FPS

Официальное электронное издание студии Clocktower Games

4

04 08

Психология компьютерных игр

Park Racer

Xtreme 3D

От редакции

Привет всем читателям!

Во-первых, спасибо всем, кто нас читает и ценит. Без вашей моральной поддержки и советов у нас вряд ли бы что-то выходило.

Во-вторых, вы видите у себя на экране очередной — четвертый — номер журнала FPS. Мы продолжаем развивать взятую нами тематику; основные темы этого месяца — 3D-графика и технология Java.

К сожалению, от обзоров приставочных игр временно пришлось отказаться по причине отсутствия материала :(но, подчеркиваю, временно.

В-третьих, хорошая новость для любителей Xtreme3D: уже месяц, как работает наш новый сайт, посвященный этому движку — http://xtreme3d.hut2.ru.Там вы можете найти примеры, уроки, полезные утилиты, игры, и собственно, скачать движок в виде нескольких разных дистрибутивов. Так что, милости просим :)

Ну, и, как всегда, можете писать нам в редакцию: clocktower89@mail.ru

Приятного чтения!

Gecko

Teddy
Скетчинг для всех и каждого!.....3
Cre8tor
Новый конструктор 3D-игр.....8
Уроки моделлинга
Моделируем дерево для Game Maker.....12
Lateral GM
Конец «битвы форматов».....16
GM Games — Park Racer
Гонки F1 от создателя Silent Walk FPS Creator.....18
Психология компьютерных игр, **ч**. I
Теория геймдизайна.....20
Знакомство с Java...
...и другими мобильными платформами.....25
Основы MIDlet Pascal
Часть 1. Анимация.....29
Программирование на Xtreme3D, **урок** 3
Создание игры от третьего лица: проверка столкновений.....32
Xtreme3D Material
Визуальный редактор материалов.....38

Teddy

Скетчинг для всех и каждого

Трехмерное моделирование у людей обычно ассоциируется со сложными программными пакетами типа 3D Studio MAX, триангуляцией, работой с вершинами, кривыми и т.д. И в большинстве случаев это верно, хотя и вовсе не так страшно и сложно, как может показаться непосвященным. А может ли моделлинг быть совсем простым, как, например, рисование? Чтобы не изучать громоздкие редакторы, не штудировать литературу, а начать прямо "с нуля" и почти сразу получить интересные результаты? Оказывается, может.

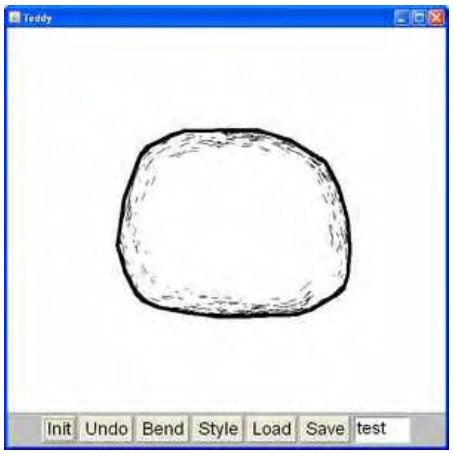
Существуют разнличные программы для "моделирования без моделирования". Для работы в них обычно не требуются специфические знания и опыт. Это Google SketchUp и IBModeller, где модели генерируются на основе фотографий, и многие другие. В этот же список можно внести и нашего сегодняшнего "гостя" - **Teddy**, создателем которой является японский программист Такео Игараши. С этой программой, названной так в честь плюшевого медведя, азы моделирования могут освоить даже дошкольники. Принцип ее действия - скетчинг, т.е. генерация объемных форм на основе заданных пользователем кривых. Все предельно просто: вы рисуете на экране курсором мыши круг, и через мгновение он превращается в шар. Можно рисовать и более сложные структуры; встроенные алгоритмы Teddy автоматически рапознают, что именно вы "имели в виду", и создадут объект с максимальной точностью. Программа имеет очень простой и интуитивный интерфейс, а если что-то все же не совсем понятно, на то имеется подробное руководство.

Так как Teddy ориентирован на создание кривых поверхностей (плоскость или точный куб сделать не получится), то и модели в нем выходят специфические - похожие на игрушечных животных. Наверно, поэтому программа так и называется. Давайте же попробуем создать кого-нибудь!

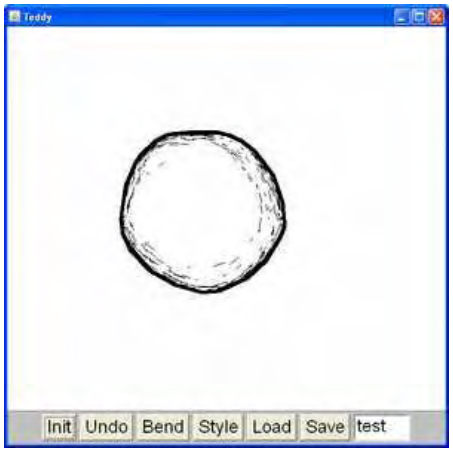
1. Teddy требует установленной на системе Java (JRE). Скачать Java можно на сайте http://www.java.com.

2. Скачайте Teddy (700 кб) по следующей ссылке: http://www-ui.is.s.u-tokyo.ac.jp/~takeo/teddy/teddy.zip.

3. Запустите *teddy.jar*. Должно появиться такое окно:



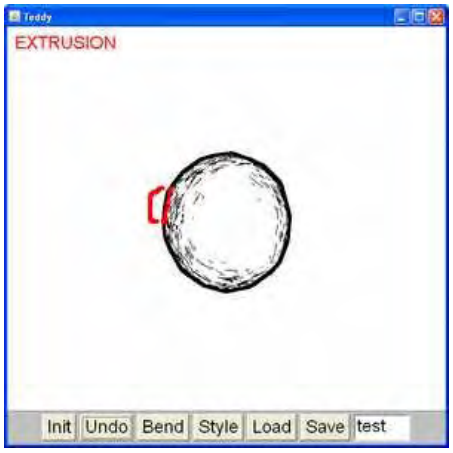
4. Нажмите кнопку **Init**. Высветится надпись "Creation". Попробуйте теперь нарисовать простую форму, например, круг или овал. Контур обязательно должен быть замкнутым, иначе получится некрасиво.



Если не понравилось, можно нажать **Undo** и нарисовать заново. Вы можете использовать правую кнопку мыши, чтобы вращать модель для осмотра со всех сторон.

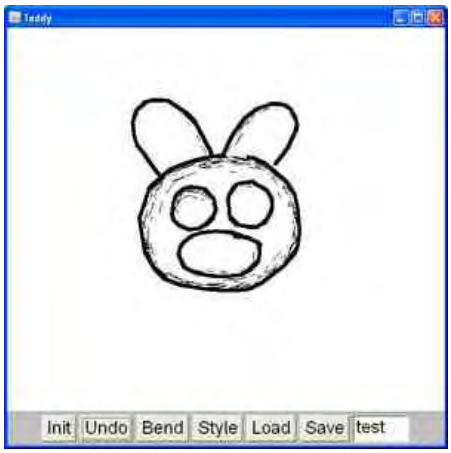
5. Если считать, что получившийся шарик - голова, можно добавить глаза. Для этого внутри контура нарисуйте кружок поменьше:

Появится надпись "Extrusion", говорящая о том, что программа готова принять данные о дополнительной поверхности. Поверните голову боком и нарисуйте выпуклость в виде полусферы:



Программа создаст предполагаемый глаз:

6. Таким же образом можно сделать второй глаз, а также любые другие выпуклости, например, уши, нос и пр.:



Правда, слишком большую детализацию задать не получится: Teddy распознает выпуклости не выше первого уровня (то есть, нельзя сделать зрачки на глазах и т.д.). Если все же попытаться, то программа скорее всего, не будет реагировать. Мне поначалу казалось даже, что Teddy после этого зависает, но на деле достаточно просто нажать **Undo** и повращать модель.

7. Теперь финальный этап - сохранение полученной фигурки. В поле, где написано "test" введите имя файла и нажмите **Save**. Модель будет сохранена в директории программы в формате OBJ, который поддерживается всеми редакторами.

Теперь спрашивается, как же раскрасить модель? Можно воспользоваться другими редакторами, создать текстуру и т.д., но новичкам в этом разбираться будет довольно сложно. Поэтому у нас два пути: скачать утилиту **Chameleon** (размер 272 кб): http://www-ui.is.s.u-tokyo.ac.jp/~takeo/chameleon/chameleon.htm, предназначенную для раскрашивания созданных в Teddy моделей, или, что более предпочтительно, скачать продвинутую версию "Медведя": **Smooth Teddy**. В ней уже имеются встроенные инструменты рисования, а также исправлены различные баги.

1. Скачиваем Smooth Teddy (1.5 Мб):
http://www-ui.is.s.u-tokyo.ac.jp/~takeo/java/smoothteddy/smoothteddy.zip.

2. Запускаем *run.bat*. Видно, что интерфейс программы значительно улучшен - добавлено затенение, контекстное меню и т.д. Скоро вы убедитесь, что изменения коснулись не только внешнего оформления - в Smooth Teddy доступно намного больше возможностей, чем у его предшественника. Принцип создания модели остался тот же, за исключением одного - для наращивания выпуклостей достаточно просто нарисовать ее основание, как указано в шаге 5 предыдущего мануала. А еще присутствует возможность перемещать выпуклости на основной поверхности: для этого кликните выпуклость - появится красный кружок, перетаскивая который можно перемещать весь объект.

3. Выберите инструмент "заливка" и закрасьте части модели каким-нибудь цветом:

4. Инструментом "кисть" можно рисовать на модели произвольные штрихи, как в графическом редакторе:

5. К сожалению, в более-менее понятный другим редакторам формат Smooth Teddy сохранять не может. Но зато есть другая возможность, которую мы рассмотрим прямо сейчас. Сохраните модель в файл ASG (*File > export to asg*).

В "веселой компании" Такео Игараши кроме двух медведей и хамелеона есть еще белка - **Squirrel**. Это аниматор, использующий очень интересную технологию - *spatial keyframing*. Реализуется она следующим образом: логический момент анимации модели зависит от пространственного положения курсора относительно ключевых точек, определяющих поворот тех или иных частей модели. Звучит несколько запутанно, но в действительности все очень просто и изящно.

1. Скачиваем Squirrel (1.9 Мб):
http://www-ui.is.s.u-tokyo.ac.jp/~takeo/research/squirrel/squirrel.zip.

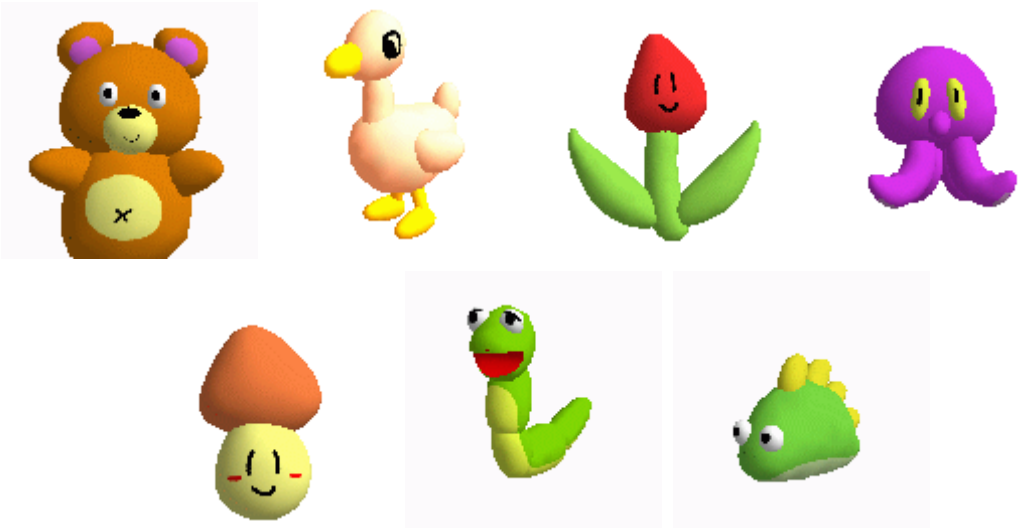
2. Импортируем созданный в Smooth Teddy файл ASG.

3. Розовый шарик - это и есть курсор. Перетащите его в любое место, поверните части модели правой кнопкой мыши и нажмите **Set**. На позиции курсора появится желтый шарик - ключ, хранящий заданное состояние модели. Перетащите курсор в другое место, измените положение частей модели, и снова нажмите **Set**. Второй ключ будет отвечать за новое состояние модели.

4. Теперь попробуем "оживить" модель. Выберите *mode* > *recording*. Перетащите курсор попеременно от одного ключа к другому и смотрите, что будет: персонаж плавно изменит свою позу из одного состояния в другое!

Это и есть основной принцип, на основе которго можно быстро сделать любую анимацию. А чтобы сохранить полученный мультфильм, вы можете записать анимацию при помощи временной шкалы внизу. Просто нажмите "запись" и начинайте анимировать. Полученную последовательность можно просмотреть кнопкой "воспроизведение" и сохранить в виде последовательности кадров - для этого выберите movie > create_movie, выставьте нужные опции и нажмите Go. Вы найдете изображения в папке movie в директории программы. Что с ними делать - решайте сами. Можете взять какой-нибудь редактор видео, например, Movie Maker, и сделать анимацию в нем, а можно воспользоваться одним из многочисленных редакторов GIF, что легко найти в интернете, и сделать аватар, анимацию для веб-страницы, игры или оформления программы. Творите!

Напоследок я приведу несколько симпатичных примеров, сделанных при помощи Smooth Teddy и Squrrel. К сожалению, формат PDF не отображает анимированные GIF'ы, но их вы можете увидеть на странице http://www-ui.is.s.u-tokyo.ac.jp/~takeo/gallery/animations/index.html.



Автор статьи: ***Gecko*** *clocktower89@mail.ru*

Всем известен популярный бесплатный трехмерный редактор Anim8tor Стивена Гленвилля. Он располагает огромным количеством инструментов моделирования, создания анимации и рендеринга сцен, что, несомненно, выгодно выделяет его на фоне остальных 3D-моделлеров. Но кто мог знать, что на основе Anim8tor существует и развивается настоящий конструктор трехмерных игр? Вот и я раньше не знал.

Конструктор называется **Cre8tor**. Он создан на движке KJX Engine с использованием библиотек Ogre, что обеспечивает высокую скорость рендеринга и неплохую физику. Кстати говоря, физика - это главный козырь Cre8tor, так как никогда еще создание реалистичных физических моделей не было таким простым и наглядным. Программа находится на стадии ранней альфы (0.075a), но тем не менее, оценить ее возможности и пророчить ей большое будущее можно уже сейчас.

Итак, первое, что нужно сделать, это скачать дистрибутив **Cre8tor** (5.3 Мб):
http://www.cre8or3d.com/downloads/Cre8or_0_07a_100807.zip

Для работы конструктору требуется установленный в системе **Microsoft .Net Framework 2.0**. Весит он сравнительно немало - 22 Мб, но скачивать все равно рано или поздно придется: сейчас все больше софта пишется именно под .Net.
http://www.cre8or3d.com/downloads/DotNet2FrameWork.zip

Если вы планируете делать игры, то понадобится и **Anim8tor** (около 1 Мб), так как всю игровую среду придется делать в нем:
http://www.anim8tor.com

Основной принцип работы Cre8tor заключается в следующем: загружается файл an8 (файл Anim8tor), все объекты конвертируются в объекты KJX, а остальное делает сам движок. Cre8tor также поддерживает и другие аниматоровские фичи, но с некторыми ограничениями:

- Поддерживаются материалы, но текстуры для них не будут загружены автоматически, это придется делать вручную.
- Динамическая физика может быть применена только к примитивам (сфера, куб, капсула), меши могут быть использованы лишь как статичные объекты.
- Поддерживаются фигуры (Figures), но они должны использовать только один меш.
- Пока не поддерживается анимация сцен.
- Пока не поддерживаются частицы.

На основе вышесказанного можно подвести некий итог - какие элементы Anim8tor чему соответствуют:

Примитивы (Primitives)	объекты с возможностью применять динамическую физику
Меши (Meshes)	объекты с возможностью применять статическую физику
Материалы (Materials)	материалы
Объекты (Objects)	игровые объекты
Фигуры (Figures)	игровые объекты с возможностью анимации
Последовательности (Sequences)	анимации (последовательности кадров) для Фигур
Сцены (Scenes)	игровые уровни

Для всех этих элементов отведены директории в дереве ресурсов. Такую структуру используют многие конструкторы (да и не только конструкторы), так как это наглядно и удобно.

Но игра - это не только трехмерные объекты, пусть даже и с физикой и анимацией. Это еще и звуки, текстуры, различные эффекты и т.д. Все это полностью поддерживается в Cre8tor. Для них отведен отдельный класс элементов *Extras*:

Звуки (Sounds)	поддержка wav,mp3,ogg
Текстуры (Textures)	поддержка bmp,jpg,gif,png,dds
Скрипты (Scripts)	на встроенном языке программирования. Поддержка загрузки из kjs, nut, sq, txt
Шейдеры (Shaders)	низкоуровневые подпрограммы для создания эффектов на основе пиксельных (поддержка 2.0 и 3.0) и вертексных шейдеров (поддержка 1.0 и 2.0). Поддержка загрузки из kjs, nut, sq, txt.
Небесные коробки (Skyboxes)	рапространенная технология для создания неба. В качестве неба используется куб, к сторонам которого применяются бесшовные изображения, визуально формирующие эффект окружающего фона (небо, пейзаж и т.д.)
Материалы (Materials)	имеется возможность писать код материалов на встроенном языке. Поддержка загрузки из kjs, nut, sq, txt
Композиторы (Compositors)	материалы + шейдеры

Принцип создания алгоритмов в Cre8tor позаимствован из Game Maker. Видимо, автор посчитал, что принцип "событие - действие" наиболее удобен. Но, в общем-то, все единогласно сошлись во мнении, что так оно и есть. Иначе GM не был бы столь популярен. Я думаю, объяснять тут что-либо излишне: любой, кто более-менее знаком с Game Maker, особых трудностей не встретит. Можно лишь упомянуть, что Cre8tor располагает меньшим, чем GM, списком событий (Create, Step, Destroy, Variable, Modes, Collision и Keyboard с поддержкой "джентльменского набора" из стрелок, Ctrl, Alt, Shift, Space, Enter и алфавитно-цифровой раскладки). Воможно, в будущих версиях будут добавлены еще события. Но, в любом случае, это с лихвой компенсируется набором действий: тут вам и стандартные структуры If, Else, и выполнение скриптов и частей кода, и действия для работы со светом, тенями, туманом и эффектами, действия образца (Entity и Instance), встроенные действия по контролю над персонажем, физикой и камерой, а также для проигрывания звуков. Визуально "событие - действие" решено, правда, не очень впечатляюще (не в пример хуже, чем в том же GM), но, думаю, это лишь вопрос времени.

Вот что хорошо, так это то, что, несмотря на альфа, в Cre8tor уже сейчас можно компилировать и собирать самостоятельные дистрибутивы проектов. А то часто бывает, что ранние версии подобных конструкторов представляют из себя в лучшем случае подобия демо, с ограниченными возможностями и отсутствием сохранения в exe-файл. Тут все наоборот. Вам даже не нужно мучиться со сборкой проекта: программа экспортирует всю игру в zip-архив, распаковав который, вы получите самостоятельную версию вашей игры. Вот так все просто!

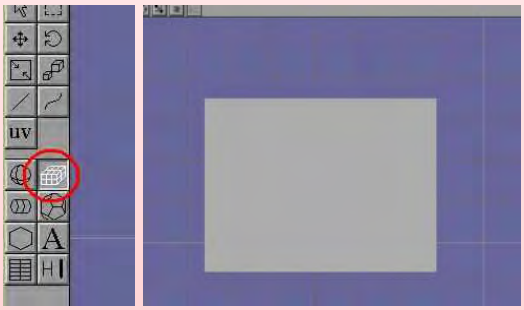
Запустив exe-файл с игрой, вы получите возможность выбрать графическую подсистему (DirectX или OpenGL) и разрешение экрана, после чего запустится сама игра. Кстати, выбор опций графики можно и отключить - в специальном элементе Game Options, аналогичном Global Game Settings в GM.

Неплохо в Cre8tor решена и помощь. Как такового, файла справки в программе вообще нет. Есть только туториалы, загружаемые непосредственно с сайта программы и информационная документация в виде Вики, загружаемая оттуда же. Также из окна помощи можно отправится на форум Cre8tor и на сайт Anim8tor. Туториалы подробные, в них шаг за шагом вы научитесь использовать возможности конструктора: сначала самые простые, затем посложнее. Правда, туториалы на английском, поэтому, для облегчения работы нашим соотечественникам, я буду публиковать их переводы.

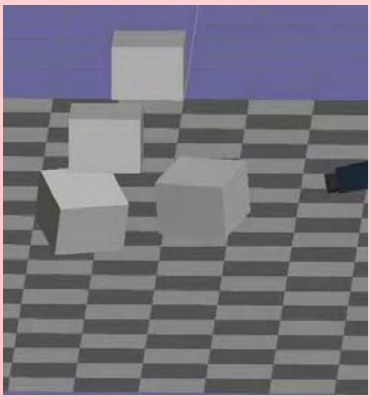
Уроки Cre8tor.

Часть 1. Самое основное.

1 Запустите Anim8tor. Создайте куб и придайте ему произвольную форму:



2 Перейдите в режим Сцены (Mode > Scene) и добавьте несколько кубов (Build > Add Object). Расположите их случайным образом и в разных поворотах:



3 Сохраните файл (File > Save). Запустите Cre8tor и выберите My An8 file > Assign Anim8tor file.

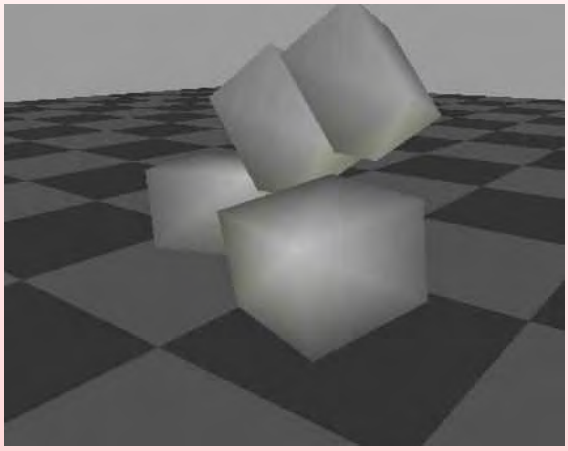
4 Загрузите свой файл an8. В дереве ресурсов найдите Objects / Object01.

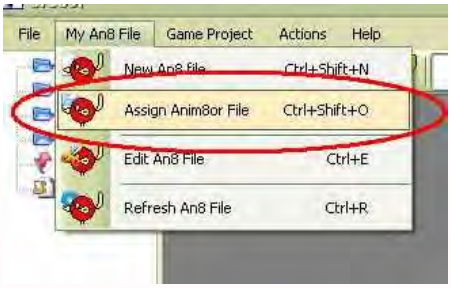
5 Кликните по нему дважды. Нажмите кнопку Edit Physical.

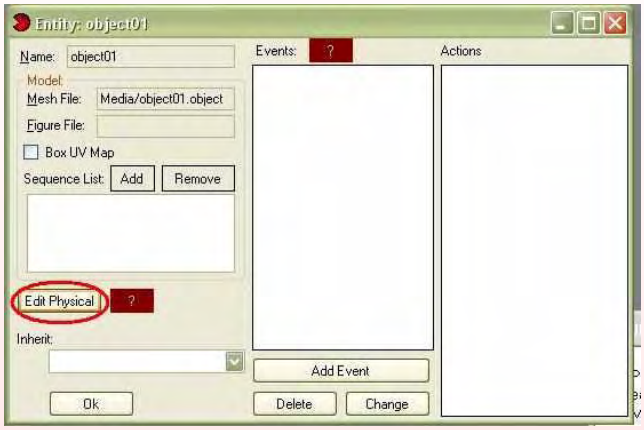
6 Выберите Cube. Программа автоматически распознает размеры и позицию куба, поэтому обведенные синим значения менять не нужно.

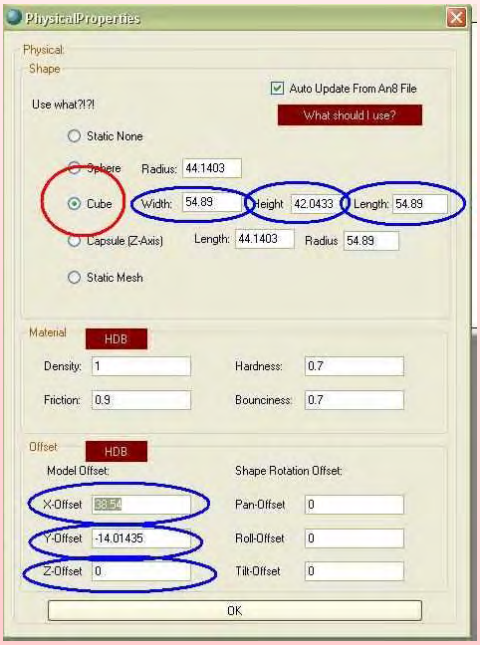
7 Нажмите кнопку Run Game и наблюдайте физику в действии! Это, правда, пока далеко не игра, но уже шаг на пути к ней.

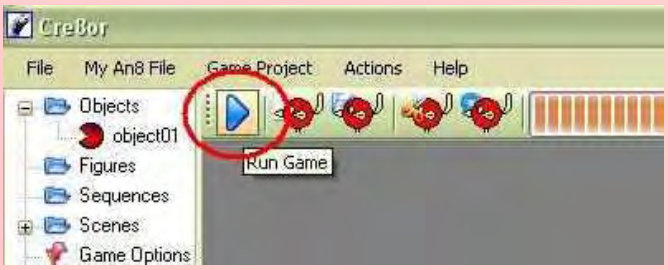
Выглядеть должно примерно так:











Справедливости ради необходимо упомянуть, что при всех ее достоинствах Cre8tor — программа еще довольно сырая. Многое нуждается в проработке, баги — в исправлении и т.д. Надеемся, в будущих версиях Cre8tor будет улучшен, и мы увидим еще один достойный продукт в коллекцию геймкриейтора.

Официальный сайт Cre8tor:
http://www.cre8tor3d.com

Официальный форум:
http://www.cre8or3d.com/forum/

Автор статьи:
Gecko *clocktower89@mail.ru*

Уроки моделлинга

Моделируем дерево для Game Maker

Моделирование растений, особенно деревьев - само по себе непростое занятие, еще сложнее - моделировать для игр, тем более для игр Game Maker. Я перепробовал много способов, пока не пришел к самому, на мой взгляд, удобному. Им сегодня я и поделюсь.

Вам понадобятся следующие бесплатные программы:

Студия 3D-графики **Anim8tor**
http://www.anim8tor.com

Редактор карт текстурирования **UVMapper**
http://www.uvmapper.com

Конвертер моделей **Marzipan**
http://www.percsisch.hu/mapzipan

Мы будем моделировать пальму, но этот принцип применим к любому дереву.

1 Сначала нам необходимо создать модели для листа и свола. Сделаем сначала лист. Открываем Anim8tor и строим многоугольник (N-Gon): *Build* > *Primitives* > *N-Gon*.
В настройках указываем количество сторон 4. Для того, чтобы создать многоугольник, просто потяните зажмите левую кнопку мыши и потяните курсор вниз.

2 Получившийся квадрат ставим в точности на центр поля при помощи инструмента *Move*. Для верности вы можете посмотреть на результат со всех сторон: *View* > *All*.

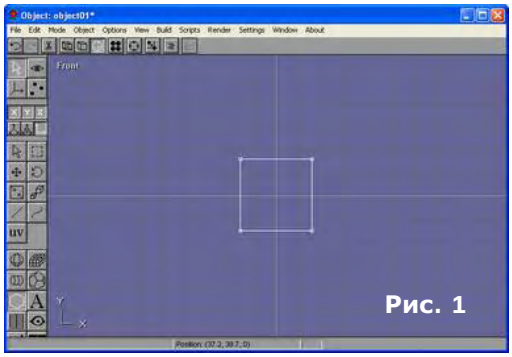


Рис. 1

3 Важная процедура: полученный квадрат представляет собой лишь линейный векторный контур, поэтому его необходимо "заполнить": *Build* > *Fill ...* При этом вы увидите, как внутри контура появится серая поверхность.

4 Сохраняем лист, причем обязательно в формате OBJ (зачем - будет ясно чуть позже): *Object* > *Export ...*

5 Создаем новую модель: *Object* > *New* и строим цилиндр: *Build* > *Primitives* > *Cylinder*. Создайте цилиндр, как описано в шаге 1. Конвертируйте его в меш *Build* > *Convert to Mesh*

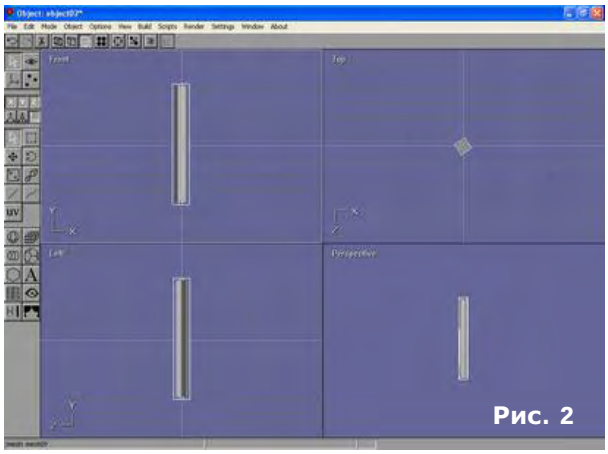


Рис. 2

6 Если цилиндр недостаточно длинен для ствола, увеличьте длину при помощи инструмента *Non-Uniform Scale*, после чего обязательно поставьте его в центр поля.

7 Сохраните модель, как в шаге 4.

8 Теперь поработаем в UVMapper. Откройте в нем модель листа. Скорее всего, появится надпись "This object has no UV texture coordinates. *Use Edit-> New UV Map to create some*". Создайте новый мэп *Edit* > *New UV Map* > *Planar*. Поставьте настройки, как на скриншоте (рис. 3). Должен получиться большой прямоугольник с диагональю. Выделите его левой кнопкой мыши и уменьшите длину примерно в два раза, потянув за рамку. Переместите как можно ближе к левому краю, чтобы сэкономить место в правой части. Получится примерно как на рис. 4.

9 Сохраните модель *File* > *Save Model...* и из опций оставьте только *Export UV Coordinates*. На всякий случай не перезаписывайте старую модель, сохраните под новым именем. Позже старый вариант можно будет удалить.

10 Теперь сохраните мэп как изображение: *File* > *Save Texture Map...* В настройках укажите размер 512x512.

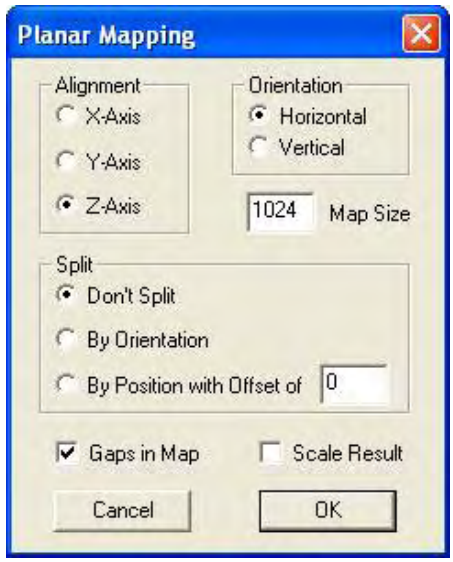


Рис. 3

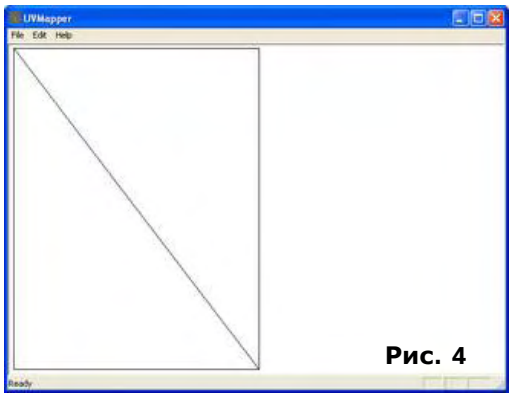


Рис. 4

11 Далее повторяем шаги 7 - 9 с моделью ствола. Когда модель будет загружена, вероятно, вы увидите мэп, созданный Anim8tor - сетку по всей поверхности. Но специфика нашей задачи такова, что нам так не надо. Выделите всю сетку целиком. Это не так просто, как может показаться. Переместите ближе к центру и уменьшите. Растягивайте и перемещайте сетку, пока не получится что-то похожее на это (рис. 4)

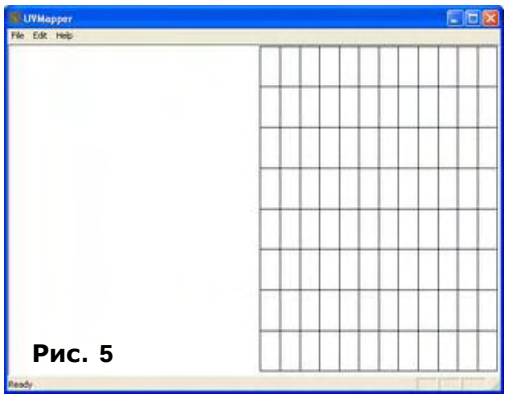


Рис. 5

12 Теперь у вас две новые модели и два мэпа. Объедините их в один, как показано на рисунке 6. Это можно сделать в любом графическом редакторе, хоть в Paint. Ни в коем случае не меняйте расположения прямоугольников, только объедините. Если получается, что один заходит на другой, откройте модель заново в UVMapper и подвиньте/уменьшите.

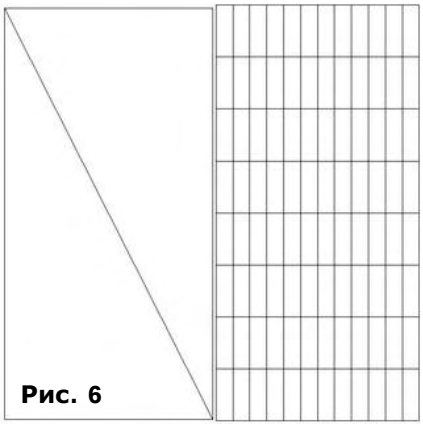


Рис. 6

Рис. 7

13 Нарисуйте текстуры на полученном мэпе. Слева у нас будет лист, справа ствол. Как рисовать, я вас учить не буду, на то есть другие уроки. Напимер, в первом выпуске "FPS" есть урок по рисованию текстуры дерева. Единственный совет: лист должен быть на однотонном фоне, причем без анти-алиасинга - это необходимо для получения частичной прозрачности.
В общем, у меня получилось так (рис. 7)

14 Я думаю, читателю стала понятна основная идея: мы сделали два отдельных объекта, использующих одну и ту же текстуру. На их основе можно сделать дерево с неограниченным количеством элементов, использующих одно и то же изображение! И сейчас последний этап: мы именно это и сделаем. Открываем Anim8tor и загружаем обе модели в разных объектах. Создаем новый, третий объект и копируем в него и ствол, и лист (*Edit* > *Copy* и *Edit* > *Paste*).

15 Чтобы посмотреть, как смотрится текстура, создадим Материал. Нажмите *Ctrl+M* и на появившейся панели кликните два раза на квадрате *New*. Нажмите кнопку Textures, на появившемся окне настроек текстур нажмите кнопку "..." напротив надписи *Ambient*. Появится окно выбора текстур, в котором следует нажать *Load Texture*. Загрузите свою текстуру и выберите ее в списке. Нажмите OK. Затем в окне настроек текстур аналогично выберте ту же текстуру для *Diffuse*, *Specular* и *Emissive*. Примените созданный Материал к стволу и листу: для этого необходимо выбрать объект и на панели Материалов нажать *Apply*. Если до этого все было сделано правильно, вы увидите, как одна и та же текстура идеально ляжет на оба объекта (рис. 8).
Частичная прозрачность не будет применена, поэтому фон листьев так и останется черным. Но на данном этапе это не важно.

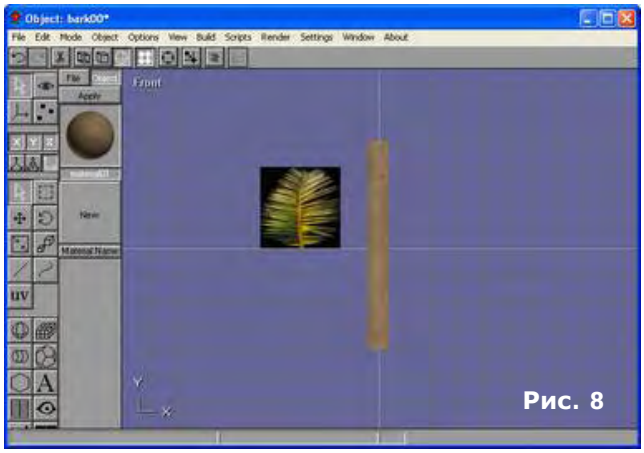


Рис. 8

16 Теперь приступаем к созданию пальмы. Выберите ствол, перейдите в режим правки точек (*Object/Point Edit*) и переключитесь на каркасный вид (*Ctrl+W*). Ствол представляет собой цилиндр из девяти вертикальных секторов - такая структура поможет придать стволу нужную нам форму, а именно - высокий усеченный конус. Для этого поочередно выбирайте сектора от низа к верху и уменьшайте их инструментом *Scale*.

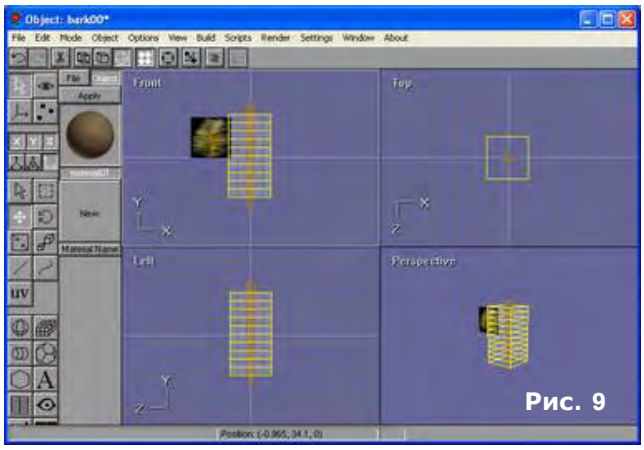


Рис. 9

17 Это еще не все. Пальмы обычно бывают изогнутые, поэтому настало время применить один из самых интересных инструментов Anim8tor: Модификатор (*Modifier*). Работает он следующим образом: вы создаете Модификатор определенного типа и привязываете его к какому-либо объекту. Теперь все изменения, которые вы сделаете в настройках Модификатора, отразятся и на объекте. Модификаторы бывают пяти типов: *Bend*, *Curvative*, *Skew*, *Taper* и *Twist*. Мы будем использовать *Bend* - Сгиб.

Выберите инструмент *Warp Modifier* и создайте Модификатор, как будто это обычный примитив. В появившемся меню выберите *Bend*. Используя инструменты поворота (*Rotate*), перемещения (*Move*) и масштабирования (*Scale*) установите Модификатор, как показано на рисунке 9.

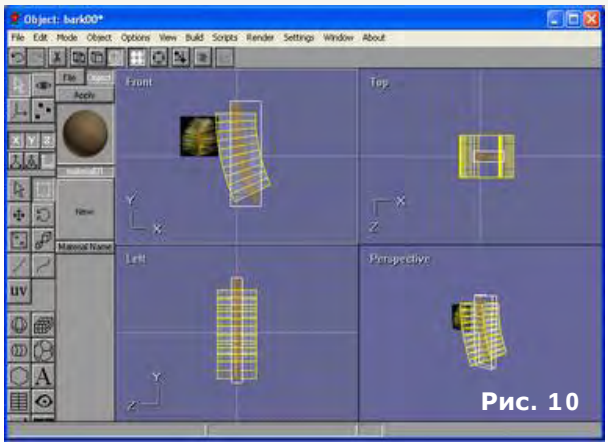


Рис. 10

18 Теперь инструментом *Drag Select* выделите Модификатор вместе со стволом и свяжите их: *Build* > *Modifiers* > *Bind Modifier*. Кликните два раза по Модификатору и в настройках укажите *Angle* как 25. При нажатии ОК ствол должен красиво согнуться, как на рис. 10. Если не понравилось, можно нажать *Undo* и попробовать другой угол.

19 Выберите *Build* > *Modifiers* > *Effect Modifier* и Модификатор исчезнет, оставив изогнутый ствол, который нужно только перевернуть. Теперь осталось расставить листья, придав сначала им нужный размер инструментом *Non-Uniform Scale*. Просто копируйте и вставляйте объекты, столько раз, сколько сочтете нужным. Расположите листься по кругу, поочередно поворачивая и перемещая. Готовую группу можно объединить: *Build* > *Group* и уже тогда поставить на вершине ствола, повернув на нужный угол (рис. 11).

20 Все! Теперь можно экспортировать модель, уже в тот формат, какой вам нужен. Я все же посоветую OBJ, так как с ним программы работают лучше всего.

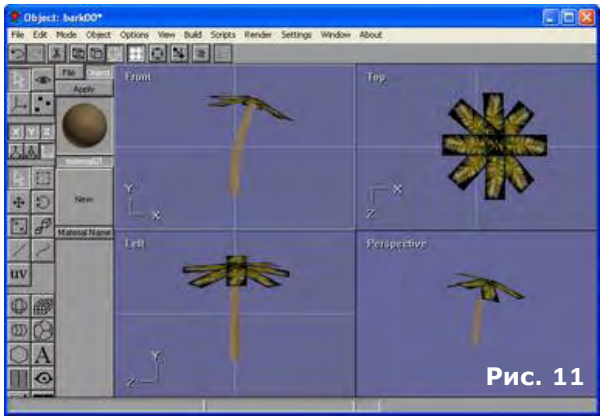


Рис. 11

Как загрузить модель в Game Maker? Я обычно пользуюсь конвертером Marzipan и делаю так: загружаю модель *File* > *Load Model*, убираю отрисовку нормалей *Tools* > *Toggle show normals*, поднимаю модель до уровня нулевой плоскости клавишей *PageUp*, смотрю, как смотрится текстура *Settings* > *Change texture* (кстати сказать, текстуру перед этим надо перевернуть, т.к. GM читает их именно так), фиксирую нормали *Tools* > *Fix normals*, смотрю, как модель смотрится в освещении (кнопка с изображением лампочки), и, если все нормально (рис. 12), экспортирую в формат DAT: *Export* > *Data Only*. Будут созданы два файла: **<name>_normals.dat** и **<name>_description.dat**, где **<name>** - название файла. Первый как раз и будет нужен в игре. Для загрузки модели в GML используется следующий скрипт:

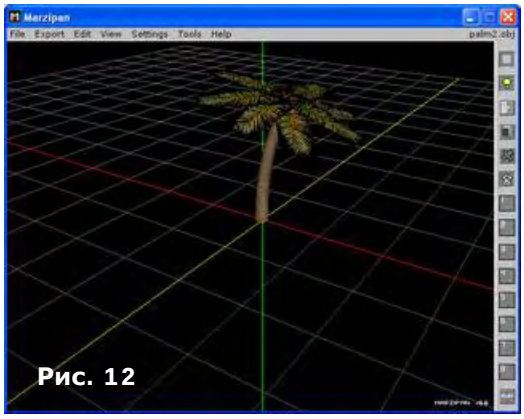


Рис. 12

```
var model,fp,vx,vy,vz,nx,ny,nz,tx,ty;
model=d3d_model_create();
d3d_model_clear(model);
d3d_model_primitive_begin(model, pr_trianglelist);
fp = file_text_open_read(argument0);
for (v=0; file_text_eof(fp)==false; v+=8) {
    vx = file_text_read_real(fp); file_text_readln(fp);
    vy = file_text_read_real(fp); file_text_readln(fp);
    vz = file_text_read_real(fp); file_text_readln(fp);
    nx = file_text_read_real(fp); file_text_readln(fp);
    ny = file_text_read_real(fp); file_text_readln(fp);
    nz = file_text_read_real(fp); file_text_readln(fp);
    tx = file_text_read_real(fp); file_text_readln(fp);
    ty = file_text_read_real(fp); file_text_readln(fp);
    d3d_model_vertex_normal_texture
(model,vx,vy,vz,nx,ny,nz,tx,ty);
}
file_text_close(fp);
d3d_model_primitive_end(model);
return model;
```

Рис. 13

где `argument0` - путь к файлу DAT. Остальное - уже за вами. Удачи!
(На рис. 13 - пальма непосредственно в игре).

Автор статьи: **Gecko** clocktower89@mail.ru

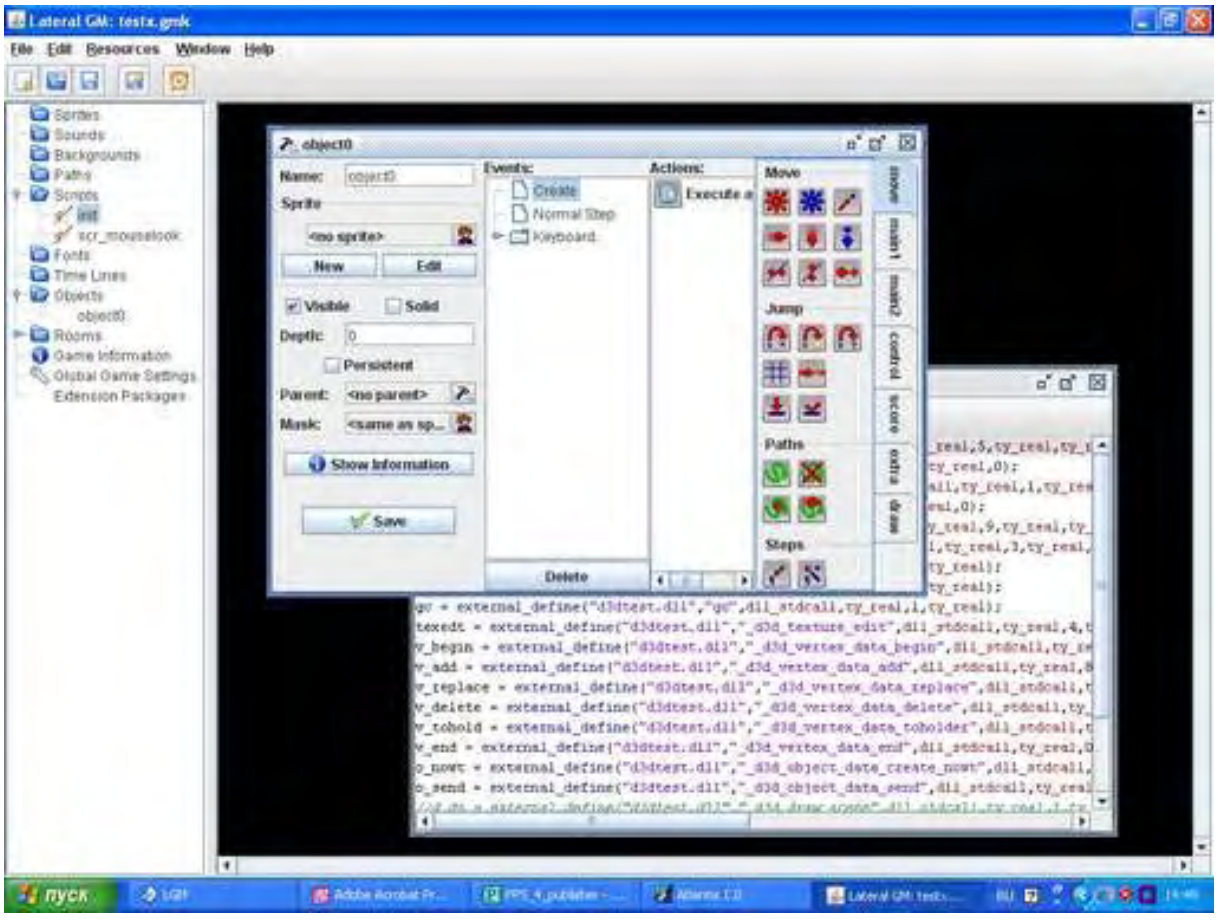
LATERAL GM

Конец «битвы форматов»

Всем известно, что в начале прошлого года вышла седьмая версия Game Maker. Можно сколь угодно ругать ее за баги, exe-шники в два мегабайта, отсутствие серьезных изменений и т.д., но от факта, что часть разработчиков все ж-таки пересели на нее, просто так не отмахнуться. Тем более, что не просто пересели. Мировое сообщество GM разделилось на два лагеря, как когда-то это было с GM5 и GM6. Одни посчитали, что возможностей шестой версии вполне хватает, и остались верны ей - от добра добра не ищут. Другие, видимо, решили, что новая версия есть новая версия, и поддержали политику YoYo Games с их неудержимой тенденцией к продвижению GM7. И все, в принципе, по-своему правы. В сложившейся ситуации, на мой взгляд, ничего катастрофичного нет, есть только одна проблема, причем довольно серьезная.

С каждой новой версией GM Марк Овермарс вводит "в игру" новый формат исходника. Поддержка старых форматов новой версией конструктора в какой-то мере сохраняется, а вот старые версии новинку "не понимают". Получается неприятное положение: скачали вы, к примеру, полезный движок или демку, и на тебе - в формате GMK! Можно, конечно, не полениться и скачать GM7, но по некоторым причинам многие предпочитают использовать старые версии GM. Лично у меня, например, когда-то GM7 был, но его, вместе с третью всех других данных, унесла внезапная поломка винчестера. С тех пор я так и не удосужился скачать GM7, и, честно говоря, не собираюсь. И не только из принципа.

А еще потому, что существует вещь, которая, уверен, для многих по большому счету просто заменит GM7. Встречайте - **Lateral GM**, или сокращенно **LGM**.



LGM - это универсальный межверсийный интерфейс Game Maker, с поддержкой GM трех поколений (версии 5.x, 6.x и 7.x). Применительно к LGM можно было бы употребить термин "конвертер", но это, в общем-то, довольно сухое понятие вряд ли в полной мере отразит весь функционал данной программы. LGM не просто свободно открывает и сохраняет форматы вышеупомянутых версий GM, но еще и позволяет довольно глубоко проникнуть в структуру и содержимое файлов, кое-что даже поддается редактированию!

В целом, LGM весьма схож со стандартным интерфейсом GM. То же дерево ресурсов слева, те же редакторы спрайтов, объектов, скриптов... Как я уже упомянул, некоторые вещи поддаются изменению. Это касается действий, скриптов и частей кода, практически всех типов ресурсов, за небольшим исключением - в редакторе комнат отсутствует возможность расстановки объектов, а в редакторе спрайтов нет непосредственно самого редактора анимации и графики. Но это все сугубо опционально, для таких вещей GM есть, как-никак :)

Представим, будто в LGM исправили все недочеты и даже добавили компилятор. То есть, из межверсийного интерфейса с урезанными возможностями, программа стала полноценной заменой GM. Причем бесплатной. Но тогда это будет просто варварским нарушением авторских прав, и LGM по понятным причинам, скорее всего, пришлось бы прекратить свое существование. А так, все чисто, никто не подкопается. Это правда, так как из текста лицензии, прилагаемой к GM6, в целом можно сделать следующий вывод: пользователь не имеет права декомпилировать исполняемый файл Game Maker и игр, сделанных в нем, но имеет полное право распространять игры любым образом без лицензионных ограничений. Честно говоря, я так и не понял толком, можно ли распространять сам GM (или его русификации, кряки, серийники и т.п.), не говоря уж о файлах GM6. Хотя строчка "Games you create with the Software you can use in any way you like. No licence fee is required" само по себе указывает на то, что игры могут быть использованы любым образом, но не уточняет, какие версии игр - исполняемые EXE или исходные GM6. Запрет на декомпилирование касается только EXE: "Stand-alone versions of games created with the Software can be distributed without any licensing fees. You are however not allowed to reverse engineer or decompile these executable." Это и есть "дырка", по которой LGM можно считать лицензионно чистым. Он работает только с исходными файлами игр, никоим образом не касаясь исполняемых файлов. Так что, пока YoYo Games не опубликуют новую, более подробную (и, соответственно, более строгую) лицензию, использование LGM вполне законно. Автор программы утверждает, что сам Марк Овермарс не был против подобной разработки. Более того, легендарный создатель GM сам зачастую выпускает утилиты, на первый взгляд, противоречащие политике выпуска новых версий. Судите сами: Game Maker 7, в отличие от старых версий GM, может похвастаться поддержкой Windows Vista, в то время как существует конвертер exe-шников, скомпилированных на GM 6.x в приложения для работы под Vista. Одним козырем GM7 меньше. А LGM - это утилита "из той же оперы", просто очень (я бы даже сказал "слишком") похожая на сам GM. И ничего более.

Тем не менее, полезность LGM, я думаю, обсуждению не подлежит. Люди реально ждали появления такого инструмента. Единственное, что может добавить ложку дегтя - это то, что LGM является приложением Java и для запуска требует установленную среду Java (JRE), желательно одной из последних версий. JRE весит около семи мегабайт, и скачать ее можно на сайте http://www.java.com. Я бы вообще на месте Microsoft непременно включал в состав Windows XP такие нужнейшие вещи, как JRE, .Net Framework и т.д., так как при нынешнем положении дел это первое, что нужно скачивать после установки ОС. Слишком уж много софта их использует...

Скачать LGM можно по следующей ссылке: http://www.ismavatar.com/lgm/lgm1.5.6.jar. По ней находится последняя недельная сборка LGM (у меня версия 1.6.0). Размер программы около 800 кб.

Автор статьи:
Gecko *clocktower89@mail.ru*

Название: Park Racer
Разработчик: Zoltán Pércsich
Платформа: PC
Жанр: аркадный симулятор автогонок
Год выхода: 2005
Ссылка на скачивание:
http://www.gamemakergames.com/?view.download&id=3416

Нечасто встречаешь бесплатные игры жанра гонок. Тем более, гонки, сделанные в GM, на котором качественных трехмерных игр - по пальцам можно пересчитать. Сегодня мы познакомимся с одной малоизвестным, но, тем не менее, достойным проектом от Zoltan Percsich - создателя небезызвестного Silent Walk FPS Creator.

Это гонки в традициях Формулы-1. Можно ли назвать их автосимулятором в полном смысле этого слова? Скорее всего, нет. Но от этого игра не становится хуже, даже наоборот. Изюминка Park Racer - в аркадных моментах, которые и превращают заурядное вождение в захватывающую гонку.

В игре четыре игровых режима: Practice, Time attack, Single race и Bomber run. Practice - просто практика. Выбирайте машину, трассу и просто ездите в одиночку. Time attack - интересный режим. Вы соревнуетесь с собственной тенью: при прохождении круга на трассе появляется автомобиль-"призрак", повторяющий ваш предыдущий круг. Single Race - одиночный заезд - гонка с оппонентами. Вы можете выбрать один из уровней сложности, определяющих мастерство соперников. На трассе появляются бонусы в качестве дозаправки и ремонта. Кстати говоря, в Park Racer наличествует учет повреждений и расхода топлива, так что необходимо быть внимательным. Наконец, Bomber run, где все то же, что и в одиночном заезде, но с возможностью сбрасывать на дорогу мины, усложняя жизнь соперникам. Но осторожно: оппоненты также наделены этой способностью, да и на собственных минах подорваться недолго.

Как я уже говорил, это скорее аркада, чем автосимулятор: реалистичной физики, тем более - какого-то революционного искственного интеллекта ждать не приходится. Соперники мчатся кучей, со стабильной скоростью согласно выбранному уровню сложности, но и повороты проходят профессионально - ни глупых столкновений, ни "пьяных водил". Управление автомобилем далеко не самое худшее, чего обычно опасаешься, но до идеала ему еще далеко. Трассы проложены строго горизонтально, без всяких горок и склонов, так что машины стоят на земле всеми четырьмя колесами, неспособные оторваться даже на сантиметр.

Кстати, о трассах: их в игре целых двенадцать штук. И все, как на подбор, отличаются лишь картой дороги - везде одни и те же деревья, трава, асфальт и пресловутые красно-белые ограничители. Это действительно несколько разочаровывает. Правда, игра на то и называется Park Racer, чтобы ездить в ней в автопарках, но ведь можно было поставить какие-нибудь ограждения, дома, гаражи с припаркованными машинами, рекламные баннеры, в конце-концов. Весь фокус в том, что, когда все эти декорации присутствуют, на них никто внимания не обращает - ну, стоят себе и стоят - а когда их нет, это сразу бросается в глаза. В этом-то и вся роль второстепенных деталей, пренебрегать которыми нельзя ни в коем случае (на заметку игроделам). Спасибо хоть на том, что ввели погодные условия: иногда льет дождь, и это ставит всю гонку с ног на голову, так как видимости никакой, машину все время заносит... Весело!

Зато конфигурация трасс, которой они между собой отличаются, - важный фактор при выборе автомобиля. Всего машин пять, и в общем-то, похожи они только внешне. Характеристики у всех разные, причем различия эти четко выражены и не представляют из себя какие-то абстрактные цифры в лошадиных силах, которые интересны только с познавательной точки зрения. Нет! Тут наглядно отображены три показателя: управляемость, максимальная скорость и ускорение. Ориентируясь на них, нетрудно подобрать подходящую машину для трассы с теми или иными особенностями. А из краткого пояснения можно узнать, какое именно качество нужно ценить в конкретном автомобиле.

О графике. Тут, надо признать, чувствуется рука мастера. А чего еще ждать от создателя знаменитого Marzipan! Весьма недурственные модели и текстуры машин, очень красиво и, главное, профессионально с точки зрения дизайна, выполненные меню, хорошо подобранные шрифты для текста... На дороге остаются следы от шин, деревья и сами автомобили отбрасывают тени. В дождь трассы окутывает серая пелена. При выборе машины модель вращается на зеркальной поверхности - очень стильно. На фоне главного меню так вообще отображается сам процесс гонок, как будто снятый на камеру. На сайте gamemakergames.com, откуда, собственно, Park Racer и перекочевала в мой компьютер, игра получила высочайшие пользовательские оценки: 8, 9 или 10 из 10, и, как можно судить по комментариям, в основном, за графику. Пускай здесь нет навороченных современных эффектов, а сама игра напоминает порт с N64 - опытные люди знают, что нельзя все на свете оценивать с позиции "детей прогресса" и технологий уровня Unreal Engine 3. Всякая графика вполне уместна там, где ее правильно используют.

Звук. Что ж, ругать здесь определенно нечего. Все очень "чисто", тщательно подобрано и звучит, когда надо звучать. Приятная музыка в жанре хаус - как раз для таких гонок, качественные звуковые эффекты, что еще скажешь.

В итоге - перед нами вполне качественный, нашедший своего потребителя продукт, неплохой подарок для любителей аркадных гонок. Ничего не остается, как только добавить: заводите моторы, господа! Шумахер отдыхает =).

Автор статьи:
Gecko *clocktower89@mail.ru*

Психология компьютерных игр

Часть первая - почему мозг любит игры

Все мы знакомы с ощущением погружения в интересную компьютерную игру. Мы концентрируем на ней свое внимание, и все остальное теряет для нас значение. Именно этому явлению и посвящена данная статья - почему появляется это ощущение и как создавать игры, дающие максимальный эффект.

В первой части статьи мы рассмотрим человеческое поведение, факторы, определяющие его, и причины, по которым нам нравится выполнять определенные действия. Сначала вам может показаться, что тема статьи находится несколько в стороне от компьютерных игр, однако скоро вы убедитесь, что игра - это лишь то, что лежит на поверхности, а на практике все значительно сложнее. Чтобы понять причины получения удовольствия от определенных компьютерных игр, нужно посмотреть, что находится под поверхностью и постараться понять, каким образом их видит наше сознание.

Вторая часть статьи концентрирует внимание на разработке игр, соответствующих нуждам, интересам и возможностям игроков. Мы узнаем, насколько важно понять аудиторию, для которой предназначена игра, и как адаптировать создаваемую игру под предполагаемого игрока. Люди одновременно имеют как общие, так и различные интересы, и дизайн игры должен учитывать этот факт.

В общем, цель данной статьи - объяснить читателю, почему хорошие компьютерные игры доставляют нам столь большое удовольствие, и рассказать о разработке игр с учетом этих причин. Я полагаю, что, взглянув на игры с позиции нашего воображения, мы сможем придумать новые способы создания компьютерных игр, способных увлечь игрока.

В течение долгого времени у людей формировались различные потребности, тесно связанные с выживанием, существованием и эволюцией. В сущности, все наши действия в повседневной жизни прямо или косвенно определены этими потребностями. Абрахам Маслоу - первый ученый, обобщивший результаты исследований поведения человека. Он создал список человеческих потребностей и организовал их в виде иерархии. В основе иерархии потребностей Маслоу лежат две группы - первичные потребности и высшие потребности. В иерархии первичных потребностей каждая потребность должна быть удовлетворена раньше тех, которые расположены выше ее. Если позднее у человека возникает потребность более низкого уровня, он сначала направляет свои действия на ее удовлетворение, и лишь затем возвращается к потребностям более высокого уровня. Когда первичные потребности удовлетворены, внимание индивида переключается на высшие потребности.

Первичные потребности:

1. Физиологические потребности: дышать, пить, есть, спать, испытывать телесный комфорт и т. д.
2. Потребности безопасности: ощущение безопасности, защищенность, отсутствие вредоносных воздействий, безопасные соседи, знание своих планов заранее.
3. Потребности привязанности и любви: присоединиться к обществу людей, быть принятым другими, принадлежать к группе, любить и быть любимым, иметь семью, общаться.
4а. Низшие потребности самооценки: быть уважаемым, получать внимание, иметь статус, силу, репутацию, достоинство, выражать себя в словах, одежде или творчестве.
4б. Высшие потребности самооценки: испытывать уважение к себе, быть дееспособным, достигнуть независимости и свободы.

Высшие потребности:

5. Познавательные потребности: Знать, понимать, исследовать, искать приключения, пробовать новое, путешествовать, чувствовать волнение.
6. Потребности самореализации: Ощущать собственный потенциал, реализовывать свои возможности.
7. Трансцендентальные потребности: Помогать другим в самоосуществлении и реализации их потенциала.

Награда за хорошее поведение

Чтобы мы прислушивались к своим потребностям, эволюция наделила наш мозг специальными зонами поощрения, стимулирующими правильное поведение, например, употребление жидкости в случае жажды. Эти зоны выделяют допамин, медиатор, связанный с чувствами удовольствия, чтобы наше тело повторяло правильные действия снова и снова. По этой причине удовлетворение потребностей часто ассоциируют с ощущениями удовольствия. Можно сказать, что наш мозг указывает нам правильное поведение, поощряя за него генерацией химических веществ, вызывающих удовольствие.

Подобные рассуждения легко применить к потребностям низкого уровня: мы испытываем голод, едим и получаем в награду насыщение. Тем не менее, все значительно усложняется, когда дело касается более высокоуровневых потребностей, например, уважения со стороны других людей. Высшие потребности имеют не физическую, а психологическую природу. Как же тогда поощрять конкретные действия или поведение, если каждый человек оценивает свои потребности по-своему?

Интересный факт заключается в том, что мы сами определяем, достигли ли своими действиями намеченной цели и заслуживаем ли поощрения. С объективной точки зрения неважно, что, как и почему мы делаем; если мы делаем то, что нужно так, как нужно, и достигаем подходящих результатов, мы получаем свою дозу допамина. Например, неважно, выиграли мы в лотерею или всего лишь так думаем; обе ситуации одинаково приятны для нас до тех пор, пока мы не начинаем тратить деньги.

Измерение результатов

Можно сказать, что на уровне нашего личного восприятия главная цель любой деятельности - получить удовольствие и/или избежать боли. Если мы знаем, что потенциальное удовольствие от какого-либо действия больше возможной неудачи или боли, то, скорее всего, мы совершим его. Количество удовольствия или боли от действия определяется нашим опытом (было ли это интересно мне?), субъективного отношения к самому действию (понравилось ли мне?) и мерой достигнутого успеха (хорошо ли у меня получилось?).

Проблемы, с которыми мы сталкиваемся в современном обществе - это огромное количество возможностей и нечеткие границы между правильным и неправильным, что очень редко позволяет нам адекватно оценить результаты собственных действий. Это означает, что нам, как правило, трудно решить, что делать, и еще труднее получить удовольствие от хороших результатов. Мы склонны соотносить себя с другими людьми, и всегда обнаруживаем того, кто умнее, счастливее, талантливее или просто выглядит лучше нас. Так как же нам обратить собственные действия себе в удовольствие?

Игры

В решении этой проблемы на помощь к нам приходят игры, причем не только видеоигры, а игры вообще. Вы можете играть с другими людьми, с самим собой, с компьютерами, и даже с силами природы, однако у всех ваших возможных противников есть общая черта - они определяют цели и правила, по которым следует играть. Так вам легче решать, что делать, и проще оценивать результаты. Кроме того, в играх мы должны обучаться, постоянно совершенствуя навыки для того, чтобы выиграть соревнование или побить собственный рекорд.

Элементы игр

Роджер Кейллоис, автор книги "Человек, Действия и Игры" (Man, Play, and Games), предложил интересную систему классификации различных элементов игр. Любая конкретная игра может включать как один, так и все перечисленные элементы.

1. Состязание
Состязание представляет собой действия, в которых игроки с помощью своих навыков преодолевают трудности, создаваемые противниками. Удовольствие заключается в развитии умения брать верх над соперником. Примеры состязаний - футбольные и шахматные матчи.

2. Случайность
Действия, на результат которых влияет элемент неопределенности. Удовольствие заключается в минимизации неопределенности и стремлении предугадать результат. Игры, основанные на случайности, также могут создавать у игроков иллюзию способности контролировать или предсказывать будущее. Примерами являются игровые автоматы и лотереи.

3. Дезориентация
Действия, в которых нарушается нормальное восприятие мира, приводящее к приятному состоянию пространственной дезориентации. Примерами являются "американские горки" и прыжки с парашютом.

4. Фантазирование
Это действия, с помощью которых мы создаем новые реальности, где не существует ограничений, присущих реальному миру. Удовольствие обусловлено приобретением свойств и способностей, которых мы лишены в обыденной жизни. Игра позволяет нам почувствовать себя так, как будто мы действительно обладаем выбранными возможностями. Примерами являются ролевые игры, театр и чтение книг.

Действия, вызывающие удовольствие

Итак, если игры способны приносить нам удовольствие, то почему же одни игры вызывают его в большей степени, чем другие? Михали Цикзентмихали (Mihaly Csakszentmihalyi), признанный специалист в исследованиях позитивной психологии, ввел понятие "потока" (flow) - состояния, которое он сам описывает следующим образом: "полное погружение в действие ради самого действия, исчезновение собственного я. Время летит; каждое последующее действие, движение и мысль неизбежно вытекает из предыдущего, словно вы играете джаз. Все ваше существо поглощено игрой, и вы используете свои навыки на полную мощь".

Люди, игравшие в игры, скорее всего знакомы с этим ощущением, и большинство из них согласится, что оно приносит большое удовольствие. Но что же является его источником? Цикзентмихали приводит список восьми факторов, определяющих состояние "потока".

1. *Задача выполнима и требует навыков*

Любая ситуация предлагает вам несколько альтернатив, или задач, выполнение которых требует определенных навыков. Если мы не в состоянии решить задачу, она теряет смысл или вызывает в нас чувство разочарования. Удовольствие же находится посредине между скукой и возбуждением и приходит в моменты, когда наши альтернативы соответствуют нашим возможностям. Хорошим примером такой ситуации является шахматная партия с сопоставимым по силе соперником. Если соперник имеет перед нами значительное преимущество, игра утрачивает смысл, поскольку мы лишены возможности влиять на ее исход. Если же соперник гораздо слабее, игра становится скучной, поскольку не представляет трудности.

2. *Концентрация на совершаемых действиях*

Когда для выполнения задачи человеку требуется применить определенные навыки, его внимание полностью поглощено собственной деятельностью. Он входит в состояние, в котором его действия становятся спонтанными, и перестает отделять себя от того, что он делает. Чтобы войти в это состояние и остаться в нем, важно избавиться от отвлекающих факторов как в самой деятельности, так и вне ее.

Во многих играх отвлекающим фактором являются сложные элементы управления. Если элементы управления недостаточно просты или интуитивно непонятны для того, чтобы позволить нам выполнять желаемые действия, мы будем постоянно отвлечены от них. Вместо того чтобы сосредоточиться на игре, мы станем концентрировать внимание на элементах управления.

3. *Четкие цели для наших действий*

Без четкой цели определить успех наших действий почти невозможно. Удовольствие не обязательно заключается в достижении цели - оно играет роль конца "линейки", с помощью которой мы измеряем свой успех. Также важно, чтобы уровень поставленных целей требовал применения навыков для их достижения, поскольку в противном случае цели утрачивают смысл.

К примеру, если в гонке перед нами стоит цель просто доехать до финишной прямой, то для нас не составляет большого труда достичь успеха. Если же необходимо прийти к финишу за короткое время, то задача, скорее всего, полностью поглотит наше внимание.

4. *Постоянная обратная связь, демонстрирующая успешность*

Чтобы иметь возможность оценить эффективность собственных действий, нам необходима обратная связь с информацией об их результатах. Некоторые наши попытки достичь целей могут быть неудачными, однако если обратной связи достаточно, мы поймем, что находимся на правильном пути, и наши усилия не бесплодны. Обратная связь - это не только способ продемонстрировать нам собственный успех. Важно представить нашу деятельность так, чтобы мы могли постоянно оценивать свой прогресс.

В играх-сражениях рядом с каждым из противников часто отображают состояние его здоровья. Это хороший способ обратной связи, позволяющий выработать способы атаки, наносящие противнику максимальный урон.

5. *Глубокое погружение в действия, отрешающее от повседневных забот*

Чтобы глубоко погрузиться в действия, важно, чтобы эти действия не были навязаны против нашей воли. Действуя по собственному желанию в условиях глубокого погружения, мы забываем о неприятностях реальной жизни. Действия требуют столь полной концентрации, что у нас не остается возможности реагировать на все остальное.

Несмотря на то, что обычно нас никто не принуждает играть в игры, через какое-то время некоторые их элементы становятся скучными или повторяющимися. В этом случае мы в меньшей степени погружены в игру, поскольку чувствуем принуждение к выполнению этих действий ради достижения успеха.

6. *Контроль над окружением*

В структурированных действиях мы часто получаем ощущение контроля. Мы избавлены от типичного для повседневной жизни страха утратить контроль - мы не боимся потерпеть неудачу. Важно, чтобы игра позволяла нам усовершенствовать свои навыки до столь высокого уровня, что риск поражения становится равным нулю. Это относится не к получению контроля, а к его развитию. Мы получаем ощущение контроля лишь в ситуациях, где результат не определен, но мы чувствуем, что можем воздействовать на него.

Наличие элементов, влияющих на исход игры, которые игрок не может контролировать, не позволит ему развивать способности к контролированию ситуации. Игры, допускающие максимальное развитие этой способности, почти всегда обеспечивают выигрыш наиболее опытного участника.

7. *Уменьшение эгоцентричности*

Когда мы полностью концентрируемся на действиях, у нас не остается возможности для саморефлексии. Все наше внимание приковано к действиям. Позднее мы взглянем на себя и убедимся в том, что наш опыт обогатился новыми навыками и достижениями.

8. *Изменение восприятия времени*

Наше чувство времени в основном зависит от восприятия изменений вокруг нас. Когда деятельность требует полной концентрации, мы лишены возможности воспринимать внешние воздействия. Это может превратить дни в часы, а часы - в минуты.

Итоги

Игры - это совокупность определенных действий, сформированных таким образом, чтобы удовольствие от нее было максимальным. Игры представляют собой способ развлечения, не зависящий от нашей текущей жизненной ситуации. В играх нас не удерживают ограничения реального мира. Мы можем создать собственные правила, позволяющие раскрыть свои таланты и быть вознагражденными за них - таланты, которые могли бы пройти незамеченными в других условиях.

Тем не менее, игры - это не только действия; они содержат множество других компонентов. В особенности это касается компьютерных и видеоигр. Например, *StarCraft* не сводится исключительно к управлению ресурсами и планированию стратегических переходов. В ней также необходимо играть роль военного предводителя далекого будущего, остановить поток пришельцев, пытающихся захватить ваш мир, бороться с изменой и предательством - в конечном счете, выбрать правильный путь и стать героем.

Во второй части статьи мы займемся рассмотрением видеоигр и узнаем, как создавать их в соответствии с потребностями и желаниями игрока.

Автор статьи: ***Андерс Хейденберг***
http://www.1337gamedesign.com/

Авторские права принадлежат CMP Media Inc и Gamasutra.com.
Оригинал статьи находится по адресу
http://www.gamasutra.com/features/20050426/ejdenberg_01.shtml.

Знакомство с JAVA

и другими мобильными платформами

Прежде чем я расскажу о специфике сущесвующих мобильных платформ, важно отметить, что по сей день программирование мобильных игр находиться в зародышевом состоянии. Это очень важно, потому что означает, что средства и технологии меняются очень быстро. Очень важно, чтобы вы держали руку на пульсе и отслеживали новые технологии и тенденции, включая беспроводные технологии, предлагаемые и поддерживаемые провайдерами.

Хотя "ландшафт" мобильных игр изменяется быстро, приходит день, когда та или иная платформа становится доминирующей. Можно составить список платформ, подходящих для написания мобильных игр:

- Java 2 Micro Edition (J2ME)
- Binary Runtime Environment for Wireless (BREW)
- Symbian
- Windows Mobile

Все эти четыре платформы поддерживаются современными телефонами. Все они предлагают разработчику широкие возможности, обеспечивая бесплатными инструментами и документацией. Ниже приведено более подробное описание каждой из платформ, что поможет вам понять их различия и особенности.

JAVA 2 Micro Edition (J2ME)

J2ME - это компактная версия популярного языка программирования Java, созданного Sun Microsystems. Многие и не подозревают, что изначально Java создавался как язык программирования мобильных устройств, поэтому он вернулся к истокам и воплотился в J2ME. J2ME содержит широкий спектр инструментов для разработки и богатый интерфейс прикладного программирования (application programming interface, API) для разработки приложений для мобильных телефонов, известных как MIDlets.

J2ME также включает виртуальную машину, которая отвечает за эмуляцию выполнения кода Java на конкретном телефоне. Создавая общий код приложения вместо собственного, J2ME позволяет без труда создавать игры, совместимые с широким кругом телефонов. На самом деле, если бы телефоны не отличались размерами экранов и графическими возможностями, не требовалось бы прикладывать и малейшего усилия, чтобы перенести игру с одного телефона на другой.

Эта платформа наиболее широко поддерживается производителями мобильных телефонов. Такие гиганты, как Motorola, Nokia, Samsung поставляют телефоны, поддерживающие J2ME.

Binary Runtime Environment for Wireless (BREW)

В отличие от J2ME, поддерживаемого широким спектром телефонных аппаратов, BREW - это платформа, ориентированная на телефоны с технологией Qualcomm CDMA (Code Division Multimple Access - множественный доступ с кодовым разделением каналов). Но это не говорит о том, что BREW имеет весьма ограниченное распространение. Для создания игр для платформы BREW разработчики часто используют язык C или C++ в сочетании с BREW API. BREW также поддерживает такие языки разработки приложений, как XML и даже Java.

Подобно J2ME, BREW может выступать промежуточным звеном между игрой и операционной системой телефона. В отличие от J2ME, платформа BREW также поддерживает и собственный код, это значит, что игра может быть скомпилирована под конкретный процессор телефона. Собственный код обрабатывается намного быстрее, чем его интерпретированный аналог, однако при его использовании могут возникать проблемы с переносом приложения с одного телефона на другой.

Платформа BREW нашла наибольшее распространение в Азии (особенно в Японии и Южной Корее). В США на сегодняшний день лидирующие поставщики беспроводных решений, предлагающие телефоны с поддержкой BREW - это Alltel и Wireless.

Symbian

Symbian - это мобильная операционная система, которая имеет существенное отличие от BREW. Оно заключается в том, что это открытая операционная система, доступная по лицензии любому производителю мобильных телефонов. ОС Symbian была разработана компанией Symbian Ltd., которая является консорциумом компаний Motorola, Nokia, Panasonic и Sony Ericsson. Благодаря простоте лицензирования, эта ОС поддерживается многими моделями телефонов.

Существует масса возможностей при разработке приложений для Symbian, поскольку эта операционная система поддерживает такие языки программирования, как C++, Java и Visual Basic. На сегодняшний день большинство мобильных игр и приложений для Symbian написаны на C++, поскольку это обеспечивает более высокое быстродействие и интеграцию с операционной системой, нежели аналоги, созданные с использованием Java. Несомненно, Java начинает компенсировать недостаток в производительности между приложениями, написанными на этом языке, и собственным кодом, но все-таки в большинстве случаев собственные программы работают эффективнее, чем Java-аналоги. Это особенно важно, если дело касается игр, где каждая доля мощности процессора на счету.

Так почему бы не остановиться на Symbian, а не на Java? Простой ответ заключается в том, что Java распространен широко, в то время как Symbian - это лишь одна из операционных систем.

Мобильные телефоны значительно отличаются от персональных компьютеров тем, что имеют весьма разнообразное программное и аппаратное обеспечение. Java - это унифицирующая технология, позволяющая использовать созданное приложение на различных типах телефонов.

Symbian

Вы ведь не думали, что Microsoft будет сидеть в стороне и спокойно следить за развитием мобильных игр без собственной операционной системы, не так ли? Конечно, нет! Компания Microsoft немного преуспела, выпустив ОС Windows Mobile, устанавливаемую на карманных компьютерах и смартфонах. Хотя некоторые КПК могут выполнять функции телефонов, все-таки они ближе к PDA, нежели к телефону, хотя бы по своим габаритным характеристикам: достаточно широкий экран (320x240), а для ввода используется световое перо (стилус).

Для разработки приложений для Windows Mobile используются многие инструменты, включая C++ и Microsoft C#.

Что такое Java?

Ранее я упоминал, что изначально Java был языком программирования, который позволял сетевым устройствам связываться друг с другом. Если быть более точным, то Java зарождался как проект Sun, целью которого было научиться внедрять компьютеры в современную жизнь. Одной из основных задач проекта было заставить все компьютеризированные устройства взаимодействовать друг с другом. Как выяснилось, Sun опередил свое время в попытке использовать Java для соединения бытовых приборов. Однако компания быстро отреагировала и завершила работу над проектом, сделав Java успешным web-ориентированным языком программирования.

Как только технология и потребность на рынке совпали с исходным назначением Java, Sun подняла свои предыдущие наработки и приспособила Java для мобильных телефонов. J2ME разработан не только с учетом ограничений мобильных телефонов, он также подходит для программирования беспроводных соединений. J2ME - это лишь подмножество более глобального инструмента Java, который состоит из языка программирования, API и среды выполнения (Runtime Environment).

Почему именно Java?

Даже если бы Java был идеально приспособлен для разработки приложений для мобильных телефонов, он бы не нашел широкого применения без поддержки промышленности. На сегодняшний день Java - доминирующая технология разработки программного обеспечения для мобильных устройств.

Разработчики активно используют Java, поскольку эта платформа является открытой. Это значит, что если вы разработали код, то его можно использовать в различных мобильных устройствах. К сожалению, такая "открытость" Java была несколько осложнена наличием API других производителей и различиями аппаратного обеспечения различных устройств. Тем не менее, вы можете написать код и, не внося никаких изменений, использовать его на разнообразных мобильных телефонах. Сравните это с платформой BREW, созданной специально для телефонов, работающих в сетях Qualcomm CDMA.

> *С технической точки зрения, несмотря на то, что Java и J2ME "более открыты", чем BREW, они до сих пор не являются языками с открытым кодом. Все разновидности Java-технологии, включая J2ME, принадлежат Sun Microsystems. К счастью, Sun была очень любезна и дала возможность свободного формирования стандартов Java, однако многие разработчики до сих пор лоббируют их, чтобы перевести Java в сообщество Open Source.*

Интересный виток взаимоотношений между Java и BREW случился в конце 2002 года, когда в свет вышла виртуальная машина Java для устройств, работающих на платформе BREW. Это означает, что BREW-устройства могут эффективно работать с Java-приложениями, как будто они изначально поддерживали Java. Но при этом Java не конкурировала с BREW как платформа для разработки игр.

Java и программирование мобильных игр

Прежде всего при программировании игр вызвают инерес следующие области технологии:

- графика и анимация
- звук
- ввод информации пользователем
- работа в сети.

Стандартный API включает поддержку всевозможных графических элементов, таких, как, например,

Графика и анимация

изображения, двумерные графические примитивы (линии, прямоугольники, эллипсы и т.д.) и анимация. В терминах анимации J2ME поддерживает спрайты - изображения, свободно перемещаемые по экрану вне зависимости от других. API среды J2ME также поддерживает проверку столкновений.

Другая интересная особенность J2ME - это замощенные слои. Вы можете выбрать небольшое изображение и замостить им фоновый слой. Такие слои используются в играх для создания больших перестраиваемых карт, что позволяет сэкономить память. Благодаря удобному менеджеру слоев, в J2ME можно с легкостью управлять несколькими слоями. Таким обрахом, можно создать один слой - полностью фоновый, который можно использовать как декорацию, и еще один слой - для создания преград на пути героя игры.

Пользовательский ввод очень важен для игр: он определяет, как удобно игроку взаимодействие с

Ввод информации пользователем

игрой. Также ввод очень важен потому, что определяет первичный интерфейс между игроком и игрой. J2ME поддерживает клавишный ввод. Существует возможность непосредственно считывать состояние клавиш на телефоне, это очень важно, если вы хотите обеспечить высокую степень взаимодействия игрока и игры.

"Большую тройку" самых важных элементов мобильных игр завершает звук. J2ME поддерживает

Использование звука в играх

воспроизведение цифрового звука в форматах PCM (WAV), MP3 и MIDI. Поддержка звука основана на Java Media API - интерфейса, предназначенного для записи и воспроизведения звука и видео на мобильных устройствах.

Самой привлекательной возможностью мобильных игр является, вероятно, возможность работы в сети.

Мобильные сети

Помня это, вы можете понять, что ориентирвоаннй на сети Java чрезвычайно удобен как платформа для мобильных игр. Сетевые возможности Java являются неотъемлемой частью его среды выполнения. В отличие от других языков, Java изначально ориентирован на поддержку сетей.

Объедините сетевую ориентированность Java и независимость этой платформы, и вы получите игровую платформу, которая преодолевает все преграды на пути доступности пользователю. Это очень важно, особенно когда вы поймете, что пользователи захотят играть в игры на различных устройствах и в различных беспроводных сетях. Игрок не должен вникать в проблемы, связанные с отличиями мобильных телефонов разных производителей. Благодаря поддержке сетей языком Java разработчикам теперь не нужно заботиться о различиях аппаратного обеспечения.

М. Моррисон, *"Создание игр для мобильных телефонов".*

Основы MIDlet Pascal

MIDlet Pascal - это паскалеподобный язык программирования, ориентированный на разработку приложений для мобильных устройств, в частности для платфомы Java (J2ME). Его можно смело рекомендовать всем начинающим Java-кодерам. Официальный сайт: http://www.midletpascal.com

Если вы прежде работали с Pascal, Delphi или Kylix, то проблем с освоением не возникнет. Язык имеет удобную встроенную среду разработки (IDE) с простым и понятным интерфейсом. Однако, осваивая новую программную среду, я заметил, что при всей кажущейся интуитивности MIDlet Pascal имеет не так уж много встроенных средств для написания игр. Многое приходится сочинять самому, например, анимацию спрайтов, скроллинговые фоны, проверку столкновений... Вот я и решил поделиться своими скромными успехами по данным вопросам.

Часть 1. Анимация.

На мой взгляд, самый легкий способ - загрузить кадры анимации из отдельных изображений в массив, а потом последовательно выводить их на экран в зависимости от значения увеличивающейся переменной.

Запускаем MIDlet Pascal, создаем новый проект. Добавляем четыре кадра анимации. Я использовал всего три изображения, так как один кадр будет повторятся дважды за цикл. Выберите *Project* > *Import resource file* и загрузите изображения. Это можно сделать сразу или по отдельности.

Помните, что Java поддерживает только формат PNG, причем с альфа-каналом, если спрайт предусматривает частичную прозрачность. Также следует учитывать, что большинство телефонов не располагают 24-битной цветопередачей (16 млн. цветов, True Color). Наилучший выход - использовать 256-цветовую индексированную палитру. К сведению, все эти дела можно легко сделать в бесплатном графическом редакторе PhotoFiltre (http://www.photofiltre.com).

Начинаем писать программу:

```
program Animation;
var background:image;
    anim:array[0..3] of image;
    frame:integer;
    cmdQuit:command;
```

Сразу вспомнились ностальгические времена Turbo Pascal... Итак, мы создаем изображение для фона, а также массив **anim** из четырех значений (от 0 до 3) типа **image** (изображение). Целочисленная переменная **frame** будет отвечать за текущий номер кадра. Переменная **cmdQuit** будет использована для выхода из программы.

Прописываем основное тело программы:

```
begin
      background:=LoadImage('/back.png');
      anim[0]:=LoadImage('/fololo1.png');
      anim[1]:=LoadImage('/fololo2.png');
      anim[2]:=LoadImage('/fololo3.png');
      anim[3]:=LoadImage('/fololo2.png');
```

Сразу после begin мы загружаем в массив изображения. Кстати, после добавления в проект, они копируются в папку /res в директории с проектом. Но копировать/удалять файлы вручную через Проводник лучше не надо. Пользуйтесь для этого интерфейсом MIDlet Pascal.

```
        cmdQuit:=CreateCommand('Quit',CM_EXIT,1);
        AddCommand(cmdQuit);
        frame:=0;
```

Команда выхода - это когда пользователь нажимает кнопку "Выход" на клавиатуре телефона. Позже мы проверим, если команда выполняется, программа завершит свое действие. Переменная frame изначально равна нулю - анимация будет проиграна с первого кадра.

Прописываем основной цикл программы:

```
        repeat
           SetClip(0,0,128,160);
           SetColor(0,0,0);
           DrawImage(background,0,0);
           DrawImage(anim[frame],20,10);
           frame:=frame+1;
           if frame=3 then frame:=0;
           repaint;
           Delay(60);
        until (GetClickedCommand = cmdQuit);
end.
```

Здесь все просто: мы устанавливаем область отбора (та область, на которой будет отрисована графика) и цвет смешивания (в данном случае - белый). Рисуем фон и изображение из массива **anim** под номером **frame**. И тут же увеличиваем **frame** на единицу. Проверяем, не достигла ли **frame** максимального значения (3), если да - обнуляем, чтобы анимация проигрывалась непрерывно. Команда **repaint** выполняет перекрашивание экрана, а **Delay** - выжидает 60 миллисекунд до следующего повторения цикла. Это нужно, чтобы анимация воспроизводилась не слишком быстро. Цикл повторяется, пока не будет выполнена команда выхода.

Все, можно компилировать и запускать (*Project > Build & run*). Ой, забыл сказать - для запуска приложений вам надо иметь эмулятор Java. Можно попробовать Midp2Exe (http://home.kimo.com.tw/kwyshell/Project/Midp2Exe), Sjboy (http://www.wedomo.com) или любой другой.

Вот полный исходник программы:

```
program Animation;
var background:image;
    anim:array[0..3] of image;
    frame:integer;
    cmdQuit:command;
begin
      background:=LoadImage('/back.png');
      anim[0]:=LoadImage('/fololo1.png');
      anim[1]:=LoadImage('/fololo2.png');
      anim[2]:=LoadImage('/fololo3.png');
      anim[3]:=LoadImage('/fololo2.png');
      cmdQuit:=CreateCommand('Quit',CM_EXIT,1);
      AddCommand(cmdQuit);
      frame:=0;
      repeat
          SetClip(0,0,128,160);
          SetColor(0,0,0);
          DrawImage(background,0,0);
          DrawImage(anim[frame],20,10);
          frame:=frame+1;
          if frame=3 then frame:=0;
           repaint;
           Delay(60);
      until (GetClickedCommand = cmdQuit);
end.
```

Теперь — дело за вами. Попробуйте изменить код, так, чтобы спрайт персонажа менялся в зависимости от нажатой клавиши. Можно пойти и дальше: дать ему возможность двигаться во всех четырех направлениях. Так получится настоящий лабиринтник, вроде Пакмена. Осталось только сделать проверку столкновений, но об этом мы будем говорить в следующий раз.

Автор статьи:
Gecko *clocktower89@mail.ru*

Программирование на Xtreme3D

Создание игры от третьего лица

Урок III: проверка столкновений

Этот урок посвящен наболевшей теме: проверке столкновений в 3D. Те, кто знает Blitz, наверняка помнят, что там такая проблема не стоит. Точная проверка уже включена в движок, вам остается лишь выбрать тип ответной реакции да назначить объекты. Здесь все по-иному. Как именно - сейчас узнаете.

Что потребуется:

1. Графический движок Xtreme3D. Скачать можно по разным ссылкам. Советую следующие:
http://www.sentinelgames.com/download/Xtreme3D/Xtreme3DV2.zip - архив версии 2.0.0.0;
http://xtreme3d.hut2.ru/files/X3DV2_full.rar - дистрибутив нашей сборки (версия 2.0.2.0).

2. Game Maker 5.x, 6.x или 7.x (зарегистрированная версия).

3. Модель уровня в формате 3DS с картой освещения (lightmap). Думаю, вы и без меня разберетесь, как ее изготовить. В крайнем случае, можно взять из ресурсов дистрибутива или скачать где-нибудь.

4. Исходник предыдущего урока (смотрите в FPS #3 или на сайте http://xtreme3d.hut2.ru)

Вообще, столкновения в Xtreme3D можно проверять разными способами:

- Использовать функции *ObjectCheckCubeVsFace*, *ObjectCheckCubeVsCube* и т.д. В большинстве случаев это подходит, но неэкономно по отношению к ресурсам, когда нужно проверять объект со сложной геометрией (например, карту) - сам проверял. Так что вариант отпадает.

- Использовать так называемый рейкастинг (*raycasting*). Под этим подразумевается принцип луча, исходящего из искомого объекта и проверяющегося на пересечение с другими объектами. Существует функция *ObjectSceneRaycast*, проверяющая на пересечение все дочерние объекты цели - так можно проверять и всю карту, но это опять-таки не очень экономно.

- Использовать *DCE*. DCE расшифровывается как *Dynamic Collision Engine* - движок динамических столкновений. Это технология GLScene, позволяющая автоматизировать процесс проверки, и во многом похожая на движок физики - и почти такая же сложная для понимания. Так как уроки ориентированы на начинающих, о DCE мы поговорим позже.

- Способ для маньяков: проверять столкновения через встроенные средства GM. Но тогда отпадает и надобность использования такой мощной системы, как Xtreme3D - ведь простую игрушку с обыкновенными плоскостными столкновениями можно состряпать и в стандартном 3D-движке GM.

- Способ для маньяков иного рода: использовать физический движок, например, ODE. Но физика уместна не во всех играх, и уж точно не для всех игровых объектов. Сюда же я бы отнес способ для энтузиастов: использование внешних библиотек столкновений. Я лично имел дело с одной - ModMod (основана на TriTri), но, как я уже сказал, это для энтузиастов, так как в большинстве эти библиотеки довольно сложны и неудобны в интеграции.

- Использовать простую, и вместе с тем мощную и универсальную управляющую структуру Xtreme3D - *FPS Manager*. Как ясно из названия, менеджер этот предназначен для игр от превого лица, но принципиальных требований к камере у него нет. Менеджер можно применить абсолютно к любому объекту.

Последний способ я и облюбовал для сегодняшнего урока. Мы попробуем оптимизировать наш движок, чтобы можно было использовать FPS Manager. Но сначала - немного о модели уровня (карте).

Xtreme3D имеет важную особенность - поддержку вторых текстурных координат. Это когда один и тот же объект (обычно типа Freeform) может иметь две по-разному наложенные текстуры. Данная технология используется в основном для создания карт освещения - лайтмэпов (*lightmaps*). При помощи специальных программ - редакторов карт (среди которых наиболее известны *3D World Studio*, *Cartography Shop*, *Gile[s], DeleD* и другие), создают и сам уровень, и генерируют лайтмэпы по заданным источникам света. Эффект получается весьма реалистичным, так как лайтмэп предусматривает качественное освещение, с тенями и всем прочим. На этом уроке мы загрузим такую карту в Xtreme3D. Я взял модель из дистрибутива X3D; используется следующие файлы:

- level.3ds - основная геометрия уровня, содержит первые текстурные координаты
- level_lm.3ds - дополнительная геометрия, содержит вторые текстурные координаты
- cigar.png, cobbles5.png, crate001.png, electro.png, facade02.png, facade03.png, facade04.png, grass3.png, tHaus.png, truck.png, tSpaceF.png, wall02.png, tBank.jpg, tHyd.jpg, tPlank.jpg - текстуры уровня
- lm.png - текстура лайтмэпа. Сложное по структуре изображение, сгенерированное редактором уровня.

Мне кажется, что внесение исправлений в старый код будет делом чересчур длинным для подробного описания, поэтому я просто приведу новый его вариант, давая по ходу комментарии там, где это необходимо.

Начало остается практически неизменным, за исключением некоторых деталей. Сравните, где что нужно изменить: я убрал освещение (пока не до него) и удалил Теневую плоскость (какой в ней толк, когда мы собираемся грузить уровень целиком из файла).

```
dll_init();
EngineCreate(window_handle());
matlib=MaterialLibraryCreate();
MaterialLibraryActivate(matlib);
view=ViewerCreate(0,0,640,480);
ViewerEnableVSync(view,vsmSync);
ViewerSetLighting(view,false);
ViewerSetBackgroundColor(view,c_black);
ViewerEnableFog(view,1);
ViewerSetFogColor(view,c_black);
ViewerSetFogDistance(view,200,500);
global.back=DummycubeCreate(0);
global.scene=DummycubeCreate(0);
global.front=DummycubeCreate(0);
global.shadow_caster=DummycubeCreate(global.scene);
```

Дальше код создания камеры и персонажа. Я уже не помню, вносил ли изменения или нет, так что проверяйте сами:

```
down=DummycubeCreate(0);
ObjectSetAbsoluteDirection(down,0,-1,0);
d=DummycubeCreate(global.shadow_caster);
ObjectSetPosition(d,-100,4,-100);
camera=CameraCreate(global.scene);
CameraSetViewDepth(camera,800);
CameraSetFocal(camera,80);
ViewerSetCamera(view,camera);
ObjectSetPositionOfObject(camera,d);
MaterialCreate('mMulti','');
matlib2=MaterialLibraryCreate();
MaterialLibraryActivate(matlib2);
MaterialCreate('m1','DATA/PROTOMAN.pcx');
MaterialCreate('m2','DATA/steel.jpg');
MaterialCreate('m3','DATA/reflection_map.jpg');
MaterialCreate('m4','DATA/PROTOMAN2.bmp',1);
MaterialSetOptions('m3',0,1);
MaterialSetBlendingMode('m2',bmAdditive);
MaterialSetBlendingMode('m3',bmModulate);
MaterialSetTextureMappingMode('m2',tmmSphere);
MaterialSetTextureMappingMode('m3',tmmSphere);
MaterialSetTextureMode('m3',tmBlend);
MaterialSetTextureImageAlpha('m4',tiaSuperBlackTransparent);
MaterialLibraryActivate(matlib);
ms=MultiMaterialShaderCreate(matlib2);
MaterialSetShader('mMulti',ms);
actor=ActorCreate('DATA/TRIS.md2',d);
ObjectSetMaterial(actor,'mMulti');
ObjectSetScale(actor,0.15,0.15,0.15);
ObjectRotate(actor,90,-90,0);
ObjectSetPosition(actor,0,3.6-4,0);
xpr=ObjectGetAbsolutePosition(actor, 0);
ypr=ObjectGetAbsolutePosition(actor, 1);
zpr=ObjectGetAbsolutePosition(actor, 2);
weapon=ActorCreate('DATA/WEAPON.md2',d);
ObjectSetScale(weapon,0.15,0.15,0.15);
ObjectRotate(weapon,90,-90,0);
ObjectSetPosition(weapon,0,3.6,0);
spd=0.2;
state='stand';
target=DummycubeCreate(d);
ObjectSetPosition(target,0,25,-30);
CameraSetTargetObject(camera,d);
jump=0;
grav=0;
set_automatic_draw(false);
```

Вот теперь-то и начинаются главные нововведения:

```
MaterialLibrarySetTexturePaths(matlib,'data/level');
level=FreeformCreate('data/level/level.3ds',global.scene);
lightmap=FreeformCreate('data/level/level_lm.3ds',0);
FreeformSetSecondTextureCoords(level,lightmap);
ObjectDestroy(lightmap);
```

Сначала следует выполнить важную процедуру: указать путь к текстурам (к папке, где лежат модель уровня и все используемые ей файлы). Если этого не сделать, система будет пытаться искать текстуры в рабочей директории игры, естественно, не найдет и попросту не отобразит.

Затем мы загружаем обе геометрии, устанавливаем вторые координаты на первую и уничтожаем вторую. Осталось немного:

```
ObjectSetDirection(level,0,1,0);
ObjectSetScale(level,0.4,0.4,0.4);
map=DummycubeCreate(global.scene);
FreeformToFreeforms(level,0,0,map);
ObjectHide(level);
```

Необходимо еще "поколдовать" над объектом уровня, чтобы привести его в нужное нам состояние: повернуть, уменьшить и разбить на отдельные объекты. Для чего нужно последнее? Об этом - чуть позже.

Ну, вот подошли и к FPS Manager' у. Ничего сложного здесь нет, мы просто создаем Менеджер, назначаем его объекту d, который в нашем случае служит позицией персонажа в пространстве, и применяем искомую карту уровня для проверки столкновений:

```
nav=NavigatorCreate();
fpsmanager=FpsManagerCreate();
FpsManagerSetNavigator(fpsmanager,nav);
FpsManagerSetMovementScale(fpsmanager,3);

FpsManagerAddMap(fpsmanager,level);

FpsSetManager(d,fpsmanager);
FpsSetSphereRadius(d,4);
FpsSetGravity(d,false);
```

А вот для чего мы разбивали уровень на составляющие: для достижения высокой скорости отрисовки карта отображается по частям, в зависимости от того, куда смотрит камера. Для этого применяется технология Quadtree culling:

```
quad=QuadtreeCreate(4,24,0.1,cmGrossCulling);
for (i=0;i<ObjectGetChildCount(map)-1;i+=1)
{
child=ObjectGetChild(map,i);
PartitionAddLeaf(quad,child);
ObjectHide(child);
}
quadtreeculling=true;
```

Единственное, что теперь осталось - это заменить функции ObjectMove и ObjectStrafe в событии Step на FPSMove и FPSStrafe:

```
if keyboard_check(vk_up)  {if (state!='attack' && state!='salute')
FpsMove(d,0.8);}
if keyboard_check(vk_down)  {if (state!='attack' && state!='salute')
FpsMove(d,-0.8);}

if keyboard_check(ord('A')) {if (state!='attack' && state!='salute')
FpsStrafe(d,-0.5);}
if keyboard_check(ord('D')) {if (state!='attack' && state!='salute')
```

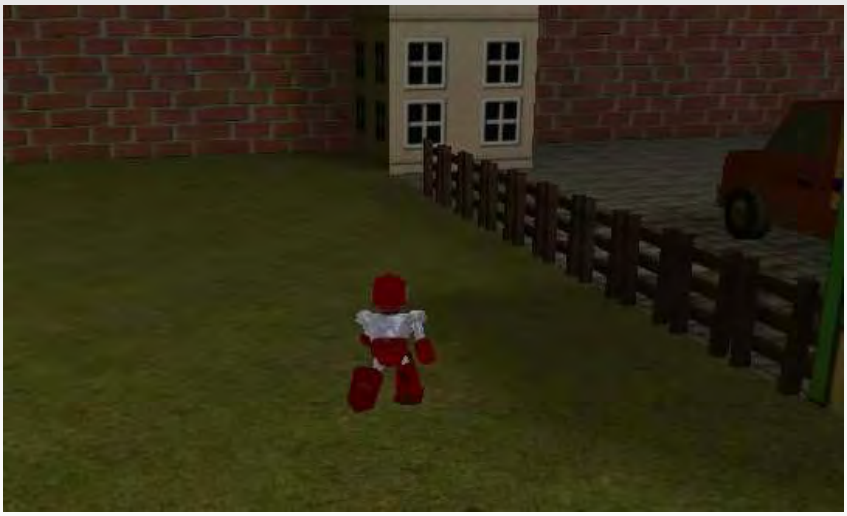
А после Update надо добавить следующее для обновления Quadtree culling:

```
Update();

if quadtreeculling=true
{
 PartitionResultHide(quad);
  qr=PartitionQueryFrustrum(quad,view);
 PartitionResultShow(quad);
}

ViewerRender(view);
```

Можно запустить и протестировать. Естественно, с первого раза редко бываешь доволен получившимся, но с умелыми руками настроить систему будет нетрудно. Например, я уверен, вам еще придется заняться размерами карты, позицией персонажа и еще много чем. Это все индивидуально для конкретных моделей и построения игры, и - открою страшную тайну! - в этом и заключается основная работа кодера: один раз что-то написать за пять минут, а потом полдня вносить исправления ;)



Когда все будет более-менее сносно, можно добавить и свет. Я не случайно оставил его на потом - освещение требует отдельной настройки, причем довольно тонкой, и лучше заниматься им в конце, когда более важная работа уже выполнена. Думаю, вы с этим справитесь и без меня. Желаю удачи. Встретимся на следующем уроке!

Автор статьи и программист: ***Gecko***
clocktower89@mail.ru

Xtreme 3D Material

Настройка Материалов в Xtreme3D - дело весьма важное. Но, подчас бывает тяжело раз за разом запускать игру, только чтобы протестировать какую-то мелочь. Это, разумеется, относится не только к Материалам, но речь в этой статье пойдет в основном о них. А именно, о полезной утилите под тривиальным названием Xtreme3D Materials.

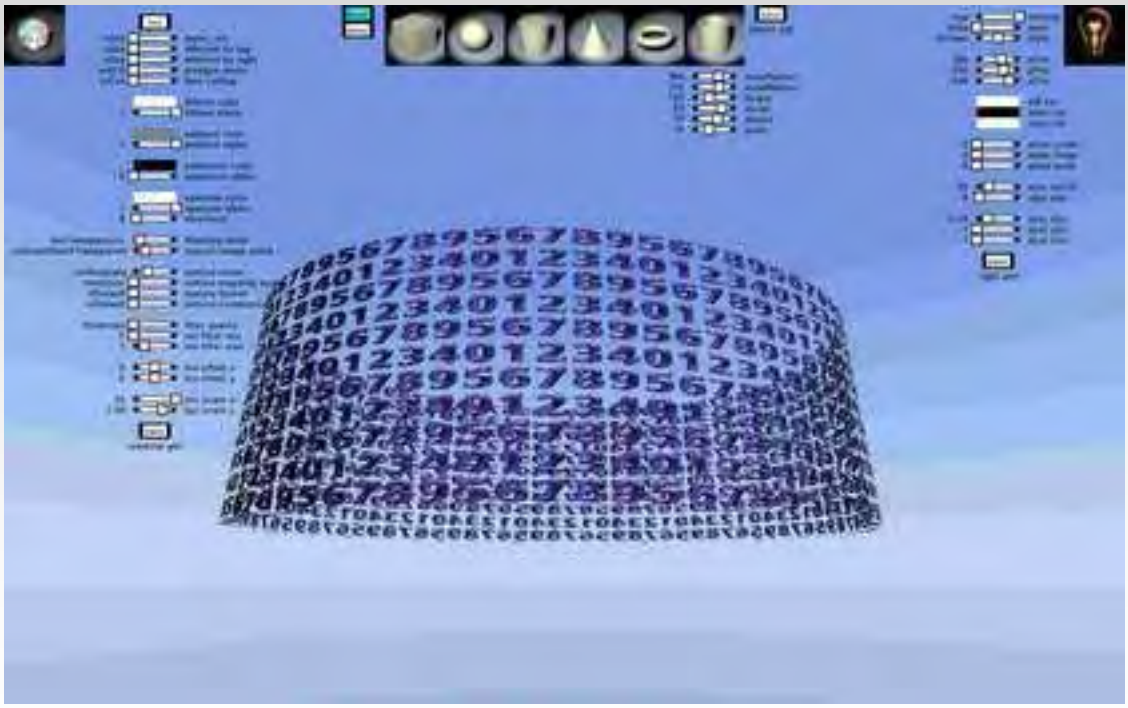
Создана она человеком по имени Jurgen Muller (студия LayArt Games) и скачать ее можно по этой ссылке:
http://layart-games.de/downloads/download.php?file=xtreme3dmaterial.zip.

Xtreme3D Material предназначена для визуализации создания и настройки Материалов. И не только их! Программа также поможет настроить источники освещения и примитивы. Вы можете изменить все опции Материала, какие только доступны в Xtreme3D (за исключением наложения шейдеров): текстуру, освещаемость, действие тумана, цвет рассеивания, цвет тени, цвет эмиссии, цвет блика, режим смешнивания, значение альфа, режим отображения, формат, тип сжатия, фильтр, смещение и масштаб. В центре экрана расположен примитив с Материалом, реагирующим на внесенные изменения. Вы можете выбрать один примитив из шести: куб, сферу, цилиндр, конус, тор и полый цилиндр (annulus). Примитивы также имеют свои настройки. Наконец, вы можете изменить освещение, задав тип, позицию, ориентацию и три компонента цвета. Материалы, примитивы и настройки света сохраняются в скрипт *.GML, который можно импортировать в игру.

Камера в программе контролируется стандартной раскладкой WASD, стрелками влево и вправо, PageUp, PageDown и правой кнопкой мыши. Хотя перемещать камеру вам вряд ли понадобится - не редактор карт все-таки :)

К недостаткам программы (таковые, как ни странно, имеются) следует отнести не очень удобный интерфейс - довольно трудно изменять значения при помощи ползунков, особенно когда дело касается точных числовых данных. Также программа работает в разрешении 1024x768, что сказывается на производительности. Лучше бы разработчик дал возможность пользователям самим выбирать себе оптимальное разрешение. Также Xtreme3D Material начинает тормозить при включенном вращении примитива, причину чему я так и не выяснил.

Но в остальном - программа очень хорошая и нужная, так как других подобных я что-то не видел. Must have, что называется!



Автор статьи: ***Gecko***
clocktower89@mail.ru

Объявление

По техническим причинам в следующие четыре месяца (май, июнь, июль, август) журнал скорее всего выходить ***не будет*** *(относительно мая вопрос решен еще не окончательно). Пропуск будет компенсирован в начале сентября специальным выпуском «FPS» с приложениями и прочими бонусами. Заранее приносим извинения и просим не делать скороспелых выводов — журнал ни в коем случае не закрывается.*

Редакция

Это все!

Надеемся, номер вышел интересным. Если так, поддержите FPS! Отправляйте статьи, обзоры, интервью и прочее на любые темы касательно игр, графики, звука, программирования и т.д. на ящик редакции: clocktower89@mail.ru.

Главный редактор журнала ***Gecko***

© 2008 Clocktower Games. Все названия и логотипы в журнале являются собственностью их законных владельцев и не используются в качестве рекламы продуктов или услуг. Редакция не несет ответственности за корректность и достоверность информации в статьях и надежность всех упоминаемых URL-адресов. Использование материалов журнала разрешено только с согласия авторов. Любое коммерческое использование журнала или его материалов без согласия редакции и авторов материалов считается незаконным.

Директор, главный редактор и оформитель: Гафаров Т. А. aka **Gecko**
Авторы статей: **Gecko**, **Андерс Хейденберг, М. Моррисон**.

По вопросам сотрудничества обращаться по адресу clocktower89@mail.ru.