Построение пользовательских интерфейсов

Глава 2

В этой главе вы научитесь строить пользовательские интерфейсы на примере создания свитка и его заполнения элементами управления пользовательского интерфейса. Построенный интерфейс вам предстоит дополнить обработчиками событий для вызова отдельных частей сценария с помощью элементов управления пользовательского интерфейса.

Задание

После изучения этой главы вы должны уметь следующее.

- Создавать перемещаемые диалоговые окна или свитки утилиты
- Вводить в свиток элементы управления пользовательского интерфейса
- Организовывать реакцию сценария на пользовательский ввод

Введение

В MAXScript имеются средства для создания специализированных пользовательских интерфейсов, которые формируются при выполнении сценария. Пользовательский ввод можно получить из свитков и диалоговых окон, а затем выполнить на его основании соответствующие команды. В этой главе описывается порядок создания элементов пользовательского интерфейса, связанных со сценарием.

Построение пользовательских интерфейсов

До сих пор вам приходилось писать сценарии, которые действуют без пользовательского ввода. Для организации ввода данных и их последующего использования в сценарии последний может быть дополнен элементами пользовательского интерфейса.

Типы пользовательских интерфейсов

В сценарии могут быть сформированы два типа интерфейсов.

- Свитки на панели Utilities
- Перемещаемое диалоговое окно

Исключение составляет *сценарный подключаемый модуль*, представляющий собой специальный сценарий, способный формировать свитки в других частях пользовательского интерфейса 3ds Max.

Виды сценариев

Существует несколько видов сценариев. Их можно приближенно разделить по типу пользовательского интерфейса, который они формируют.

- Сценарная функция — состоит из одной или более функций, но не формирует пользовательский интерфейс. После выполнения сценария такие функции можно вызывать из приемника команд или другого сценария.

- Сценарная утилита — это сценарий, определяющий один или более свитков на панели Utilities в качестве элементов своего пользовательского интерфейса.
- Сценарный подключаемый модуль — это специальный вид сценария, создающий новые инструменты 3ds Max или расширяющий уже существующие. Во всех остальных видах сценариев применяется код для выполнения действий, которые уже доступны из пользовательского интерфейса 3ds Max, например создание параллелепипедов и сфер, назначение контроллеров и конфигурирование видовых окон. Сценарный подключаемый модуль позволяет создавать новые геометрические объекты, карты, модификаторы и прочие элементы оформления сцены. Пользовательский интерфейс нового инструмента появляется в соответствующей части интерфейса 3ds Max. Например, новая карта появляется в окне Material/Map Browser, тогда как новый модификатор — в списке Modifier List.
- Общие сценарии — любые сценарии, которые могут быть отнесены к категории общих. Такие сценарии позволяют сформировать перемещаемое диалоговое окно в качестве элемента пользовательского интерфейса либо вообще отказаться от интерфейса. Например, можно написать сценарий, создающий ряд сфер аналогично сценариям, приведенным в главе 1. Такой сценарий не будет формировать пользовательский интерфейс.

Макросценарии

Достаточно ввести несколько строк кода в начале общего сценария, чтобы превратить его в исполнительный элемент в диалоговом окне Customize User Interface (Специальная настройка пользовательского интерфейса). Такой элемент можно ввести на панели инструментов или в другом месте пользовательского интерфейса, чтобы вызывать оттуда сценарий. А кроме того, сценарий можно вызвать или выполнить с помощью назначенного клавиатурного эквивалента команды.

Подобный сценарий называется *MacroScript* (т.е. макросценарий). Он позволяет сформировать пользовательский интерфейс и определить его функции, хотя это и не обязательно. Это просто общий сценарий с командой, которая превращает его в исполнительный элемент.

Название MacroScript происходит от основного назначения макросценария, который призван помочь пользователям, не знакомым с MAXScript, создавать сценарии из кода, отображаемого в приемнике команд. Так, неоднократно повторяющуюся последовательность команд можно выполнить вручную лишь один раз, чтобы вывести ее в зарегистрированном средствами Macro Recorder виде в верхнем (розовом) подокне приемника команд. А затем зарегистрированную последовательность команд можно скопировать и вставить в окне редактора MAXScript Editor, выполнить полученный в итоге сценарий и далее работать с его исполнительным элементом в диалоговом окне Customize User Interface.

Для превращения любого общего сценария в MacroScript достаточно сделать одно из двух.

- Ввести нужный код в начале сценария, чтобы обозначить его как макросценарий, а затем выполнить сценарий.

- Выделить содержимое сценария в редакторе MAXScript Editor (или в любом другом месте 3ds Max, где отображается код, например в приемнике команд). При этом автоматически формируется внутреннее имя макросценария, например `Macro1`, `Macro2` и т.д.

Файлы сценариев

Как правило, сценарий сохраняется в файле с расширением `.ms`. Для хранения сценариев в организованном порядке макросценарии следует сохранять в файлах с расширением `.mcr`. Формально оба вида сценариев можно сохранять в файле с любым из двух указанных расширений, и при этом сценарий будет выполняться. Однако правильно выбранное расширение указывает на конкретный вид используемого сценария.

Сценарий можно выполнять автоматически при запуске 3ds Max. Для организации такого режима работы сценария достаточно поместить сценарий в любую из следующих папок.

- `3dsmax/stdplugs/stdscripts`
- `3dsmax/plugins` или любую подпапку внутри папки с подключаемыми модулями
- `3dsmax/ui/macroscripts` или в любую другую подпапку
- `3dsmax/scripts/startup`

При запуске 3ds Max программа запуска осуществляет поиск файлов с расширениями `.ms`, `.mcr` и `.mse` в папках, перечисленных в указанном выше порядке. (Расширение `.mse` используется для шифрованных файлов сценариев. Более подробные сведения по данному вопросу приведены в разделе “Encrypting Script Files” (Шифрование файлов сценариев) справочного руководства по MAXScript.)

Свитки

Свитки составляют саму основу любого пользовательского интерфейса, формируемого средствами MAXScript. Прежде чем создавать какие-либо другие элементы пользовательского интерфейса, в том числе флажки и счетчики, необходимо сформировать хотя бы один свиток, в котором они будут находиться. Под термином “свиток” в MAXScript подразумевается любой свиток, известный вам из пользовательского интерфейса 3ds Max, например панелей команд или диалоговых окон. Но, если требуется, свиток может быть отображен средствами MAXScript, как отдельное диалоговое окно.

Правда, создать специальный свиток средствами MAXScript и заполнить его элементами пользовательского интерфейса не составляет большого труда.

Для создания свитка используется следующая конструкция:

```
rollout <переменная> "Имя свитка"

(
```

```
  < Элементы пользовательского интерфейса, в том числе
  флажки, кнопки и счетчики >

)
```

В переменной хранится внутреннее имя свитка, используемое в сценарии для обращения к свитку. А строка “Имя свитка” определяет название, появляющееся в заголовке свитка.

Код, находящийся в круглых скобках, называется *выражением свитка*. Внутри выражения свитка определяются все элементы пользовательского интерфейса и их функции.

Для начала в следующем упражнении будет создан самый простой свиток.

1. Откройте новое окно редактора MAXScript Editor и введите следующий фрагмент кода:

   ```
   rollout a "Something New"
   (
     spinner b "Enter a value: "
     button c "Click Me"
   )
   ```

   В этом коде определяется свиток, обозначенный как “Something New” (Нечто новое). Переменная `a` содержит внутреннюю ссылку на свиток, которую можно использовать в остальных частях сценария для обращения к данному свитку.

   Команды `spinner` и `button` создают соответственно счетчик и кнопку с метками, указанными в кавычках. А переменные `b` и `c` содержат внутренние ссылки на счетчик и кнопку.

2. Выполните созданный сценарий.

   При этом ничего особенного не произойдет, поскольку в данном сценарии было задано лишь местоположение и внешний вид свитка.

3. Введите в конце сценария следующую строку кода:

   ```
   createDialog a 20 50
   ```

4. Вычислите код сценария.

5. Свиток появится в виде диалогового окна Something New шириной `20` и высотой `50` единиц. Он содержит два элемента пользовательского интерфейса: счетчик и кнопку.

   По команде `createDialog` создается новое перемещаемое диалоговое окно с использованием команд создания свитка. В связи с тем, что `a` — это внутреннее имя свитка, команда `createDialog a` обращается к определению свитка, сделанному ранее в данном сценарии.

   Если изменить значение в счетчике и щелкнуть на кнопке, то ничего не произойдет. Для того чтобы что-то произошло при взаимодействии пользователя с элементами интерфейса в данном диалоговом окне, в сценарий необходимо ввести обработчики событий.

Для ввода обработчиков событий в сценарий выполните следующее упражнение.

1. Закройте диалоговое окно Something New. Введите в окне редактора MAXScript Editor следующий код после строки с определением счетчика:

```
on c pressed do
(
  d = b.value
  sphere pos:[d,0,0]
)
```

   Вычислите код сценария. Диалоговое окно появится, как и прежде.

2. Измените значение в счетчике и щелкните на кнопке. После каждого такого щелчка создается сфера, расположенная по оси X так, как указано в счетчике. Рассмотрим более подробно, каким образом действует обработчик событий для кнопки `c`, указанный в строке кода `on c pressed do`. В этой строке сценарию предписывается выполнить код в круглых скобках, когда осуществляется щелчок на кнопке `c`.

   Счетчик хранится в переменной `b`. А значение, введенное в его поле, может быть получено с помощью свойства `.value` данной переменной. Следовательно, `b.value` — это значение, введенное в поле счетчика `b`.

3. После щелчка на кнопке `c` значение из счетчика присваивается в сценарии переменной `d`, а затем создается сфера, располагаемая по оси X так, как указано в счетчике.

Для того чтобы преобразовать данный сценарий в макросценарий, выполните следующее упражнение. В этом упражнении вам предстоит перетащить исходный код сценария на выбранную панель инструментов, чтобы организовать доступ к нему на панели инструментов.

1. Закройте диалоговое окно Something New.
2. Выделите весь исходный код сценария в окне редактора MAXScript Editor, нажав комбинацию клавиш <Ctrl+A>.
3. Перетащите выделенный текст на пустой участок между двумя кнопками на основной панели инструментов. Как только курсор примет вид стрелки со знаком “плюс”, отпустите кнопку мыши.

   На основной панели инструментов появится новая кнопка. Кнопки сценариев похожи на миниатюрные окна MAXScript Editor или MAXScript Listener.

4. Щелкните на новой кнопке.

   Откроется диалоговое окно Something New, в котором можно создать сферу описанным выше образом.

5. Закройте диалоговое окно Something New.
6. Щелкните правой кнопкой мыши на кнопке New и выберите команду Edit Macro Script (Править макросценарий).

   Откроется новое окно редактора сценариев с новым вариантом создаваемого вами сценария. В начале этого сценария введен ряд строк для его преоб-

разования в макросценарий. Этот новый вариант сценария сохранен в файле `DragAndDrop-Macro#.mcr`, где # — порядковый номер текущей версии сценария.

Примечание. Создание нового варианта сценария не оказывает никакого влияния на его исходный вариант, который по-прежнему находится в собственном окне редактора MAXScript Editor.

7. Закройте окно редактора MAXScript Editor с новым вариантом сценария, который был перенесен на основную панель инструментов.

   Созданный в данном упражнении сценарий является общим. Его нетрудно преобразовать в утилиту, изменив лишь несколько строк кода.

Для преобразования сценария в утилиту выполните следующее упражнение.

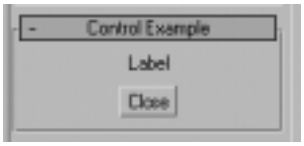
1. Закройте диалоговое окно Something New.
2. Замените слово `rollout` словом `utility` в первой строке кода сценария.
3. Удалите последнюю строку кода с командой `createDialog` отображения перемещаемого диалогового окна.
4. Вычислите код сценария.
5. Выберите кнопку MAXScript на панели Utilities. Затем выберите вариант Something New из раскрывающегося списка Utilities в свитке MAXScript.

   На панели Utilities появится свиток Something New, в котором можно воспользоваться счетчиком и кнопкой таким же образом, как и ранее в перемещаемом диалоговом окне.

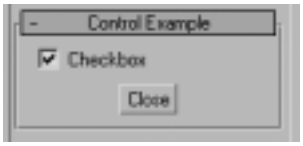
Ввод элементов пользовательского интерфейса

Ранее в этой главе было показано, каким образом элементы пользовательского интерфейса вводятся в свиток. Существует еще немало средств для ввода меток, флажков, кнопок-переключателей и других элементов интерфейса. При этом можно контролировать расположение элементов интерфейса справа, слева либо по центру диалогового окна или в единственной строке свитка. Ниже перечислены наиболее часто используемые элементы пользовательского интерфейса в том виде, в каком они называются и применяются в сценариях MAXScript.

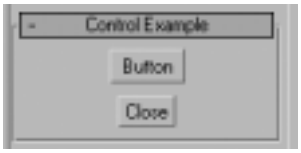
- `Label` (Метка) — статический элемент управления, предназначенный для отображения текста. Пользователь не может изменить текст метки, однако это можно сделать в сценарии.



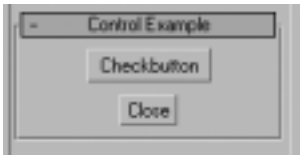
- `Checkbox` (Флажок) — элемент, который может быть включен или выключен пользователем.



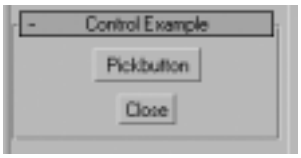
- `Button` (Кнопка) — элемент, который утапливается при нажатии и приподнимается обратно при отпускании.



- `Checkbutton` (Фиксирующая кнопка) — элемент, который утапливается при первоначальном нажатии и приподнимается при повторном нажатии.

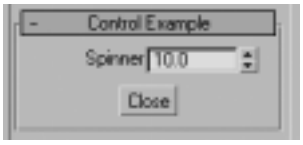


- `Pickbutton` (Кнопка выбора) — элемент выбора объектов на сцене.

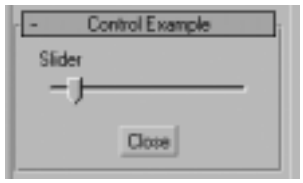


- `Radiobuttons` (Кнопки-переключатели) — ряд переключателей в свитке. Одновременно можно выбрать лишь один из них.

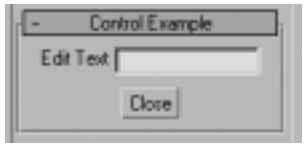
- `Spinner` (Счетчик) — элемент, предназначенный для ввода числового значения в свитке. Он состоит из поля ввода и стрелок. У счетчика имеются два параметра: диапазон значений и устанавливаемое по умолчанию значение.



- `Slider` (Ползунковый регулятор) — альтернативный счетчику элемент. Пользователь перемещает ползунок в ту или иную сторону. У ползункового регулятора те же самые параметры: диапазон значений и устанавливаемое по умолчанию значение.



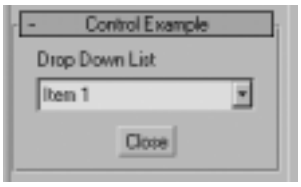
- `Edittext` (Поле редактирования текста) — текстовое поле, в котором пользователь вводит и редактирует текст.



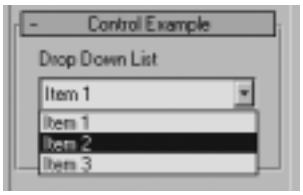
- `Listbox` (Списковое окно) — список элементов, предоставляемых пользователю на выбор. Пользователь может прокручивать список для выбора нужного элемента из списка.

- `Combobox` (Комбинированное окно) — комбинация списка выбора и поля редактирования текста. Список всегда отображается в свитке полностью, хотя длинные списки могут быть снабжены полосами прокрутки. В расположенном вверху поле редактирования отображается выделенный в данный момент элемент списка.

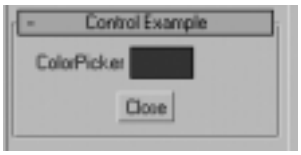
- `Dropdownlist` (Раскрывающийся список) — аналогичен комбинированному окну, но в данном случае список раскрывается, когда пользователь щелкает на кнопке со стрелкой вниз. Следует иметь в виду, что текст в раскрывающемся списке не вводится.



- `Colorpicker` (Селектор цвета) — элемент, предназначенный для отображения диалогового окна Color Selector (Селектор цвета). В свитке он представлен образцом цвета. Пользователь может щелкнуть на образце цвета, чтобы открыть диалоговое окно Color Selector.



- `Progressbar` (Индикатор выполнения) — элемент, предназначенный для отображения хода действия или выполнения процесса.



- `Mapbutton` (Кнопка карты) и `Materialbutton` (Кнопка материала) — элементы, предназначенные для отображения карт и материалов в диалоговом окне Material/Map Browser.
- `Bitmap` (Растр) — элемент, предназначенный для отображения растрового изображения в свитке.

Для каждого элемента интерфейса, указываемого в выражении свитка, создается соответствующий объект со своими свойствами. Доступ к этим свойствам осуществляется по имени данного элемента интерфейса. Имеются свойства, которые являются общими для всех элементов интерфейса (например, свойство положения), а также свойства, характерные только для элемента конкретного типа.

При создании в 3ds Max элементы интерфейса размещаются на панели со стандартными параметрами расположения. Если требуется другая схема их расположения, конструктору каждого такого элемента можно передать ряд следующих параметров.

- `align` — указывается следующим образом: `#left`, `#right` или `#center`. Знак "решетки" должен быть указан обязательно. Данный параметр определяет выравнивание элемента интерфейса по левому краю, по правому краю или же по центру.
- `pos` — определяет положение элемента интерфейса в конкретной точке с координатами x и y, указываемыми в пикселях. Значение этого параметра указывается с помощью типа данных `Point2 ([x, y])`. Такое положение элемента интерфейса определяется относительно левого верхнего угла свитка.
- `width` — задает ширину элемента интерфейса в пикселях.
- `height` — задает высоту элемента интерфейса в пикселях. Для спискового и комбинированного окон высота задается в зависимости от числа элементов в списке. Так, для отображения N элементов в списковом окне его высоту следует задать равной N. А для отображения N элементов в комбинированном окне его высоту следует задать равной N+2.
- `offset` — задает смещение относительно устанавливаемого по умолчанию положения элемента интерфейса. В качестве единиц изменения используются пиксели, а в качестве типа данных — `Point2`.
- `across` — задает положение элементов интерфейса по горизонтали, а не по вертикали. Применяется к элементу и следующим за ним (N – 1) элементам, где N — число, указываемое после слова `across`. Этот параметр более подробно рассматривается в следующем разделе.

Как правило, лучше сначала предоставить 3ds Max возможность расположить элементы интерфейса с помощью устанавливаемых по умолчанию параметров, а затем внести необходимые коррективы в расположение элементов. Текстовые метки должны быть краткими, чтобы они не выходили за край свитка, а элементы интерфейса не должны перекрываться.

Обработчики событий

Общая форма обработчика событий имеет следующий вид:

```
on <имя элемента интерфейса> <имя события> <аргументы> do
(
  [выполняемые команды]
)
```

Ниже приведен перечень типичных событий, используемых в сценариях.

- `pressed` — вызывается после щелчка на кнопке.
- `changed` — вызывается при изменении состояния управляющего элемента, например при установке флажка или правке значения в счетчике.
- `picked` — вызывается для кнопки выбора, когда пользователь выбирает элемент в сцене.

- `entered` — вызывается при вводе числа в поле редактирования счетчика и последующем нажатии клавиши <Enter>.
- `selected` — вызывается при выборе элемента из списка в списковом или комбинированном окне.

В версии 3ds Max 8 событие `RightClick` введено для следующих видов кнопок.

- `Button`
- `CheckButton`
- `MapButton`
- `MaterialButton`
- `PickButton`
- `ImgTag`

Многие элементы пользовательского интерфейса были усовершенствованы в версии 3dsMax 7. Более подробно с перечнем подобных нововведений можно ознакомиться, перейдя к подразделу “User Interface Controls” (Элементы управления пользовательского интерфейса) в разделе “What was New in MAXScript in 3dsMax 8” (Нововведения в MAXScript версии 3dsMax 7) справочного руководства по MAXScript.

Группы в свитке

По мере укрупнения и усложнения утилит возникает потребность сгруппировать элементы в свитке на панели Utilities. Элементы можно сгруппировать в логической последовательности, используя выражение группы. Независимо от того, как и для чего группируются элементы, группирование должно разумно разграничивать функции панели. Группа очерчивается на панели контуром с меткой в левом верхнем углу. А синтаксис группы следующий:

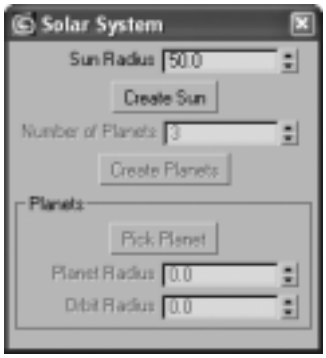
```
group "описание группы"
(
<элементы пользовательского интерфейса>
)
```

Создание рабочего сценария

А теперь вам предстоит написать сценарий для формирования Солнечной системы на основании пользовательского ввода. Но прежде чем создавать пользовательский интерфейс, необходимо выяснить, что именно должен делать сценарий. В данном случае требуется написать сценарий, выполняющий следующее.

- Для того чтобы начать процесс формирования Солнечной системы, пользователь должен создать Солнце. В качестве Солнца может служить сфера, расположенная в точке с координатами [0,0,0]. Пользователь должен задать радиус сферы.

- В версии 3ds Max 8 введено новое свойство — возможность обрабатывать для разных типов кнопок пользовательские события, связанные со щелчком правой кнопкой мыши (см. перечень событий, приведенных в предыдущем разделе). Для того чтобы воспользоваться этим свойством в сценарии, придется организовать процесс создания сферы в зависимости того, какой кнопкой мыши (левой или правой) пользователь щелкнет на кнопке создания Солнца. Так, если пользователь щелкнет левой кнопкой, будет создана обычная сфера. А если он щелкнет правой кнопкой, будет создана геосфера.
- Как только будет создано Солнце, пользователь может сформировать дополняющие его планеты. У него должна быть возможность выбора числа планет.
- Как только пользователь щелкнет на кнопке Create Planets (Создать планеты), в сценарии должны быть созданы планеты произвольного радиуса. Для каждой планеты вокруг Солнца будет очерчен круг, обозначающий ее орбиту, причем движение планеты будет ограничено этим кругом с помощью ограничения по линии пути.
- После создания планет пользователь может выделить каждую планету и изменить ее радиус и орбиту. На основании всей этой информации вы можете построить такой пользовательский интерфейс, как на приведенном ниже рисунке.



Создание пользовательского интерфейса

А теперь вы можете приступать к созданию и проверке пользовательского интерфейса. В дальнейшем вам предстоит ввести обработчики событий в этот интерфейс для того, чтобы он выполнял конкретные функции.

Для создания пользовательского интерфейса выполните следующее упражнение.

1. Откройте окно редактора MAXScript Editor.
2. Введите в верхней части этого окна следующую строку:

   ```
   -- SolarSystem.ms
   ```

   Вы можете дополнить этот комментарий, например, своим именем и датой создания сценария.

3. Введите следующий фрагмент кода:

```
rollout ssRoll "Solar System"
(
  spinner spn_sunRadius "Sun Radius"
  button but_createSun "Create Sun"
  spinner spn_numPlanets "Number of Planets"
  button but_createPlanets "Create Planets"
  pickbutton pbt_pickPlanet "Pick Planet"
  spinner spn_planetRadius "Planet Radius"
  spinner spn_orbitRadius "Orbit Radius"
)
createDialog ssRoll 20 20
```

Многие программисты пользуются специальными условными обозначениями элементов пользовательского интерфейса. В приведенной ниже таблице представлены префиксы, используемые для обозначения переменных по типу элемента пользовательского интерфейса.

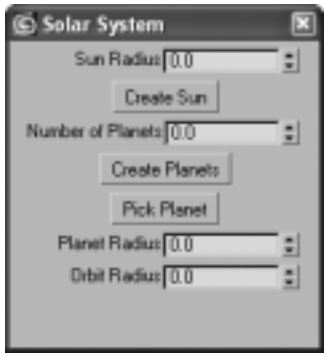
Префикс типа элемента	Что обозначает
`spn_`	Счетчик
`but_`	Кнопка
`pbt_`	Кнопка выбора

После префикса типа элемента указывается имя переменной, содержащее текстовую метку данного элемента пользовательского интерфейса, причем первое слово должно быть введено строчными буквами, а все последующие слова должны начинаться с прописной буквы. Так, переменная для счетчика Sun Radius (Радиус Солнца) должна называться следующим образом: `spn_sunRadius`.

Вы можете, конечно, выбрать другие условные обозначения в своих сценариях. Главное — быть последовательным в их соблюдении.

4. Вычислите код сценария.

Откроется диалоговое окно с элементами пользовательского интерфейса, указанными в коде данного сценария.



Внешний вид этого диалогового окна можно усовершенствовать, выделив в отдельную группу элементы пользовательского интерфейса, предназначенные для правки внешнего вида планет.

5. Введите перед кодом первой кнопки выбора следующий фрагмент кода:

```
group "Planets"
(
```

6. Введите закрывающую круглую скобку после кода последнего счетчика.

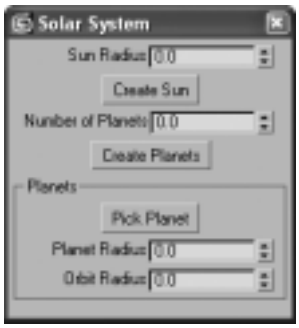
Код трех последних элементов пользовательского интерфейса должен выглядеть следующим образом:

```
group "Planets"
(
pickbutton pbt_pickPlanet "Pick Planet"
spinner spn_planetRadius "Planet Radius"
spinner spn_orbitRadius "Orbit Radius"
)
```

Этот код выделяет три последних элемента пользовательского интерфейса в отдельную группу.

7. Вычислите код сценария.

Элементы пользовательского интерфейса, предназначенные для правки внешнего вида планет, теперь находятся в отдельной группе.



Для улучшения внешнего вида пользовательского интерфейса можно также ввести пробел после метки каждого элемента, т.е. перед завершающими кавычками.

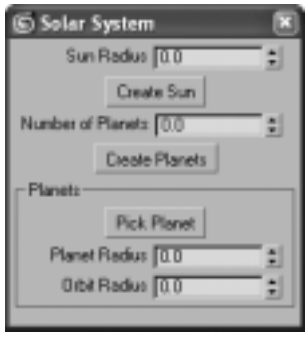
8. Внесите изменения в строку кода каждого счетчика, содержащую метку, чтобы ввести пробел после метки счетчика:

```
rollout ssRoll "Solar System"
(
  spinner spn_sunRadius "Sun Radius "
  button but_createSun "Create Sun"
  spinner spn_numPlanets "Number of Planets "
  button but_createPlanets "Create Planets"
  pickbutton pbt_pickPlanet "Pick Planet"
  spinner spn_planetRadius "Planet Radius "
  spinner spn_orbitRadius "Orbit Radius "
```

```
  group "Planets"
  (
  pickbutton pbt_pickPlanet "Pick Planet"
  spinner spn_planetRadius "Planet Radius "
  spinner spn_orbitRadius "Orbit Radius "
  )
)
```

9. Вычислите код сценария.

   Между меткой каждого счетчика и самим счетчиком должен быть пробел.



10. Сохраните вновь созданный сценарий в файле `mySolarSystem.ms`.

Примечание. Сохранять сценарий без лишних напоминаний полезно после очередного этапа работы над ним — это общепринятая практика программирования. Однако, начиная с версии 3ds Max 8, при аварийном завершении работы 3ds Max, вызванном сбоями в MAXScript, должно появиться диалоговое окно, в котором предлагается сохранить все открытые сценарии.

Ввод обработчиков событий

В следующем упражнении вам предстоит ввести обработчики событий для создания Солнца и планет.

1. Введите следующий фрагмент кода перед последней закрывающей скобкой в выражении свитка:

```
on but_createSun pressed do
(
)
on but_createSun rightclick do
(
)
```

   В этом коде создаются два обработчика событий для кнопки Create Sun. Если щелкнуть на ней левой кнопкой мыши, в сценарии должна быть создана сфера радиусом, определяемым параметром Sun Radius. Если же щелкнуть на ней правой кнопкой мыши, в сценарии на этот раз должна быть создана геосфера.

2. Введите внутри обработчика событий `pressed` следующую строку кода:

   ```
   sun = sphere radius:spn_sunRadius.value
   ```

   Теперь данный обработчик событий должен иметь следующий вид:

   ```
   on but_createSun pressed do
   (
   sun = sphere radius:spn_sunRadius.value
   )
   ```

   Значение переменной `spn_sunRadius` хранится в ее свойстве `.value`. Выражение `spn_sunRadius.value` возвращает любое значение, находящееся в поле редактирования счетчика Sun Radius в момент нажатия кнопки Create Sun.

3. Введите внутри обработчика событий `rightclick` следующий фрагмент кода:

   ```
   on but_createSun rightclick do
   (
     sun = geosphere radius:spn_sunRadius.value
   )
   ```

4. Вычислите код сценария.

5. Измените значение параметра Sun Radius в открывшемся диалоговом окне, введя в его поле число, большее нуля, а затем щелкните левой кнопкой мыши на кнопке Create Sun. В сцене будет создана сфера с указанным радиусом. Переместите сферу по сцене и попробуйте щелкнуть правой кнопкой мыши, чтобы создать геосферу. Если сценарий не позволяет этого сделать, проанализируйте его код в приемнике команд, выявите ошибку и исправьте ее, прежде чем продолжить работу над данным сценарием.

6. Создайте обработчик событий для формирования планет и их орбит. Для этого введите следующий фрагмент кода после только что введенного кода обработчика событий:

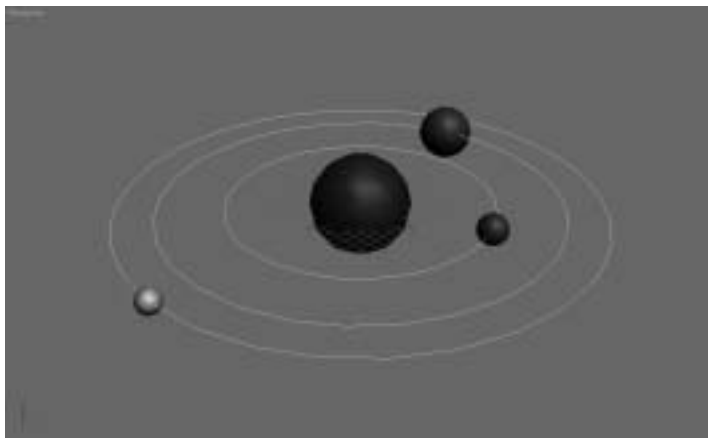
   ```
   on but_createPlanets pressed do
   (
     for i = 1 to spn_numPlanets.value do
     (
       -- Создание планеты
       planet = sphere() -- Создать планету
       -- Установить радиус планеты в виде случайного числа
       -- в пределах от 10 до 30
       planet.radius = random 10.0 30.0
       -- Создать окружность для орбиты планеты
       orbit = circle radius:(i*10)
       -- Поместить планету на орбиту
       planet.pos.controller = Path_Constraint()
       planet.pos.controller.path = orbit
       -- Повернуть орбиту, чтобы она заняла произвольное
       -- исходное положение
       orbit.rotation.z_rotation = random 0.0 360.0
     )
   )
   ```

Этот обработчик событий создает ряд указанных планет с произвольной орбитой в виде окружности и ограничивает их движение по кругу с помощью ограничения по линии пути. И наконец, он поворачивает орбиту каждой планеты на случайную величину, чтобы исходное положение орбит было разным.

7. Вычислите код сценария.

   Установите значение 3 в поле счетчика Number of Planets (Число планет) и щелкните на кнопке Create Planets.

   Вокруг Солнца будут созданы две планеты со своими орбитами.



Точная настройка пользовательского интерфейса

В поле счетчика Number of Planets отображается число с плавающей точкой в пределах от 0 до 10. Это стандартный диапазон значений для всех счетчиков. Однако для данного счетчика больше подходит диапазон целых чисел от 1 до 10.

1. Введите следующий код в конце строки `spinner spn_numPlanets`:

   ```
   type:#integer range:[1,10,3]
   ```

   Теперь эта строка кода должна иметь следующий вид:

   ```
   spinner spn_numPlanets "Number of Planets" type:#integer
   range:[1,10,3]
   ```

   Данная строка кода предписывает отображать в счетчике Number of Planets только целые числа в пределах от 1 до 10, причем по умолчанию отображается значение 3.

2. Вычислите код сценария.

3. Измените значение в счетчике Number of Planets. По умолчанию в этом счетчике отображается значение 3, которое может изменяться лишь от `1` до `10`.

   В версии 3ds Max 7 появилось новое свойство элемента управления `pickButton`, позволяющее автоматически отображать имя выбранного объекта в надписи кнопки выбора. Ранее это приходилось делать вручную.

Итак, дополните строки кода для отображения кнопки выбора следующим ключевым параметром:

```
autoDisplay: true
```

Автоматическое удаление диалоговых окон

В данный момент открыто несколько диалоговых окон Solar System (Солнечная система). При каждом вычислении кода сценария появляется новое окно данного типа, а старые не закрываются. Для того чтобы закрыть автоматически любые открытые ранее варианты диалогового окна, когда приходится в очередной раз вычислять код сценария, достаточно ввести в его начале следующий код:

```
destroyDialog ssRoll
```

Но при вычислении этой строки кода может возникнуть ошибка, если открытое диалоговое окно `ssRoll` отсутствует. Следовательно, диалоговое окно `ssRoll` нужно удалить в сценарии, если оно существует, и ничего не предпринимать, если оно отсутствует.

Для этого в начале сценария можно ввести следующий фрагмент кода:

```
if ((ssRoll != undefined) and (ssRoll.isdisplayed)) do
  (destroyDialog ssRoll)
```

В этом коде сначала проверяется тот факт, что диалоговое окно `ssRoll` определено. Если затем обнаруживается, что данное окно отображается, оно удаляется (т.е. закрывается). Следует иметь в виду, что для удаления диалогового окна должны выполняться оба условия, т.е. результат их проверки должен быть истинным (`true`). Если же не выполняется хотя бы одно из этих условий, т.е. результат его проверки оказывается ложным (`false`), ничего не происходит.

Примечание. Приведенный выше условный оператор является относительно новым и применяется благодаря внедрению в версии 3ds Max 7 свойства свитков, называемого `.isdisplayed`. Данное свойство возвращает логическое значение `true`, если свиток отображается (или виден); в противном случае оно возвращает логическое значение `false`. В предыдущем издании книги для достижения того же результата в сценарии применялось следующее действие:

```
try (destroyDialog ssRoll) catch()
```

Для автоматического удаления диалоговых окон выполните следующее упражнение.

1. Закройте все варианты диалогового окна Solar System.
2. Введите в сценарии следующий фрагмент кода перед началом выражения свитка:

   ```
   if ((ssRoll != undefined) and (ssRoll.isdisplayed)) do
     (destroyDialog ssRoll)
   ```

3. Вычислите код сценария.

   Как видите, остается лишь один вариант данного диалогового окна.

   Теперь код сценария должен иметь следующий вид:

```
-- SolarSystem.ms

if ((ssRoll != undefined) and (ssRoll.isdisplayed)) do
  (destroyDialog ssRoll)

rollout ssRoll "Solar System"
(
  spinner spn_sunRadius "Sun Radius"
  button but_createSun "Create Sun"
  spinner spn_NumPlanets "Number of Planets" type:#integer range:[1,10,3]
  button but_createPlanets "Create Planets"

  group "Planets"
    (
    pickbutton pbt_pickPlanet "Pick Planet" autoDisplay: true
    spinner spn_planetRadius "Planet Radius"
    spinner spn_orbitRadius "Orbit Radius"
  )

  on but_createSun pressed do
  (
    sun = sphere radius:spn_sunRadius.value
  )

  on but_createPlanets pressed do
  (
    for i = 1 to spn_numPlanets.value do
    (
    -- Создание планеты
    planet = sphere() -- Создать планету
    -- Установить радиус планеты в виде случайного числа
    -- в пределах от 10 до 30
    planet.radius = random 10.0 30.0
    -- Создать окружность для орбиты планеты
    orbit = circle radius:(i*10)
    -- Поместить планету на орбиту
    planet.pos.controller = Path_Constraint()
    planet.pos.controller.path = orbit
    -- Повернуть орбиту, чтобы она заняла произвольное
    -- исходное положение
    orbit.rotation.z_rotation = random 0.0 360.0
    )
  )

)
createDialog ssRoll 20 20
```

Ввод событий для кнопки выбора

Организуйте событие для кнопки Pick Planets, выполнив следующее упражнение.

1. Введите следующий код непосредственно перед последней круглой скобкой в выражении свитка:

```
on pbt_pickPlanet picked aPlanet do
(
  -- Установить радиус выбранной планеты в счетчике Planet Radius
```

```
    spn_planetRadius.value = aPlanet.radius
    -- Установить радиус орбиты планеты в счетчике Orbit Radius
    pOrbit = aPlanet.pos.controller.path
    spn_orbitRadius.value = pOrbit.radius
)
```

Если щелкнуть на кнопке Pick Planet, выбранный объект сохранится в локальной переменной `aPlanet`. Имя выбранного объекта теперь автоматически отображается в надписи кнопки выбора Pick Planet.

Примечание. Программирующие на MAXScript часто пользуются переменной `obj` для обозначения того факта, что объект выбран. Имя переменной `obj` не является ни ключевым словом, ни любым другим именем специальной переменной. Это имя упоминается здесь лишь потому, что оно часто встречается в сценариях.

2. Вычислите код сценария.
3. Щелкните сначала на кнопке Pick Planet, а затем на самой планете.

   Независимо от того, какая планета выбрана, в поле счетчика Orbit Radius появляется значение `10,0`, хотя радиус орбиты может достигать и 10. Напомним, что диапазон значений в счетчике ограничивается верхним пределом 10. Для правильного отображения числовых значений в счетчике необходимо изменить этот предел.
4. Замените строку `spn_orbitRadius` следующей строкой кода:

   ```
   spinner spn_orbitRadius "Orbit Radius" range:[0,100,0]
   ```
5. Вычислите код сценария.
6. Щелкните на кнопке Pick Planet и выберите планету. В поле счетчика Orbit Radius появится правильно отображаемое значение радиуса орбиты.

Использование локальных переменных

В следующем упражнении вам предстоит ввести обработчики событий, изменяющие радиус выбранной планеты и ее орбиты в результате изменений в счетчиках Planet Radius и Orbit Radius.

1. Введите следующий фрагмент кода после введенного последним обработчика событий:

   ```
   on spn_planetRadius changed value do
   (
   )
   ```

   Данное событие возникает всякий раз, когда пользователь изменяет значение свойства `.value` в переменной счетчика, вводя новое значение с клавиатуры или настраивая счетчик с помощью мыши.

   Но дело в том, что при вызове данного обработчика событий требуется изменить радиус объекта, выбранного в предыдущем обработчике событий, а имя этого объекта хранится в переменной `aPlanet`, которая является локальной для данного блока кода. Следовательно, для того чтобы использовать

эту переменную в данном обработчике событий, нужно каким-то образом сделать ее доступной вне области действия предыдущего обработчика событий.

Для этой цели необходимо объявить переменную `pPlanet` вне области действия всех обработчиков событий и хранить в ней имя объекта. Это даст возможность использовать ее в любом обработчике событий внутри данного сценария.

2. Введите следующую строку кода перед обработчиком событий `pbt_pickPlanet`:

```
local pPlanet
```

3. Введите в начале кода обработчика событий `pbt_pickPlanet` следующую строку:

```
pPlanet = aPlanet
```

В этой строке значение переменной `pPlanet` устанавливается в соответствии с выбранной планетой. Теперь переменную `pPlanet` можно использовать в остальных обработчиках событий.

4. Введите внутри обработчика событий `spn_planetRadius` следующую строку кода:

```
pPlanet.radius = spn_planetRadius.value
```

Данный фрагмент кода теперь должен иметь следующий вид:

```
local pPlanet
on pbt_pickPlanet picked aPlanet do
(
  pPlanet = aPlanet
  -- Установить радиус выбранной планеты в счетчике Planet Radius
  spn_planetRadius.value = aPlanet.radius
  -- Установите радиус орбиты планеты в счетчике Orbit Radius
  pOrbit = aPlanet.pos.controller.path
  spn_orbitRadius.value = pOrbit.radius
)

on spn_planetRadius changed value do
(
  pPlanet.radius = spn_planetRadius.value
)
```

5. Вычислите код сценария.

6. Щелкните сначала на кнопке Pick Planet, а затем на планете.

7. Измените значение в счетчике Planet Radius. Радиус планеты соответственно изменится.

Сделайте то же самое для орбиты планеты. И в этом случае возникнет аналогичная проблема с переменными, поскольку переменная `pOrbit` является локальной для обработчика событий `pbt_pickPlanet`. Для того чтобы сделать доступной эту переменную, ее следует объявить локально вне обработчиков событий.

8. Введите следующую строку кода после строки объявления локальной переменной `pPlanet`:

```
local pOrbit
```

9. Введите следующий фрагмент кода после введенного последним обработчика событий:

```
on spn_orbitRadius changed value do
(
  pOrbit.radius = spn_orbitRadius.value
)
```

10. Вычислите код сценария.
11. Щелкните сначала на кнопке Pick Planet, а затем на планете.
12. Измените значение в счетчике Orbit Radius. Радиус орбиты планеты соответственно изменится.

Включение и отключение элементов пользовательского интерфейса

Итак, сценарий работает, хотя он и недостаточно защищен от ошибок. В частности, кнопка Create Planets оказывается доступной даже в тот момент, когда пользователь еще не создал Солнце. Всякая попытка создать планеты до Солнца может привести к ошибке.

Во избежание пользовательских ошибок отдельные элементы интерфейса включаются и отключаются в зависимости от конкретных действий пользователя в интерфейсе. Например, при открытии диалогового окна Solar System можно отключить все кнопки и счетчики до тех пор, пока пользователь не создаст Солнце.

1. Введите во всех строках, определяющих счетчики и кнопки, кроме первых двух, следующую строку кода:

```
enabled:false
```

Измененный фрагмент кода сценария должен теперь выглядеть следующим образом:

```
(
  spinner spn_sunRadius "Sun Radius "
  button but_createSun "Create Sun"
  spinner spn_NumPlanets "Number of Planets " type:#integer \
  range:[1,10,3] enabled:false
  button but_createPlanets "Create Planets" enabled:false

  group "Planets"
(
  pickbutton pbt_pickPlanet "Pick Planet" autoDisplay:true \
  enabled:false
  spinner spn_planetRadius "Planet Radius " enabled:false
  spinner spn_orbitRadius "Orbit Radius " range:[0,100,0] \
  enabled:false
)
```

При вычислении кода сценария в диалоговом окне становятся доступными лишь два первых элемента пользовательского интерфейса. Остальные элементы будут активизированы в обработчиках событий.

2. Введите следующий фрагмент кода внутри обработчика событий `but_createSun` для отображения элементов пользовательского интерфейса, предназначенных для создания планет:

   ```
   spn_numPlanets.enabled = true
   but_createPlanets.enabled = true
   ```

   Оба элемента пользовательского интерфейса становятся доступными после того, как пользователь создаст Солнце того или иного типа.

3. Введите следующий фрагмент кода внутри обработчика событий `but_createPlanets`, но непременно вне цикла `for`, т.е. до или после него:

   ```
   pbt_pickPlanet.enabled = true
   spn_planetRadius.enabled = true
   spn_orbitRadius.enabled = true
   ```

4. Сохраните сценарий.

5. Установите 3ds Max в исходное состояние и очистите сцену.

6. Вычислите код сценария.

   Сначала доступными оказываются лишь два первых элемента. После создания Солнца появляется возможность создать планеты и далее изменить радиус как отдельных планет, так и их орбит.

   А теперь необходимо сделать еще одно, последнее изменение в сценарии, чтобы он стал еще более защищенным от ошибок. При выполнении данного сценария в первый раз значение параметра Sun Radius устанавливается по умолчанию равным нулю. Если не изменить его, Солнце будет создано с нулевым радиусом. Нетрудно представить, как неопытный пользователь будет щелкать снова и снова на кнопке Create Sun, добиваясь того, чтобы Солнце появилось на сцене. В итоге сцена заполнится десятками невидимых Солнц, прежде чем пользователь поймет, в чем состоит ошибка.

   Во избежание этого радиус сферы следует установить по умолчанию равным 50 или другому числовому значению, превышающему нуль.

7. Дополните строку кода, определяющую счетчик Sun Radius, следующим кодом:

   ```
   range:[1,100,50]
   ```

   Эта строка должна теперь иметь следующий вид:

   ```
   spinner spn_sunRadius "Sun Radius " range:[1,100,50]
   ```

8. Сохраните сценарий.

   Вариант этого сценария можно найти в файле `\chapter2\Solar_System.ms` на прилагаемом к этой книге CD-ROM.

Для усовершенствования данного сценария в него можно внести целый ряд дополнений. Например, пользователю можно разрешить следующее.

- Указывать начальные радиусы орбит с приращением вместо исходного значения 10. Для этого придется расширить диапазон значений параметра Orbit Radius в сторону больших чисел.

- Наклонить каждую орбиту или сделать ее эллиптической. Для этого потребуются навыки выполнения преобразований, которые более подробно рассматриваются в главе 5.

Заключение

В этой главе вы научились строить пользовательские интерфейсы, создав свиток и заполнив его элементами пользовательского интерфейса, а затем введя в него обработчики событий для вызова отдельных частей сценария с помощью элементов управления пользовательского интерфейса. Кроме того, вы научились отличать общий сценарий от макросценария и создавать каждый из них в отдельности.