ТЕХНОЛОГИЧЕСКОЕ ОБРАЗОВАНИЕ

Мазин Максим Александрович,
Парфенов Владимир Глебович,
Шалыто Анатолий Абрамович

АНИМАЦИЯ. FLASH-ТЕХНОЛОГИЯ. АВТОМАТЫ

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.

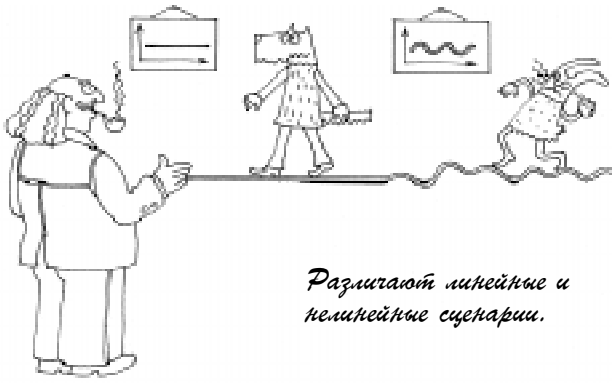
Однако если необходимо разрабатывать приложения для сети Интернет, то наиболее предпочтительным оказывается использование пакета *Macromedia Flash* [4], являющегося основой *Flash-технологии*. Большая популярность этой технологии в мире [5] объясняется рядом ее достоинств, среди которых следует отметить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

В настоящее время интерактивная анимация широко используется в различных приложениях...



Различают линейные и нелинейные сценарии.

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

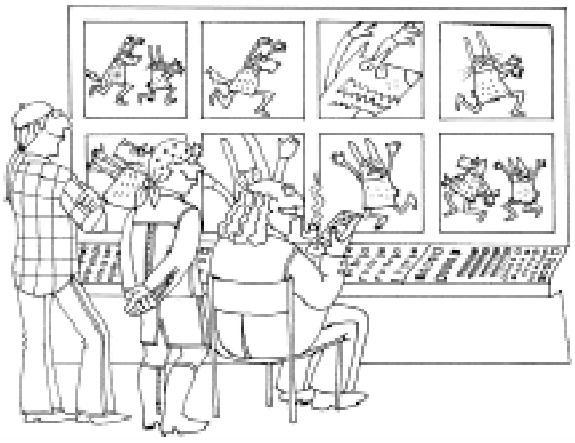
РАЗРАБОТКА ИНТЕРАКТИВНОЙ АНИМАЦИИ

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 *программированием 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.



Решать указанные подзадачи будем на основе SWITCH-технологии...

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.

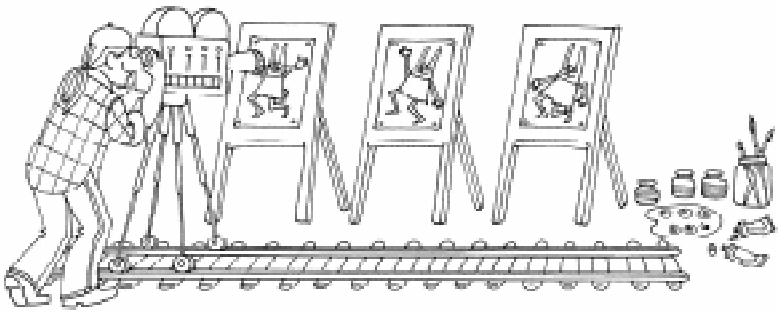
Изложенное позволяет построить схему связей автомата, которая описывает его интерфейс. При этом автомат управляет статическими изображениями, анимацией и средой исполнения.

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.



...художник «рисует» статические изображения и выполняет раскадровку анимации...

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.

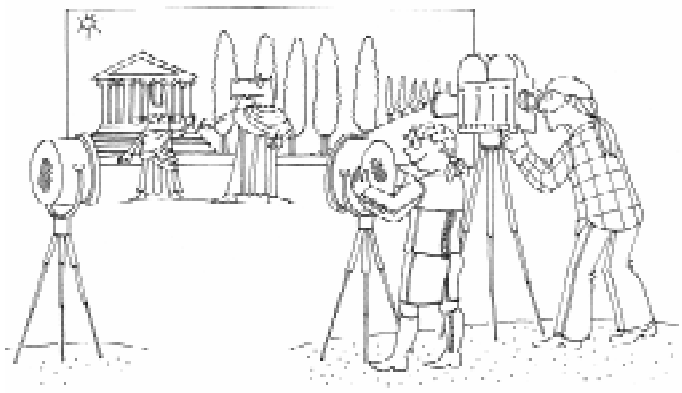
В тетради (на пергаменте) появляется надпись «Длина палки h = ___ локтей». Тетрадь выполнена из пергаментных листов. Шрифт надписи должен быть стилизован. Если пользователь ввел неверный ответ, то появляется надпись: «Неверно! Попробуй еще раз».

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.

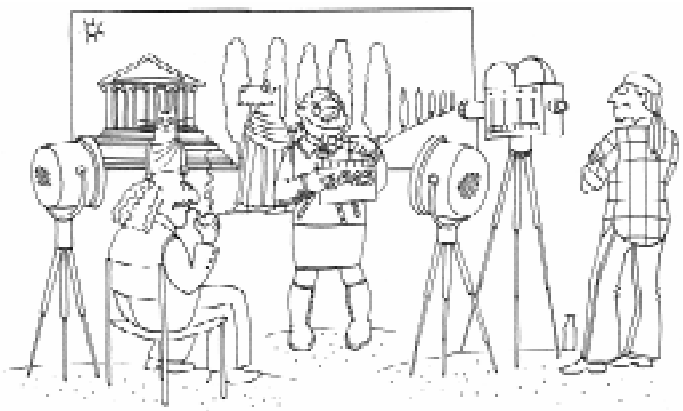
Появляется надпись: «Посмотри на тени. Измерь и занеси их длины в тетрадь».

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».

Если пользователь ввел верный ответ, появляется надпись: «Теперь у тебя есть все данные, чтобы вычислить высоту дерева». В тетради появляется окно для



...в сценарии будут использованы древнегреческие образы...



Кнопка «Повторить задачу» —
... задача запускается с новыми значениями величин...

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;

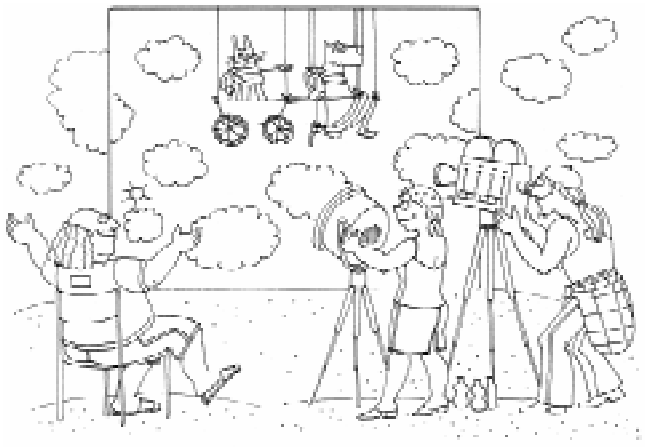
– кнопка «Готово» – пользователь нажимает эту кнопку после окончания ввода ответа в соответствующие поля. По ее нажатию проверяется правильность введенных значений;

– 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 пе-



... изображение Гелиоса, пересекающего небосклон.

реход. 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 завершении 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). При нажатии 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).

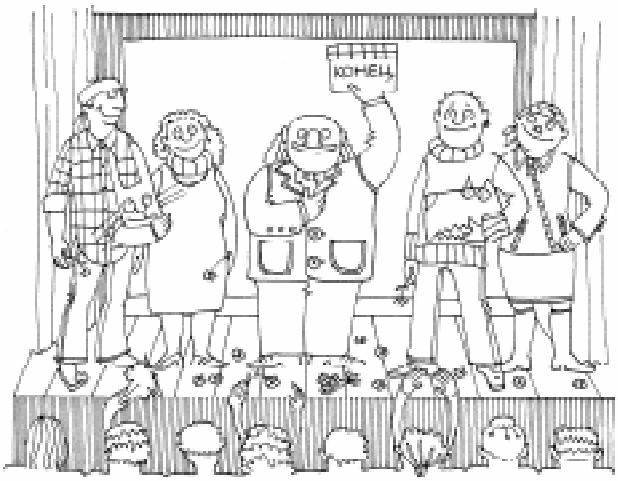
(3, 4). Если пользователь ввел (e1) верный ответ (x2), появляется надпись: «Теперь у тебя есть все данные, чтобы вычислить высоту дерева», Старец кивает (za5).

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



На экране постоянно присутствует кнопка «Выход», нажатие на которую завершает работу флэш-плеера.

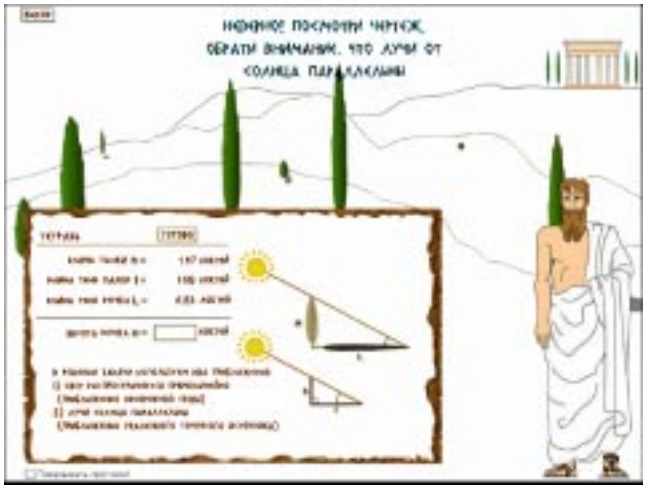


Рисунок 1. Изображение после перехода из состояния 4 в состояние 5

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 сердится (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) При нажатии 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 (рисунок 1).

Для написания программы проведем дальнейшую формализацию задачи. Построим схему связей автомата, определяющую его интерфейс (рисунок 3).

Граф переходов, построенный с помощью шаблона (*stencil*), описанного в работе [10], приведен на рисунке 4.

Обратим внимание, что граф переходов является не «картинкой», а математической моделью, по которой текст фрагмента программы, реализующего логику задачи, строится формально и изоморфно.

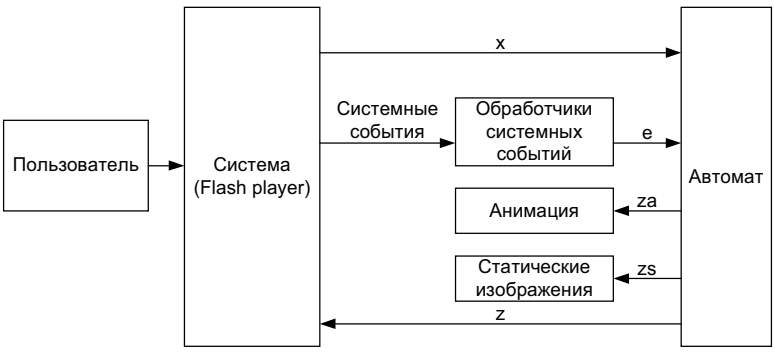


Рисунок 2. Структурная схема программы.

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.

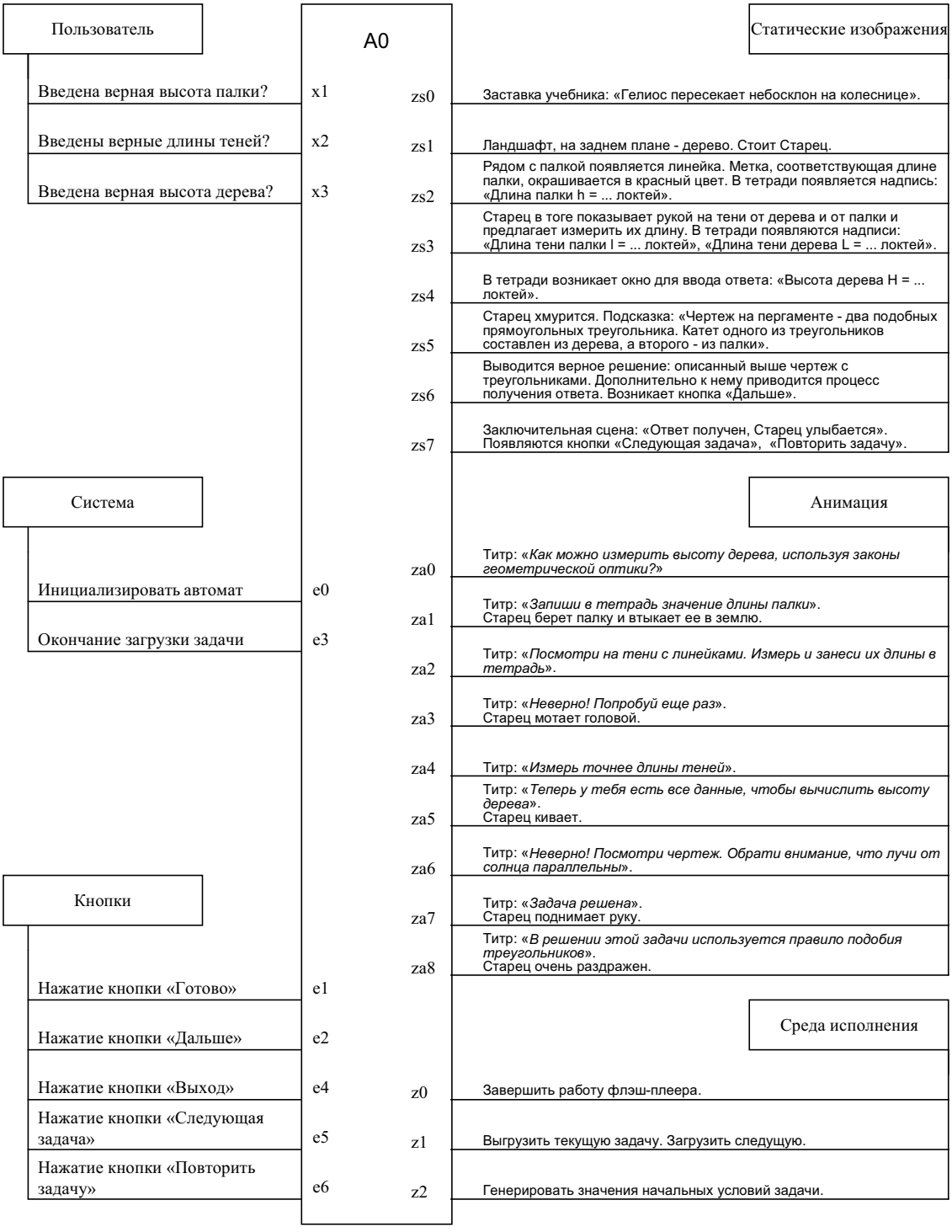


Рисунок 3. Схема связей, описывающая интерфейс автомата

Структурная схема программы приведена на рисунке 2.

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 рас-

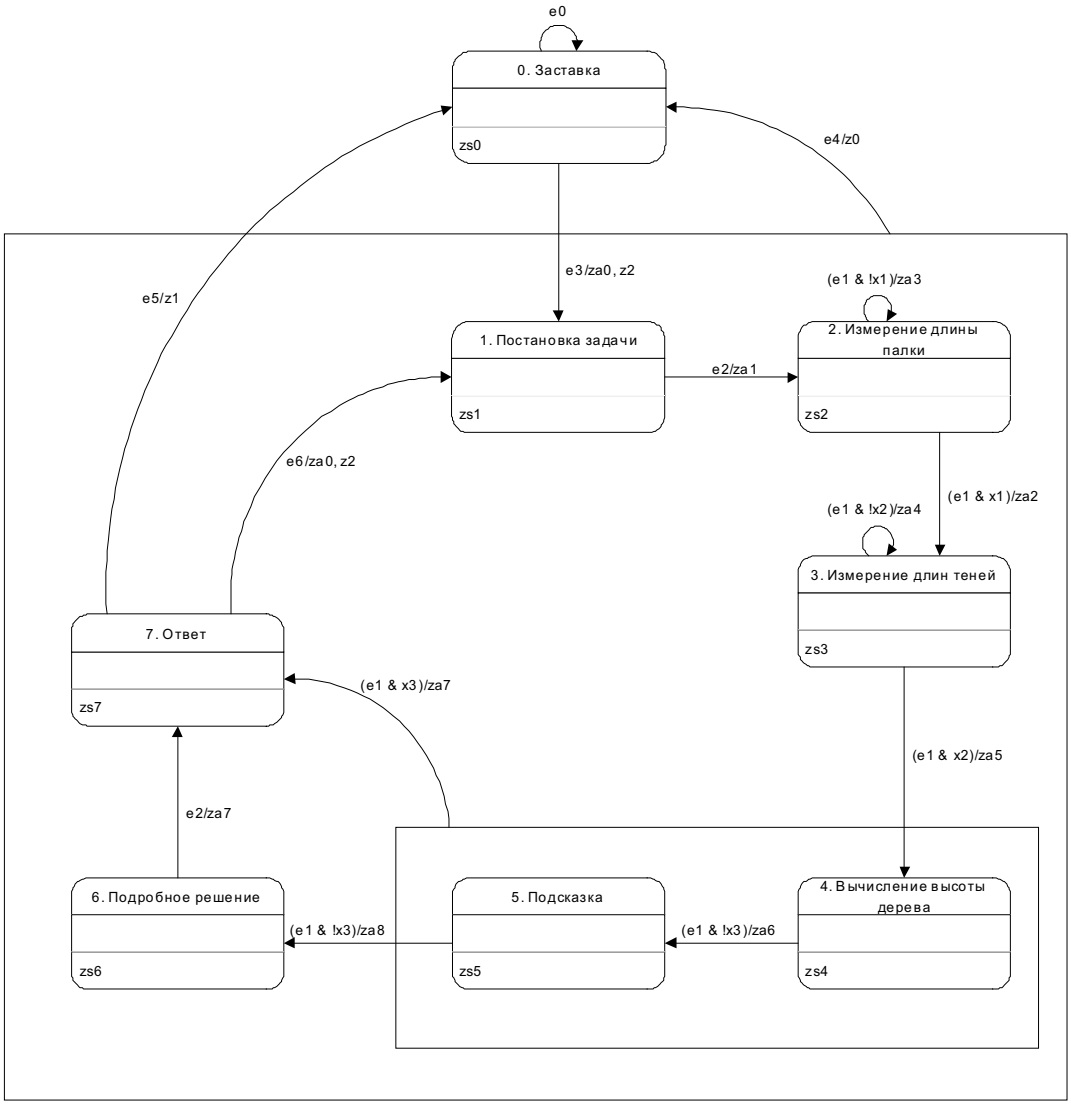


Рисунок 4. Граф переходов, реализующий логику задачи.

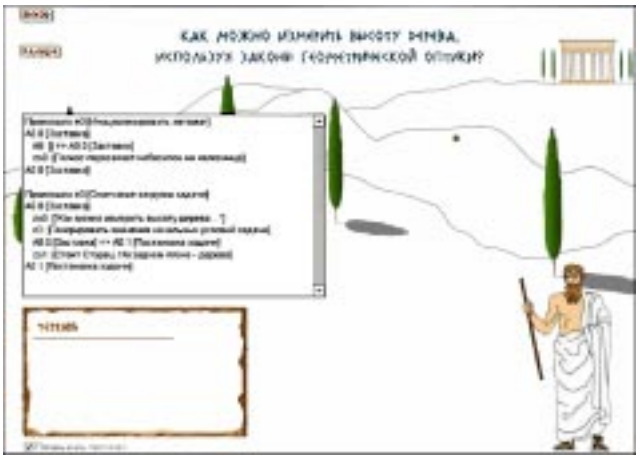


Рисунок 5. Вывод протоколов на экран.

средоточена по обработчикам событий, а централизована, что резко упрощает отладку, которую в работе [9] предлагается проводить с помощью протоколирования, выполняемого в терминах автоматов. Этот модуль, вызываемый из автомата обработчиков событий и функций выходных действий для упрощения структурной схемы на рисунке 2 не показан. При отладке возможен вывод протоколов на экран (рисунок 5). Для отключения и включения вывода протоколов на экран используется флажок в левом нижнем углу экрана.

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*.

Листинги программ приведены в Приложении на диске к журналу, а исходные тексты и исполняемый файл – на сайте http://is.ifmo.ru в разделе «Статьи».

Работа выполнена при поддержке Российского фонда фундаментальных исследований по гранту № 02-07-90114 «Разработка технологии автоматного программирования».

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. Раздел «Последователи».

Мазин Максим Александрович,
студент кафедры «Компьютерные технологии» Санкт-Петербургского государственного института точной механики и оптики (технического университета) – СПбГИТМО (ТУ),
Парфенов Владимир Глебович,
д.т.н., профессор, декан факультета «Информационные технологии и программирование» СПбГИТМО (ТУ),
Шалыто Анатолий Абрамович,
д.т.н., профессор кафедры «Компьютерные технологии» СПбГИТМО (ТУ).