IgriZdes Engine

(izeng)

Описание интерфейса компоненты

Ниже представлено описание игрового движка, выполненного в виде компоненты COM Windows. Движок предназначен для вывода 2D графики (он использует DirectX для этого) и простейшего моделирования движения объектов с возможностью управлять ими. Документация не дописана до конца на данный момент.

Примеры простейшего использования

Примером использования будет служить серия уроков, в конце которой мы получим целую игру. Можете прочитать описание, если интересно или сразу двинуться к изучению уроков. Вот их список:

Урок1: Подготовка

Описание игры

В космосе, по плоскости, ограниченной с 4-х сторон летают космические корабли. Они могут стрелять заморозкой — разрядом, выводящим из строя корабль на некоторое время. Также на уровне есть мяч (тоже космический! :)), его нужно взять. Корабль, взявший мяч, не может стрелять, но зато летает в 2 раза быстрее других. Если по нему попали, то, кроме выхода из строя, корабль теряет мяч и он появляется рядом.

Задача игрока — взять мяч и продержаться с ним как можно дольше. На раунд отводится определенное время. Корабль, продержавший мяч наибольшее время считается победившим.

Управление осуществляется стрелками (или WASD) и стрельба на «0 » (или пробел). Возможно 2 игрока за одной клавиатурой.

В конце раунда выводится рейтинг игроков (highscore).

Также на уровне есть препятствия квадратной формы, от них мяч и корабли отскакивают. Корабли также отскакивают и друг от друга при столкновении и от барьеров, ограничивающих поле.

Материалы по урокам

Материалы лежат в папке Tutorials. Каждому примеру соответствует отдельная папка — Tutorial1, Tutorial2 и т.д. В каждой такой папке есть папка Resources с ресурсами, которые использует Tutorial. Вы можете их использовать при обучении. По окончании каждого tutorial

у вас получатся определенные файлы с кодом. Если не получается — в папке Source_codes есть готовые примеры. В папке MSVS_projects лежат готовые проекты по tutorial.

В Tutorial1\Source_codes\WinAPI_project\main.cpp – шаблон оконного приложения Windows.

В Tutorial1\Source_codes\Tutorial1\main.cpp — шаблон приложения на движке IgriZdes.

Урок 1: Подготовка

1) Создаем новый Win32 empty проект. Назовем его Tutorial1 для определенности.

2) Добавляем в source files новый файл main.cpp с вашим любимым шаблонным кодом оконного Windows приложения. Дополним шаблон следующим:

```
Game_Init();

while(TRUE) {

        if(PeekMessage(&msg,NULL,0,0,PM_REMOVE)) {

                if(msg.message == WM_QUIT)

                        break;

                TranslateMessage(&msg);

                DispatchMessage(&msg);

        }

        //шаг (step), аналогичен кадру игры

        Game_Main();

}

//завершение игры и освобождение ресурсов

Game_Shutdown();
```

Где Game_Init(), Game_Main(), Game_Shutdown() - функции, вызываемые соответственно при инициализации, шаге (кадре) и завершении игры.

Или, можете взять готовый шаблон Source_codes\WinAPI_project\main.cpp.

3) Подключаем файл IZObj.h & IZObj.cpp в проект. Первый включаем в наш главный исполняемый файл (main.cpp).

4) Создадим глобальный объект для управления движком:

```
IZEng* izEngine  = NULL;
```

5) В Game_Init(...) пишем инициализацию игры. Для начала инициализируем движок и экран:

```
izEngine = new IZEng("izeng.dll"); //загрузка из dll
```

```
izEngine->pScreen->ResetDisplay(g_hMainWnd,1024,768,0,false,30);
```

Где g_hMainWnd — дескриптор окна приложения. Остальные параметры означают установить разрешение в 1024 на 768, частоту кадров по умолчанию (0) и отобразить в окне (не разворачивать на весь экран). Важно в режиме отладки ВСЕГДА ИСПОЛЬЗОВАТЬ ОКОННЫЙ РЕЖИМ, А НЕ ПОЛНОЭКРАННЫЙ. Т.к., если программа будет вести себя не так, как ожидалось, возможно, вы не сможете ее свернуть и придется жать «reset». Также число FPS установлено в 30.

Я рекоммендую все объекты движка создавать через оператор new() и удалять через delete или delete[], когда они вам больше не нужны. Также можно удалить сразу все объекты вызовом pScreen->ShutDown(), который также еще и освободит ресурсы и экран.

Также отобразим картинку на фон, для этого создадим глобальный объект:

```
IZObj* izBackgrnd  = NULL;
```

И инициализируем в Game_Init(...):

```
izBackgrnd = new IZObj(); //вызывается конструктор по умолчанию, т.к. dll уже загружена выше

izBackgrnd->pAnim->LoadSimpleAnim("backgrnd.png",1024,768,1,1,FALSE,0);

POINTF ptPos; ptPos.x = 0; ptPos.y = 0;
izBackgrnd->pParams->SetPos(&ptPos);
```

Любой объект с точки зрения движка, если он выводится на экран, значит, он — анимация. Картинка — это анимация из одного кадра. Поэтому, мы используем LoadSimpleAnim() для загрузки фона. В данном случае вся анимация состоит из 1 кадра с размерами 1024 на 768 и без прозрачности. Подробнее о методах — читайте описание интерфейса компоненты.

Обратили внимание на расширение картинки фона? Я рекоммендую использовать для всех изображений формат PNG. Для фона можно также использовать JPG — зависит от того, какого качества картинка вам нужна.

Чтобы наш фон отобразился на экране нужно добавить его в список выводимых объектов:

```
izBackgrnd->pDraw->Show(TRUE);
```

Делать это нужно один раз, т.е. при инициализации, как вы, наверно, догадались. Чтобы скрыть объект, достаточно вызвать Show(FALSE).

6) Сделаем возможность закрыть окно нажатием escape. Для этого добавим в Game_Main():

```
if(izEngine->KeyDown(VK_ESCAPE))

    SendMessage(g_hMainWnd,WM_DESTROY,0,0);
```

7) Чтобы на экран раз за разом отображались все новые и новые кадры используем след. метод: (учтите, что скорость обновления экрана зависит от установленных нами выше FPS = 30)

```
izEngine->pScreen->UpdateScreen(); //обновление кадра
```

Таким образом, Game_Main() примет вид:

```
int Game_Main(void *parms,int num_parms) {

    if(KeyDown(VK_ESCAPE))

        SendMessage(g_hMainWnd,WM_DESTROY,0,0);
```

```
	izEngine->pScreen->UpdateScreen(); //обновление кадра
	return(0);
}
```

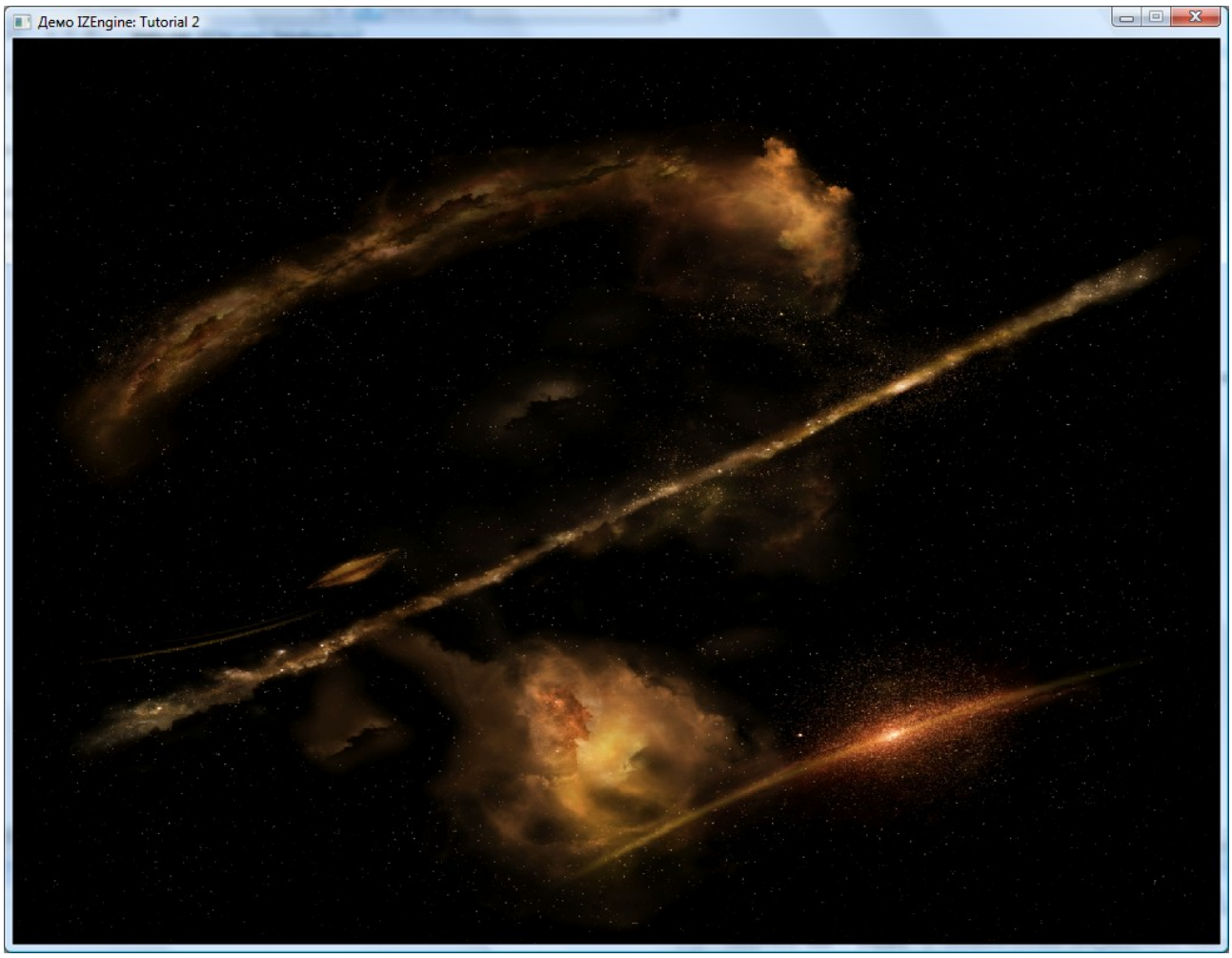
8) Все, что осталось, это — освободить ресурсы, вот код:

```
int Game_Shutdown(void *parms,int num_parms) {
	izEngine->pScreen->ShutDown();
	if(g_bMusic)
		PlaySound(NULL,g_hInst,SND_PURGE);
	return(0);
}
```

Обратите внимание, что после вызова pScreen->ShutDown() ни один объект IZObj/IZEngine не будет работать.

В Source_codes\Tutorial1 все это проделано (и даже больше). Для любителей MS Visual Studio есть MSVS_projects\Tutorial1. (версия VS 2008)

Вот итог:



Урок 2: Добавим динамики: летучий корабль. Разбираемся с движущимся объектом и управлением.

Добавим космический корабль игрока. Создаем корабль глобально и инициализируем его в Game_Init() как мы делали ранее с фоном:

```
IZObj* izShip = NULL;
int Game_Init(void *parms,int num_parms) {
        <...>
        //ship
        izShip = new IZObj;
        <...>
}
```

Для разнообразия пусть у него будет две анимации — одна при полете, а другая — при столкновении с чем-либо. Делается это следующим образом:

```
izShip->pAnim->LoadFrameTable(”ship.png”,h(0.1),h(0.1),0,1,TRUE,0);
```

Что за заковыристый метод? Читайте теорию движка. Здесь добавлю лишь, что он открывает рисунок, состоящий из кадров -друг за другом идущих в горизонтальную линию — ряд, поэтому число рядов = 1, а 0 означает, что число колонок вычисляется. h(0.1) означает, что наш корабль будет размером в 1/10 от ширины экрана. Метод ничего не спрашивает о

размерах изображения — считается, что кадры квадратные.

Для различия одной и второй анимации добавим enum глобально:

enum {ANIM_SHIP_FLY, ANIM_SHIP_COLLISION};

Для создания линейной[1] анимации на основе таблицы кадров (Frame Table – см. имя метода выше и теорию) используем метод pAnim->CreateLinearAnim(ANIM_SHIP_FLY,<стартовый кадр>, <конечный кадр>).

Число кадров в таблице можно узнать через метод: GetFrameAmountInTable(...)

Соответственно, код:

```
int nShipFrameTableFrames;
izShip->pAnim->GetFrameAmountInTable(&nShipFrameTableFrames);
izShip->pAnim->CreateLinearAnim(ANIM_SHIP_FLY,0,nShipFrameTableFrames*0.57-1);
izShip->pAnim->CreateLinearAnim(ANIM_SHIP_COLLISION,
        nShipFrameTableFrames*0.57,nShipFrameTableFrames-1);
ptPos.x = w(0.2); ptPos.y = h(0.2);
izShip->pParams->SetPos(&ptPos);
```

Обратите внимание, что мы используем приблизительную оценку числа кадров на первую и вторую анимации - nShipFrameTableFrames*0.57-1. Пусть мы имеем 100 кадров, тогда 57 (от 0 до 56-го) будут использоваться в первой анимации и от 57 до 100-го — во второй. Это сделано для того, чтобы приложение не зависело сильно от графики, идущей вместе с игрой. Вы можете точно задавать числа кадров на одну, вторую и т.д. анимации. Мне удобнее так.

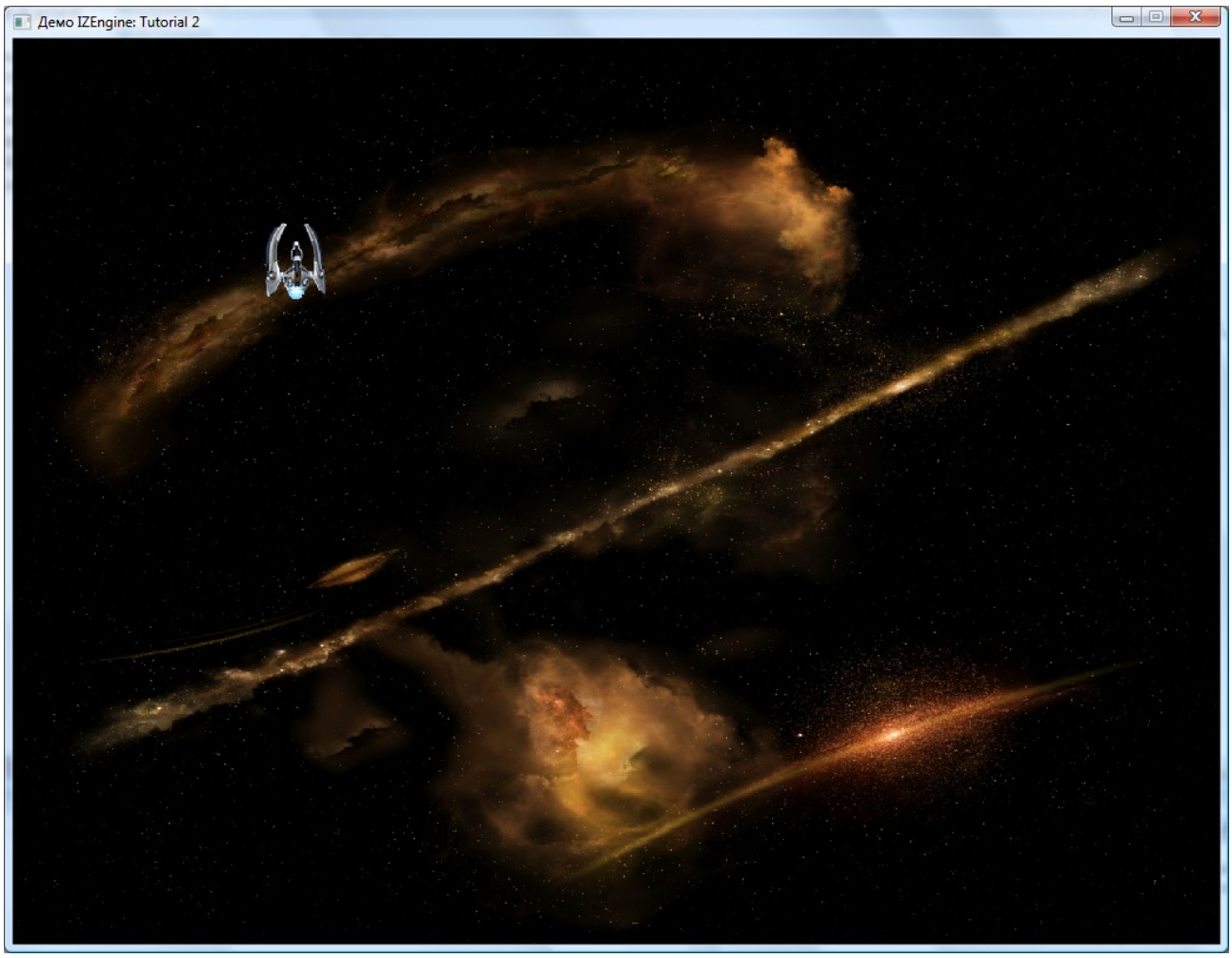
И, как и раньше, для вывода корабля на экран добавим в Game_Init() след.:

```
izShip->pDraw->Show(TRUE);
```

Добавим эту строку после вывода фона, чтобы корабль отображался поверх него, а не наоборот.

Получим следующее:

1 См. терминологию.



Реализуем космический хоккей — добавим одного противника и мяч, также поделим поле пополам. Пусть наш корабль смотрит всегда вверх, а противника вниз. Задачи игрока — отбивать мяч в сторону противника и не дать ему попасть в ворота (добавим сверху и снизу). Мяч отлетает от всех границ экрана, а после попадания в ворота появляется у пропустившего мяч.

Для начала, сделаем наш корабль движущимся. Для проверки нажатых клавиш, опять же, используем izEngine->KeyDown(<код клавиши>). Движение корабля сделаем с ускорением. Для управления используем след. клавиши:

W, S, A, D – вверх, вниз, влево, вправо соответственно.

Вот код:

```
//инициализация движения с ускорением

izShip->pAccelMove->SetAccel(h(0.002)); //устанавливаем ускорение

izShip->pAccelMove->SetMaxSpeed(h(0.05)); //уст. макс. возможную скорость

izShip->pAccelMove->EnableAccel(TYPE_ACCEL); //активируем ускорение (можно еще активировать тормоз)

//управление движением с ускорением
```

```
POINTF vecShipMove;

izShip->pAccelMove->SetAccelDirY(0);

izShip->pAccelMove->SetAccelDirX(0); //если пользователь ничего не жмет

if(izEngine->KeyDown('W'))

        izShip->pAccelMove->SetAccelDirY(-1); //-1 - направление,

//всего направлений 3:

//0 - стоим

//1 - движемся в сторону возрастания оси X или Y

//-1 -в сторону уменьшения

//Обратите внимание, что ось Y направлена вниз

if(izEngine->KeyDown('S'))

        izShip->pAccelMove->SetAccelDirY(1);

if(izEngine->KeyDown('A'))

        izShip->pAccelMove->SetAccelDirX(-1);

if(izEngine->KeyDown('D'))

        izShip->pAccelMove->SetAccelDirX(1);
```

Теперь корабль летает согласно нажатым клавишам, причем, чем дольше игрок жмет клавишу, тем быстрее летит корабль.

Все бы хорошо, но вот корабль вылетает за экран. Для борьбы с этим в IZEngine есть средства интерфейса pPhysics.

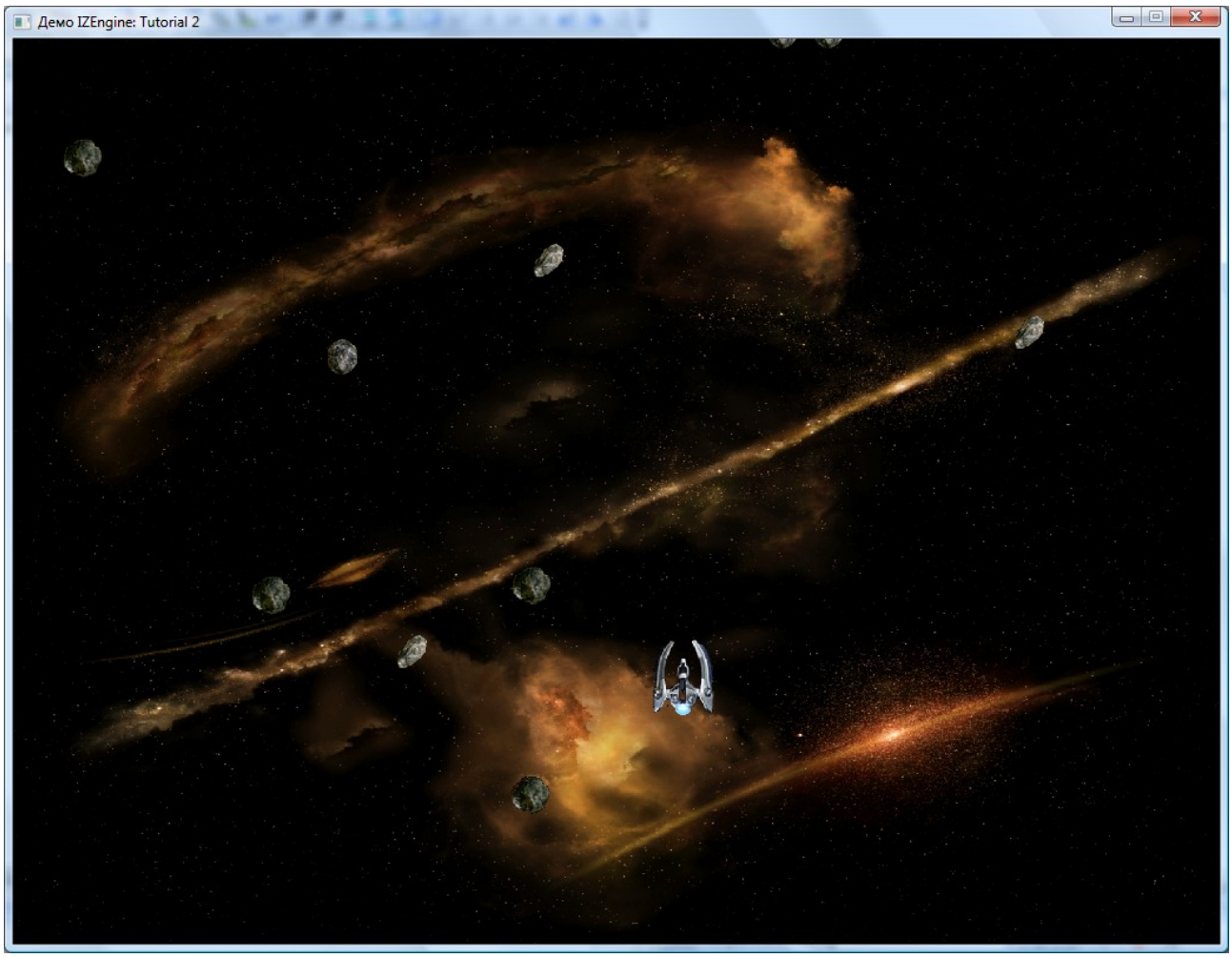
```
RECT rcShipBounds;

rcShipBounds.left = 0; rcShipBounds.top = h(0.5); rcShipBounds.right = w(1); rcShipBounds.bottom = h(1);

izShip->pPhysics->SetBounds(rcShipBounds,FALSE); //ограничим область для корабля нижней
        половиной экрана

izShip->pPhysics->SetBoundsReact(REACT_STOP); //пусть при попытке выхода за область корабль
        тормозит
```

Поэкспериментируйте и выставите также REACT_BOUNCE, REACT_TELEPORT, REACT_RAND_TELEPORT. Последние два больше подходят для случаев, когда ограничивающая область — весь экран. За их счет можно сделать, скажем, летящие астероиды:



Давайте добавим их и разнообразим геймплей. Вот как это можно сделать:

```
IZObj* izAsteroids; //глобально

const int g_nAsteroidsCount = 10;

Game_Init() {

	izAsteroids = new IZObj[g_nAsteroidsCount];

	for(int i=0;i<g_nAsteroidsCount;i++) {
		IZObj* izAsteroid = izAsteroids + i;
		int nTemp;
		izAsteroid->pAnim->LoadOrUseExistingSimpleAnim(
			(UCHAR*)"Ast100_0"+i+".png",h(0.05),h(0.05),0,1,TRUE,0);
		POINTF ptPos; ptPos.y = rand()%rcScreen.bottom; //установим астероид в произвольную
		//позицию по y
		//по x то же самое делать нужды нет - мы используем REACT_RAND_TELEPORT
		izAsteroid->pParams->SetPos(&ptPos);
		izAsteroid->pPhysics->SetBounds(rcScreen,FALSE); //область ограничения - экран
		izAsteroid->pPhysics->SetBoundsReact(REACT_RAND_TELEPORT); //пусть появляются с
		//другой стороны
		//экрана и с произвольной координатой x
		izAsteroid->pLineMove->SetSpeed(h(0.01));
		izAsteroid->pLineMove->SetDirY(1,false);  //пусть летят вниз навстречу игроку
	}

}
```

Несколько слов о LoadOrUseExistingSimpleAnim() - этот метод аналогичен LoadSimpleAnim(), но, если вы заметили, мы для нескольких объектов используем в данном случае один и тот же файл с графикой (Ast100_0I.png где I – от 0 до 2). Если бы мы воспользовались вторым методом, то один и тот же файл был бы загружен в память несколько раз, что, сами понимаете, накладно.

Не забудьте добавить астероиды на экран методом pDraw->Show(TRUE) аналогично тому, как мы делали с кораблем.

Запустите и посмотрите — теперь астероиды летят навстречу кораблю один за другим, вылетая из разных точек из-за экрана. (и исчезая внизу за экраном на самом деле, но игрок об этом знать не должен) А корабль управляется с клавиатуры.

Вроде все :) Рабочий код в Source_codes\Tutorial2\main.cpp. А проект под MS VS 2008 в MSVS_projects\Tutorial2.

Урок 3: Простые столкновения.

У нас есть корабль. У нас есть летящие астероиды. Они летят, а игрок наблюдает. Это скучно. Сделаем из милого зрелища какое-то подобие игры: пусть астероиды выводят корабль из строя при столкновении на какое-то время.

Для начала нам понадобится узнать, произошло ли столкновение астероида с кораблем. Для этого используем средства pPhysics->CheckCollisionCircleCircle(). Этот метод проверяет столкновение двух объектов на основе сопоставления вписанных в них окружностей. Другими словами, представьте, что ваши объекты (корабль и астероид) сферической формы. Это приближение, но для нашей аркадной игры этого вполне достаточно.

для проверки столкновения используем метод: (в реальном времени, т.е. в Game_Main())
izShip->pPhysics->CheckCollisionCircleCircle(izAsteroid);
И, в случае столкновения, отключим возможность управления.
Установим будильник:
izShip->pAlarm->SetAlarm(0,3000); //установим будильник 0 (а всего их 16) на 3 секунды
Проверим, «тикает» ли он:
BOOL bTicking;
izShip->pAlarm->CheckTicking(0,&bTicking);
Теперь, если он «тикает» - не будем обрабатывать нажатия клавиш от игрока:
if(!bTicking) {

```
	if(izEngine->KeyDown('W'))
		izShip->pAccelMove->SetAccelDirY(-1);
	if(izEngine->KeyDown('S'))
		izShip->pAccelMove->SetAccelDirY(1);
	if(izEngine->KeyDown('A'))
		izShip->pAccelMove->SetAccelDirX(-1);
	if(izEngine->KeyDown('D'))
		izShip->pAccelMove->SetAccelDirX(1);
```

}
Что делать с астероидом? Можно было бы его «отбросить» от корабля, но отскоки при столкновениях — тема одного из следующих уроков. Сейчас же ограничемся его уничтожением. Но уничтожать астероид нам особо незачем —

всего астероидов не так много (у меня их 10) и, если все астероиды будут уничтожены, игроку станет неинтересно.
Т.о. пусть у нас при столкновении появится анимация разлома астероида на части.
Реализация разлома на части
Вы можете реализовать сами. Подсказка: чтобы узнать, закончилась ли анимация, используйте метод pAnim->CheckEndAnim(); тип анимации PLAY_ONCE.
Или воспользоваться стандартными средствами движка. В IZEngine есть классы-расширения. В принципе, ими необязательно пользоваться, они существуют для удобства. Лежат они в папке \ext.
Нам понадобится ext\IZOneAnim. подключите его к проекту. Это — класс, создающий «одну» анимацию. Т.е. по окончании ее проигрывания она исчезает. Взрывы, эффекты при стрельбе и т.д.

```
void collideObjs() {
	for(int i=0;i<g_nAsteroidsCount;i++) {
		BOOL bCollision = FALSE;
		IZObj2* izAst = izAsteroids+i;
		izShip->pPhysics->CheckCollisionCircleCircle(izAst->pParams,&bCollision);
		if(bCollision) {
			IZOneAnim::Create(izAst->CentreCoordsPt(),g_nBigBang);
			//создаем анимацию разлома астероида
			POINTF ptf = izAst->WidthAndHeight();
			ptf.x = -ptf.x - 1; ptf.y = -ptf.y; //-ptf.x - 1 даст произвольную позицию по x
			izAst->pParams->SetPos(&ptf); //поместим астероид за экран — он еще пригодится
		}
	}
}
```

Уроки ниже еще не готовы

Это все, что написано на текущий момент. Ниже идут наброски на возможное будущее движка.

Урок N: Разлетаются во все стороны.

Описание реализации столкновений большого числа объектов с возможным отлетом.

Чтобы объект «учавствовал» в столкновениях — нужно сделать его «твердым». Да-да, объекты по умолчанию эфимерные :) Делается это след. образом:

izShip->pCollision->SetSolidShape(SHAPE_CIRCLE);

Теперь, при анализе столкновений, этот объект будет считаться окружностью.

То же самое нужно проделать для всех астероидов. Теперь, для проверки столкновения используем метод: (в реальном времени, т.е. в Game_Main())
POINTF
izShip->CheckCollision(izAsteroid);

Для каждого астероида.
В интерфейсе также есть методы реакций на столкновение:
Bounce(<объект, от которого отскакиваем>) - отскок; движемся в противоположную сторону от объекта.
Пусть же наш корабль отлетит от встречного астероида и будет неконтролируемым на несколько секунд.

```
//если есть столкновение
izShip->pCollision->Bounce(izAsteroid);
```

Теория движка.

Идея движка — универсальный объект, предельно мало занимающий в памяти и представляющий собой что угодно — анимацию, картинку, автомобиль, несущийся по дороге, корабль, летящий в космосе, планету на орбите звезды-гиганта и т.д. Загрузка графики осуществляется через интерфейс pScreen (иногда через pAnim / pAngledAnim) и вся графика хранится в куче запущенного приложения. Подключается по мере необходимости. На одной и той же графике может быть создано сколько угодно анимаций.

Расскажу о FrameTable, AngledTable, Frame, Anim, etc. Что под ними понимает движок и как они связаны с конкретными рисунками, выводимыми на экран.

Описание функций движка

Описание IZEng функций и функций библиотеки.

IScreen интерфейс

Интерфейс управления экраном, камерой, загрузкой/выгрузкой графики

Терминология

Линейная анимация — анимация, использующая кадры из таблицы кадров в их исходном порядке, не переставляя, друг за другом, ни одного не пропуская, от i до j, где i и j – порядковые номера кадров и i<j.

Будильник — таймер, отсчитывающий опред. число милисекунд и извещающий об окончании отсчета. Таймер отсчитывает каждый кадр, а извещает об окончании в течение одного кадра. В следующем кадре он возвращен в изначальное состояние.