Министерство образования и науки Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
Карельский государственный педагогический университет

студент 551 гр.
Туктаров А.А.

Использование Flash-анимации для разработки интерактивных компьютерных моделей по физике

Дипломная работа

Кафедра информатики
Научный руководитель:
ассистент кафедры информатики
А.С. Кюршунов

Петрозаводск
2004г.

Содержание:

Введение 3
1. Выбор технологии Macromedia Flash для разработки интерактивных компьютерных моделей 6
2. Разработка интерактивных компьютерных моделей 8
2.1. Математический маятник 8
2.2 Движение тела, брошенного под углом к горизонту 11
2.2.1. Создание интерактивного вектора 13
2.2.2. Построение графика функции 16
2.3 Движение тела по наклонной плоскости 26
2.3.1. Создание направляющего элемента 27
2.3.2. Организация движения 29
2.3.3. Отображение векторов 30
3. Практические рекомендации по разработке интерактивных компьютерных моделей 34
3.1 Различные подходы к организации интерактивности компьютерной модели 34
3.2. Два способа размещения программного кода для организации жизненных циклов моделей 35
3.3. Принцип построения графиков в среде Macromedia Flash 36
3.4. Организация движения и взаимодействия объектов 37
3.5. Избирательное отображение элементов модели 39
Выводы 40
Литература. 41

Введение

В последние годы в России и за рубежом существует тенденция к более широкому применению компьютерных технологий в обучении.

В нашей стране это подтверждается рядом нормативных документов Министерства Образования РФ. В рамках документа «О Концепции модернизации российского образования на период до 2010 года» можно говорить о том, что существует тенденция к информатизации образования и развитию дистанционного обучения.

В связи с указанной тенденцией 3-4 декабря 2003 г. в г. Москве был проведен научно-методический симпозиум «Компьютерное моделирование в обучении точным наукам», в котором были обозначены направления, которые связаны с необходимостью разработки компьютерных моделей для обучения, такие как [1]:

- применение виртуальных лабораторных практикумов по различным дисциплинам и курсам в области точных наук;
- формирование у школьников и студентов системного естественнонаучного мировоззрения на основе создания опорных образовательных образов в этой области;
- изучение природы микро- и макромиров, окружающих человечество, которое с помощью физического лабораторного оборудования современной школы и вуза практически невозможно;
- более глубокий анализ физических, химических, биологических и других процессов и явлений за счет имитации и учета существенно большего количества параметров и факторов по сравнению с возможностями физического лабораторного оборудования образовательных учреждений.

Использование компьютерных моделей повышает качество и эффективность обучения естественнонаучным дисциплинам в школах и вузах. Поэтому появляется необходимость разработки компьютерных моделей для различных дисциплин.

Существует два способа разработки компьютерных моделей: с помощью специализированных программных средств и прямого программирования.

Первый способ прост для использования, так как специализированные программные средства позволяют быстро и удобно создать компьютерную модель, но только ту, которая ограничена набором объектов и методов, существующих в таких программных средах. Таким образом, хотя этот способ не требует серьезных знаний программирования, он ограничивает разработчика и не является гибким.

Второй способ позволяет создавать любую модель, так как с помощью прямого программирования можно создавать любые отношения между графическими объектами. Этот способ является трудоемким и требует хорошего знания языков программирования.

Среди средств разработки компьютерных моделей можно выделить Macromedia Flash. Этот продукт не является специализированным средством, но позволяет разработчику совмещать встроенный инструментарий для рисования графических объектов и описывать отношения между ними с помощью встроенного объектно-ориентированного языка программирования Action Script. Таким образом, графический редактор среды Macromedia Flash является простым средством для рисования внешнего вида объектов и создания анимации, а Action Script описывает связи между этими объектами. Этот способ является наиболее удобным, так как позволяет разработчику сократить время, требующееся для разработки модели.

Компьютерные модели можно использовать как в дистанционном обучении, так и в традиционном обучении, в том числе и с сетевой поддержкой. Кроме того, для обучения с использованием компьютерных моделей важно организовать взаимодействие учащегося с моделью для вовлечения его в активную исследовательскую деятельность, поэтому эффективным является использование интерактивных компьютерных моделей.

Под интерактивной компьютерной моделью понимается компьютерная модель, которая позволяет организовать взаимодействие с пользователем для изменения начальных условий и свойств этой модели.

Рассмотрим некоторые условия применения интерактивных компьютерных моделей на уроках.

В условиях класса часто невозможно провести реальный эксперимент из-за отсутствия оборудования или продолжительности его по времени. Также эксперимент может быть опасен для здоровья учащихся. Поэтому удобно использовать интерактивные компьютерные модели для замены или поддержки лабораторного эксперимента [2].

В дистанционном обучении вообще невозможно проведение лабораторных работ. В этом случае интерактивная модель полностью заменяет эксперимент.

В традиционном обучении интерактивные модели можно использовать до или после проведения эксперимента. В первом случае ученики наблюдают за поведением объектов модели, внося изменения в её параметры, анализируют модель, а затем выполняют лабораторную работу в реальных условиях. Во втором случае, после проведения реального эксперимента, ученики могут изменять те условия, которые невозможно изменить в условиях класса. Таким образом, происходит более глубокий анализ моделируемых явлений и процессов.

Целью данной работы является описание принципов разработки интерактивных компьютерных моделей с помощью Macromedia Flash. Данная работа предназначена для пользователей, умеющих работать с графикой и Action Script во Flash; может быть использована студентами как материал для написания курсовых и дипломных работ, преподавателями и учителями при создании интерактивных компьютерных моделей с помощью Macromedia Flash.

Данная работа состоит из введения, трех глав, выводов и библиографии. Глава 1 включает в себя рассмотрение особенностей Macromedia Flash как средства для создания интерактивных компьютерных моделей, его преимуществ и недостатков. В главе 2 рассматриваются примеры создания компьютерных моделей на материале по физике (математический маятник; движение тела, брошенного под углом к горизонту; движение тела по наклонной плоскости). Глава 3 представляет собой практические рекомендации по созданию интерактивных компьютерных моделей.

1. Выбор технологии Macromedia Flash для разработки интерактивных компьютерных моделей

Macromedia Flash является программной графической средой, предназначенной для удобной работы с векторной графикой и анимацией, в первую очередь предназначенной для использования в Web-документах. Также Macromedia Flash позволяет организовывать интерактивные проекты, используя Action Script.

Как уже отмечалось, Macromedia Flash обладает рядом преимуществом перед другими средствами создания интерактивных компьютерных моделей. Macromedia Flash – единственная среда со встроенным редактором для рисования графических объектов, которая позволяет также описывать отношения между ними с помощью встроенного объектно-ориентированного языка программирования Action Script. Таким образом, наличие Action Script дает инструментарий для создания компьютерных моделей.

Проекты, разработанные на Macromedia Flash, являются платформенно-независимыми, то есть поддерживаются различными компьютерными платформами. Это является очень важным при использовании компьютерных моделей в дистанционном обучении, основой, для которой является среда Интернет, так как учащиеся могут пользоваться различными компьютерными платформами. Также проекты, реализованные в среде Macromedia Flash, имеют малый размер файлов.

Рассмотрим некоторые характеристики Action Script.

Во-первых, он является объектно-ориентированным языком, что соответствует современному подходу к программированию. Видеоролик или символ во Flash считается объектом, если ему присвоено имя (instance name). В данной работе все символы будут иметь свое имя, поэтому символ часто будет называться объектом.

Во-вторых, он направлен на манипулирование роликами и учитывает все особенности Macromedia Flash как среды создания графики и анимации.

Рассмотрим некоторые недостатки среды Macromedia Flash как средства для создания моделей.

Несмотря на то, что Flash-проекты платформенно-независимы сам редактор для создания Flash роликов рассчитан только на Windows, что не позволяет разработчику

создавать модели в других операционных системах. Также Macromedia Flash является коммерческим продуктом. Еще одним недостатком Macromedia Flash является то, что он не позволяет разрабатывать трехмерные компьютерные модели. Также существует ряд недостатков, связанных с конкретными моделями и графическими объектами.

2. Разработка интерактивных компьютерных моделей

Рассмотрим технологию разработки интерактивных моделей на примере следующих физических задач: математический маятник, движение тела, брошенного под углом к горизонту и движение тела по наклонной плоскости. В каждой модели создаются символы (объекты), а отношения между ними описываются на языке Action Script.

2.1. Математический маятник

Простейшей интерактивной моделью является модель математического маятника. Рассмотрим тело, подвешенное на нити и совершающее затухающие колебания. Исходными данными являются: начальный угол наклона, декремент затухания (рис. 1.).

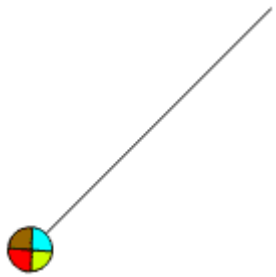

Рис. 1

Данная модель описывается следующим уравнением.

$\alpha = \alpha_0 e^{-\sigma t} \sin(wt+\Delta\varphi)$ – уравнение, описывающее изменение угла наклона с течением времени [5], где

α – угол наклона;

α_0 – начальный угол наклона;

σ – декремент затухания;

t – время;

Данная модель отображает качение маятника (в соответствии с уравнением движения математического маятника). Пользователь задает начальный угол наклона, декремент затухания через поля ввода. Также предусмотрены кнопка «Пуск», запускающая движение и «Стоп» для остановки.

В основном видеоролике создадим кнопку, запускающую качение маятника и разместим в ней программный код:

```
on (release) {                    // метод, вызываемый по нажатии на кнопку
    _root.pusk = true;            // переменной pusk (находится в коренном ролике) присваивается значение истина (запуск движения).
}
```

В основном видеоролике создадим кнопку, останавливающую качение маятника и разместим в ней программный код:

```
on (release) {
    _root.pusk = false;           // переменной pusk присваивается значение ложь (остановка движения).
}
```

В основном видеоролике создадим поля ввода начального угла наклона и декремента затухания (рис. 2).

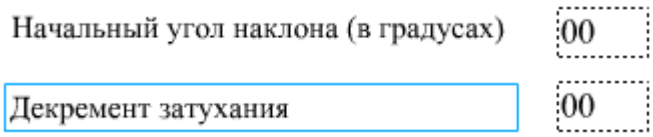

Рис. 2

Причем для текстового поля установим свойство input text (ввод текста) и свяжем с переменной num (num1 для коэффициента затухания) – по этой переменной передается значение этого поля в программный код (в панели character) (рис 3.).

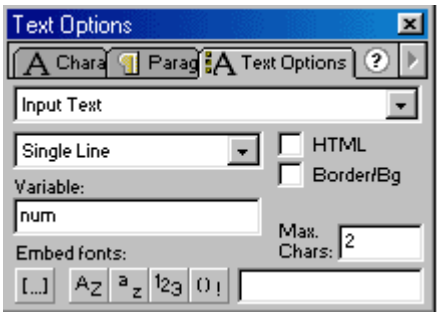

Рис. 3

Создадим объект-видеоролик маятник, разместим на нем программный код:

```
onClipEvent (enterFrame) {    //          метод события вызываемый при смене кадра
   if (!_root.pusk) {                  // если не запущено движение
      t = 0;              // время равно 0
     _root.may._rotation = _root.num;
// Считываем начальное отклонение из поля с переменной num

   } else {              // иначе
_root.may._rotation = Math.exp (-_root.num1*t)*_root.num*Math.sin(4*t+1.5);
// уравнение движения математического маятника

      t = t+0.03;                                    // меняем время
   }
}
```

2.2 Движение тела, брошенного под углом к горизонту

Рассмотрим движение тела, брошенное под углом к горизонту. Исходными данными являются: величина и угол наклона вектора начальной скорости. Построим траекторию движения тела (рис 4.).

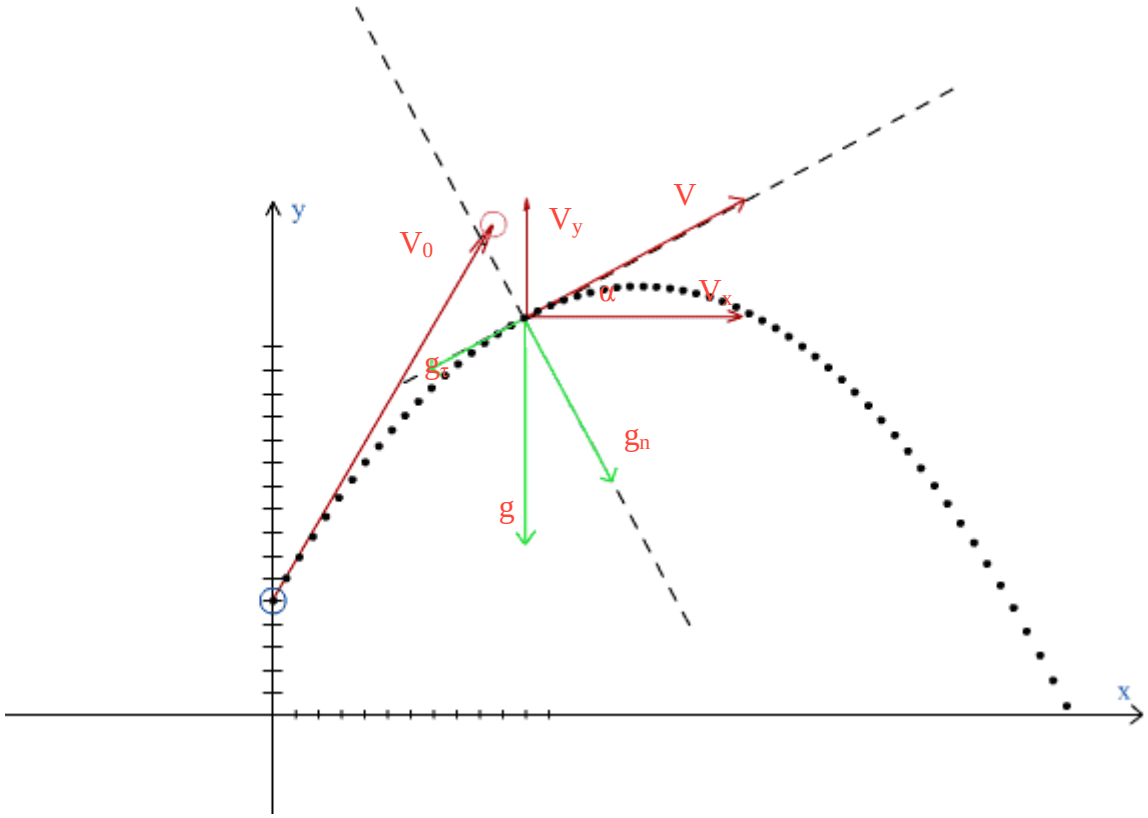

Рис. 4.

Данная модель описывается следующей системой уравнений [6] [3].

$$\begin{cases} y = y_0 + v_{0y}t - gt^2/2; \\ x = x_0 + v_{0x}t; \\ v_x = v_{0x}; \\ v_y = v_0 y - gt; \\ R_{кр} = |V|^2/g_n; \end{cases} \quad (1)$$

где

y – координата тела по оси Oy;

y_0 – начальное положение тела по оси Oy;

t – время;

g – ускорение свободного падения;

x - координата тела по оси Ox;

x_0 – начальное положение тела по оси Ox;

v_{0x} – начальная скорость по оси Ox;

v_{0y} – начальная скорость по оси Oy;

Rкр – радиус кривизны;

v – скорость;

g_n – нормальная составляющая ускорения g;

Данная модель демонстрирует траекторию тела, брошенного под углом к горизонту, где пользователь (ученик) изменяет величину, а также угол наклона начального вектора скорости. Причем с помощью красного кружка изменяется длина и угол наклона начального вектора скорости, а с помощью синего кружка перетаскивается параллельно весь вектор вдоль оси Oy. Следовательно, происходит изменение начальных свойств за счет перетаскивания мышью, что удобно для пользователя.

Также при наведении мышью на любую точку траектории отображаются касательные оси, вектор скорости, проекции вектора скорости, вектор ускорение свободного падения, проекции g_τ, g_n., а также радиус кривизны. И существует возможность выбора показывать или не показывать каждый из векторов.

2.2.1. Создание интерактивного вектора

Создадим вектор v_0 и два направляющих элемента (кружки). Направляющий элемент служит для изменения длины и направления вектора.

Интерактивный вектор является очень удобным, может быть использован для многих интерактивных компьютерных моделей и позволяет решать различные задачи. Создаем объект с именем, например sys (movie clip) и помещаем в него три элемента (movie clip): два кружка (ruler и yruler) и вектор (vector). Для того чтобы к этим элементам можно было обращаться из Action Script кода, следует задать их instance name (панель instance, поле name). В данной работе, имя объекта и его instance name будем делать одинаковыми (рис. 5.).

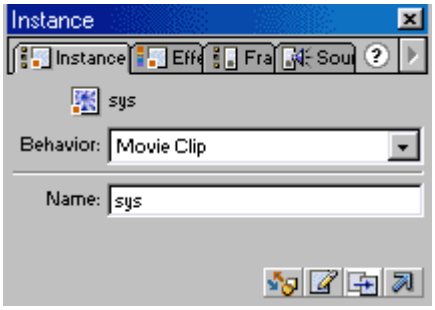

Рис. 5.

Объект sys используется как хранилище трех символов, для того чтобы при этом можно были перетаскивать мышью всю систему из трех элементов.

Action Script для vector:

```
onClipEvent (load) {                                    //при загрузке клипа
   var x1;                                              // задаем переменные
   var y1;
}
onClipEvent (enterFrame) {                              //при каждом входе в кадр
```

x1 = _root.sys.ruler._x; // считываем координаты

y1 = _root.sys.ruler._y;

ar = Math.atan2(y1,x1); // арктангенс от отношения y1, x1 (т.к. координаты yruler 0,0) задание угла через координаты ruler

a = ar*180/Math.PI; // перевод в градусы, т.к. свойство _rotation задается в градусах.

setProperty (_root.sys.vector, _rotation, 0); // обязательно перед тем как задавать изменяющиеся ширину и длину надо обнулить угол поворота.

setProperty (_root.sys.vector, _width, x1/Math.cos(ar)); // задание ширины вектора через угол

setProperty (_root.sys.vector, _rotation, a); // поворот на угол a.

}

Таким образом, задается ширина и поворот через координаты направляющего элемента ruler. Для этого начало рисунка вектора надо поместить в середину movie clip (рис. 6.). Также при растяжении векторов возникает погрешность, так как вектор пропорционально увеличивается и в ширину.

Рис 6.

Рассмотрим программный код для объекта ruler:

onClipEvent (mouseDown) { // если клик на данном объекте

if (this.hitTest(_root._xmouse,_root._ymouse,false)) { // hitTest() – метод, который возвращает истину, если объект пересекает область, заданную координатами (первые два параметра), третий параметр указывает включать ли в область всю форму объекта).

startDrag (_root.sys.ruler, true, 0, -180, 146, 180); // начало и ограничение перетаскивания внутри прямоугольника.

}

}

onClipEvent (mouseUp) { // конец перетаскивания когда отпускаем кнопку мыши

stopDrag ();

}

С помощью этого кода удобно организовывать перетаскивание мышью любого объекта.

Рассмотрим программный код для объекта yruler.

onClipEvent (mouseDown) {

if (this.hitTest(_root._xmouse,_root._ymouse,false)) {

startDrag (_root.sys, true, 46.5, 313, 46.5, _root.sys.ruler._y);

}

}

onClipEvent (mouseUp) {

stopDrag ();

}

Аналогичный код перетаскивания объекта, но внутри вертикальной линии.

Таким образом, параметры объекта vector (угол наклона, длину) можно использовать для дальнейшей работы с моделью.

2.2.2. Построение графика функции

В Macromedia Flash Action Script нет операторов для рисования примитивов, например линия, точка. Поэтому невозможно строить графики традиционным способом, однако данную проблему можно решить следующим способом.

Во Flash следует использовать метод дублирования элементов (например, объект – видеоролик изображающий пятнышко) с помощью duplicateMovieClip. А затем помещаем скопированный объект в определенную точку экрана, используя параметры _x, _y (координаты).

Также при построении учитывается то, что экранные координаты Flash не являются координатами модели (центр в левом верхнем углу, положительное направление Oy вниз, а оси Ox вправо).

В основном видеоролике создадим кнопку и разместим в ней программный код, результатом работы которого будет построение траектории движения тела.

Рассмотрим программный код для кнопки.

```
on (release) {
    mas = 10;                                   // переменная (масштаб)
    t = 0;                                      // обнулим параметр (время)
    y0 = (-sys._y+313)/mas;                     // задаем y0 как у элемента sys (содержащий вектор и направляющие) причем с учетом  экранной системы координат.
    v0x = (sys.vector.x1)/mas;                  // задаем v0x, обращаемся к переменной x1, находящейся в коде элемента sys.vector.x1 (с учетом масштаба)
    v0y = (-sys.vector.y1)/mas ;                // задаем v0y, обращаемся к переменной y1, находящейся в коде элемента sys.vector.y1
    while (t<4.5) {                             // запускаем цикл (параметр t)
```

duplicateMovieClip (_root.plot, "mc"+i, i); // оператор создающий копию объекта _root.plot; "mc"+i – новое имя созданного объекта; i – номер уровня (виртуального слоя, в который помещается скопированный объект). Причем важно, чтобы каждый новый скопированный элемент помещался в отдельный слой, чтобы не происходило замещение одного объекта другим.

```
y = y0+v0y*t-0.5*9.8*t*t;    //уравнение равноускоренного движения
x = v0x*t;
if (y>0) {                          // изображаем, пока не тело не достигло земли
   setProperty ("mc"+i, _x, mas*x+46.5);
   setProperty ("mc"+i, _y, -mas*y+313);
```

// помещаем скопированный объект в определенную точку экрана, используя свойство _x, _y (координаты) с учетом экранной системы координат.

```
}
t = t+0.06;                         // изменение времени
i = i+1;                // изменение переменной, использующейся в  duplicateMovieClip
  }
}
```

Для того чтобы при наведении мышью на любое положение тела отображать вектор скорости, проекции вектора скорости, вектор g, проекции вектора ускорения свободного падения, вектора g_τ, g_n. а также радиус кривизны, следует обращаться к большому количеству дублированных элементов. Если обращаться к ним извне и использовать абсолютную адресацию (например _root.mc14._x) это очень трудоемкий процесс, из-за их количества. Гораздо удобнее задавать программный код внутри каждого дублированного объекта и использовать относительную адресацию (например, this._x). Причем программный код дублированных элементов не сохраняется, если он находится на элементе (правая кнопка -> actions). Следовательно, программный код задается внутри объекта (на кадре внутри него). Используется трехкадровая система. В первом кадре объявляются переменные, задаются их начальные значения; во втором кадре находится

основной программный код; в третьем возврат на второй. Таким образом, обновление происходит не по загрузке movie clip (enterFrame), а по возврату во второй кадр.

В данной модели следует предусмотреть кнопки, предназначенные для того, чтобы показывать или не показывать каждый из векторов.

Эту задачу удобно решить следующим образом. Для копируемого объекта точка (plot) создается набор векторов, касательные оси, а их параметры (длина, угол наклона) изменяются в соответствии с координатами дублированного элемента (точки), зададим их свойство _visible (видимость) false, а по наведении мышью поменяем на true (объекты будут появляться при наведении мышью).

Для этого создаем объекты внутри plot. Причем каждую группу элементов в своем слое (и отдельный слой, для того чтобы в его кадры поместить программный код).

Зададим имена для объектов:

g — объект – хранилище vx и vy;

vx, vy — вектора проекции вектора v;

gg — вектор g;

pl — объект – хранилище g_τ, gn, vsk, а также касательных осей координат.

g_τ, gn — вектора проекции вектора g;

vsk — вектор скорости;

pl1 — объект (рисунок точки);

Структурная схема размещения объектов модели изображена на рисунке 7.

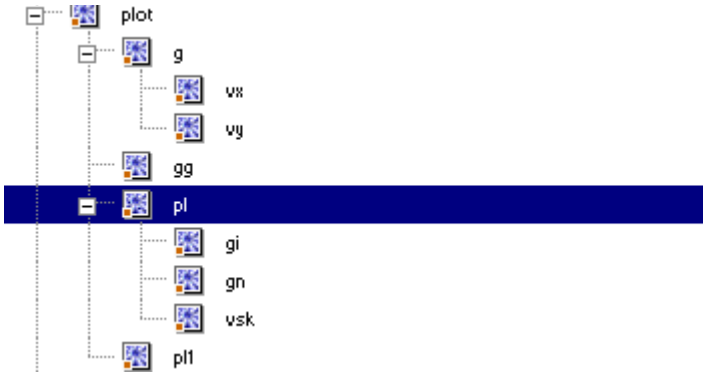

Рис. 7

В верхний слой поместим объект pl1, так как сам рисунок всегда будет видимым. На слой ниже поместим объект pl. Еще на слой ниже поместим g и gg. А в слое программного кода будет три кадра.

Рассмотрим программный код для 1-го кадра.

Здесь объявляются переменные и задаются их начальные значения.

```
mas = 10;                                   // переменная (масштаб)

v0x = (_root.sys.vector.x1)/mas;            // вычисляем v0x, обращаемся к
        переменной x1, находящейся в коде элемента sys.vector.x1 (с учетом масштаба)

 v0y = (-_root.sys.vector.y1)/mas ;         // вычисляем v0y,
            обращаемся к переменной y1, находящейся в коде элемента sys.vector.y1

x = (this._x-46.5)/mas;                     // координата x считывается относительно
положения данного дублированного объекта (с учетом масштаба и экранной системы
координат)

t = x/v0x;                                  // выражаем t из (1.)

vy = v0y-9.8*t;                             // см. (1.)

ar = Math.atan2(vy, v0x);                   // вычисляем угол наклона через арктангенс
(для pl)

a = -ar*180/Math.PI ;                       // переводим в градусы
```

setProperty (pl, _rotation, a); // устанавливаем угол наклона объекта pl (касательные оси координат)

setProperty (pl.vsk, _width, Math.sqrt(v0x*v0x+vy*vy)*mas); // устанавливаем длину вектора скорости(по теореме Пифагора см. (1.))

setProperty (g.vx, _width, v0x*mas); // устанавливаем длину вектора как v0x см.(1.)

setProperty (pl.gn, _height, Math.cos (ar)*98-5); // устанавливаем длину вектора gn см.(1.)

rcr = pl.vsk._width*pl.vsk._width/pl.gn._height; // устанавливаем радиус кривизны.

if (ar>=0) { // если угол наклона касательных осей координат >= 0 (когда угол >0 Vy направлен вверх, а g_τ влево ,

иначе Vy направлен вниз, а g_τ вправо)

setProperty (g.vy, _height, vy*(mas+1.4)); // устанавливаем длину вектора vy

setProperty (pl.gi, _width, Math.sin (ar)*98); // устанавливаем длину вектора g_τ

} else {

setProperty (g.vy, _rotation, -180); // разворачиваем на 180 градусов

setProperty (g.vy, _height, -vy*(mas+1.4)); // устанавливаем длину вектора vy

setProperty (pl.gi, _rotation, 180); // разворачиваем на 180 градусов

setProperty (pl.gi, _width, -(Math.sin (ar)*98)); // устанавливаем длину вектора g_τ

}

Action Script для 2-го кадра:

if (pl1.hitTest(_root._xmouse, _root._ymouse, false)) { // если наводим мышью на объект pl1

setProperty (pl, _visible, true); // делаем pl видимым

setProperty (pl.vsk, _visible, _root.c); // делаем vsk видимым, если переменная c = true (см. кнопки переключения видимости для каждого вектора)

setProperty (g, _visible, _root.c1); // делаем g видимым, если переменная c1 = true

setProperty (pl.gn, _visible, _root.c2); // делаем gn видимым, если переменная c2 = true

setProperty (gg, _visible, _root.c3); // делаем gg видимым, если переменная c3 = true

setProperty (pl.gi, _visible, _root.c4); // делаем $g_т$ видимым, если переменная c4 = true

_root.num4 = Math.round (this.rcr)/10; // присвоим переменной поля, отображающего радиус кривизны для данного дублированного элемента (с учетом масштаба и округления).

}

Рассмотрим программный код для 2-го кадра.

gotoAndPlay (2); // возвращаемся на второй кадр для того, чтобы контролировать наведение мыши в течение всей работы модели.

Рассмотрим программный код для объекта pl.

setProperty (this, _visible, false); // если не наводим мышью, то объект невидим. Причем этот кадр следует растянуть и на второй для того, чтобы происходило обновление

Action Script для 1-го кадра объекта g.

setProperty (this, _visible, false); // если не наводим мышью, то объект невидим. Причем этот кадр следует растянуть и на второй для того, чтобы происходило обновление

Рассмотрим программный код для объекта gg.

setProperty (this, _visible, false); // если не наводим мышью, то объект невидим. Причем этот кадр следует растянуть и на второй для того, чтобы происходило обновление.

Изображение всех векторов затрудняет анализ модели обучаемым, следовательно, следует организовать возможность выбора показывать или не показывать каждый из векторов следует создать поле с именем элемента, кнопку-переключатель, поле «ДА» «НЕТ» (рис. 8.).

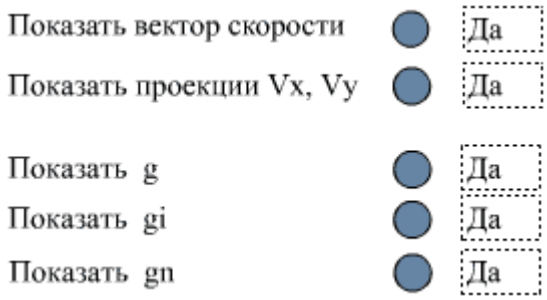

Рис. 8.

Создадим кнопку и скопируем 4 раза. Для полей «ДА» «НЕТ» установим Dynamic Text и переменные num1, num2, num3, num5, num6(рис 9.).

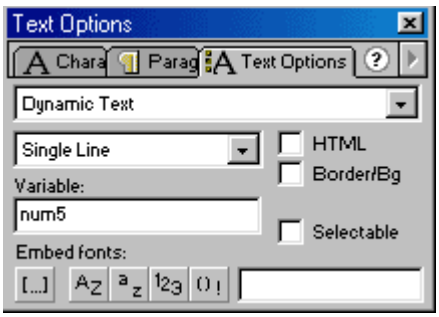

Рис. 9.

В первом кадре основного клипа поместим программный код:

c = true;

c1 = true;

c2 = true;

c3 = true;

c4 = true; // вычисляем начальные значения переменных, предназначенных для организации взаимодействия между кнопкой и поля «ДА» «НЕТ».

Рассмотрим программный код для кнопок.

Для первой кнопки:

on (release) {

_root.c = !_root.c; // при нажатии на кнопку если true, то присваиваем false, если false, то присваиваем true

if (_root.c) { // переменной поля присваиваем значения «ДА» или «НЕТ» в зависимости от значения переменной с.

num1 = "Да";

} else {

num1 = "Нет";

}

}

Далее программный код аналогичен.

Для второй кнопки:

on (release) {

_root.c1 = !_root.c1;

if (_root.c1) {

```
        num2 = "Да";
    } else {
        num2 = "Нет";
    }
}
```

Для третьей кнопки:

```
on (release) {
    _root.c3 = !_root.c3;
    if (_root.c3) {
        num5 = "Да";
    } else {
        num5 = "Нет";
    }
}
```

Для четвертой кнопки:

```
on (release) {
    _root.c4 = !_root.c4;
    if (_root.c4) {
        num6 = "Да";
    } else {
        num6 = "Нет";
    }
```

```
}
```

Для пятой кнопки:

```
on (release) {
    _root.c2 = !_root.c2;
    if (_root.c2) {
        num3 = "Да";
    } else {
        num3 = "Нет";
    }
}
```

Также для радиуса кривизны установим динамическое поле с переменной num4, отображающее радиус кривизны для каждой скопированной точки (см. программный код для 1-го кадра plot)(рис 10.).

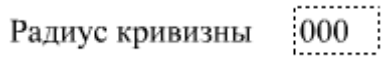

Рис. 10.

Следовательно, при нажатии на определенную кнопку меняется видимость вектора, а также значение поля «ДА» «НЕТ».

2.3 Движение тела по наклонной плоскости

Рассмотрим движение тела по наклонной плоскости. Исходными данными являются: коэффициент трения, угол наклона плоскости (рис. 11.).

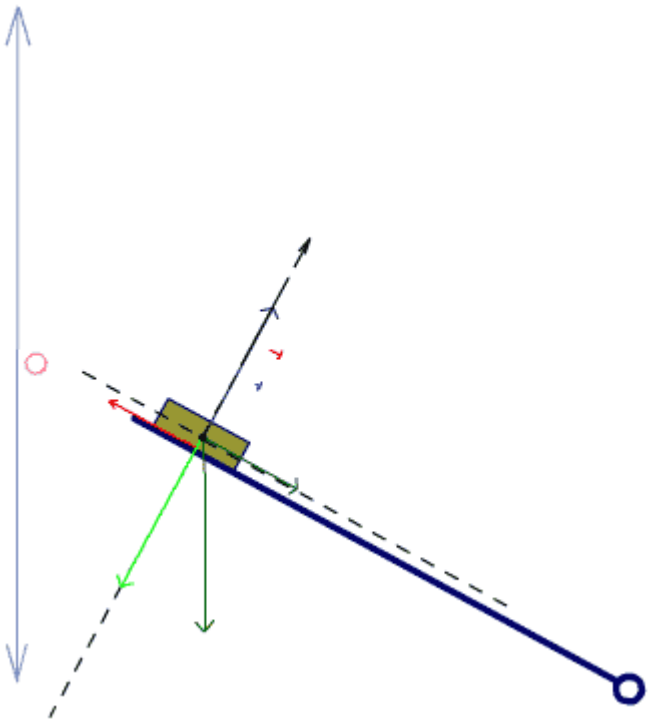

Рис. 11.

Данная модель описывается следующей системой уравнений [4].

$F_{тр}=\mu N$

$N=|mg|\cos\alpha$

$$\begin{cases} Ox: -F_{тр}+mg*\sin\alpha=ma \\ Oy: -N-mg*\cos\alpha=0 \end{cases} \Rightarrow -\mu N + N + mg(\sin\alpha - \cos\alpha) = ma$$

$$a= g (\sin\alpha -\mu*\cos\alpha)$$

$$\begin{cases} x = x0+v0t+at2/2 \quad (2) \\ v= v0+at \end{cases}$$

где

t – время;

g – ускорение свободного падения;

x - координата тела по оси Ox;

x_0 – начальное положение тела по оси Ox;

v_0 – начальная скорость;

v – скорость;

Fтр – сила трения;

μ – коэффициент трения;

N – реакция опоры;

m – масса тела;

Данная интерактивная модель демонстрирует реалистично двигающееся тело (в соответствии с системой уравнений, описывающей движения по наклонной плоскости). Причем угол наклона изменяется с помощью направляющего элемента (синий кружок), который двигается вертикально. Также отображаются угол наклона плоскости, вектора g, N, V, проекция g на наклонную ось x в соответствии с законом движения тела. Коэффициент трения пользователь задает в поле (с переменной num). Существует возможность перетаскивать тело по плоскости.

2.3.1. Создание направляющего элемента

Создадим рисунок, и сделаем его символом с именем, например, ruler. Поместим на него программный код.

```
onClipEvent (mouseDown) {
    if (this.hitTest(_root._xmouse, _root._ymouse, false)) {
        startDrag (_root.ruler, true, 200, 300, 200, 1);          // перемещение внутри
вертикальной оси.
    }
}
onClipEvent (mouseUp) {
    stopDrag ();
}
```

(аналогичный программный код для перетаскивания элементов в предыдущей модели)

Для того чтобы угол наклона плоскости изменялся в соответствии с положением направляющего элемента, создадим объект, содержащий плоскость, тело, набор векторов (имя объекта sys). Поворачиваться будет весь объект sys, что удобно для задания движения (движение будет происходить горизонтально относительно sys, и под углом относительно внешнего ролика). Поместим центр объекта в правый нижний его угол, а плоскость и тело в соответствии с рисунком (рис 12.)..

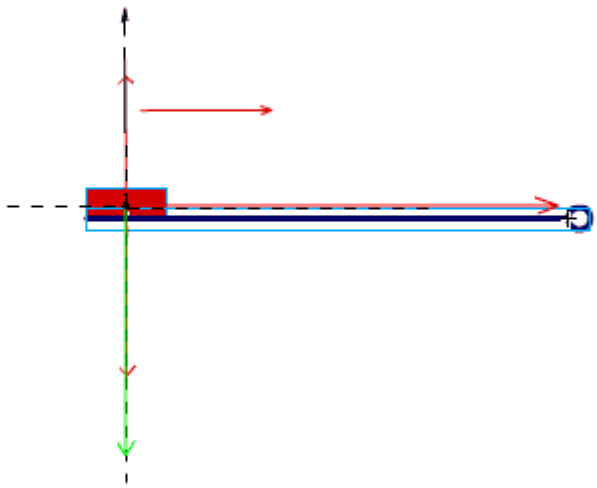

Рис. 12.

Поле для коэффициента трения (рис .13)

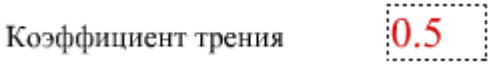

Рис .13

2.3.2. Организация движения

Добавим весь программный код движения в объект ruler.

```
onClipEvent (load) {                    // при загрузке вычисляем начальные значения переменных
 t = 0;                                 // время равно 0
 xn = _root.sys.telo._x;                // начальное перемещение соответствует положению тела
 v0 = 0;                                // начальная скорость равна 0
 v = 0;                                 // скорость равна 0
}
onClipEvent (enterFrame) {              // при каждом входе в кадр
 if ((_root.sys.telo._x<0)&&(_root.sys.telo.h)) {  // если тело не достигло конца плоскости и его не перетаскивают (см. код для telo), то реализуем движение
  x1 = _root.ruler._x-root.sys._x;
  y1 = -_root.ruler._y+_root.sys._y;                // высчитаем разность между координатами направляющего элемента и начала ролика sys для вычисления угла поворота sys через арктангенс.
  ar = Math.atan2(y1, x1);                  // узнаем угол
  a = ar*180/Math.PI;                       // перевод в градусы
  setProperty (_root.sys, _rotation, a);    // поворот на этот угол
  m1 = _root.num;                           // считаем коэффициент трения из поля
  usk = 9.8*(Math.sin(ar)-m1*Math.cos(ar)); // ускорение см. (2.)
  if (usk!=oldusk) {                        // таким образом проверяем поменялся ли угол наклона во время движения тела (старое ускорение не равно новому (ускорение напрямую зависит от угла))
```

```
    xn = _root.sys.telo._x;                    // начальное перемещение снова соответствует положению тела
    v0 = v;                                    // начальная скорость равна v
    t = 0;                                     // время равно 0
    }
     t = t+0.05;                               // при каждом входе в кадр обновляем время
     if ((usk>=0)||(v>=0)) {                   // условие движения тела (тело остановится когда ускорение <0 и скорость <0
        _root.sys.telo._x = xn+v0*t+(usk*t*t)/2;      // закон движения см. (2.)
        v = v0+usk*t;         // закон движения см. (2.)
        oldusk = usk;        // старое ускорение (для контроля изменения ускорения)
     }
  }
  }
```

2.3.3. Отображение векторов

Подход к отображению векторов в данной модели такой же, как и в предыдущей.

Создадим объект plot, который включает в себя все вектора.

g — вектор;

pl — хранилище элементов gn, vsk;

gn — проекция g на наклонную ось Oy;

vsk — проекция g на Ox;

vecsk — вектор скорости движения тела;

Структурная схема размещения объектов модели изображена на рисунке 14.

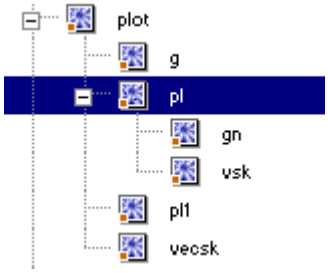

Рис. 14.

В верхний слой поместим объект pl1. На слой ниже поместим объект pl. Еще на слой ниже поместим g, рисунок вектора N, вектор ускорения и vecsk. А в слое программного кода будет два кадра.

Рассмотрим программный код для объекта plot.

```
ar = _root.ruler.ar;                                    // считываем угол (переменная ar) объекта ruler

setProperty (pl.vsk, _width, Math.sin(ar)*98);          // установим ширину vsk через синус угла (см. (2.))

setProperty (g, _rotation, -_root.ruler.a);             // вектор g не меняет угол вместе с sys (поворачиваем его обратно на тот же угол)

setProperty (pl.gn, _height, Math.cos(ar)*g._height);   // установим длину gn через угол и длину вектора g.

setProperty (vecsk, _visible, false);                   // если тело не двигается, вектор скорости не виден.

setProperty (usk, _visible, false);                     // если тело не двигается, вектор ускорения не виден.

    this._width = _root.num*65;                         //  длина вектора силы трения

if (_root.ruler.v>0) {                                  // если скорость больше 0
```

setProperty (vecsk, _visible, true); // если тело двигается, вектор скорости виден.

setProperty (vecsk, _width, _root.ruler.v); // установим длину в соответствии с переменной скорости (в объекте ruler)

if (_root.ruler.usk>0) {

setProperty (usk, _visible, true);

usk._width = _root.ruler.usk*5; // установим длину в соответствии с переменной скорости (в объекте ruler)

}

}

Action Script для 3-го кадра plot:

gotoAndPlay (1);

Также следует поместить весь набор векторов в соответствии с положением тела

Action Script для объект plot:

onClipEvent (enterFrame) {

setProperty (this, _x, _root.sys.telo._x+1);

setProperty (this, _y, _root.sys.telo._y-7); // устанавливаем координаты набора векторов на тело.

}

и текстовое поле для ввода коэффициента (input text с переменной num) (рис 15.).

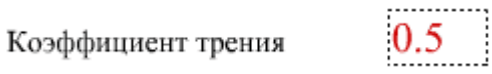

Рис. 15.

Для предоставления возможности перетаскивания тела по наклонной плоскости на объект тело (sys.telo) поместим следующий программный код.

```
    onClipEvent (mouseDown) {
        if (this.hitTest(_root._xmouse, _root._ymouse, false)) {
            startDrag (this, true, -220, 0, 0, 0);          // перетаскиваем вдоль плоскости
            h = false;                  // когда перетаскиваем, движения не происходит (см. код для ruler)
        }
    }
    onClipEvent (mouseUp) {
        stopDrag ();
        h = true;
        if (this.hitTest(_root._xmouse, _root._ymouse, false)) {
            _root.ruler.t = 0;
            _root.ruler.v0 = 0;
            _root.ruler.xn = this._x;        // в конце перетаскивания задаем начальные параметры движения
        }
    }
    onClipEvent (load) {
        h = true;                           // начальное значение h.
```

3. Практические рекомендации по разработке интерактивных компьютерных моделей

В ходе работы над созданием интерактивных компьютерных моделей, описанных во второй главе, можно выделить некоторые принципы и подходы создания интерактивных компьютерных моделей в среде Macromedia Flash.

3.1 Различные подходы к организации интерактивности компьютерной модели

При создании интерактивных компьютерных моделей необходим диалог с пользователем, при котором модель получает начальные данные и в соответствии с ними изменяется ее поведение. Во-первых, пользователь может вводить информацию в поля ввода. Во-вторых, изменять начальные условия с помощью перетаскивания мышью различных элементов.

Первый случай является наиболее простым:

Создаем текстовое поле для ввода информации. Устанавливаем свойство input text и переменную (здесь num) – по этой переменной передается значение этого поля в программный код (в панели Text Options). В поле Max Chars устанавливаем максимальное количество символов данного поля. Контроль над значением переменной осуществляется через программный код по имени переменной (рис. 17.).

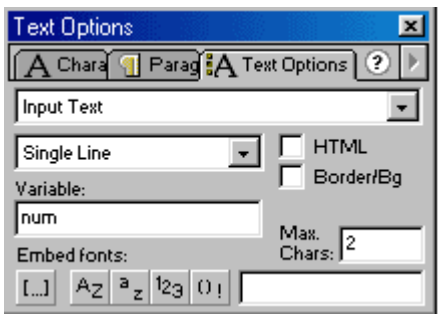

Рис. 17

Во втором случае удобно использовать направляющие элементы (ruler) и программный код, предназначенный для перетаскивания мышью (описан в моделях движение тела по наклонной плоскости, а также тело, брошенное под углом к горизонту). Таким образом, пользователь взаимодействует с моделью, перетаскивая эти элементы. Изменение работы модели происходит за счет контроля координат (параметры ._x и ._y) направляющего элемента. Например, во второй модели динамически изменяются длина и направление начального вектора скорости при перетаскивании мышью элемента ruler (так как длина и направление зависят от координат ruler). Также для контроля перетаскивания важно использовать метод **объект.hitTest()** (возвращающий истину, если нарисованная область объекта пересекает область другого объекта или определённую точку экрана). К тому же это позволяет перетаскивать тот элемент, на который наведена мышь.

При создании моделей физических явлений удобно использовать уже готовый объект интерактивный вектор (описанный во второй главе) для задания начального движения.

Также следует организовать вывод информации о результатах работы модели. Здесь в определенных случаях также можно использовать динамические текстовые поля и передавать туда значения через переменную поля. Устанавливаем в панели Text Options свойство Dynamic Text (как при отображении радиуса кривизны во второй модели).

Также результатом работы будут являться различные визуальные изменения (перемещение, появление, изменение скорости объектов), в соответствии программному коду.

3.2. Два способа размещения программного кода для организации жизненных циклов моделей

Существуют два способа размещения программного кода: на объекте (правая кнопка -> Actions) и на кадрах (трехкадровая система).

В первом случае выделяется два блока.

Первый блок:

onClipEvent (load) {}

Программный код, помещенный в этот блок, выполняется при первой загрузке данного объекта (ролика), то есть один раз вначале. Поэтому этот блок используется для начальной инициализации переменных.

Второй блок

onClipEvent (enterFrame) {}

При проигрывании ролика генератор постоянно загружает его заново, поэтому по событию смены кадра (enterFrame) будет происходить постоянное обновление программного кода, то есть здесь удобно размещать основные операторы, контроль над перемещениями объектов, мыши. Также важно, что такой блок является почти единственным способом организации больших циклов во Flash (см. вторую модель), т.к. обыкновенная организация циклов часто заставляет «зависать» Flash генератор.

При размещении программного кода на кадрах (внутри объекта) используется трехкадровая система. В первом кадре происходит начальная инициализация переменных (как в блоке с load). Во втором кадре размещается основной код, и организуются циклы (как в блоке с enterFrame), так как в третьем кадре происходит возврат на второй.

Оба способа нужны для разных целей. Например, второй способ является единственным способом сохранения кода при дублировании элементов, так как программный код, помещенный на объект, при дублировании не сохраняется.

3.3. Принцип построения графиков в среде Macromedia Flash

Как уже было отмечено, в Action Script нет операторов для рисования точек и других графических элементов. Поэтому невозможно строить графики обычным способом.

Во Flash следует использовать метод дублирования точек, используя метод duplicateMovieClip. Этот метод создает дубликат объекта, его синтаксис следующий:

duplicateMovieClip (имя_копируемого_объекта, новое_имя_дублированного объекта, уровень_(слой)_на_который_помещается_новый_объект);

Например: duplicateMovieClip (_root.plot, "mc"+i, i);

где i – переменная, которая обновляется при каждом входе в кадр, поэтому имя каждого нового элемента будет уникально, и он будет помещаться на свой слой. Если каждый скопированный объект не будет находиться на своем слое, то происходит замещение объектов (скорее всего они не будут видны). Поэтому всегда следует помещать каждый объект на свой уникальный слой.

А затем помещаем скопированный объект (точка) в определенную точку экрана, используя свойство _x, _y (координаты дублированного объекта). Соответственно в функции будет зависимость координаты y (свойство _y) от x (свойство _x). При этом важно учитывать, что экранная система координат не совпадает с системой координат модели (центр в левом верхнем углу, положительное направление Oy вниз, а оси Ox вправо). Также следует учитывать масштаб.

3.4. Организация движения и взаимодействия объектов

При разработке интерактивных компьютерных моделей всегда приходится организовывать движение объектов, контролировать их перемещение, организовывать и контролировать перетаскивание элементов пользователем, а также их взаимодействие друг с другом (рис. 18.).

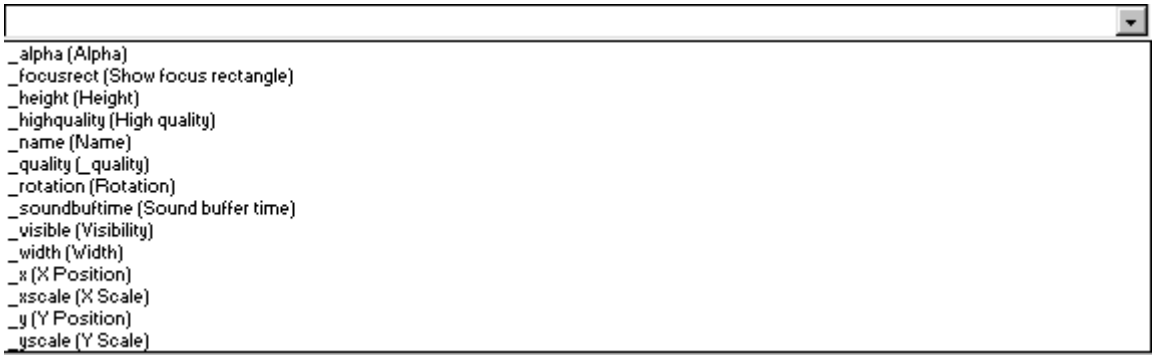

Рис. 18.

Задание движения графического объекта происходит в соответствии с законом его движения. То есть сначала вычисляются координаты, исходя из физических формул, а затем параметрам _x и _y присваиваются значения с учетом масштаба и экранной системы координат. Контроль над перемещением производится считыванием свойств _x и _y, а также других свойств объекта (основные параметры перечислены при использовании оператора setProperty).

При использовании свойств ширины и длины ролика (_width, _height), следует помнить об угле наклона (_rotation) этого объекта. Следует его обнулять для того, чтобы работать с горизонтально расположенным роликом.

При контроле над взаимодействием объектов друг с другом удобно использовать метод объект.hitTest(), возвращающий истину, если нарисованная область объекта пересекает область другого объекта

Для создания элемента, который пользователь будет перетаскивать, чаще всего используется программный код, описанный в моделях этой работы. Этот направляющий элемент можно эффективно использовать для создания интерактивных элементов с динамически изменяющейся длиной и направлением (см. создание интерактивного вектора).

Естественно весь программный код, контролирующий взаимодействие и перемещение объектов должен находиться в блоке onClipEvent (enterFrame) {}, иначе операторы не будут выполняться в течение всей работы модели.

Также для организации движения и взаимодействия объектов удобно использовать направляющие элементы. Они служат для изменения отображения объектов модели (см. модели).

3.5. Избирательное отображение элементов модели

Во время работы некоторых моделей отображается большое количество элементов, что затрудняет анализ модели пользователем (учащимся). Таким образом, следует организовать возможность выбора показывать или не показывать каждый из элементов.

Для этой цели удобно создавать поля с кнопками включения и выключения определенных объектов, как это организовано в модели **тело, брошенное под углом к горизонту**. Вид таких полей может быть следующим (см. рис. 8). На первом месте текстовое поле с описанием элемента, который следует включить или отключить. Далее сама кнопка, которая меняет значение логической переменной, отвечающей за отображение элемента (параметр _visible). И на третьем месте текстовое поле «ДА» «НЕТ», отображающее виден элемент или нет.

В программном коде по смене кадра (enterFrame) присвоим параметру _visible значение логической переменной, отвечающей за отображение соответствующего элемента.

Выводы

В данной работе были рассмотрены модели: математический маятник; движение тела, брошенного под углом к горизонту; движение тела по наклонной плоскости. Были проанализированы особенности Macromedia Flash как средства для создания интерактивных компьютерных моделей, его преимуществ и недостатков.

Также рассмотрены практические рекомендации по созданию моделей в Macromedia Flash, выявлены некоторые подходы и принципы их разработки.

Данная работа может быть использована преподавателями и учителями при создании интерактивных компьютерных моделей, студентами как материал для написания курсовых и дипломных работ. Модели, описанные в работе, могут применяться на уроках физики.

В результате сделан вывод о том, что с некоторыми ограничениями Macromedia Flash можно рекомендовать для разработки интерактивных компьютерных моделей. При разработке компьютерных моделей с помощью Macromedia Flash упрощается работа с графикой. Также можно менять графическое оформление объектов, не изменяя программный код. Кроме того, можно сделать вывод о том, что среда Macromedia Flash удобна для организации интерактивности компьютерной модели.

В ходе работы были выявлены следующие недостатки Macromedia Flash: невозможность создания трехмерных моделей, проблемы с некорректным масштабированием объектов.

Литература.

1. Рекомендации научно-методического симпозиума «Компьютерное моделирование в обучении точным наукам»// Педагогическая информатика. 2004, №1.

2. Бутиков Е.И. Интерактивные компьютерные модели в преподавании физики// «Компьютерное моделирование 2003»: Труды 4-й Международной научно-технической конференции. СПб.: Нестор, 2003.

3. Касьянов В.А. Физика 10 класс: Учебник для общеобразовательных учреждений. М.: Дрофа. 2003.

4. Рейнхардт Р., Лотт Джой Macromedia Flash MX Action Script М., СПб., Киев: Диалектика 2003.

5. Савельев С.И. Курс общей физики. Т. 1. М.: Наука, 1986.

6. Трофимова Т.И. Курс физики. М.: Высшая Школа 1999.