокола, а также реализует несколько простых методов для его ведения. В терминологии паттернов отношение классов `Log` и `FileOutputStream` называется *Facade* [6].

Класс `RockmanGameView` наследуется от стандартного класса `JCanvas` платформы *Java*. Он отображает состояние математической модели, используя набор картинок. Для более удобного вывода на экран спрайтов (спрайт - совокупность картинок) класс `RockmanGameView` использует класс `GraphicsME` вместо стандартного контекста `Graphics`. Класс `GraphicsME` (рис. 1) содержит стандартный класс `Graphics` и предоставляет методы, более удобные для отображения спрайтов. Это еще один пример реализации паттерна *Facade*.

4. Автомат

Программа содержит один автомат, реализующий логику игры. Его описание и реализация выполнены на основе работы [8].

4.1. Нумерация и перечень состояний

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

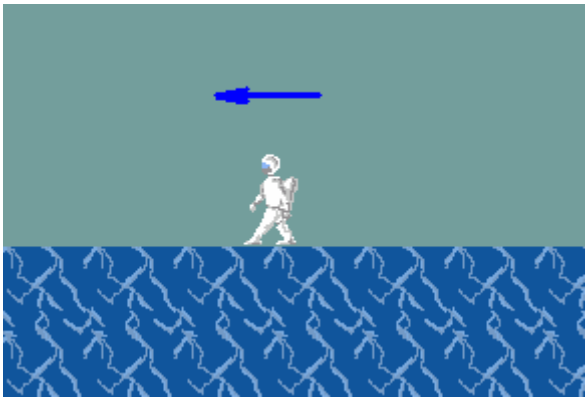


Рис. 3. Состояние 0. Ходьба

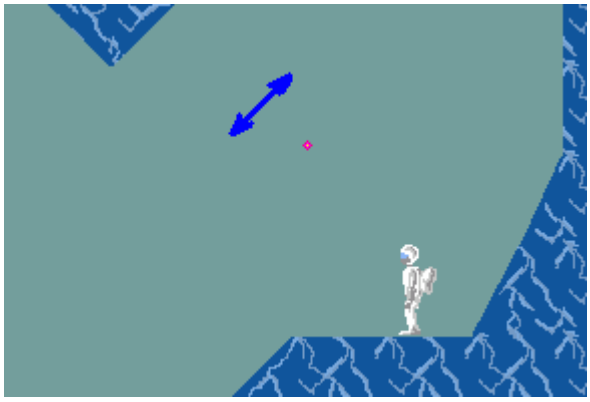


Рис. 4. Состояние 1. Прицеливание

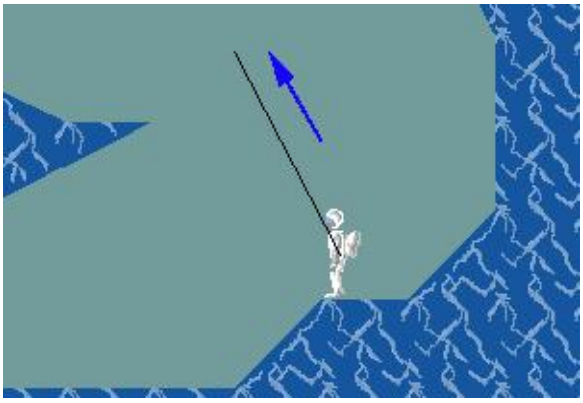


Рис. 5. Состояние 2. Стрельба

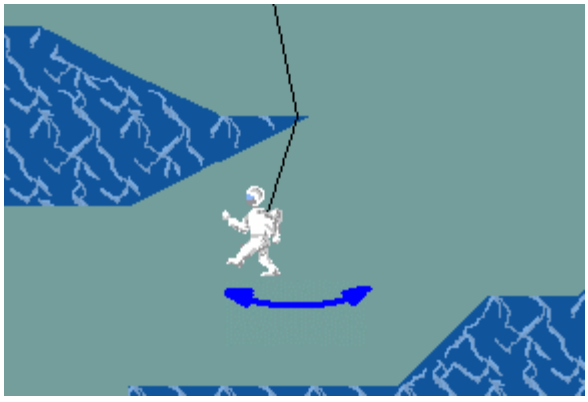


Рис. 6. Состояние 3. Качание

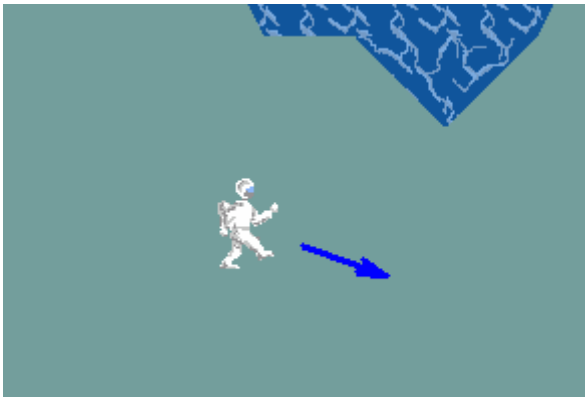


Рис. 7. Состояние 4. Полет

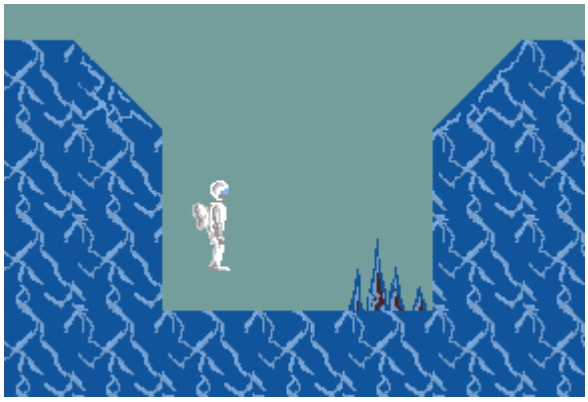
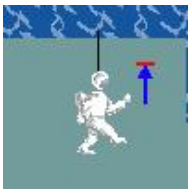


Рис. 8. Состояние 5. Смерть

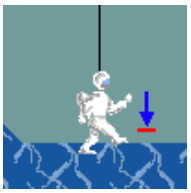
4.2. Нумерация и перечень событий

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↓»

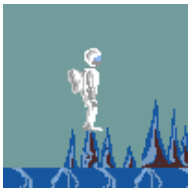
4.3. Перечень входных переменных и их описание



X0
Прикосновение головы с потолком при укорачивании веревки.



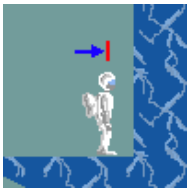
X1
Прикосновение ног с полом при удлинении веревки.



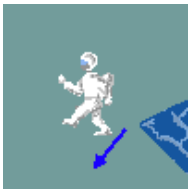
X2
Смертельный участок.

X3
Космонавт на выходе.

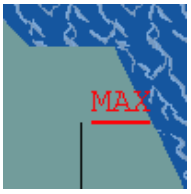
X4
Прикосновение к стене слева.



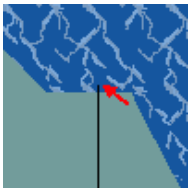
X5
Прикосновение к стене справа.



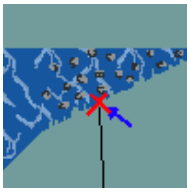
X6
Потеря опоры под ногами.



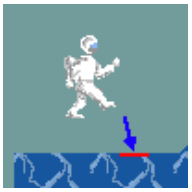
X7
Не хватает длины веревки.



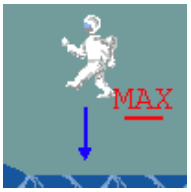
X8
Гарпун ударился в «хорошую» скалу.



X9
Гарпун ударился о «плохую» скалу.



X10
Космонавт приземлился после падения.



X11
Высота падения смертельна.

Входную переменную не описать картинкой.

X12
Помигал достаточно.

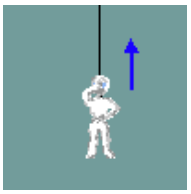
4.4. Нумерация и перечень выходных воздействий

Воздействие не описать картинкой.

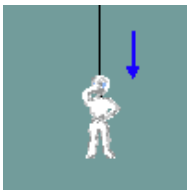
Z0
Расчет некоторых параметров при качании с нажатой кнопкой «Влево».

Воздействие не описать картинкой.

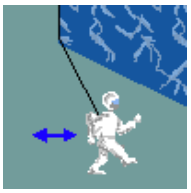
Z1
Расчет некоторых параметров при качании с нажатой кнопкой «Вправо».



Z2
Подъем по веревке. Длина веревки уменьшается.



Z3
Спуск по веверке. Длина веревки увеличивается.



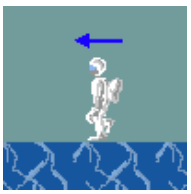
Z4
Расчет координат, скорости, ускорения при раскачке.

Воздействие не описать картинкой.

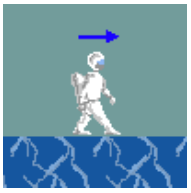
Z5
Вычисления для перехода в состояние «Полет» из состояния «Качание».

Воздействие не описать картинкой.

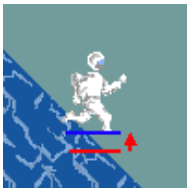
Z6
Инициализация состояния «Смерть».



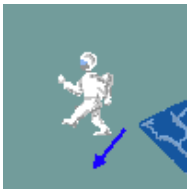
Z7
Сдвиг координат космонавта при шаге влево.



Z8
Сдвиг координат космонавта при шаге вправо.



Z9
Стабилизация, чтобы его ноги оказались на поверхности.



Z10
Инициализация состояния «Полет» после падения влево из состояния «Ходьбы».

Z11
Инициализация состояния «Полет» после падения вправо из состояния «Ходьбы».

Z12
Устанавливает нужный спрайт при ходьбе вправо в зависимости от наклона поверхности.

Z13
Устанавливает нужный спрайт при ходьбе влево в зависимости от наклона поверхности.

Z14
Устанавливает спрайт при покое космонавта.

Воздействие не описать картинкой.

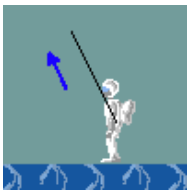
Z15
Инициализация состояния «Прицеливание».

Z16
Поворот прицела по часовой стрелки.

Z17
Поворот прицела против часовой стрелки

Воздействие не описать картинкой.

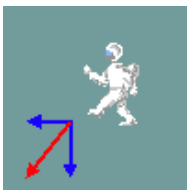
Z18
Инициализация состояния «Выстрел».



Z19
Удлиняет веревку в состоянии «Выстрел».

Воздействие не описать картинкой.

Z20
Инициализация состояния «Качание».



Z21
Рассчет физики полета.

Z22
Увеличение счетчика миганий 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.

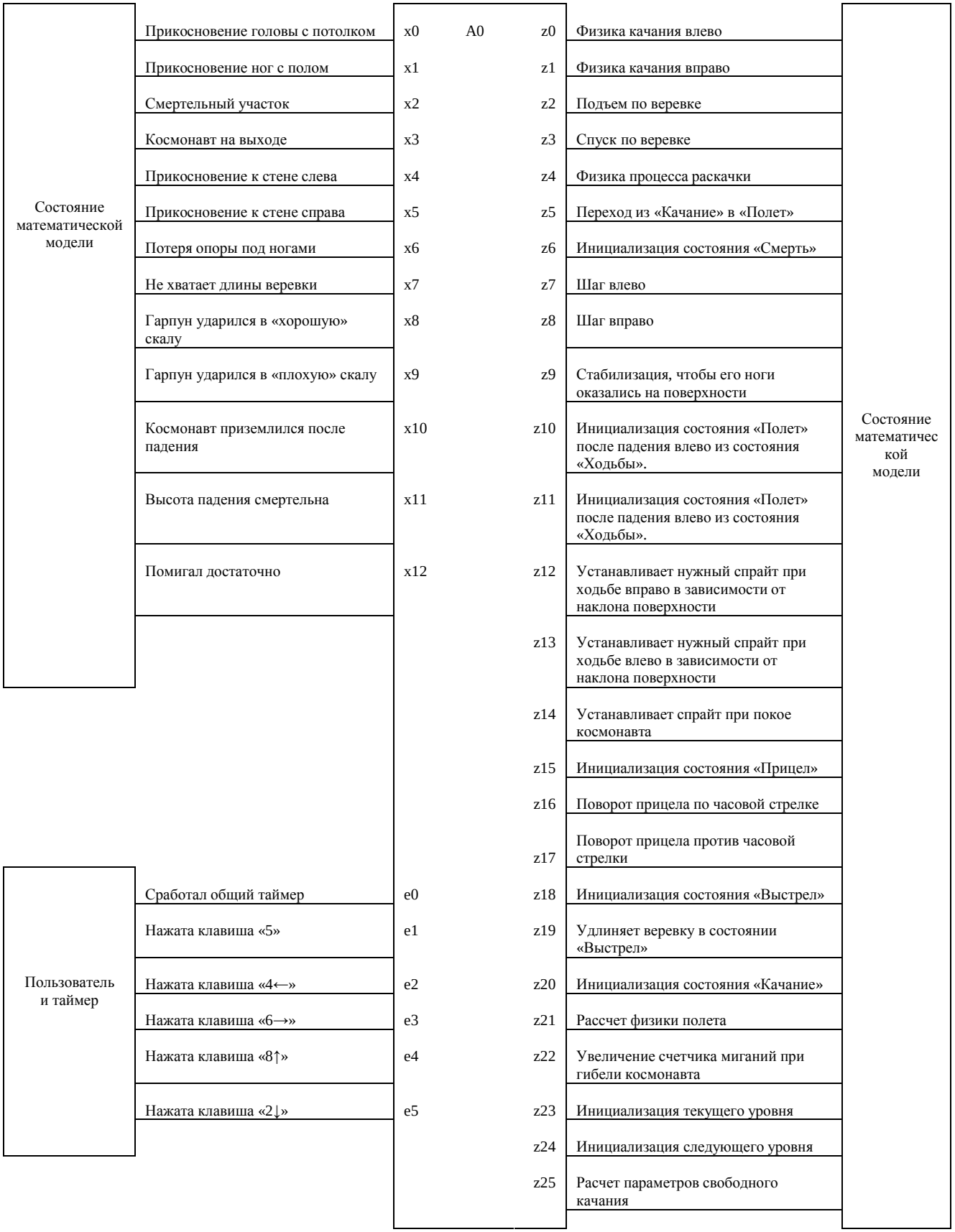


Рис. 9. Схема связей автомата

4.6. Граф переходов

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

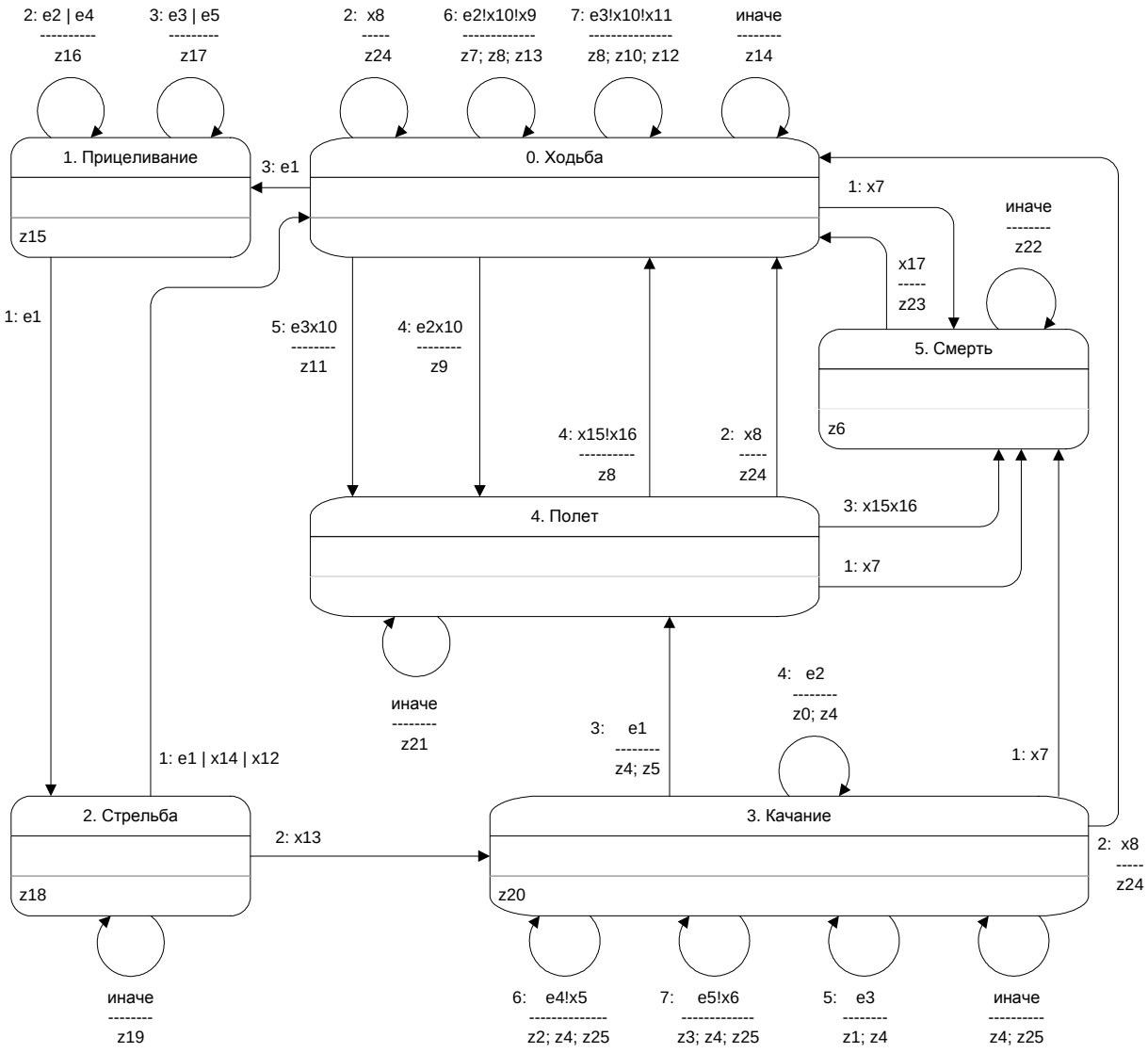


Рис. 10. Граф переходов автомата

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

Рассчеты координат, скорости, углового ускорения космонавта в состоянии «Качание» реализованы 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.

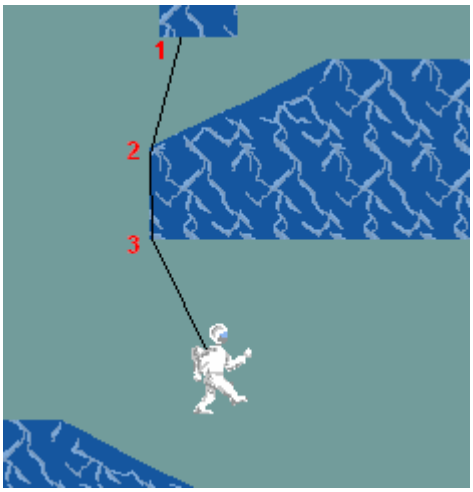


Рис. 11. Качание

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.

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

Проектная документация, исходный код программы, пример протокола ее работы, *JavaDoc*-документация преведены в соответствующем проекте на сайте http://is.ifmo.ru , раздел «Проекты». Отметим, что *JavaDoc*-документация не заменяет проектную документацию, а лишь дополняет ее, что не является общепринятым.

Литература

1. *Ла Мот А., Ратклифф Д., Суминаторе М. и др.* Секреты программирования игр. СПб.: Питер, 1995.
2. *Шалыто А.А., Туккель Н.И.* От тьюрингова программирования к автоматному // Мир ПК. 2002. № 2.
3. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++. М.: Бином; СПб.: Невский диалект, 1998.
4. *Лавров С.* Программирование. Математические основы, средства, теория. СПб.: БХВ, 2002.
5. ***Benveniste A., Le Guernic P.*** Hybrid Dinamical System Theory and the Singal Language //IEEE Trans. on Automatic Control. 1990, vol. 35, № 5.
6. ***Stelting S., Maassen O.*** Applied Java Patterns. NJ: Prentice Hall. 2001.
7. ***Cooper J.*** The Design Patterns Java Companion. NY: Addison-Wesley. 1998.
8. *Туккель Н.И., Шалыто А.А.* Система управления дизель-генератором. Программирование с явным выделением состояний. (http://is.ifmo.ru, раздел «Проекты»).

Об авторах

Беляев Антон Васильевич – студент кафедры «Компьютерные технологии» Санкт-Петербургского государственного университета информационных технологий, механики и оптики (СПбГУ ИТМО). E-mail: bell@rain.ifmo.ru

Суясов Дмитрий Игоревич – студент кафедры «Компьютерные технологии» СПбГУ ИТМО. E-mail: suyasov@rain.ifmo.ru

Шалыто Анатолий Абрамович – докт. техн. наук, профессор, заведующий кафедрой «Технологии программирования» СПбГУ ИТМО. E-mail: shalyto@mail.ifmo.ru