Статья опубликована в методической газете для учителей информатики "Информатика". 2006. № 11 (516), с. 36 – 47.

Вариант статьи опубликован ранее в журнале "Компьютерные инструменты в образовании". 2003. № 4, с.39 – 47.

# Анимация. Flash-технология. Автоматы

**М. А. Мазин**, **А. А. Шалыто**

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

## 1. Введение

В настоящее время интерактивная анимация широко используется в различных приложениях, таких как веб-дизайн, электронные обучающие пособия и т.д.

Создание такой анимации возможно при непосредственном программировании на различных языках, например, *C++* или *Java*. В этой области традиционно применяются графические библиотеки, из которых наиболее распространены *OpenGL* [1] и *DirectX* [2]. Первая характеризуется независимостью от платформы, а вторая, кроме средств работы с графикой, предоставляет также широкий спектр других средств разработки мультимедийных приложений.

Существуют средства, позволяющие ускорить процесс создания анимации за счет использования редакторов трехмерной графики, например, пакет *3D-Max* [3]. Обычно такие пакеты применяются совместно с указанными выше графическими библиотеками.

Однако если необходимо разрабатывать приложения для сети Интернет, то наиболее предпочтительным оказывается использование пакета *Macromedia Flash* [4], являющегося основой *Flash-технологии*. Большая популярность этой технологии в мире [5] объясняется рядом ее достоинств, среди которых следует отметить ее поддержку большинством современных браузеров и компактность исполняемого файла.

Как правило, содержание анимации задается ее сценарием. Различают линейные и нелинейные сценарии.

Если для реализации *линейных сценариев*, в которых отсутствуют разветвления, достаточно средств встроенного редактора *Macromedia Flash*, то для *нелинейных сценариев*, отличающихся наличием разветвлений, дополнительно применяется язык *ActionScript* [6]. В интерактивной анимации, в отличие от мультфильмов, обычно используются нелинейные сценарии.

Язык *ActionScript* в основном применяется для реализации достаточно простых нелинейных сценариев. В более сложных случаях его традиционное применение затруднительно. Так, например, интерпретатор не сообщает об использовании в выражениях не инициализированных переменных, что приводит к тому, что ошибку в имени переменной можно искать часами.

Авторами предлагается применять автоматную парадигму в для формализации перехода от заданного сценария к коду программы на языке *ActionScript.* Это позволяет устранить указанные недостатки.

## 2. Разработка интерактивной анимации

Разработка анимации предполагает решение четырех подзадач:

- создание сценария;
- формализация сценария;

- «рисование»;
- программирование.

Решать указанные подзадачи будем на основе *SWITCH-технологии* [8, 9], базирующейся на парадигме автоматного программирования, которая называется также программирование с явным выделением состояний.

По вербальному заданию (тексту сценария) строится его математическая модель в виде графа переходов конечного автомата. Для этого сюжет разбивается на сцены — статические изображения. В каждой сцене присутствуют элементы управления, с помощью которых пользователь осуществляет ввод данных. Каждая сцена определяет состояние автомата, в котором указанное изображение формируется как выходное действие.

При этом пользовательский ввод генерирует входные переменные и порождает события, с которыми вызывается автомат. В зависимости от типа события и от значений входных переменных, вызываемый автомат может осуществлять переходы.

Анимация, связанная с задачей, составляет совокупность дополнительных выходных действий, которые выполняются при переходах автомата.

Кроме того, при переходах выполняются выходные действия, связанные с управлением средой исполнения, например, таким действием является завершение работы флэш-плеера.

Изложенное позволяет построить схему связей автомата, которая описывает его интерфейс. При этом автомат управляет статическими изображениями, анимацией и средой исполнения.

По построенному графу переходов формально и изоморфно строится текст подпрограммы, который реализует его на основе конструкции *switch* языка *ActionScript*. Функции входных переменных и выходных действий на этом этапе заменяются функциями-заглушками. Обработчики событий выделяют из потока обрабатываемые события и вызывают с этими событиями автомат, который в зависимости от своего текущего состояния и значений входных переменных может осуществлять переходы и выходные действия.

После реализации графа переходов программист пишет программные модули, соответствующие входным переменным, обработчикам событий и выходным действиям. В то же время, руководствуясь схемой связей автомата, художник «рисует» статические изображения и выполняет раскадровку анимации.

Тем самым создается программа, состоящая из обработчиков событий, конструкции *switch* и функций, вызываемых из нее.

Отладка программы в рамках предлагаемого подхода осуществляется с помощью протоколов (логов), для построения которых, вводятся функции протоколирования, вызываемые из конструкции *switch*, функций выходных действий, входных переменных и обработчиков событий.

## 3. Пример

Рассмотрим предлагаемый процесс на примере разработки интерактивной задачи по физике. В качестве основы для сценария используем упражнение из задачника [7]: «Ученик заметил, что палка длиной 1.2 м, поставленная вертикально, отбрасывает тень длиной 0.8 м. А длина тени от дерева оказалась в это же время равной 9.6 м. Какова высота дерева?»

Создадим по формулировке задачи сценарий. Так как в древней Греции геометрическая оптика получила особенное развитие, в сценарии будут использованы древнегреческие образы: Старец в тоге, древнегреческий храм, кипарис, высоту которого измеряем в задаче, и Гелиос на колеснице.

Приведем неформальный текст сценария.

*Ландшафт, на заднем плане – дерево. Стоит Старец. Появляется надпись: «Как можно измерить высоту дерева, используя законы геометрической оптики?»*

*Старец берет палку, втыкает ее в землю. Появляется надпись «Запиши в тетрадь значение длины палки». Рядом с палкой появляется линейка. Крупные метки на линейке позволяют зафиксировать ее длину. Метка, соответствующая длине палки, окрашивается в другой цвет.*

*В тетради (на пергаменте) появляется надпись «Длина палки h = _____ локтей». Тетрадь выполнена из пергаментных листов. Шрифт надписи должен быть стилизован. Если пользователь ввел неверный ответ, то появляется надпись: «Неверно! Попробуй еще раз».*

*При верном ответе. Старец обращает внимание ученика на тени. Это выражается в том, что он показывает рукой на тени и предлагает измерить длину теней – от дерева и от палки.*

*Появляется надпись: «Посмотри на тени. Измерь и занеси их длины в тетрадь».*

*В тетради появляются надписи: «Длина тени палки l = _____ локтей», «Длина тени дерева = _____ локтей». Если пользователь ввел неверные длины теней, то появляется надпись «Измерь точнее длины теней».*

*Когда пользователь ввел верный ответ, появляется надпись: «Теперь у тебя есть все данные, чтобы вычислить высоту дерева». В тетради появляется окно для ввода ответа «Высота дерева H = _____ локтей». Если введен неправильный ответ, то появляется подсказка.*

*Подсказка: наблюдаем геометрическое построение (на пергаменте) – два подобных треугольника. Вертикальные стороны этих треугольников составлены для большего треугольника – из дерева, а для меньшего – из палки. Появляется надпись: «Неверно! Посмотри чертеж. Обрати внимание, что лучи от солнца параллельны».*

*Если пользователь опять вводит неправильное число, то выводится решение – описанный чертеж и вывод формулы для вычисления высоты дерева. Возникает надпись: «В решении этой задачи используется правило подобия треугольников».*

Перейдем к формализации сценария. Первоначально введем элементы управления, а также выделим входные переменные, события и выходные действия.

Для управления анимацией введем следующие кнопки и поля:

- кнопка «Выход» — по ее нажатию флэш-плеер прекращает работу;
- кнопка «Следующая задача» — по ее нажатию начинается загрузка следующей задачи. В примере рассматривается одна задача, и поэтому следующей задачей также является рассматриваемая;
- кнопка «Повторить задачу» — по ее нажатию задача запускается с новыми значениями величин, фигурирующих в ее условии, например, новыми длиной палки и высотой дерева;
- четыре текстовых поля для ввода длины палки, длин теней палки и дерева, а также высоты дерева. В тексте сценария полям соответствуют подчеркнутые пробелы;
- кнопка «Готово» — пользователь нажимает эту кнопку после окончания ввода ответа в соответствующие поля. По ее нажатию проверяется правильность введенных значений;
- кнопка «Дальше» — ее нажатие инициирует переход от одной статической сцены к другой.

Несмотря на то, что при запуске автомат уже находится в начальном состоянии, его необходимо инициализировать событием *e0* для того, чтобы автомат смог сформировать статическое изображение в начальном состоянии.

Во время загрузки файла с анимацией, для комфорта пользователя, на экран выводится «предзагрузчик» (короткая циклическая анимация) — изображение Гелиоса, пересекающего небосклон.

Выделим сцены, каждой из которых присвоим номер, соответствующий вершине графа переходов. Переходы обозначим двойным номером *(i, j)*, где *‘i’* – номер сцены, из которой

осуществляется переход, а *‘j’* – сцена, в которую осуществляется переход. Введем также обозначение событий *‘e’*, инициирующих переходы. Входные переменные обозначим *‘x’*, если условием перехода является истинное значение переменной, и *‘!x’* – в противном случае. Кроме того, введем обозначения выходных действий, которые могут быть трех типов: статические изображения *‘zs’*, анимация *‘za’* и системные действия *‘z’*. При этом статические изображения формируются в вершинах, а анимация и системные действия — на переходах.

Приведем формальный текст сценария.

*0. Предзагрузчик: Гелиос пересекает небосклон на колеснице (zs0).*

*(0, 1) По завершению загрузки (e3) появляется надпись: «Как можно измерить высоту дерева, используя законы геометрической оптики?» (za0), происходит инициализация значений начальных условий задачи (z2).*

*1. Ландшафт, на заднем плане – дерево. Стоит Старец (zs1).*

*(1, 2). По нажатию кнопки «Дальше» (e2) Старец берет палку и втыкает ее в землю. Появляется надпись: «Запиши в тетрадь значение длины палки» (za1).*

*2. Рядом с палкой появляется линейка. Крупные метки на линейке позволяют зафиксировать ее длину. Метка, соответствующая длине палки, окрашивается в красный цвет. В тетради (на пергаменте) появляется надпись «Длина палки h = _____ локтей». Шрифт надписи стилизован — используется шрифт «капитальное письмо» (zs2).*

*(2, 2). Если пользователь ввел (e1) неверный ответ (!x1), то появляется надпись: «Неверно! Попробуй еще раз», Старец мотает головой (za3). Ответ вводится в поле «Длина палки h = ___». Для подтверждения ввода пользователь нажимает кнопку «Готово».*

*(2, 3). В случае ввода (e1) верного ответа (x1) появляется надпись: «Посмотри на тени с линейками. Измерь и занеси их длины в тетрадь» (za2).*

*3. Старец в тоге обращает внимание пользователя на тени. Это выражается в том, что он показывает рукой на тени и предлагает измерить длину теней – от дерева и от палки. В тетради появляются надписи: «Длина тени палки l = _____ локтей», «Длина тени дерева L = _____ локтей» (zs3).*

*(3, 3). Если пользователь ввел (e1) неверный ответ — ошибочные значения длин теней (!x2), то появляется надпись «Измерь точнее длины теней» (za4).*

*(3, 4). Когда пользователь ввел (e1) верный ответ (x2), появляется надпись: «Теперь у тебя есть все данные, чтобы вычислить высоту дерева», Старец кивает (za5).*

*4. В тетради возникает окно для ввода ответа «Высота дерева H = _____ локтей» (zs4).*

*(4, 5). В случае ввода (e1) неправильного ответа — высота дерева введена ошибочно (!x3), появляется надпись «Неверно! Посмотри чертеж. Обрати внимание, что лучи от солнца параллельны» (za6).*

*(4, 7). Если пользователь ввел (e1) верный ответ (x3), появляется надпись об успешном решении задачи: «Задача решена», Старец поднимает руку (za7).*

*5. Подсказка: «Чертеж на пергаменте – два подобных прямоугольных треугольника. Катет одного из треугольников составлен из дерева, а второго — из палки (zs5).*

*(5, 6). Если опять вводится (e1) неверный ответ (!x3), то появляется надпись: «В решении этой задачи используется правило подобия треугольников». Старец сердит (za8).*

*(5, 7). Если пользователь ввел (e1) верный ответ (x3) после подсказки, появляется надпись об успешном решении задачи. Старец поднимает руку (za7).*

*6. Выводится верное решение. Оно представляет собой описанный выше чертеж с треугольниками, но дополнительно к нему приводится процесс получения ответа. Возникает кнопка «Дальше» (zs6).*

*(6, 7) Нажав на кнопку «Дальше» (e2), переходим к заключительной сцене.*

*7. Заключительная сцена: «Ответ получен». Старец улыбается. Появляются кнопки «Следующая задача», «Повторить задачу» (zs7).*

*(7, 0) Нажатие на кнопку «Следующая задача» (e5) приводит к выгрузке текущей задачи и загрузке следующей (z1).*

*(7, 1) По нажатию на кнопку «Повторить задачу» (e6) генерируются новые значения начальных условий (z2). Вновь выводится надпись: «Как можно измерить высоту дерева, используя законы геометрической оптики?» (za0).*

**Замечание.** На экране постоянно присутствует кнопка «Выход», нажатие на которую *(e4)* завершает работу флэш-плеера.

В качестве примера приведем изображение, формируемое выходным действием *zs5* в состоянии 5 и выходным действием (*za6*) при переходе из состояния 4 в состояние 5 (рис. 1).

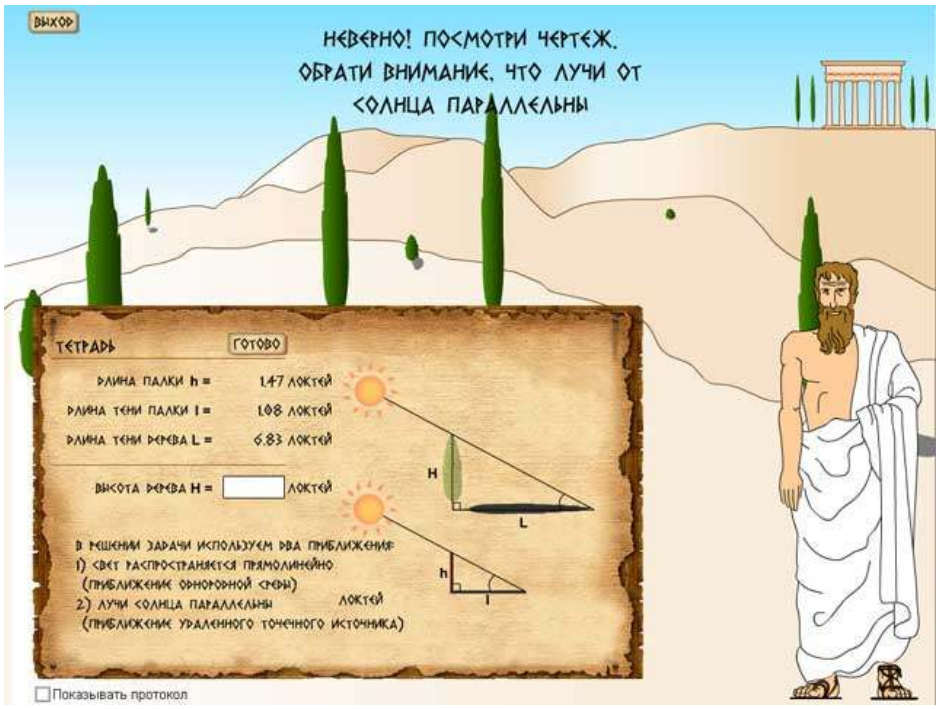


Рис. 1. Изображение после перехода из состояния 4 в состояние 5

Для написания программы проведем дальнейшую формализацию задачи. Построим схему связей автомата, определяющую его интерфейс (рис. 2).

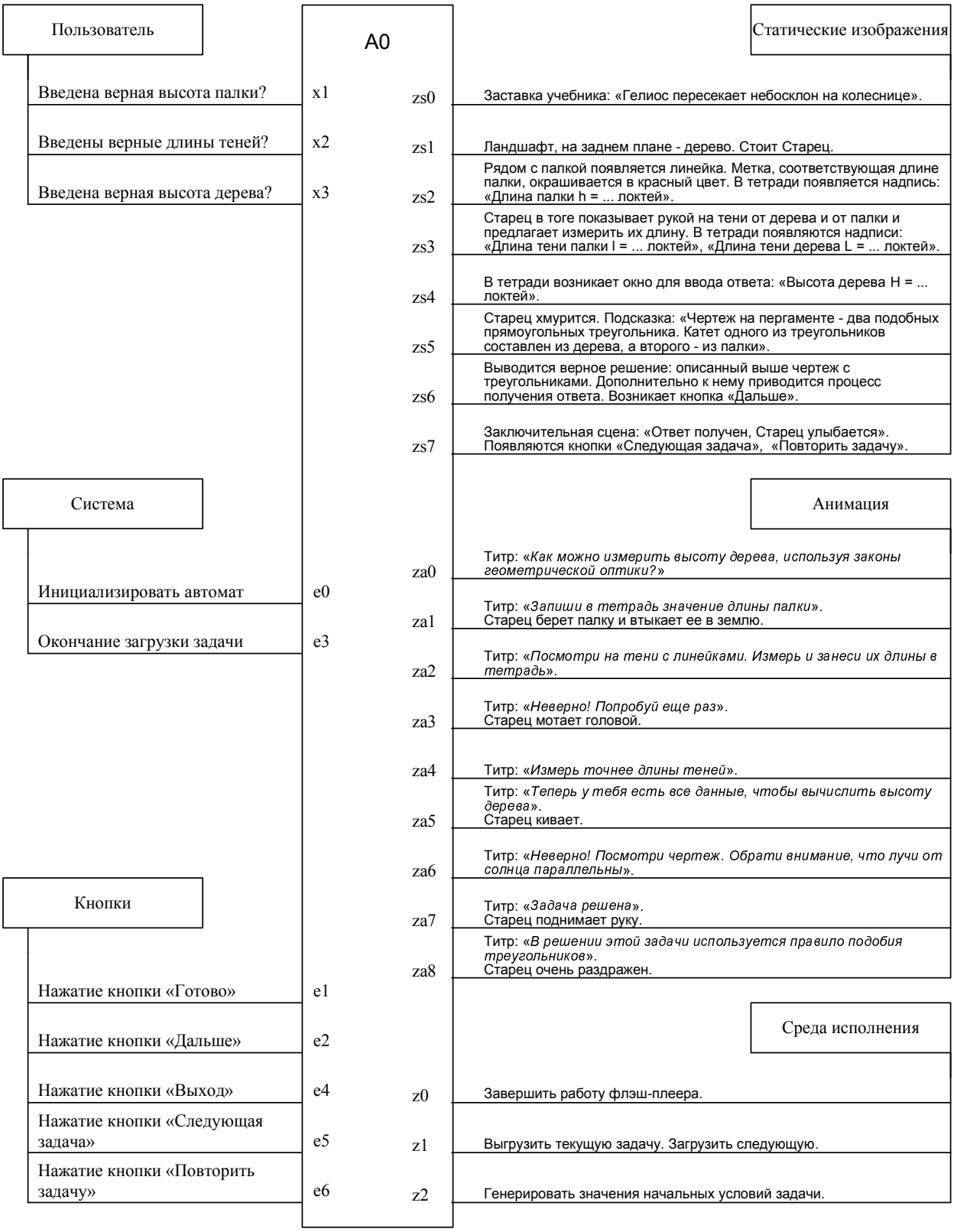


Рис. 2. Схема связей, описывающая интерфейс автомата

Граф переходов, построенный с помощью шаблона (*stencil*), описанного в работе [10], приведен на рис. 3.

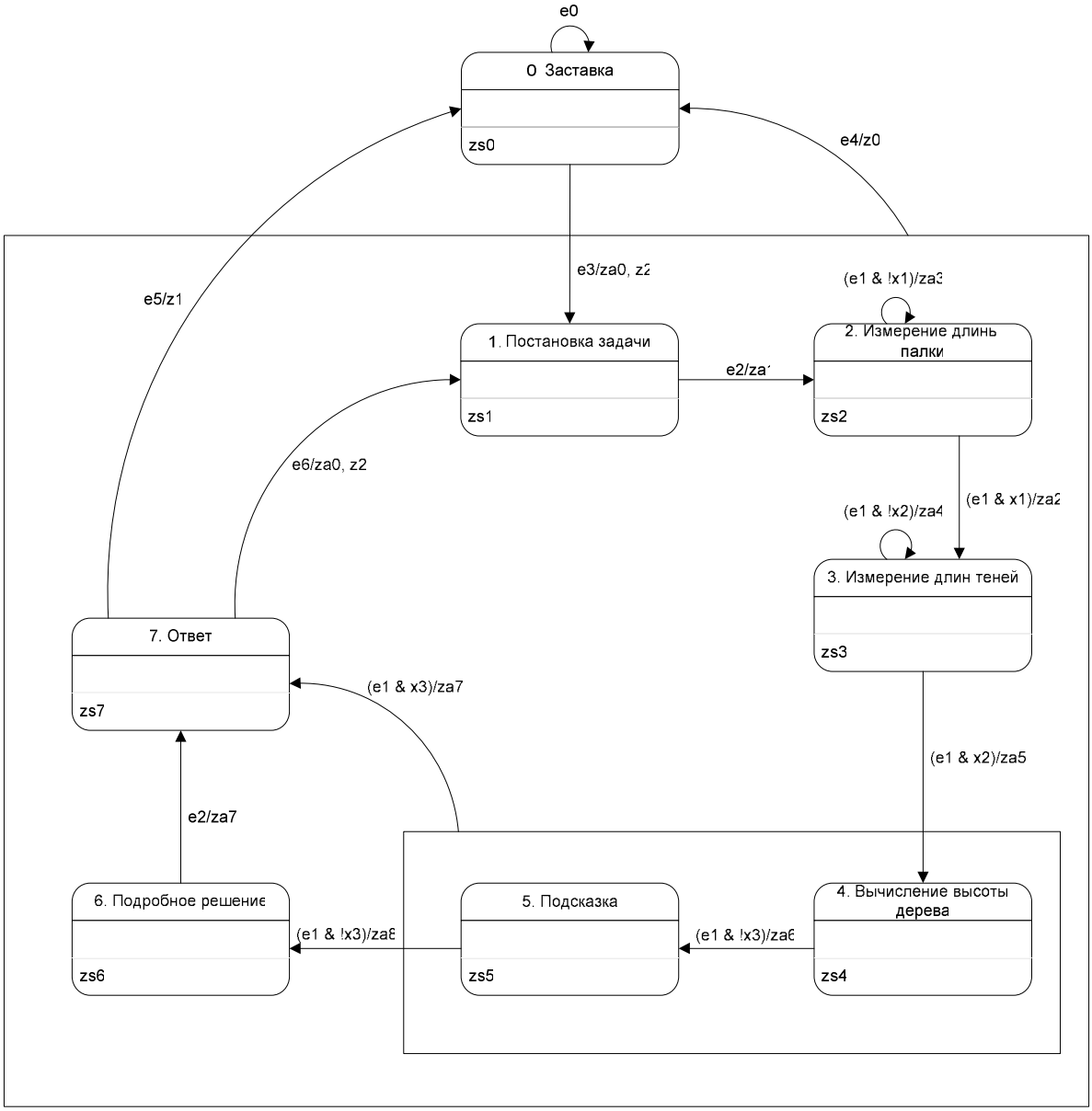


Рис. 3. Граф переходов, реализующий логику задачи

Обратим внимание, что граф переходов является не «картинкой», а математической моделью, по которой текст фрагмента программы, реализующего логику задачи, строится формально и изоморфно. Этот текст на языке *ActionScript* приведен в Листинге 1.

Для окончательного построения программы, реализующей автомат, необходимо разработать тексты подпрограмм входных переменных, обработчиков событий и выходных действий. Так как в перечисленных подпрограммах практически отсутствует логическая обработка, они реализуются традиционным путем (Листинг 2, 3).

Структурная схема программы приведена на рис. 4.

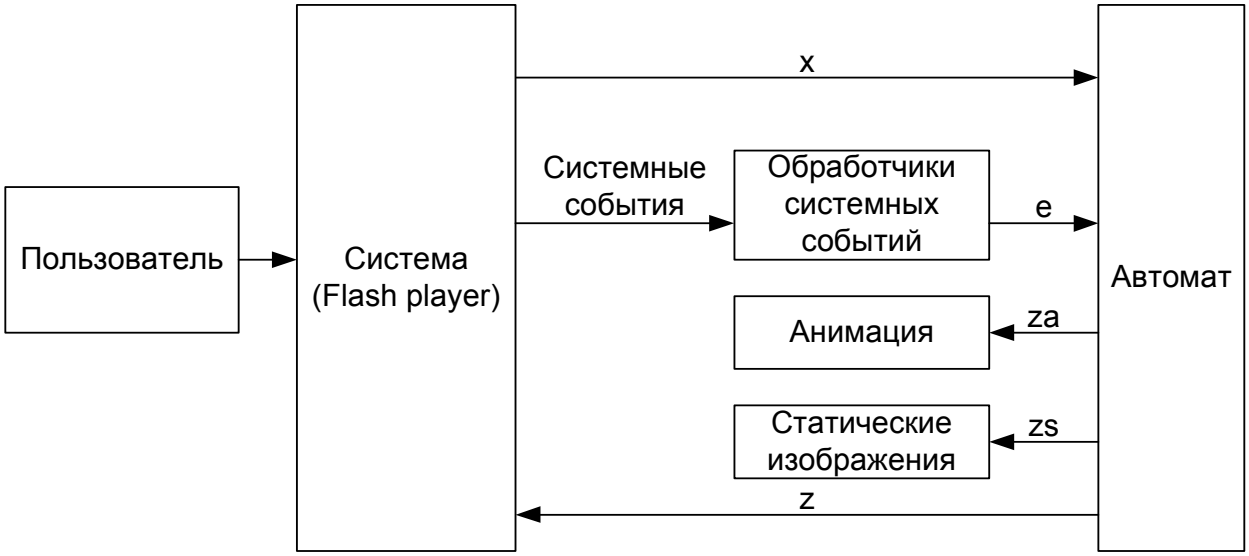


Рис. 4. Структурная схема программы

Обработчик системного события запускается в связи с этим событием, например, от устройства «мышь» или клавиатуры, выделяет это событие, и вызывает с ним (в качестве аргумента) автоматную функцию.

Из изложенного следует, что при применении предлагаемого подхода логика, в отличие от традиционного подхода, не рассредоточена по обработчикам событий, а централизована, что резко упрощает отладку, которую в работе [9] предлагается проводить с помощью протоколирования, выполняемого в терминах автоматов. Листинг 4 содержит модуль программы, обеспечивающий протоколирование, а пример протокола приведен в Листинге 5. Этот модуль, вызываемый из автомата, обработчиков событий и функций выходных действий, для упрощения структурной схемы на рис. 4 не показан. При отладке возможен вывод протоколов на экран (рис. 5). Для отключения и включения вывода протоколов на экран можно использовать флажок в левом нижнем углу экрана.

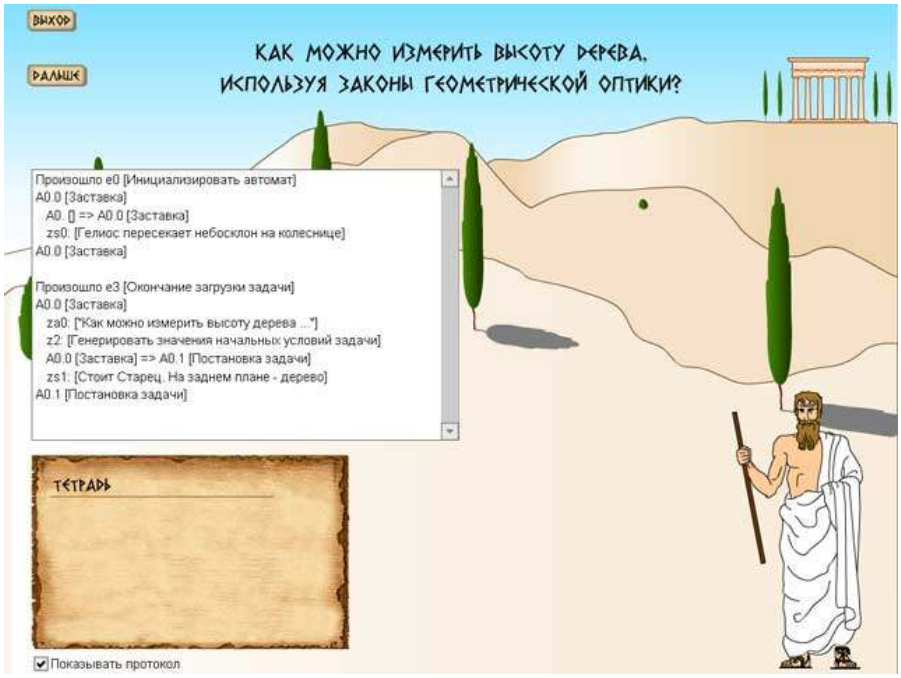


Рис. 5. Вывод протоколов на экран

## 4. Заключение

Подход, предлагаемый в настоящей работе, позволяет использовать автоматы при спецификации, в тексте программы и при протоколировании.

Таким образом, в настоящей работе предложена новая технология программирования анимации на языке *ActionScript*.

Указанные выше листинги программ приведены в Приложении.

Работа выполнена в лаборатории *Технологии программировании*, организованной *СПбГУ ИТМО* и центром разработки корпорации *Borland*.

## Литература

1. *Тарасов И.А.* Основы программирования в OpenGL. М.: Горячая линия – телеком. 2001.
2. *Гончаров Д., Салихов Т.* Книга DirectX 7.0 для программистов. СПб.: Питер. 2001.
3. *Маэстри Дж.* Компьютерная анимация персонажей. СПб.: Питер. 2001.
4. *Рейнхардт Р., Ленц Дж.* Flash 5. Библия пользователя. М.: Вильямс. 2001.
5. *Туйкин М.* CeBIT 2002 // Программист. 2002. № 4.
6. *Сандерс Б.* Flash ActionScript. СПб.: Питер. 2001.
7. *Степанова Г.Н., Степанов А.П.* Сборник вопросов и задач по физике. СПб.: Основная школа. 2001.
8. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука. 1998.
9. *Шалыто А.А., Туккель Н.И.* SWITCH-технология — автоматный подход к созданию программного обеспечения "РЕАКТИВНЫХ" систем // Программирование. 2001. № 5. http://is.ifmo.ru. Раздел «Статьи».
10. *Головешин А.* Конвертер Visio2Switch. http://is.ifmo.ru/progeny/visio2switch/.

## Приложение

### Листинг 1. Рализация логики задачи

Файл *automat.as*

```
#include "log.as"
#include "animation.as"
#include "static.as"
#include "out.as"
#include "var.as"

iProblem2.A0 = function(e) {
    if (Y0 == undefined)
        Y0 = 0;
    else
        Yold = Y0;

    switch (Y0)
    {
        case 0: logBegin("A0", "0", "Заставка");
            if (e == 0)           {               Y0=0;}
            if (e == 3)           {za0(); z2();   Y0=1;}
        break;

        case 1: logBegin("A0", "1", "Постановка задачи");
            if (e == 2)           {za1();         Y0=2;}
            else
            if (e == 4)           {z0();          Y0=0;}
        break;

        case 2: logBegin("A0", "2", "Измерение длины палки");
            var _x1 = x1();
            if (e == 1 &&  _x1)   {za2();         Y0=3;}
            else
            if (e == 1 && !_x1)   {za3();              }
            else
            if (e == 4)           {z0();          Y0=0;}
        break;

        case 3: logBegin("A0", "3", "Измерение длин теней");
            var _x2 = x2();
            if (e == 1 &&  _x2)   {za5();         Y0=4;}
            else
            if (e == 1 && !_x2)   {za4();              }
            else
            if (e == 4)           {z0();          Y0=0;}
        break;

        case 4: logBegin("A0", "4", "Вычисление высоты дерева");
            var _x3 = x3();
            if (e == 1 &&  _x3)   {za7();         Y0=7;}
            else
            if (e == 1 && !_x3)   {za6();         Y0=5;}
            else
            if (e == 4)           {z0();          Y0=0;}
        break;

        case 5: logBegin("A0", "5", "Подсказка");
            var _x3 = x3();
            if (e == 1 &&  _x3)   {za7();         Y0=7;}
            else
            if (e == 1 && !_x3)   {za8();         Y0=6;}
            else
            if (e == 4)           {z0();          Y0=0;}
        break;

        case 6: logBegin("A0", "6", "Подробное решение");
            if (e == 2)           {za7();         Y0=7;}
            else
            if (e == 4)           {z0();          Y0=0;}
        break;

        case 7: logBegin("A0", "7", "Ответ");
            if (e == 4)          {z0();          Y0=0;}
            else
            if (e == 5)          {z1();          Y0=0;}
            else
            if (e == 6)          {za0(); z2();   Y0=1;}
        break;
    }

    if (Yold != Y0) {
        switch (Y0) {
            case 0: logTrans("A0", Yold, "0", "Заставка");
                zs0();
            break;
```

```
            case 1: logTrans("A0", Yold, "1", "Постановка задачи");
                zs1();
            break;

            case 2: logTrans("A0", Yold, "2", "Измерение длины палки");
                zs2();
            break;

            case 3: logTrans("A0", Yold, "3", "Измерение длин теней");
                zs3();
            break;

            case 4: logTrans("A0", Yold, "4", "Вычисление высоты дерева");
                zs4();
            break;

            case 5: logTrans("A0", Yold, "5", "Подсказка");
                zs5();
            break;

            case 6: logTrans("A0", Yold, "6", "Подробное решение");
                zs6();
            break;

            case 7: logTrans("A0", Yold, "7", "Ответ");
                zs7();
            break;

        }
    }

    logEnd("A0",Y0);
};
```

**Листинг 2. Реализация выходных действий**

Файл *static.as*

```
iProblem2.zs0 = function() {
    logStaticOut(0 ,"Гелиос пересекает небосклон на колеснице");
    ShowButtons(false,false,false,false,false); // Спрятать все кнопки
    _root.iPreLoader._visible = true;           // Показать предзагрузчик
    _alpha = 0;                                 // Скрыть загружаемые объекты
};

iProblem2.zs1 = function() {
    logStaticOut(1 ,"Стоит Старец. На заднем плане - дерево");
    _root.iPreLoader._visible = false;          // Скрыть предзагрузчик
    _alpha = 100;                               // Показывать загружаемые объекты
    ShowButtons(true,true,false,false,false);   // Показывать кнопки "Дальше" и "Выход"
    ShowMaster("WithStick");                    // Показать Старца с палкой
    ShowStick(false,false,false,false);         // Спрятать воткнутую палку
    ShowTree(true,true,false,false);            // Показать дерево и его тень
    ShowWorkBook("Empty");                      // Показать пустую тетрадь
};

iProblem2.zs2 = function() {
    logStaticOut(2 ,"Линейка рядом с палкой");
    ShowButtons(false,true,false,false,false);  // Показывать кнопку "Выход"
    ShowMaster("WithOutStick");                 // Показать Старца без палки
    ShowStick(true,true,true,false);            // Показать воткнутую палку с тенью и ленейкой
    ShowWorkBook("StickHeight");                // Показать тетрадь с полем для ввода длины палки
};

iProblem2.zs3 = function() {
    logStaticOut(3 ,"Линейки рядом с тенями");
    ShowMaster("PointingOnShadow");             // Показать Старца, указывающего на тени
    ShowStick(true,true,false,true);            // Показать палку с тенью и ленейкой у тени
    ShowTree(true,true,false,true);             // Показать дерево с тенью и ленейкой у тени
    ShowWorkBook("ShadowsLengths");             // Показать тетрадь с полем для ввода длин теней
};

iProblem2.zs4 = function() {
    logStaticOut(4 ,"Линейки убраны");
    ShowMaster("WithOutStick");                 // Показать Старца без палки
    ShowStick(true,true,false,false);           // Показать воткнутую палку с тенью
    ShowTree(true,true,false,false);            // Показать дерево с тенью
    ShowWorkBook("TreeHeight");                 // Показать тетрадь с приглашением ввести высоту
дерева
};

iProblem2.zs5 = function() {
    logStaticOut(5 ,"Старец увеличен. В тетради чертеж");
    ShowMaster("Torso");                        // Показать увеличенного Старца
    ShowWorkBook("Scheme");                     // Показать чертеж
};

iProblem2.zs6 = function() {
    logStaticOut(6 ,"Старец недоволен. Решение в тетради");
    ShowButtons(true,true,false,false,false);   // Показать кнопки "Дальше" и "Выход"
    ShowMaster("TorsoAngry");                   // Показать озлобленного старца
    ShowWorkBook("Formula");                    // Показать вывод решения
};

iProblem2.zs7 = function() {
    logStaticOut(7 ,"Старец радуется");
    ShowButtons(false,true,true,true,false);    // Показать кнопки
    ShowMaster("Happy");                        // Показать счастливого старца
    ShowStick(true,true,false,false);           // Показать палку с тенью
    ShowTree(true,true,true,false);             // Показать дерево с линейкой и тенью
    ShowWorkBook("Answere");                    // Показать в тетради правильный ответ
};
```

Файл *animation.as*

```
iProblem2.za0 = function() {
    logAnimationOut(0, "\"Как можно измерить высоту дерева ...\"");
    ShowComment("TreeMeasureProblem");          // Титр с постановкой задачи
};

iProblem2.za1 = function() {
    logAnimationOut(1,"\"Запиши в тетрадь значение длины палки\". Старец втыкает палку");
    ShowComment("StickHeight");                 // Титр: "Запиши в тетрадь значение длины палки"
    iWorkBook.txfStickHeight = "";              // Очистить поле ввода длины палки
};

iProblem2.za2 = function() {
    logAnimationOut(2,"\"Посмотри на тени с линейками ...\"");
    ShowComment("ShadowsLengths");              // Титр: "Посмотри на тени с линейками ..."
    iWorkBook.txfStickShadowLength = "";        // Очистить поле ввода длины тени от палки
    iWorkBook.txfTreeShadowLength = "";         // Очистить поле ввода длины тени от дерева
};
```

```
iProblem2.za3 = function() {
    logAnimationOut(3,"\"Неверно! Попробуй еще раз\". Старец мотает головой");
    ShowComment("WrongHeight");                 // Титр: "Неверно! Попробуй еще раз".
    AnimateMaster("HeadShake");                 // Старец мотает головой
    iWorkBook.txfStickHeight = "";              // Очистить поле ввода длины палки
};

iProblem2.za4 = function() {
    logAnimationOut(4,"\"Измерь точнее длины теней\". Старец мотает головой");
    ShowComment("WrongLengths");                // Титр: "Измерь точнее длины теней".
    AnimateMaster("HeadShake");                 // Старец мотает головой
    if (!CheckData(iWorkBook.txfStickShadowLength, StickShadowLength(), 2))
        iWorkBook.txfStickShadowLength = "";    // Очистить поле ввода длины тени от палки
    if (!CheckData(iWorkBook.txfTreeShadowLength, TreeShadowLength(), 2))
        iWorkBook.txfTreeShadowLength = "";     // Очистить поле ввода длины тени от дерева
};

iProblem2.za5 = function() {
    logAnimationOut(5,"\"Теперь у тебя есть все данные ...\". Старец кивает");
    ShowComment("TreeHeight");                  // Титр: "Теперь у тебя есть все данные ..."
    AnimateMaster("HeadNod");                   // Старец кивает
    iWorkBook.txfTreeHeight = "";               // Очистить поле ввода высоты дерева
};

iProblem2.za6 = function() {
    logAnimationOut(6,"\"Неверно. Посмотри чертеж ...\"");
    ShowComment("WatchScheme");                 // Титр: "Неверно. Посмотри чертеж ...".
    iWorkBook.txfTreeHeight = "";               // Очистить поле ввода высоты дерева
};

iProblem2.za7 = function() {
    logAnimationOut(7,"\"Задача решена\". Старец поднимает руку");
    ShowComment("ProblemSolved");               // Титр: "Задача решена"
    AnimateMaster("HandUp");                    // Старец поднимает руку
};

iProblem2.za8 =function() {
    logAnimationOut(8,"\"В решении этой задачи используется ...\". Старец очень раздражен");
    ShowComment("WatchFormula");                // Титр: "В решении этой задачи используется ..."
    AnimateMaster("Angry");                     // Старец очень раздражен.
    iWorkBook.txfTreeHeight = "";               // Очистить поле ввода высоты дерева
};
```

Файл *out.as*

```
iProblem2.z0 = function() {
    logOut("0","Завершить работу флэш-плеера");
    fscommand("quit","");                       // Передать плееру команду выход
};

iProblem2.z1 = function() {
    logOut("1","Выгрузить текущую задачу. Загрузить следущую");
    _root.NextProblem();                        // Выгрузить текущую задачу, загрузить следующую
};

iProblem2.z2 = function() {
    logOut("2","Генерировать значения начальных условий задачи");
    InitConstants();                            // Инициализировать используемые константы
    InitData();                                 //   и переменные
}
```

**Листинг 3. Реализация входных переменных и обработчиков событий**

Файл *var.as* — входные переменные

```
iProblem2.x1 = function() {
    uSH = iWorkBook.txfStickHeight;
     SH = StickHeight;
    res = CheckData(uSH,SH,2);
    logVar("1", res?1:0, "Введена верная высота палки?");
    return res;
};

iProblem2.x2 = function() {
    uSL = iWorkBook.txfStickShadowLength;
     SL = StickShadowLength();
    uTL = iWorkBook.txfTreeShadowLength;
     TL = TreeShadowLength();
    res = CheckData(uSL, SL, 2) && CheckData(uTL, TL, 2);
    logVar("2", res?1:0, "Введены верные длины теней?");
    return res;
};

iProblem2.x3 = function() {
    uTH = iWorkBook.txfTreeHeight;
     TH = TreeHeight;
    res = CheckData(uTH, TH, 1);
    logVar("3", res?1:0, "Введена верная высота дерева?");
    return res;
};
```

Файл *events.as* — обработчики событий

```
automatInit = function() {
    logEvent(0,"Инициализировать автомат");
    A0(0);
}

// Загрузка задачи окончена [e3]
loadingFinished = function() {
    logEvent(3,"Окончание загрузки задачи");
    A0(3);
};

// Сообщения от графических кнопок [e1,e2,e4,e5,e6]
onMouseUp = function() {
    var xm = _root._xmouse;
    var ym = _root._ymouse;

    if (iWorkBook.iOK.hitTest(xm,ym,true) && iWorkBook.iOK._visible) {
        logEvent(1,"Нажатие кнопки \"Готово\"");
        A0(1);
    } else
    if (iButNext.hitTest(xm,ym,true) && iButNext._visible) {
        logEvent(2,"Нажатие кнопки \"Дальше\"");
        A0(2);
    } else
    if (iButExit.hitTest(xm,ym,true) && iButExit._visible) {
        logEvent(4,"Нажатие кнопки \"Выход\"");
        A0(4);
    } else
    if (iButNextProblem.hitTest(xm,ym,true) && iButNextProblem._visible) {
        logEvent(5,"Нажатие кнопки \"Следующая задача\"");
        A0(5);
    } else
    if (iButReplayProblem.hitTest(xm,ym,true) && iButReplayProblem._visible) {
        logEvent(6,"Нажатие кнопки \"Повторить задачу\"");
        A0(6);
    }
};


// Сообщения от клавиатуры [e1,e2]
onKeyUp = function () {
    switch (Key.getCode())
    {
    case Key.ENTER:
        logEvent(1,"Нажатие кнопки \"Готово\"");
        A0(1);
    break;

    case Key.SPACE:
        logEvent(2,"Нажатие кнопки \"Дальше\"");
        A0(2);
    break;
    }
};

// Обрабатывать сообщения от клавиатуры
Key.addListener(this);
```

**Листинг 4. Реализация функций протоколирования**

Файл *log.as*

```
iProblem2.logToWin          = true;
iProblem2.doLogBegin        = true;
iProblem2.doLogTrans        = true;
iProblem2.doLogEnd          = true;
iProblem2.doLogOut          = true;
iProblem2.doLogEvent        = true;
iProblem2.doLogAnimationOut = true;
iProblem2.doLogStaticOut    = true;
iProblem2.doLogVar          = true;

iProblem2.yNames = new Array();

// Для вывода протоколов может быть использовано окно отладчика редактора Macromedia Flash
//  или окно, созданное средствами стандартных компонент и доступное не только во время
//  во время отладки. Для практических применений технологии рекомендуется пользоваться
//  окном отладчика (logToWin == false). Здесь используется компонентное окно, оно добавлено
//  из соображений наглядности протоколирования.
iProblem2.log = function(str) {
    if (logToWin) {
        lstLog.addItem(str);
        lstLog.setScrollPosition(Math.max(0,lstLog.getLength() - lstLog.getRowCount()));
    }
    else
        trace(str);
};

iProblem2.logBegin = function(A, Y, yName) {
    if (doLogBegin) {
        log(A+"."+Y+" ["+yName+"]");
        if (yNames[Y] == undefined)
            yNames[Y] = yName;
    }
};

iProblem2.logTrans = function(A, Y_old, Y, yName) {
    if (doLogTrans) {
        log("   "+A+"."+Y_old+" ["+yNames[Y_old]+"] => "+A+"."+Y+" ["+yName+"]");
        if (yNames[Y] == undefined)
            yNames[Y] = yName;
    }
};

iProblem2.logEnd = function(A, Y) {
    if (doLogEnd) {
        log(A+"."+Y+" ["+yNames[Y]+"]");
        log("");
    }
};

iProblem2.logOut = function(z, descr) {
    if (doLogOut)
        log("   z"+z+": ["+descr+"]");
};

iProblem2.logEvent = function(e, descr) {
    if (doLogEvent)
        log("Произошло e"+e+" ["+descr+"]");
};

iProblem2.logAnimationOut = function(z, descr) {
    if (doLogAnimationOut)
        logOut("a"+z, descr);
};

iProblem2.logStaticOut = function(z, descr) {
    if (doLogStaticOut)
        logOut("s"+z, descr);
};

iProblem2.logVar = function(x, value, descr) {
    if (doLogVar)
        log("   x"+x+"="+value+": ["+descr+"] - " + (value?"ДА":"НЕТ"));
};
```

**Листинг 5. Пример протокола**

Приведенный протокол соответствует случаю, когда пользователь решил задачу, не воспользовавшись ни одной подсказкой.

```
Произошло e0 [Инициализировать автомат]
A0. []
   A0. [] => A0.0 [Заставка]
   zs0: [Гелиос пересекает небосклон на колеснице]
A0.0 [Заставка]

Произошло e3 [Окончание загрузки задачи]
A0.0 [Заставка]
   za0: ["Как можно измерить высоту дерева ..."]
   z2: [Генерировать значения начальных условий задачи]
   A0.0 [Заставка] => A0.1 [Постановка задачи]
   zs1: [Стоит Старец. На заднем плане - дерево]
A0.1 [Постановка задачи]

Произошло e2 [Нажатие кнопки "Дальше"]
A0.1 [Постановка задачи]
   za1: ["Запиши в тетрадь значение длины палки". Старец втыкает палку]
   A0.1 [Постановка задачи] => A0.2 [Измерение длины палки]
   zs2: [Линейка рядом с палкой]
A0.2 [Измерение длины палки]

Произошло e1 [Нажатие кнопки "Готово"]
A0.2 [Измерение длины палки]
   x1=1: [Введена верная высота палки?] - ДА
   za2: ["Посмотри на тени с линейками ..."]
   A0.2 [Измерение длины палки] => A0.3 [Измерение длин теней]
   zs3: [Линейки рядом с тенями]
A0.3 [Измерение длин теней]

Произошло e1 [Нажатие кнопки "Готово"]
A0.3 [Измерение длин теней]
   x2=1: [Введены верные длины теней?] - ДА
   za5: ["Теперь у тебя есть все данные ...". Старец кивает]
   A0.3 [Измерение длин теней] => A0.4 [Вычисление высоты дерева]
   zs4: [Линейки убраны]
A0.4 [Вычисление высоты дерева]

Произошло e1 [Нажатие кнопки "Готово"]
A0.4 [Вычисление высоты дерева]
   x3=1: [Введена верная высота дерева?] - ДА
   za7: ["Задача решена". Старец поднимает руку]
   A0.4 [Вычисление высоты дерева] => A0.7 [Ответ]
   zs7: [Старец радуется]
A0.7 [Ответ]

Произошло e4 [Нажатие кнопки "Выход"]
A0.7 [Ответ]
   z0: [Завершить работу флэш-плеера]
   A0.7 [Ответ] => A0.0 [Заставка]
   zs0: [Гелиос пересекает небосклон на колеснице]
A0.0 [Заставка]
```