

Март 2010

№1

[ПРОГРАММИСТ]

Программирование и алгоритмизация.
Для молодых и активных людей..

Пилотный выпуск. Мы начинаем!

```
Shift: TShiftState);
begin
  case Key of
    37: begin // клавиша "стрелка" - влево
      if xS1 <= 0 then EXIT; { если звездолет у левого края формы
                             xS1 не изменяется}
      dxS1:= -5;
      xS1:= xS1 + (dxS1+2); // движение звездолета влево
    end;
    39: begin // клавиша "стрелка" - вправо
      if xS1 >= 630 then EXIT; { если звездолет у правого края формы
                                xS1 не изменяется}
      dxS1:= 5;
      xS1:= xS1 + (dxS1-2); // движение звездолета вправо
    end;
  end;
end;
```

WinApi

Простейшая программа Windows API
на C++

В номере:

{
//Применение
изометрических координат
в Delphi.

// Установка отступов для
логических блоков
программы

//Как работать с графикой
на канве в среде Delphi.

//Быстрое преобразование
Фурье. Практика
использования. Часть 2
или... REAL-TIME
спектроанализ

// Простейшая программа
на WinAPI и C. Цикл
статей

И многое другое....

Издается с марта 2010. Выходит ежемесячно
№1, март 2010 г.

Редакция:
Utkin, JTG, Алексей Шульга, Сергей Бадло

Дизайн и верстка:
Егор Горохов, Indian, Сергей Бадло

Авторский состав:
Utkin, Виктор Кон, Владимир Дегтярь,
Дмитрий Федорков, Руслан Аблязов,
Сергей Бадло

Контакты:
Вопросы и авторские статьи направляйте на
maindatacentr@gmail.com
Информационная поддержка
www.programmersforum.ru

Примечание:
Мнение авторов не всегда совпадает с мнением
редакции. Перепечатка материалов журнала и
использование их в любой форме, в том числе в
электронных СМИ, возможно только с
разрешения редакции.

ТЕМА НОМЕРА	
Мы начинаем	с.0x02
НЕВЕРОЯТНО, НО ФАКТ	
Любопытные факты	с.0x03
НОВОСТИ ПРОГРАММИРОВАНИЯ	
Персональный язык программирования	с.0x07
АЛГОРИТМЫ	
Установка отступов для логических блоков программы	с.0x0D
АЛГОРИТМЫ. ГРАФИКА В DELPHI	
Применение изометрических координат в Дельфи	с.0x10
Как работать с графикой на канве в среде Дельфи. Урок 1-2	с.0x15
РЕАЛИЗАЦИЯ. НАЧИНАЮЩИМ	
Простейшая программа на WinAPI на C	с.0x1B
ЛАБОРАТОРИЯ. В ЗАПИСНУЮ КНИЖКУ ИНЖЕНЕРА	
Быстрое преобразование Фурье. Практика использования. Ч-2	с.0x20
ИГРОВАЯ ПЛОЩАДКА	
Игра Fortress. Конкурс на создание лучшего бота. Итоги	с.0x2B

О журнале

Дорогие друзья! С марта 2010 года выходит в свет первый номер нового журнала «Программист». Это совместный проект клуба программистов www.programmersforum.ru.

Все его содержание создается начинающими и профессиональными программистами, электронщиками и инженерами, и рассчитано на широкий круг читателей. Мы считаем, что материалы журнала должны носить не только теоретический, но и практический характер и быть действительно полезны программисту в его повседневной работе. Об уровне и характере статей вы можете судить по материалам нашего форума. На нашем форуме вы можете пообщаться с авторами статей, задать коллегам вопросы и поделиться с ними мнениями по интересующим Вас темам.

Рубрики журнала (плавающие)

- Новости программирования (новые языки, концепции, среды, дополнительные утилиты)
- Отдел тестирования (оценка удобства использования той или иной средой)
- Общие вопросы (вопросы правового использования, личные мнения о тех или иных механизмах и языках и т.д.)
- Алгоритмы (описание некоторых методов работы без привязки к языку и платформе)
- Юмор (специфические хохмы программистов)
- Реализация (описание различных тонкостей программирования)
- Разработка (циклы статей по созданию программных проектов от этапа постановки задачи до получения работающей программы)
- Переводные материалы (перевод статей по программированию)
- Рубрика про железки / Лаборатория

Периодичность выхода – раз в месяц. Ящик для корреспонденции и вопросов: maindatacentr@gmail.com

Общие требования к присылаемым материалам

У нас нет категоричных требований к оформлению материалов, но в связи с особенностями верстки (используется свободное ПО «SCRIBUS») и облегчения труда редакторов, есть некоторый желательный минимум:

- статья должна иметь выраженную структуру с разделами и содержать – название статьи, сведения об авторах, экскурс или введение, сведения об используемых средствах разработки, теоретическую и/или практическую часть, заключение (выводы, чего добились) и ресурсы к статье (ссылки на код, интернет-ресурсы, литературу)
- текст статьи в формате MS Word или обычным текстовым файлом
- шрифт Arial 10
- все рисунки, таблицы должны иметь упоминание в тексте и иметь подпись
- рисунки (скрины) к статье должны прилагаться в виде отдельных файлов в формате PNG, BMP или TIF
- разделы статьи отделять двумя <ENTER>
- не используйте табуляцию без необходимости, в конечном итоге присланный материал все-равно переводится в текстовый формат
- по присланным материалам автор получает рецензию и корректирует статью согласно замечаниям

Замечание редактора обязательно должно вызывать реакцию автора. Если автор не согласен с комментарием или исправлением редактора, он должен разъяснить вопрос в виде комментария в статье на форуме "Обсуждение статьи" или на ящик-копилку. Если редактор согласен с тем, что ошибся, вопрос снимается. Если редактор не согласен, решает группа.

С уважением, редакция журнала «Программист»

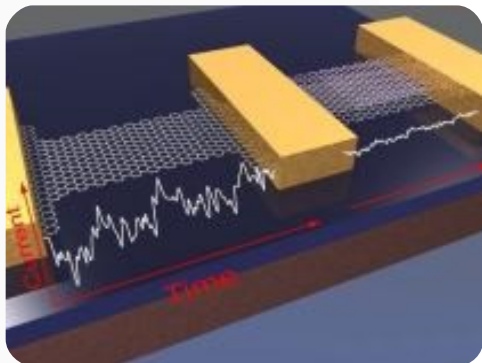
Любопытные факты...

Данная рубрика пожалуй самая динамичная в нашем журнале, так как будет включать в себя различные «вкусности», как: новости из мира технологий, исторические факты, новинки электроники, радиотехники и программирования. Может быть даже стоит переименовать ее в "Мировую солянку"?

р.с.: редакции требуется активный ведущий данного раздела

Разгон 3G.USB модема: для повышения уровня сигнала и как следствие стабильности коннекта и скорости, можно воспользоваться простым, не требующий затрат, способом. Подопытным выступил «Билайновский» модем ZTE MF-622. Нужен провод ~20см длины, желательно медный (в резиновой оплётке или без, неважно). Оборачиваем несколько витков вокруг модема, расстояние между витками 1см. Также нам понадобится CD или DVD диск, любой ненужный, и проутюжив его, слегка согните как показано на фото. Направляем наш импровизированный девайс в ту сторону, где находится вышка оператора и... вуа-ля.

(автор, наш форумчанин **Vasek123**)



Высокочастотный транзистор на основе графена: продемонстрировали исследователи из IBM, работающий на рекордной граничной частоте, когда-либо достигнутой графеновым полупроводниковым элементом, 100 млрд рабочих циклов в секунду или 100 ГГц. Графен представляет собой слой атомов углерода толщиной в один атом, соединенных в гексагональную сотоподобную кристаллическую решетку. Эта двумерная форма углерода обладает уникальными электрическими,

оптическими, механическими и тепловыми свойствами. Примечательно, что рабочая частота графенового элемента уже превышает граничную частоту современных кремниевых транзисторов с такой же длиной затвора (40 ГГц). Подобная производительность была достигнута у элементов на основе графена, полученных из природного графита.

Первым в мире программистом: была женщина – англичанка Ада Лавлэйс. В середине 19 века она составила план операций для прообраза современной ЭВМ - аналитической машины Чарльза Беббиджа, с помощью которых можно было решить уравнение Бернулли, выражающее закон сохранения энергии движущейся жидкости. В 1975 году Министерство обороны США приняло решение о начале разработки универсального языка программирования. Министр прочитал подготовленный секретарями исторический экскурс и без колебаний одобрил и сам проект, и предполагаемое название для будущего языка «Ада». 10 декабря 1980 года был утвержден стандарт языка.



Иракские боевики «прослушивали» американские БПЛА: с помощью российской программы SkyGrabber. При этом, чтобы достичь своих целей боевикам практически не пришлось тратить какие-то огромные деньги. Они просто воспользовались совершенно не сложной утилитой. Как оказалось, SkyGrabber способна перехватывать сигналы и с беспилотников. Используя ее, иракские повстанцы получили возможность добывать важные сведения об операциях воинских контингентов.

Защитные диоды на входах КМОП логики: можно использовать как диодный мост. Достаточно подать переменное напряжение на входы элемента 2И-НЕ (2ИЛИ-НЕ) и получаем питание для остальных 3-х элементов.

Программа для записи дисков Nero Burning ROM: получила своё название неслучайно. Это каламбур, в буквальном переводе «Нерон, жгущий Рим».

Если у вас завалилась советская логика: в частности микросхема 155ЛП7, то ее можно распилить на транзисторы. В неё входят два логических элемента и два транзистора и довольно неплохие, около $U_{ke}=30V$ и $I_{ki}=300-500mA$. Чтобы логическая часть не мешалась под ногами, отпиливаем ее в прямом смысле - ножовкой по металлу между 3-м и 4-м выводами (или между 11-м и 12-м). Получается 2-х транзисторная сборка в 8-ми выводном корпусе.

Получить солнечный элемент: можно пропилив отверстие в корпусе транзистора. Особенно хороши модели советского периода типа МП и П. КПД однако маловат...

Блок управления от старой микроволновки: можно использовать для плавной регулировки мощности паяльника, так как она содержит готовый ШИМ контроллер.

Отечественные стабилизаторы КРЕН142 или импортные LM317: замечательно работают как... задающие генераторы при подключении на выход колебательного контура с КПЕ.

Микросхемы динамической памяти DRAM: (NEC 4164 и обычные KP565PY5) можно использовать как видеокамеру. Сам принцип использования основан на эффекте ускорения разряда конденсаторов в ячейках памяти при воздействии на них света. Для этого нужно по всем адресам микросхемы со снятой крышкой записать единицы, через некоторое время (без регенерации) - прочитать. Те ячейки матрицы памяти, на которые падает свет, разряжаются и переходят в состояние 0 раньше, которые остались в тени - позже.

DRAM является микросхемой памяти, поэтому распределение логического адреса должно соответствовать физическому месту на кристалле. Чтобы узнавать это распределение необходимо провести несколько испытаний. В микросхеме DRAM, адреса ячеек которой состоят из двух составляющих - адреса строк и адреса столбцов,



Логические адреса в пределах строки соответствуют разумеется не физическому месту ячейке памяти на интегральной схеме на монокристалле. В обработанном изображении видно это ошибочное распределение перестановленными строками и столбцами. Перестановка разрядов адреса позволяет этот дефект устранить, но, тем не менее, здесь все-таки нужен экспериментальный подход. Геометрия интегральных схем на монокристалле различных изготовителей может быть другой.

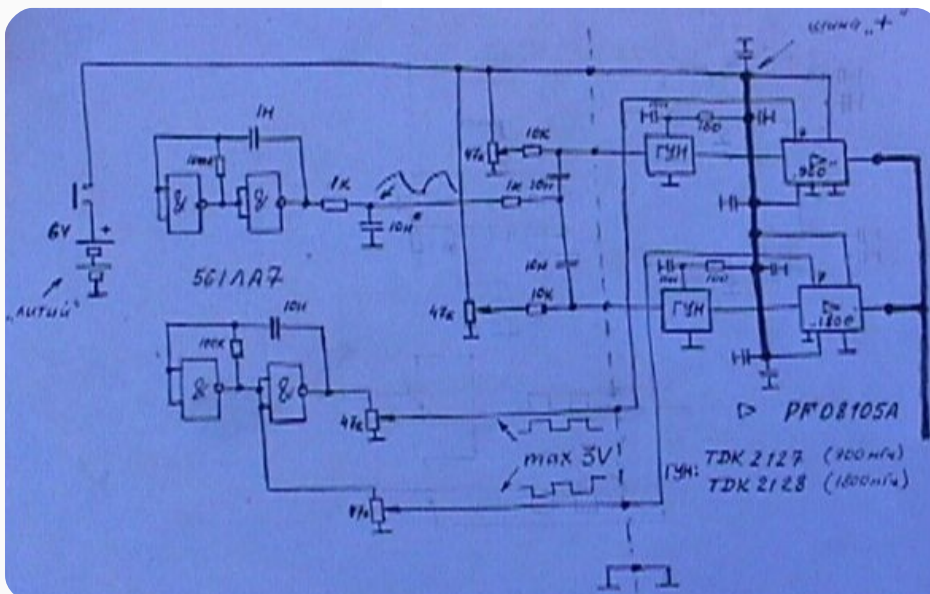
NEC 4164

Speichermatrizen

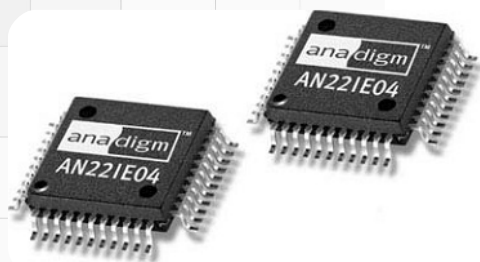
Kleinbild

DRAM

Малобюджетный вариант глушилки GSM: можно собрать и в любительских условиях. Для этого вам понадобятся две микросхемы ГУН TDK-2127/2128 из старых мобилок Siemens C25 и отечественная КМОП логика К561ЛА7 для ГКЧ. Для повышения выходной мощности можно добавить, снятые с этой-же мобилки, ВЧ усилители PF08105A. Питание – обычный 6В аккумулятор от брелка сигнализации.

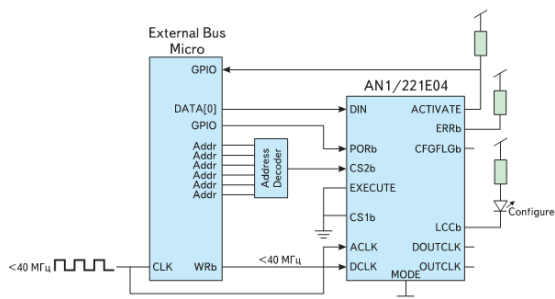


0x05

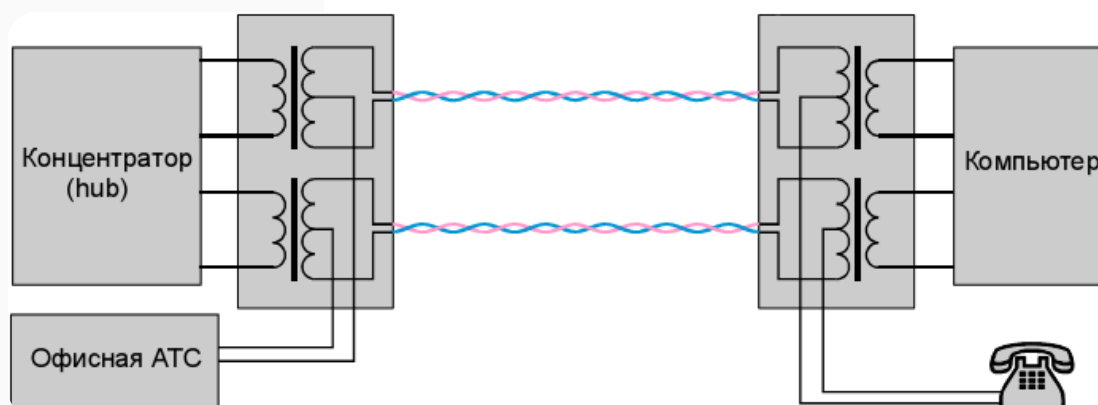


Новые образцы ПАИС: (программируемой аналоговой ИС) стали доступны массовому потребителю, в частности AN221E04. В отличие от ПЛИС, тут вы можете разрабатывать любые **аналоговые** цепи, приемники, фильтры. Основное преимущество – динамическое переконфигурирование в процессе работы (на лету).

Нужен приемник с фильтром, заливаете конфигурацию и вот оно, нужен регистратор сейсмических колебаний – получите его мгновенно. Новая конфигурация активируется за один такт синхронизации. Фирмой-разработчиком прилагается бесплатное ПО для проектирования AnadigmDesigner с поддержкой СИ - кода. Правда, цена на сами микросхемы до сих пор относительно «кусачая», около 25 уе.



Передача данных по фантомным цепям: известно, что кабельные линии связи, содержащие группу витых пар медных проводов, обладают некоторыми «скрытыми возможностями». Они заключаются в том, что помимо использования витых пар по прямому назначению (для передачи данных) каждая из них может рассматриваться как отдельный провод. Совокупность таких проводов позволяет без дополнительного увеличения числа жил кабеля сформировать дополнительные (фантомные) каналы связи или (и) цепи питания удалённых устройств. Основные идеи построения фантомных каналов и цепей предложены более 30 лет назад и, по существу, с тех пор мало изменились. Тем не менее, представленное решение может оказаться полезным при проектировании систем передачи данных, например при использовании общей кабельной инфраструктуры для построения офисной телефонной сети из сети Ethernet (см. рисунок):



Чтобы не вводить изменения в существующую разводку, блок трансформаторов размещается в вилке тройнике, содержащим две розетки для подключения кабеля телефонного аппарата и кабеля сетевой карты компьютера. Фантомные цепи обеспечивают повышенную дальность связи (или скорость передачи) благодаря удвоенному сечению жилы медного проводника (витая пара рассматривается как единый провод) и меньшей погонной емкости между витыми парами, чем между проводами витой пары.

Персональный язык программирования

Статья имеет целью познакомить читателя с одним представителем огромной армии неизвестных языков программирования. Этот язык я придумал сам для себя, он ни на какой язык не похож и этим может быть интересен.

Виктор Кон <http://kohnvict.narod.ru>

Про языки программирования написано очень много статей и философских размышлений. Их много и есть необходимость в их классификации. Главный аспект – это то, что языки могут быть многоуровневыми. Есть ассемблер, есть базовые языки более высокого уровня: Basic, C++, Java и много других. С их помощью можно написать компиляторы и интерпретаторы других языков еще более высокого уровня и так далее. Второй аспект – разделение языков на компилируемые и интерпретируемые. Первые используются для создания независимых и конечных программ, например программы решения квадратного уравнения. Вторые являются текстовыми инструкциями для одной единственной программы – интерпретатора. Бывают и пересечения, когда программа частично компилируется в промежуточный код, который затем интерпретируется. Ну и, наконец, третий аспект – структура самой программы, тут три уровня. Вся программа одним куском, программа с процедурами, программа с объектами. Какой язык и какая программа удобнее для конкретного пользователя зависит от свойств самого пользователя.

Вместо введения

Я хочу рассказать о себе. Я пользователь в том смысле, что пишу программы для своей научной работы. Почти все программы моделируют те или другие физические процессы и часто используются только один раз. Условно конечно, много раз в процессе отладки



**Кон Виктор
Германович**

родился 22.09.1944,
физик, теоретик,
доктор физ.-мат. наук.

Постоянная должность: главный научный сотрудник Теоретического Отдела Института Сверхпроводимости и Физики Твердого Тела Российского Научного Центра «Курчатовский Институт» (РНЦ КИ).

Адрес РНЦ КИ:
пл. Курчатова 1, 123182 Москва, Россия.

до тех пор пока не получен достоверный результат. После того, как результат получен – программа уже не нужна. Опять неточно, нужен ее код, чтобы развивать его дальше и учесть более сложные процессы, но на каждом этапе программа пишется на результат, который получается конечным числом ее запусков. Как писать такие программы? Можно конечно одним куском и на ассемблере. Но это будет огромный труд с очень низким коэффициентом полезного действия. Можно писать процедуры и компилировать их в новые сборки программ. Но и это не оптимально. В каждой программе будет очень много одного и того же кода, который будет повторяться, засоряя носители информации. Это не так важно сейчас, но было важно раньше, когда компьютеры были маломощные. Сама практика подсказала, что наиболее оптимальным путем является программа-интерпретатор, выполняющая

команды одну за другой, а наиболее удобным и простым для выполнения конкретных работ является командный язык программирования с возможностью группировать код в процедуры.

Так получилось, что я пришел к этому выводу в далеком 1991-м году, когда программ еще было мало, все работали в операционной системе ДОС, которая мало что умела и о программах типа Mathematica, IGOR-pro и им аналогичных даже и не мечтали. Моя идея состояла в том, что нужно сделать интерпретатор языка программирования высокого уровня, который имел бы команды всех уровней, начиная от прерываний ассемблера и кончая запуском редактора текстов или программы построения графика одной командой. В то же время структура языка должна быть такой, чтобы четкие и конечные операции можно было сделать написав несколько строк текста. Я потратил два года работы и сделал себе такую программу. Работая в ДОС, она имела графические возможности, необычный редактор текстов с уникальными возможностями, хорошие возможности по интерфейсу и очень мне помогала в работе. Самый главный плюс такой программы состоял в том, что я мог очень быстро получить ответ на многие сложные вопросы. А быстрота получения ответа иногда имеет решающее значение. Часто просто нет времени на написание и отладку огромного кода, ответ нужен, как говорят, еще вчера. И я мог это делать.

Минусы у программы тоже были – отсутствие шрифтов. Конечно, я создал свой масштабируемый фонт, который можно было использовать, но это было не очень быстро и не очень красиво. Когда появилась Виндовс, а компьютеры стали более мощными, необходимость в интерпретаторе отпала. Самый главный минус был в том, что программа могла работать только в ДОС. Виндовс не разрешал многие операции, которые она могла делать, да и просто ее не запускал по соображениям безопасности. Я снова рассыпал программу на отдельные куски, но и они постепенно становились все менее привлекательными.

Прошло время – появились КПК (карманные персональные компьютеры), это чудо на ладони с большими возможностями по части поиграть музыку или показать кино. Но они не умели делать вычисления, не было программ. А уж о том, чтобы программировать прямо на КПК не было и речи. Я снова вспомнил про свой язык программирования и свой интерпретатор. Код интерпретатора был написан на фортране, но для КПК нужно было писать на чем-то другом. Так получилось, что я в то время (это уже был 2004 год) начал работать на Java, поэтому выбор был сделан. Я просто перетранслировал вручную весь код с фортрана на Java, даже порой уже не понимая, что этот код конкретно делает. И все получилось – код заработал. Конечно, пришлось отказаться от прерываний, немного по-другому написать редактор текстов, по-другому сделать интерфейс. Но сам язык программирования и его интерпретация почти не изменились. Эта программа до сих пор работает у меня на КПК, который я как раз и купил в начале 2004 года. Он до сих пор работает с той же самой батареей. Это отдельная тема и я не буду ее развивать здесь.

Важно, что мне снова понравилось работать на своем языке, и я сделал клон этой программы также и для ПК, то есть настольного компьютера и ноутбука. Эта программа имеет больше возможностей и настолько удобна, по крайней мере, для меня, что я уже не мыслю себе другой жизни. Вот вчера мне понадобилось выделить из pdf файла книги в 150 страниц всего 4 страницы и записать в другой pdf файл. На моем языке это делается в несколько строк, но и их писать не надо – они уже написаны. А вычислить, скажем,

таблицу значений функции Бесселя и построить график можно с помощью программы в одну строчку.

Но это только примеры. Реально программа умеет очень много всего, практически все, что лично мне нужно и самое главное – если чего-то нехватает, то просто дописываем интерпретатор и создаем новую команду. Сложность только в том, что, как и у любого интерпретируемого языка, команд очень много. Но они хорошо структурированы и есть описание с хорошей навигацией. Так что перечитав описание можно все легко вспомнить. Наизусть я свой язык не помню.

Вместо рекламы

Цель этой статьи – просто познакомить уважаемую публику с тем, что у меня есть. С самого начала я не стремился делать коммерческий продукт, но в какой-то момент работы над программой я подготовил вариант, который содержит только команды общего назначения, могущие представлять интерес для всех. Этот вариант я выложил в интернет на отдельный сайт <http://vkacl.narod.ru>, а также разместил в каталоги программ. Неожиданно программа стала популярной на сайте FreeSoft, где ее скачали уже более 40 тыс. раз. Были и письма, из которых я понял, что кое-кто действительно научился пользоваться. У программы есть несколько вариантов. Первый, основной, содержит интерпретатор языка, который я назвал ACL (advanced command language), среду разработки, включающую встроенное описание языка, и примеры готовых программ. Эта программа называется vkACL.jar. Как любая Java программа, она сама запускается через виртуальную машину Java или JRE, которую предварительно надо установить на компьютер. Дистрибутив JRE можно бесплатно скачать хоть с родного сайта, хоть с разных зеркальных сайтов, например [1]. Скриншот программы можно посмотреть вот по этому адресу [2].

Есть также вариант проигрывателя с окном, то есть это программа, которую можно настроить таким образом, что она будет показывать меню и выполнять (интерпретировать) одну ACL программу. Такой вариант не содержит помощи и выглядит как конечная программа на одну функцию. При этом jar-файл может иметь любое имя. Программу в таком варианте я использую для разработки конкретных программ, с которыми могли бы работать мои коллеги, не умеющие программировать. Эти программы выглядят как обычные Java программы. Есть еще и второй вариант проигрывателя, который совсем не имеет головного окна. Такая программа либо выполняет свою работу, не общаясь с пользователем совсем (так работали старые программы в эпоху до ПК), либо средства общения с пользователем написаны в самой ACL программе. Иногда такой вариант бывает оптимальным.

Наконец, есть вариант программы в виде Java-апплета, который сразу работает в интернете и запускается вот по этому адресу [3]. Этот вариант сделан совсем недавно и я его сейчас развиваю. Если у кого есть быстрый интернет, то можно сразу пользоваться программой с любого компьютера. К сожалению Java-апплет не может работать с файлами пользователя, но я организовал ввод и вывод данных через редактор текстов. Любую информацию, в том числе и картинки, можно ввести в программу и вывести из программы копированием текста через буфер обмена (клавиши Ctrl-A, Ctrl-C, Ctrl-V), а готовые картинки можно скопировать с экрана, используя любую программу с функцией snapshot, то есть фотографирования части экрана в файл. Таких программ много, подробности можно прочитать в моей статье <http://kohnvict.narod.ru/Data/advice.htm>.

Java - апплет содержит несколько готовых программ общего назначения, в частности таблицу Менделеева в виде книги, а также мои научные программы для коллег. Также каждый может скопировать и выполнить свою программу.

Что такое ACL?

Невозможно в двух словах объяснить, как работает космический корабль, слишком там много деталей. Такая же проблема с языком программирования, который рассчитан на то, чтобы делать все на свете. Когда я начинал, то не имел никакого опыта, да тогда вообще все было в очень ограниченном ассортименте. Поэтому я ничего не читал, а все придумал сам. Первый момент - никаких строк и меток. Символ конца строки ничего не значит. В эпоху ДОС это было революционно. Программа пишется таким образом, что как раз комментарии ничем не выделяются, они пишутся свободно, а вот сами команды языка программирования начинаются с символа (#). Этот символ для интерпретатора служит знаком, что пора начинать работу. После этого знака пишется имя команды. Принцип написания всех имен следующий: можно писать сколько угодно букв, но слитно (без пробелов). Реально команды различаются максимум по трех первым буквам, а часто даже просто по одной букве. Я не люблю много писать, поэтому мне не нравится даже Java, хотя я считаю ее лучшим языком программирования. Команды из одной или двух букв абстрактны, но это дело привычки. Любой язык хорош, который знаешь. Вот у меня фамилия Кон. И это намного удобнее, чем Константинополь. Итак, читается слово до пробела и выбирается три первые буквы или меньше, если их меньше.

Что дальше?

Каждая команда - это отдельный блок программы, который знает, умеет и выполняет свою работу. Но часто ему нужны данные, конкретизирующие задачу. Входные данные разделены на два типа. Первый тип - это целочисленные параметры, которые имеют имена. Сделано это исключительно для более удобного понимания программы. Параметры универсальные и используются разными командами, причем один раз определенный параметр сохраняет свое значение до следующего определения. Определять параметры можно после каждой команды в поле, заключенном в квадратные скобки. Есть дополнительно к целочисленным несколько текстовых параметров. А всю остальную информацию надо записывать как аргументы сразу после квадратных скобок. Аргументы задаются как во многих командах многих интерпретаторов, хоть в ДОС (батч-файлы), хоть в компиляторах и прочих программах с командной строкой. Потому язык и называется командным.

Самое интересное, что больше ничего и нет. Точнее нет грамматики языка. Одни команды. И вся грамматика тоже реализована через команды. Одна команда запускает огромную готовую программу, другая реализует грамматику, но структурно они неразличимы. Возникает законный вопрос - а как делать вычисления? А где циклы, условия, логика? Ведь языки программирования, как и разговорные языки - как ни пиши, а предмет то один и тот же. Все верно. Но у меня «быстрый» язык, он кстати в первой редакции назывался quick. Для него главное - запускать готовые блоки, а все остальное вспомогательное. Поэтому все остальное реализовано по минимуму. Переменные все реальные и их имена короткие - одной буквой от (a) до (z), затем от (A) до (Z) и еще три символа (\$), (%), (&). Можно писать одну букву и одну цифру, например (z1=x3;). И все - больше переменных нет, а зачем много? Массивы - есть один целый массив i(), один реальный массив r() и один массив символов (уникодов) t(). Индексы в

круглых скобках. Каждый из массивов имеет фиксированный (большой) размер, а все команды работают с частями этих массивов, причем есть специальные параметры (b [begin]) и (le [length]), которые как раз и выделяют индексы массивов, используемые в программе. То есть фактически вместо имени массива используется его начальный индекс в большом массиве. А зачем имена? Проблема выбора для некоторых людей – непреодолимое препятствие. А так все массивы плотно и динамически упаковываются в одном. Нет строк. Массив символов и строка – одно и то же. Более того, символы и числа – тоже одно и то же. Элементы символьного массива могут участвовать в вычислениях.

Как делаются вычисления? А очень просто. Открывается команда вычислений `#cal` или `#c` или просто `#`. Это единственная команда, которая может иметь нулевое имя. И все формулы вычислений являются ее аргументами. Она вычисляет и переопределяет переменные и массивы, это ее работа. В вычислениях можно использовать имена функций, кроме стандартных есть еще функции Бесселя и интегралы Френеля. Пример: `# z=sin(3.14*20/180); X1=5/exp(-2.7)+r(25); #%` А что с циклами? Есть простой цикл, реализованный командой `#rep N (repeat)` с одним аргументом (числом повторений). Все команды после этой выполняются до команды `#end`, которая возвращает сканирование программы в самую первую команду после `#rep` и так ровно N раз. Все индексы и переменные должны переопределяться внутри цикла вручную. А условные циклы? Вот тут интереснее. Все сделано с помощью всего одной команды – и условия, и условные циклы. Зачем много? Опять проклятая проблема выбора. Просто есть еще одна команда цикла, она называется `#case N`.

Эта команда так же точно, выполняет все команды до команды `#end`. Но при одном условии, что ее аргумент совпадает со значением переменной (&). Если не совпадает, то вообще ничего не делается. В данном случае команда `#end` снова проверяет это же самое условие. Если не изменить переменную (&), то цикл будет делаться бесконечно. Поэтому внутри тела цикла ее надо изменить как раз тогда, когда надо. Таким способом можно сделать выбор: `# &=2; #case 1 . . . #end | #case 2 . . . #end | #case 3 . . . #end |` Вот в этом коде выполнится только многоточие после `#case 2`. Знак вертикальной черты (|) после `#end` через пробел означает автоматическую смену переменной (&) на значение 12345. Таким образом, тело цикла выполнится один раз. Перебор вариантов мы делаем. Но нам нужна логика. Программа должна уметь мыслить.

На это я могу авторитетно заявить. Не умеет компьютер мыслить. Все что он умеет – это сравнивать два числа на больше и меньше, а еще точнее – определять знак числа. И одной этой операции ему хватает, чтобы обыгрывать человека в шахматы. И не только в шахматы, а вообще делать разумную работу. Я не стал напрягаться и делать как у всех. Я просто ввел две новые операции `<` и `>` в математические вычисления. Результатом операции `a<b` является минимум из двух значений `a` и `b`. Аналогично `a>b` вычисляет максимум. И все, больше ничего не надо. Это позволяет писать полноценные программы, реализующие любую логику при переборе вариантов.

Заключение

Есть еще много интересных нюансов в языке и много такого, что позволяет ему иметь неограниченные возможности по созданию очень большой программы, включающей разные куски кода, написанные в разных файлах где угодно, в том числе и на серверах в интернете. Есть готовая научная графика, возможность работы с картинками, zip-архивами, делать любые операции с файлами и текстом. Но об этом можно очень долго

писать. Моя задача состояла в том, чтобы заинтересовать читателя обратиться на указанные выше сайты за более подробной информацией. Я надеюсь, что эту задачу я выполнил. Напоследок скажу чего нет. Нет работы с базами данных стандартного вида, хотя аппарат работы с небольшими базами данных есть. Я не работаю с базами данных и мне это не нужно. До сих пор я не освоил аппарат работы с xml файлами, каюсь. Но и это мне не нужно. Персональный язык программирования не обязан быть универсальным.

Ресурсы

- . Дистрибутив JRE <http://java.sun.com> или <http://www.java.com/ru/download/index.jsp>
- . Скриншот программы
http://img-fotki.yandex.ru/get/3904/kohnvict.12/0_24d12_9b5a6cf9_orig
- . Вариант программы в виде Java-апплета <http://vkacl.narod.ru/applets/00/vkACLa.htm>

Установка отступов для логических блоков программы

Данная статья раскрывает некоторые тонкости алгоритма одного из элементов форматирования исходных текстов программ. Рассчитана на широкий круг программистов и не требует специальных знаний.

by Utkin www.programmersforum.ru

Наверное, каждый замечал – в хороших редакторах код, заключенный в операторные скобки, такие как `{-}`, `begin - - end` и пр., автоматически выравнивается по определенному отступу слева. Это делает код более удобным для чтения и придает ему определенный стиль. Как это выполнить самому? Давайте представим условный редактор кода и научим его выполнять такой элемент автоформатирования...

Краткий экскурс...

Прежде всего, как советуют классики, нужно определиться со структурами данных, в которых будет проводиться основная работа по форматированию. Большинство языков программирования имеют поддержку массивов различных типов данных. Поэтому будем представлять входящий текст программы как массив строк. Теперь рассмотрим вопрос о представлении отступа. В зависимости от реализации это может быть задание отступа отрисовки символов в компоненте вывода текста на экран, либо задание символа табуляции (в случае если транслятор позволяет использовать табуляцию в программах), либо просто определенное количество символов пробела в качестве заменителя символа табуляции. Теперь о самих отступах – все, что нам требуется, это либо добавление в начало строки определенного символа (или группы символов, в случае если символ табуляции будет заменен на пробелы), либо передача информации компоненту отрисовки текста программы о величине отступа от левого края для конкретной строки. Важное условие – величина отступа меняется в зависимости от уровня вложенности логических блоков программы. Еще немного конкретики – нам нужно знать, как в данной программе представляются операторные скобки. Дадим данному понятию свое определение – маркер начала логического блока и маркер конца логического блока. Для нашего алгоритма это просто строка-образец, содержащая в себе операторную скобку.

Также нам нужно где-то помнить текущую величину отступа, в чем она будет выражаться – в пикселях отступа для компонента, число символов табуляции или число символов пробела, зависит только от конкретных реализаций данного алгоритма. Теперь введем также определение шага отступа – то есть, на сколько смещается отступ от левого края в зависимости от уровня вложенности логических блоков программы. Так, величина отступа для содержимого логического блока будет равна сумме общего отступа и отступа текущего блока, а шаг отступа будет равен отступу текущего блока (см. таблицу):

Таблица. Условное представление логических блоков

ЛОГИЧЕСКИЙ БЛОК ПРОГРАММЫ		
общий отступ	маркер начала логического блока	
общий отступ	отступ текущего блока	СОДЕРЖИМОЕ БЛОКА
общий отступ	маркер конца логического блока	

0x0D

В общем, имеющейся информации уже достаточно для построения алгоритма установки отступов для логических блоков программы.

Итак, приступим

Сначала обычно имеет место быть инициализация, то есть создание необходимых структур и установка конкретных значений для величины отступа, шага отступа, маркеров начала и конца логических блоков. Весь алгоритм* будет представлять собой один цикл по всем элементам массива строк – хранилище нашего кода программы:

1. Получим очередную строку программы из массива строк.
2. Удалим из левой ее части все пробелы и символы табуляции, если они там имеются.
3. Получим первую лексему языка программирования из нашей строки. Это может быть символ или ключевое слово.
4. Сверим наше слово с маркерами.
5. Если слово является маркером начала логического блока, то...
 - 5.1. В зависимости от особенностей реализации – добавим в начало строки символы табуляции или символы пробела, либо передадим информацию в компонент по отрисовке строк программы в соответствии с текущим отступом.
 - 5.2. Увеличим отступ на величину шага отступа.
 - 5.3. Закончим текущую итерацию цикла.
6. Если слово является маркером конца логического блока, то...
 - 6.1. Если разница между отступом и шагом отступа будет больше или равно нулю, то уменьшим отступ на величину шага отступа.
7. В зависимости от особенностей реализации – добавим в начало строки символы табуляции или символы пробела, либо передадим информацию в компонент по отрисовке строк программы в соответствии с текущим отступом.

* такой способ весьма прост, но имеет один недостаток – его нельзя использовать для языков программирования, в которых отступы влияют на процесс вычисления программы – это Питон, некоторые версии Хаскела и т.д.

Некоторые комментарии по пунктам алгоритма

Второй пункт алгоритма нацелен на удаление существовавшего до этого форматирования. Программист или какая-либо другая программа могли использовать другие принципы форматирования кода, и если бы данного пункта не было, то возникла-бы путаница: отступы наложились друг на друга и результат отличался от ожидаемого.

Реализация третьего пункта зависит от конкретного языка программирования программу, на котором предстоит форматировать. Для многих языков программирования лексему можно определить как последовательность символов (или одиночный символ) отделенный от другой лексемы разделителем, в качестве которого могут выступать пробелы, символы табуляции и другие лексемы – спецсимволы, скобки т.д.

Пункты 5 и 6 могут быть изменены в зависимости от языка программирования. Это связано с тем, что некоторые из языков воспринимают регистр введенных лексем,

а некоторые его игнорируют. Если программа, которую нужно форматировать составлена на языке программирования не чувствительного к регистру, то перед проверкой первой лексемы на равенство маркерам начала и конца логического блока их необходимо преобразовать к одному регистру.

Пункт 5.3 предназначен для того, чтобы сразу не сдвигался маркер начала логического блока.

Пункт 6.1 нужен для того, чтобы отступ не получил отрицательного значения (не стал выступом). По сути, программа рассматривается как некий поток команд для стека. Итак, перед нами стек – это текущая величина отступа. Каждый маркер начала логического блока увеличивает его на величину шага отступа, а каждый маркер конца логического блока уменьшает его на ту же величину. Т.е. пункт 6.1. ответственен за защиту от опустошения стека: нельзя брать из стека больше, чем там имеется. В тоже время, в самой программе такие ошибки вполне возможны – к примеру, программист при наборе программы мог забыть поставить одну (или несколько) из открывающих операторных скобок...

Заключение

Данный алгоритм является только примером способов обработки исходных текстов программ. Как правило, в настоящих средах разработки подобные алгоритмы входят в состав более сложного алгоритма включающего в себя также функции по разделению строки на синтаксические элементы, автозамены, приведение к заданному регистру, удалению лишних пробелов и т.д. В тоже время он дает начальное представление о работе подобных систем в редакторах текста (не обязательно редакторах исходного текста программ).

Реализация рассмотренного алгоритма на Дельфи 7 прилагается в ресурсах к статье на <http://www.programmersforum.ru> в разделе «Журнал клуба программистов. Первый выпуск».

Применение изометрических координат в Дельфи

В данной статье рассмотрены методы применения изометрии на канве, позволяющие получить псевдо-эффект 3D на двумерной плоскости.

Владимир Дегтярь
by DeKot degvv@mail.ru

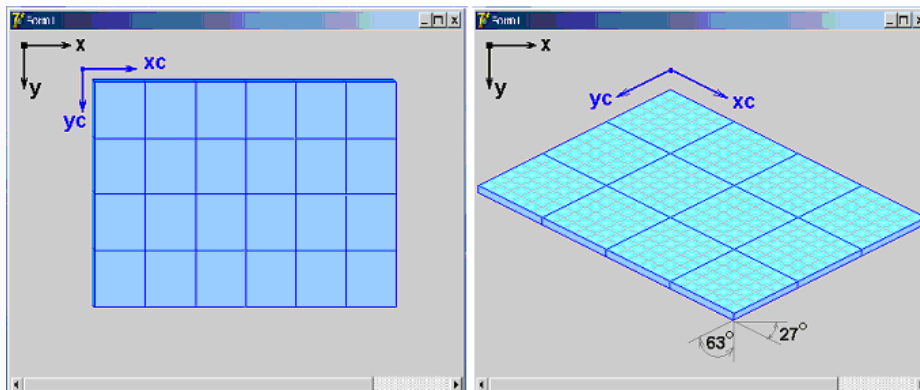


Рис. 1. Игровое поле в прямоугольных и изометрических координатах

При создании простых 2D игр (аркады, «стрелялки» и т.п.) обычно для построения игрового поля используется двумерная матрица с координатами, привязанными к координатам экрана (формы). При этом, направления координат игрового поля и экрана совпадают, и плоскость поля располагается как бы вертикально перед пользователем. Некоторую объемность изображения и эффект перспективы, правда, можно получить за счет манипулирования масштабом графических элементов. Но все это достигается путем значительного усложнения кода программы и требует сложных математических вычислений.

Построение изометрической матрицы

Значительно лучший визуальный эффект можно получить при применении изометрических координат для игрового поля. В этом случае поле для пользователя как бы наклонено под определенными углами по отношению к плоскости, а значит и координатам, экрана. При применении матрицы в изометрических координатах требуется привести координаты ячеек поля и индексы массивов, описывающих такую матрицу к прямоугольным координатам экрана. Для случая, когда матрица игрового поля выполнена в изометрических координатах, можно применить следующие функции для определения индексов массива по координатам формы или же координаты ячеек массива по индексам. Необходимые данные приведены на рисунке 2:

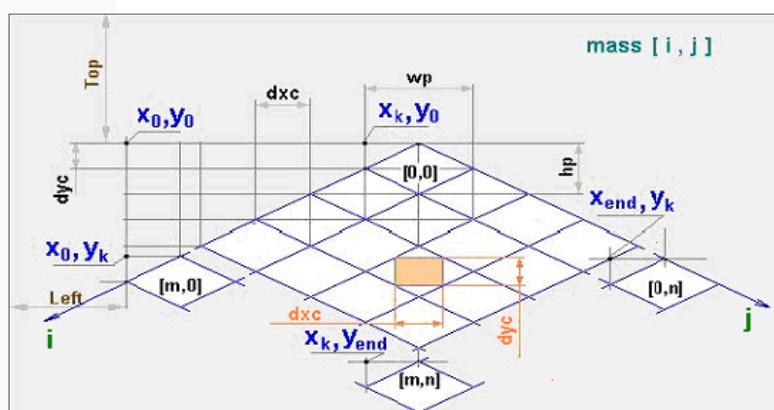


Рис. 2. Необходимые данные

0x10

Думаю понятно, что такая матрица описывается следующим массивом:

```
mas: array [ 0..m,0..n ] of < тип данных >,
где i - в диапазоне 0 .. m, j - в диапазоне 0..n;
Left, Top - смещение от края формы;
dxc, dyc - шаг координат ячеек матрицы;
dxc := 2 * dyc;
wp, hp - ширина и высота ячейки матрицы в координатах формы;
wp := 2 * dxc;    hp := 2 * dyc;
x0 = Left;        y0 = Top;
```

Для определения координат ячейки по индексам:

```
function Kord_X(i , j : byte) : integer;
begin
  Result:= ((m + ( j - i)) * dxc ) + Left
end ;
x := Kord_X(i , j);

function Kord_Y(i , j : byte) : integer;
begin
  Result := ((i + j) * dyc) + Top
end;
y := Kord_Y(i , j);
```

Для определения индексов по координатам:

```
function Ind_I (x , y : integer) : integer;
begin
  Result := ((m - ((x - Left) div dxc)) + ((y -Top) div dyc)) div 2
end;
i := Ind_I (x , y);

function Ind_J (x , y : integer) : integer;
begin
  Result := (((x -Left) div dxc) + (((y -Top) div dyc) - m)) div 2
end;
j := Ind_J (x , y);
```

При определении координат по курсору мыши, необходимо назначить область «чувствительности» курсора в пределах области, показанной на рисунке оранжевым цветом. Тогда координаты **x, y** ячейки матрицы по координатам курсора **xm, ym** определяются следующим образом:

```
procedure TForm1.FormMouseUp (Sender : TObject ; Button : TMouseButton ;
  Shift : TShiftState ; xm , ym : Integer) ;
begin
  x := (((xm - Left) + (dxc div 2)) div dxc) * dxc) + Left;
  y := (((ym - Top) + (dyc div 2)) div dyc) * dyc) + Top
end;
```

Графические объекты в изометрических координатах

Для удобства работы с графикой в изометрических координатах следует тщательно подходить к соотношениям размеров объектов и размерами ячеек матрицы. Так, углы расположения матрицы - 27° и 63° указаны не случайно. При работе с пиксельными изображениями форматов BMP, JPG, PNG и аналогичных этот наклон наиболее удобен для отображения различных элементов.

Для движущихся объектов, реализуемых в виде отдельных рисунков или спрайтов следует применять следующие пропорции в размерах:

```
Sprite.Width:= 1 / 3 * wp;  Sprite.Height:= 2 / 3 * hp;  
или  
Sprite.Width:= 2 / 3 * wp;  Sprite.Height:= hp;
```

где: **wp** и **hp** - соответственно ширина и высота ячейки матрицы (см. рис.3).

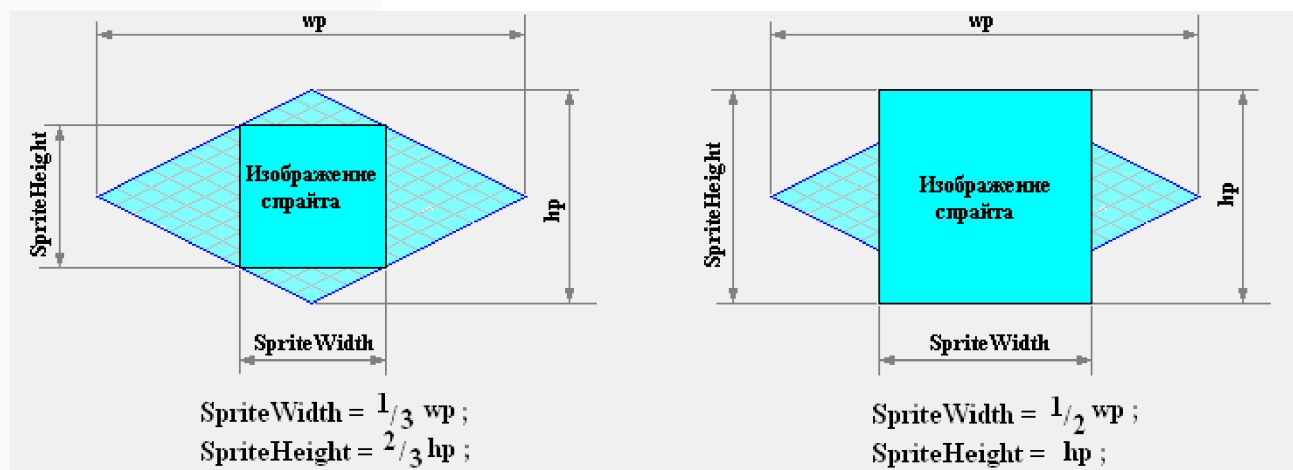


Рис. 3. Организация движения спрайта

При организации движения спрайта приращения по координатам dx и dy должны иметь соотношение 2:1 и соответствовать условию:

```
N_step = dxc / dx,  или N_step = dyc / dy;
```

где: **N_step** - количество приращений за один такт (шаг) в цикле или по таймеру; **dxc**, **dyc** - шаг координат ячеек изометрической матрицы. Приведем пример:

```
for i:= 1 to N_step do begin  
  Sprite(x,y) ; // процедура вывода спрайта на форму в координатах x , y;  
  x := x + dx ; y := y + dy ;  
end;
```

При выполнении такого условия координаты спрайта всегда после выполнения шага движения попадают в координаты следующей ячейки. При использовании обработчика нажатия клавиш «стрелки» приращения координат спрайта принимают следующие значения (см. рис.4):

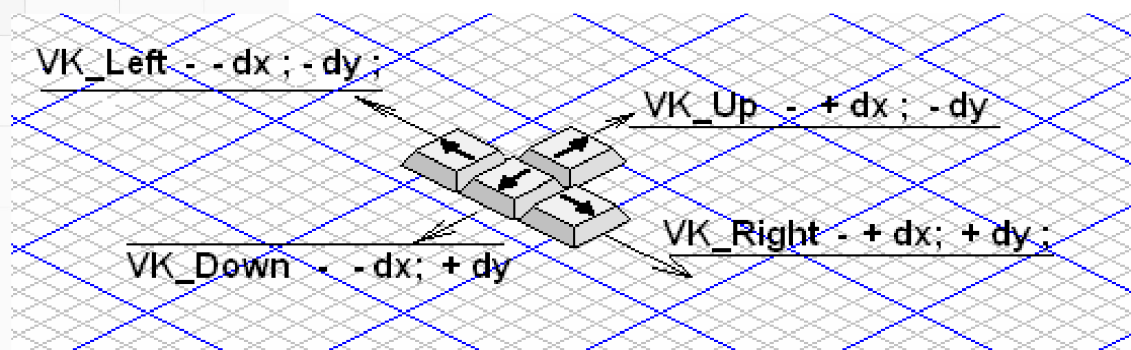


Рис. 4. Приращение координат спрайта

Многомерная матрица игрового поля в изометрических координатах

До сих пор мы рассматривали двухмерную изометрическую матрицу, расположенную в одной плоскости. Для получения реального трехмерного изображения можно применять многомерную матрицу в трех изометрических координатах (см. рис.5):

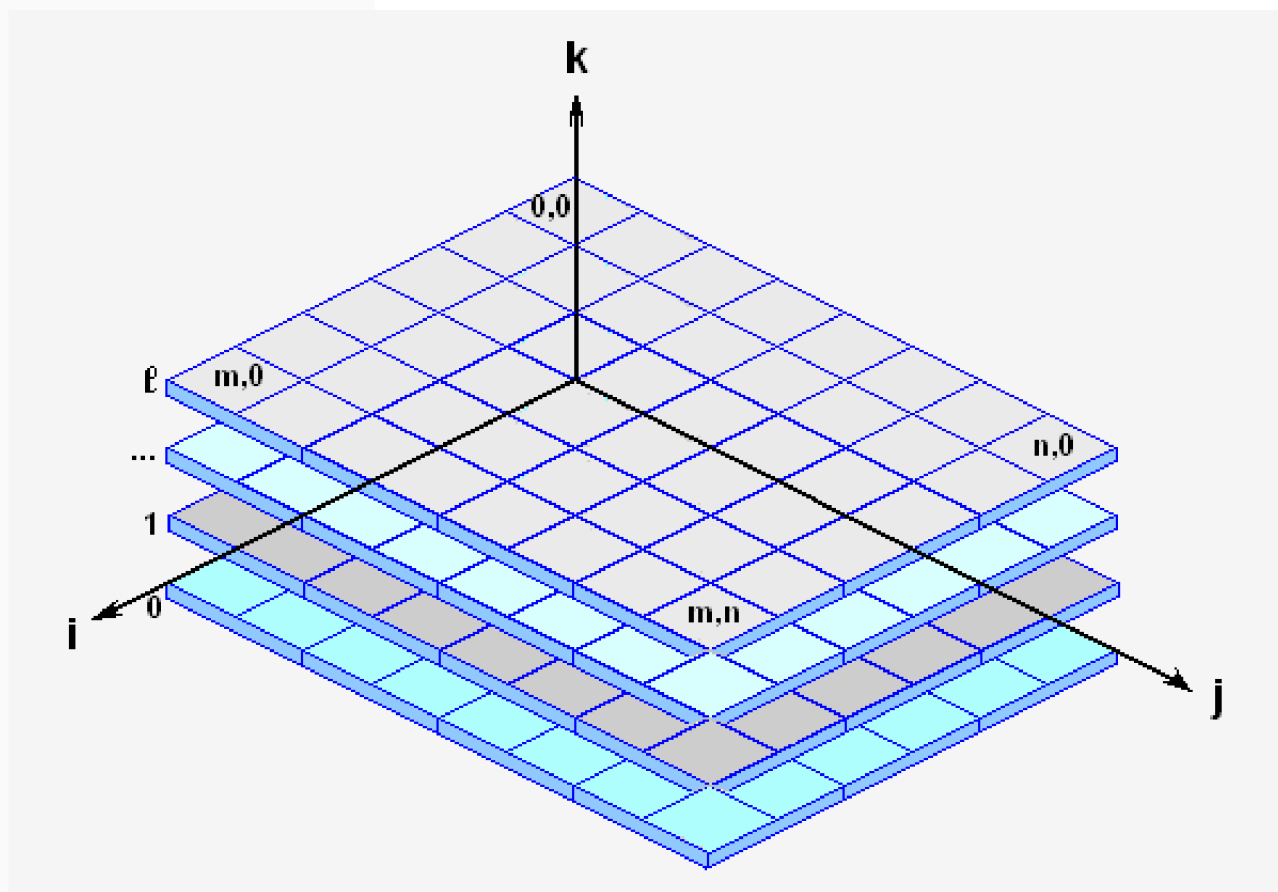


Рис. 5. Отображение многомерной матрицы в изометрических координатах

Такая матрица описывается следующим массивом:

```
mas_index : array [ 0 .. l , 0 .. m , 0 .. n ] of < тип данных > или...
mas_index [ k , i , j ] ;    здесь индекс k находится в диапазоне значений 0..l;
```

где: i - $0..m$; j - $0..n$. Работа с такой матрицей в пределах одного слоя аналогична описанию в начале нашей статьи в соответствии с рисунком 1. Однако при переходе с одного уровня на другой следует учитывать следующие особенности:

- каждый последующий слой в экранных прямоугольных координатах сдвигается на величину **dyc**
- при организации движения графических объектов приращения координат объекта в экранных координатах задаются одинаковыми **dx := dy** и выполняется условие **dx * N_step = dxc**

В этом случае, при переходах между уровнями (при использовании обработчика клавиш «стрелки») изменения индексов ячеек матрицы следующие (см. таблицу и рисунок 6):

Таблица. Изменение индексов ячеек матрицы

Начальные значения индексов	Следующие значения индексов	Приращения координат
VK_Left mas_index [k , i , j]	mas_index [k + 1 , i , j - 2]	- dx; - dy;
VK_Right mas_index [k , i , j]	mas_index [k - 1 , i , j + 2]	+ dx; + dy;
VK_Up mas_index [k , i , j]	mas_index [k + 1 , i - 2 , j]	+ dx; - dy;
VK_Down mas_index [k , i , j]	mas_index [k - 1 , i + 2 , j]	- dx; + dy;

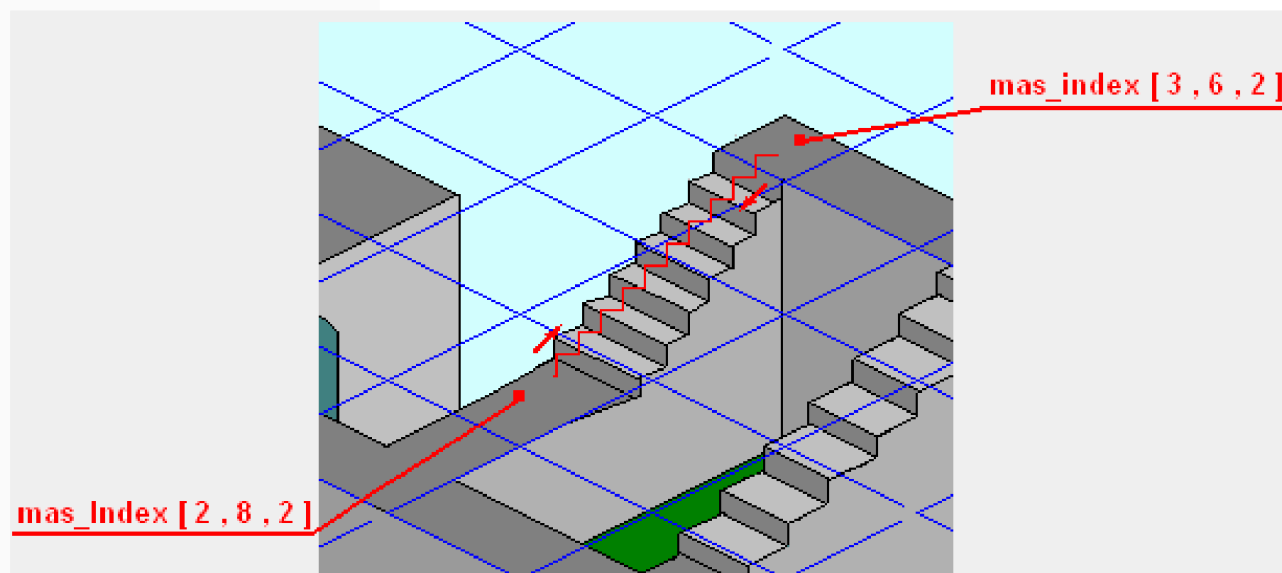


Рис. 6. Визуализация переходов

Далее, после перехода на следующий уровень обработка кода происходит как с двумерной матрицей с учетом новых индексов в массиве индексов.

Заключение

Пример применения многомерной изометрической матрицы (полный проект небольшой «бродилки») приведен в ресурсах к статье на <http://www.programmersforum.ru> в разделе «Журнал клуба программистов. Первый выпуск». В следующих уроках мы научимся работать с графикой на канве в среде Дельфи.

Как работать с графикой на канве в среде Дельфи. Урок 1-2

Понятия и методы работы с графикой в среде Дельфи для начинающих (полезно для создающих первые игры). Все сопровождается подробными примерами «космической стрельжки»...

Владимир Дегтярь
by DeKot degvv@mail.ru

Операционная среда WINDOWS является графической средой и для вывода графической информации на экран или принтер использует функции GDI (Graphics Devices Interface – Интерфейс графических устройств). GDI – функции являются аппаратно-независимыми, поэтому взаимодействие приложений (в том числе и созданных в среде Delphi) с аппаратными устройствами компьютера осуществляются через драйвера устройств и через специальную структуру данных – называемой контекстом отображения (дисплейный контекст – DC (Display Context)). Контекст отображения содержит основные характеристики устройств вывода графической информации, а также инструменты для рисования (шрифт, перо и кисть).

Графика в Delphi (немного теории). Урок 1

Система Delphi предлагает специальные классы, упрощающие использование графических средств:

- . TCanvas – для контекста отображения
- . TFont – для шрифта
- . TPen – для пера
- . TBrush – для кисти

Связанные с этими классами объекты получают соответствующие свойства – canvas, font, pen, brush, которые уже как объекты, в свою очередь, имеют ряд своих свойств.

Для работы с рисунками или изображениями Delphi предлагает классы: TGraphic, TPicture, TImage, TBitmap, TjpegImage, TShape, TIcon, TMetaFile и др. Основной класс для связанных с рисованием графических операций – это TCanvas. С помощью его свойств и методов можно рисовать на поверхности визуальных объектов, которые включают в себя этот класс и имеют свойство canvas. Для выполнения различных графических операций используются типы TPoint и TRect, описываемые следующим образом (см. листинг):

```
TPoint = record    // задание координат точки
  X : LongInt;
  Y : LongInt;
TRect = record    // определение прямоугольной области
  Left : Integer;
  Top : Integer;
  Right : Integer;
  Bottom : Integer;
```

или

```
...
TRect = record      // определение прямоугольной области
  TopLeft : TPoint;
  BottomRight : TPoint;
...
TRect := Bounds ( X,Y, Width, Height : Integer );
```

где: X, Y – координаты левого верхнего угла области; Width, Height – ширина и высота прямоугольной области.

Одним из основных объектов для рисования является – поверхность рисования (она же холст или канва) – объект класса TCanvas. У холста есть ряд свойств и методов для отображения графической информации, перемещения графических объектов по поверхности рисования, копирования изображений и/или их отдельных областей, вывода текстовой информации.

Наиболее частыми применяемыми методами отображения (вывода) графики являются:

- . Canvas < рисование геометрических фигур (примитивы) > Arc (дуга), Pie (сектор), Ellipse (эллипс, круг), Rectangle (прямоугольник) и другие
- . Canvas.Draw – вывод графической информации
- . Canvas.StretchDraw – вывод графической информации с изменением масштаба
- . Canvas.CopyRect – копирование графической области
- . Canvas.TextOut – вывод текстовой информации

Кроме этого еще используются методы загрузки графических объектов из файлов, из других компонентов или объектов.

Самое простое приложение с выводом графики. Урок 2

Создадим новый проект в среде Delphi «File => New => Application» (при запуске Delphi новый проект создается автоматически). Сразу же сохраним проект – File => SaveAll. На первый запрос – сохраняем модуль под именем предложенным Delphi – Unit 1. По второму запросу изменим имя проекта с Project1 на Lesson 1. Delphi в новом проекте создает объект Form1 и в редакторе кода модуля Unit1 появляется заготовка кода (см. рис.1):

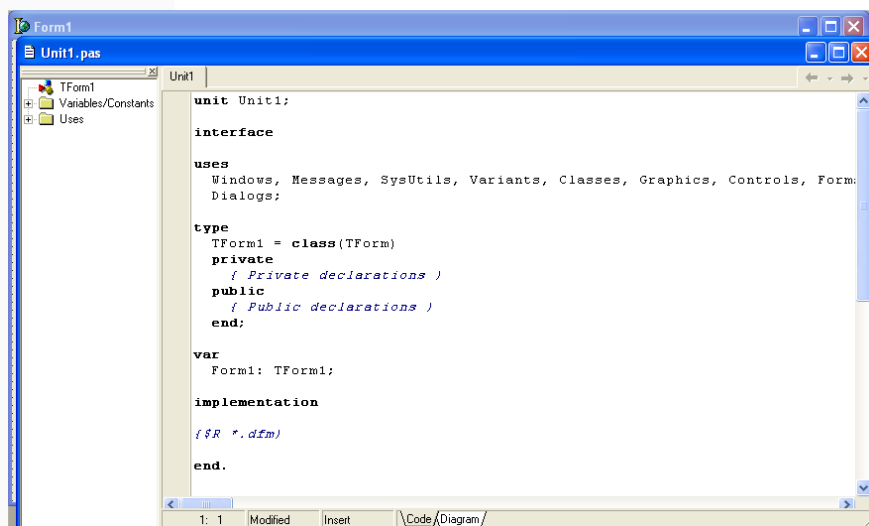


Рис. 1. Создаем новый проект

0x16

В Object Inspector в свойствах формы Form1 изменим заголовок Caption на «Урок по графике №1» и выставим размеры окна формы: Top (100*), Left (230), Width (700), Height (575), ClientWidth (700), ClientHeight (540).

* почему именно эти цифры разберем позже

Теперь с помощью методов графических примитивов нарисуем, что-либо на форме. В Object Inspector перейдем на вкладку Events (События) и «кликнем» дважды по событию OnPaint(). В редакторе кода появится шаблон процедуры обработчика события On Paint (см. рис.2):

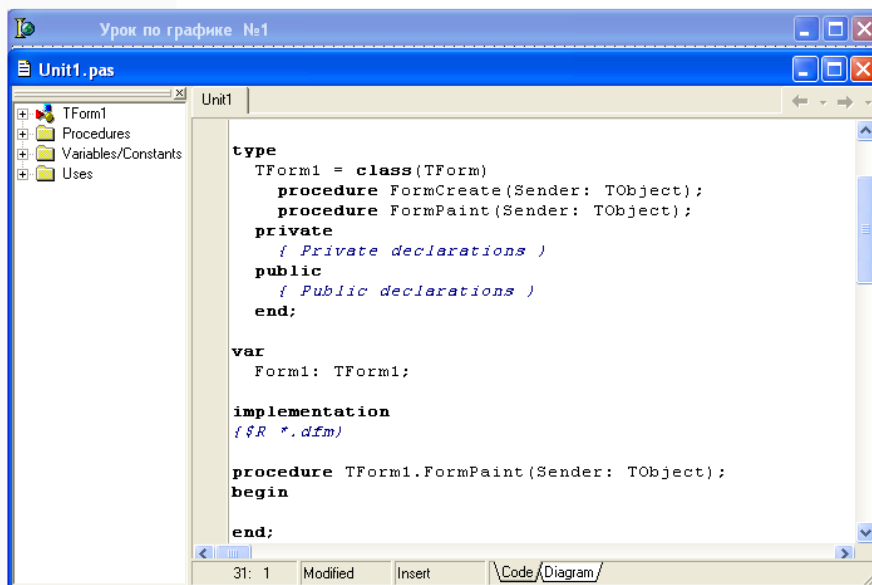



Рис. 2. Создание обработчика OnPaint()

Теперь запишем в этой процедуре следующий код (см. листинг):

```
with form1.canvas do begin
  pen.color:= clred;           // цвет пера
  rectangle(350, 50, 550, 100); // рисуем прямоугольник с координатами
                                // верхнего левого угла x1=350, y1=50 и
                                // правого нижнего x2=550, y2=100

  pen.color:= clgreen;
  pen.width:= 4;               // ширина пера
  brush.color:= clskyblue;     // цвет заполнения фигуры
  ellipse(60, 100, 250, 400)   // эллипс, вписанный в прямоугольник
end;
```

Запустите проект (Run или F9 или ) и посмотрите результат. Конечно, это слишком простой проект, поэтому усложним нашу задачу с использованием рисунков находящихся в файлах (формат файлов .bmp). Графические файлы, которые нам понадобятся для последующих проектов, находятся в папке <data> в соответствующих папках проектов (см. ресурсы к статье).

Далее, выведем на форму изображение звездолета (файл «ship1.bmp») на фоне звездного неба (файл «star1.bmp»). В файле «ship1.bmp» два изображения звездолета (спрайты – они нам понадобятся для организации движения звездолета), файл «star1.bmp»

используется для создания фона и имеет размер 700x540 (под эти размеры и установлены размеры окна формы через Object Inspector). Нам также понадобятся объекты типа TBitmap: BufFon (буфер для загрузки фона из файла «star1.bmp»), BufSpr (буфер для загрузки спрайтов из файла «ship1.bmp»), BufPic (буфер для загрузки рисунка одного из спрайтов из BufSpr), Buffer (общий буфер для объединения всех рисунков с последующим выводом на форму).

Размеры BufFon и BufSpr устанавливаются в соответствии с размерами изображений при загрузке. Размер общего буфера Buffer устанавливаем равным BufFon, а размер BufPic – равен размеру одного спрайта, что в общем случае определяется следующим образом:

```
BufPic.Width := round ( BufSpr.Width / n );  
BufPic.Height:= round ( BufSpr.Height / m );
```

где: n – кол-во спрайтов в горизонтальном ряду изображений в файле «sprite», m – кол-во рядов с изображением спрайтов в файле.

Инициализацию буферов проведем в процедуре OnCreate() формы (см. рис.3):

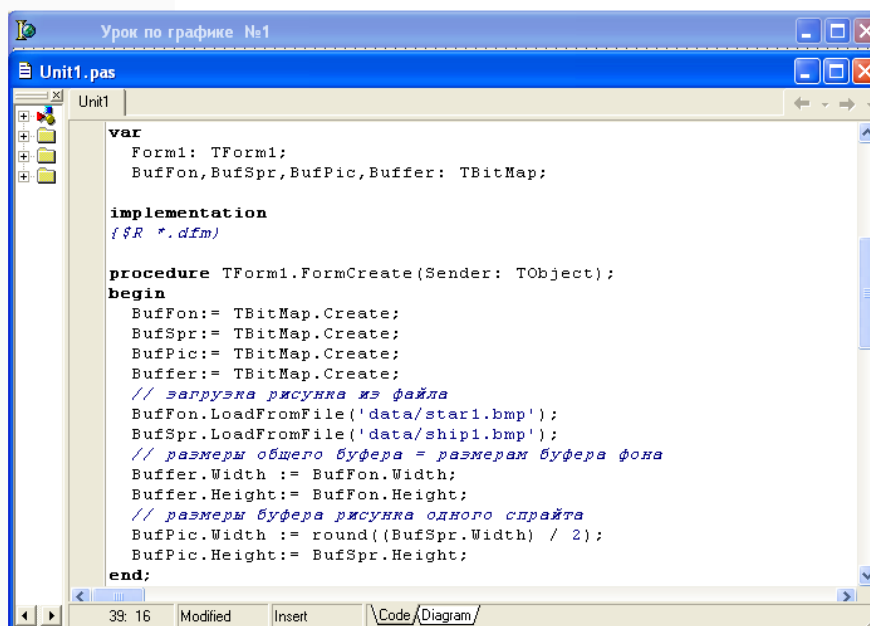


Рис. 3. Инициализация буферов

Для вывода одного спрайта через BufPic** создаем процедуру копирования спрайта из BufSpr в BufPic методом CopyRect (см. листинг):

```
procedure DrawShip1 ( i: byte); // загрузка одного спрайта в буфер рисунка  
begin  
  BufPic.Canvas.CopyRect(bounds(0, 0, BufPic.Width, BufPic.Height),  
    BufSpr.Canvas,bounds( i * 66, 0, BufPic.Width,  
    BufPic.Height));  
end;
```

Вывод изображений производим аналогично предыдущему примеру (рисование прямоугольника и эллипса) в процедуре OnPaint(). Необходимо ввести переменные xs1 и

ys1 (координаты вывода звездолета). Процедура DrawShip1(0) с параметром 0 выводит первый спрайт в буфер рисунка BufPic. Далее выводим фон и спрайт на канву дополнительного буфера Buffer и затем из него выводим все на форму. Удалите из процедуры код предыдущего примера и вставьте следующий (см. листинг):

```
// загрузка одного спрайта в буфер рисунка
procedure DrawShip1(i: byte);
begin
  BufPic.Canvas.CopyRect(bounds(0, 0, BufPic.Width, BufPic.Height),
    BufSpr.Canvas, bounds(i * 66, 0, BufPic.Width,
    BufPic.Height));
end;

procedure TForm1.FormPaint(sender: tobject);
var xs1, ys1: integer;           // координаты звездолета SHIP1
begin
  xs1:= 250; ys1:= 466;
  DrawShip1(0);
  Buffer.canvas.draw(0, 0, BufFon); // выводим фон в общий буфер
  Buffer.canvas.draw(xs1, ys1, BufPic); // выводим рисунок спрайта поверх
  // фона в общий
  Buffer.canvas.draw(0, 0, Buffer); // вывод обеих рисунков (общего буфера)
  // на форму
end;
```

После запуска проекта и компиляции получаем следующую картинку (см. рис.4):



Рис. 4. Тестовый проект звездолета

Заключение

Мы получили статическое изображение и теперь в последующих уроках создадим движущиеся графические объекты. Но для начала познакомимся с основными принципами получения «эффекта» движения объектов (папка Lesson1***).

Рассматриваемые в данной статье проекты полностью приведены в ресурсах к статье на <http://www.programmersforum.ru> в разделе «Журнал клуба программистов. Первый выпуск».

** на Canvas BufPic в область с координатами левого верхнего угла $X = 0$ и $Y = 0$, шириной и высотой соответствующие размерам буфера BufPic копируем изображение спрайта из области с координатами левого верхнего угла $X = i * 66$ и $Y = 0$, шириной и высотой соответствующие размерам буфера BufPic. В координате X цифра 66 соответствует ширине одного спрайта. В переменной i передается номер спрайта (0 - 1-й, 1 - 2-й).

*** перед запуском в среде Дельфи скопируйте в папку с проектом папку <data> с графическими файлами.

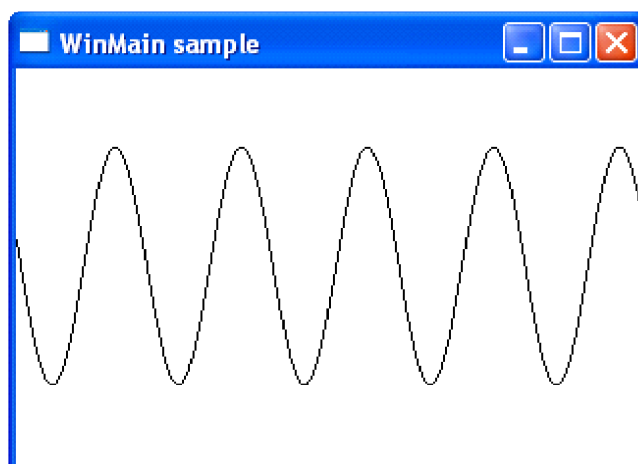
Простейшая программа WinAPI на C++

Многие, кто переходит с «учебного» ДОСовского компилятора, вроде Borland C++, на визуальное программирование быстро запутываются в сложных библиотеках типа MFC или VCL, особенно из-за того, что новые создаваемые проекты уже содержат с десяток файлов и сложную структуру классов. Рано или поздно встает вопрос: «...а почему нельзя написать оконную программу с простой линейной структурой, состоящую из одного файла .cpp?» На самом деле можно. Для этого нужно работать с низкоуровневыми функциями операционной системы - API.

Дмитрий Федорков
by ds.Dante www.programmersforum.ru

Windows API (application programming interfaces) - общее наименование целого набора базовых функций интерфейсов, программирования приложений операционных систем семейств Windows и Windows NT корпорации «Майкрософт». Является самым прямым способом взаимодействия приложений с Windows...

Википедия



Все что делает любая программа - делает либо непосредственно с помощью инструкций процессора, обращаясь к функциям биоса (хотя их прямые вызовы используются всё реже), либо через системные функции (API). К последним относятся: прорисовка окон, получение координат мыши, чтение файлов и т. д.

Зачем нам вообще API

WinAPI - это основа, в который должен разбираться любой программист, пишущий под Windows, независимо от того, использует-ли он библиотеки вроде MFC (Microsoft Visual C++) или VCL (Borland Delphi / C++ Builder). Часто бывает проще написать простенькую программу, состоящую из одного файла, чем настраивать относительно сложный проект, созданный визардами. Я не говорю уже, что программа получается гораздо оптимизированнее (всё-таки низкоуровневое программирование) и в несколько раз меньше. К тому же у них не возникает проблем совместимости, если у конечного пользователя не хватает каких-то библиотек, чем иногда грешит MFC.

Наша программа

Напишем простую программу: окно, в нем - синусоида, которая движется влево, как график функции $y = \sin(x + t)$. Если кликнуть мышкой по окну, анимация приостановится, или наоборот продолжится. Чтобы было проще разобраться, я сразу приведу весь исходный код, а потом прокомментирую ключевые места. Попробуйте самостоятельно модифицировать разные части программы, попробуйте оптимизировать мою программу, может вам даже удастся найти ошибки в коде (см. листинг):

```

#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <cmath>

LRESULT CALLBACK WindowProc (HWND, UINT, WPARAM, LPARAM);

HDC dc;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow)
{
    // Create window
    WNDCLASS wc      = {0};
    wc.style          = CS_OWNDC | CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc    = WindowProc;
    wc.hInstance      = hInstance;
    wc.hCursor        = LoadCursor (NULL, IDC_ARROW);
    wc.lpszClassName= L"CMyWnd";
    RegisterClass (&wc);
    HWND hWnd = CreateWindow (L"CMyWnd", L"WinMain sample",
                               WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, 0, 320, 240,
                               NULL, NULL, hInstance, NULL);

    dc = GetDC (hWnd);
    ShowWindow (hWnd, nCmdShow);

    // Message loop (timer, etc)
    SetTimer (hWnd, 1, USER_TIMER_MINIMUM, NULL);
    MSG msg;
    while (GetMessage(&msg,NULL,0,0) > 0) // while not WM_QUIT (0)
                                           // nor some error (-1)
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }

    return msg.wParam;
}

// Message processing function
LRESULT CALLBACK WindowProc (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    static bool Move = true;
    static int Phase=0, Width, Height;

    switch (message)
    {
    case WM_LBUTTONDOWN:
    case WM_RBUTTONDOWN:

```



```

        Move = !Move;
        // no break

    case WM_TIMER:
        if (Move)
            Phase++;
        // no break
        else
            break;

    case WM_PAINT:
        Rectangle (dc, -1, -1, Width+1, Height+1);
        MoveToEx (dc, 0, Height * (0.5 + 0.3*sin(0.1*Phase)), NULL);
        for (int i=0; i<Width; i++)
            LineTo (dc, i, Height * (0.5 + 0.3*sin(0.1*(i+Phase))) );
        break;

    case WM_SIZE:
        Width  = LOWORD(lParam),
        Height = HIWORD(lParam);
        break;

    case WM_KEYDOWN:
        if (wParam != VK_ESCAPE)
            break;
        // else no break

    case WM_DESTROY:
        PostQuitMessage (0);
    }

    return DefWindowProc (hWnd, message, wParam, lParam);
}

```

Обращаю ваше внимание на то, что эта программа писалась под Visual C++. У Билдера может быть проблема из-за заголовка `<cmath>`, вместо него нужен `<math.h>`. Для этой программы понадобится пустой проект с единственным файлом `.cpp`. В Visual Studio в свойствах создаваемого проекта нужно отметить галочку «Empty project». Итак, приступим...

Пройдемся по коду

В программе мы добавляем заголовочный файл **<cmath>**, который нужен для расчета синусоиды, и **<windows.h>**, который содержит все функции WinAPI. Строчка `#define WIN32_LEAN_AND_MEAN` отключает некоторые редко используемые функции и ускоряет компиляцию.

Функцию **WindowProc()** пока пропустим, оставив на сладкое.

HDC – контекст устройства рисования. Не будем подробно останавливаться на графике –

0x1D

это не основная тема статьи, да и используется не очень часто. Скажу лишь, что эта переменная глобальная, потому-что используется в обеих функциях. Надо добавить, что буква «H» в типе данных WinAPI (в «HDC») обычно означает «Handle» (хэндл), т.е. переменную, дающую доступ к самым разным устройствам, объектам и прочим сущностям WinAPI. Хэндл – представляет собой обычный указатель, работа с которым зависит от контекста (от типа переменной). Вообще, хэндлы – сложная тема, без которой тоже поначалу вполне можно обойтись.

Теперь главное (main) – точка входа. В консольных программах функция main может возвращать либо void, либо int, а также может иметь или не иметь аргументы (int argc, char **argv). Итого 4 варианта. В нашем случае используется функция **WinMain()**, которая может иметь только такой вид, как в примере. Слово **WINAPI** (которое подменяется препроцессором на `__stdcall`) означает, что аргументы функции передаются через стек (подробнее – в учебниках по ассемблеру). Аргумент **HINSTANCE hInstance** – хэндл текущего процесса, который бывает нужен в некоторых ситуациях. Назначение следующего аргумента **HINSTANCE hPrevInstance** весьма смутное, известно только, что эта переменная всегда равна NULL. В исходниках квейка можно даже найти такую строчку: `if (hPrevInstance != NULL) return 0.`

Аргумент **LPSTR lpCmdLine** – командная строка. В отличие от консольного main (int argc, char **argv), эта строка не разделена на отдельные аргументы и включает имя самой программы (что-нибудь типа “C:\WINDOWS\system32\format.com C: \u”). Далее int nCmdShow определяет параметры окна, указанные например, в свойствах ярлыка (это будет нужно при создании окна).

Перейдем, наконец, к выполняемому коду. В первую очередь нам нужно создать окно. Структура WNDCLASS хранит свойства окна, например текст заголовка и значок. 4-ре из 9-ти полей структуры должны быть нулевыми, поэтому сразу инициализируем ее нулями. Далее **CS_HREDRAW | CS_VREDRAW** означает, что окно будет перерисовываться при изменении размера окна. `wc.hInstance` задаёт текущий процесс (тут-то и понадобился этот аргумент из WinMain). Еще также нужно явно указать мышинный курсор, иначе, если это поле оставить нулевым, курсор не будет меняться, скажем, при переходе с границы окна на само окно (попробуйте сами). **wc.lpfnWndProc** – это адрес функции, которая будет обрабатывать все события. Такие как нажатие клавиши, движение мыши, перетаскивание окна и т. д. После знака «=» просто указываем имя нашей функции. Позже мы напишем эту функцию, которая и будет определять реакцию программы на интересующие нас события.

WNDCLASS – это не само окно, а класс (форма), экземпляр которого и будет нашим окном. Но перед созданием окна нужно зарегистрировать в системе этот класс. Задаем его имя в системе **CMyWnd** и регистрируем класс.

Функция создания окна `CreateWindow()` достаточно простая и вместо того, чтобы перечислять все ее аргументы, опять сошлюсь на интернет. Кому мало одиннадцати аргументов, могут попробовать функцию `CreateWindowEx()`. Обратите внимание, все строковые аргументы предваряются буквой L, что означает, что они – юникодовые. Для многих функций WinAPI существует по два варианта: обычный ANSI и юникодовый. Соответственно они имеют суффикс A или W, например `CreateWindowA` и `CreateWindowW`. Если вы посмотрите определение функции в `<windows.h>`, то увидите что-то типа `#define CreateWindow CreateWindowW`. Вместо `CreateWindow()` мы можем явно вызывать `CreateWindowA()` с обычными строками (без приставки L).

Описание **GetDC()** и **ShowWindow()** снова пропущу (кому интересно – тот легко найдет). Далее начинается самое интересное – работа с событиями. Для начала создадим таймер, который будет генерировать соответствующее событие 65 раз в секунду (фактически максимальная частота, по крайней мере для Windows XP). Если вместо последнего аргумента **SetTimer()** написать имя подходящей функции, она будет вызываться вместо генерации события.

Далее идет то, что называется message loop – цикл обработки событий. Мы принимаем событие и обрабатываем его. В нашем случае можно убрать **TranslateMessage(&msg)**, но эта функция понадобится, если на основе этого примера кто-нибудь будет создавать более сложную программу (с обработкой скан-кодов клавиатуры). Если мы получаем событие выхода программы, то **GetMessage()** возвращает ноль. В случае ошибки возвращается отрицательное значение. В обоих случаях выходим из цикла и возвращаем код выхода программы.

Теперь займемся функцией обработки событий **WindowProc()**, которую мы оставили на сладкое. Эта функция вызывается при любом событии. Какое именно сейчас у нас событие – определяется аргументом message. Дополнительные параметры (например, координаты мыши в событии «мышь двинулась») находятся в аргументах **wParam** и **lParam**. В зависимости от того, чему равно message, мы совершаем те или иные (или вообще никакие) действия, а потом в любом случае вызываем **DefWindowProc**, чтобы не блокировать естественные реакции окна на разные события.

Вообще то, что я сделал с оператором **switch** больше похоже на стиль ассемблера и порицается большинством серьезных разработчиков. Я имею в виду сквозные переходы между case-ми (там, где нет break). Но пример простой, к тому же у меня было настроение «похакерить», так что не буду лишать вас удовольствия разобраться в этом коде.

Имена констант message говорят сами за себя и уже знакомы тем, кто работал с MFC. При событии WM_PAINT рисуем белый прямоугольник, а на нём — чёрную синусоиду. На каждый WM_TIMER смещаем фазу синусоиды и перерисовываем ее. На клик мыши запускаем или останавливаем движение, при этом, если нажать обе кнопки одновременно, то фаза увеличится ровно на 1, для чего здесь и убран break (см. рисунок). При изменении размера окна мы обновляем переменные **Width** и **Height** за счёт того, что в **lParam** хранятся новые размеры. Всегда нужно вручную обрабатывать событие **WM_DESTROY**, иначе окно не будет закрываться. Наша программа закрывается также при нажатии клавиши <Escape>.

Где это еще может пригодиться

Вот и вся программа. Человек, имевший опыт написания под WinAPI сразу обращает внимание на характерные структуры – цикл обработки событий и выборку в функции WindowProc(), даже если эта программа написана на ассемблере. Вообще, знание принципов работы операционной системы иногда бывает очень полезным, особенно, при оптимизации программы. Не говоря уже о том, что знание WinAPI избавит вас от «изобретения велосипеда». Например, если программа должна кэшировать файлы. Также многие места в работе MFC и VCL станут гораздо понятнее. Тех, кто хочет изучить эту тему поподробнее, отошлю к классической литературе: Джеффри Рихтер «Создание эффективных WIN32 приложений с учетом специфики 64-разрядной версии Windows». Если непонятна какая-то функция – не гнушайтесь пользоваться Гуглом.

0x1F

Традиционно, измерительный прибор представляет собой автономное специализированное устройство, к которому подключаются анализируемые входные сигналы и на выходе которого имеется некий результат. Причем внутренняя архитектура остается неизменной, в отличие от виртуальных приборов. И если для изменения функциональности в первом, требуется существенная переработка схемы и конструкции, то для вторых достаточно изменить программу. Продолжая наш цикл по практике использования быстрого преобразования Фурье [1], сегодня мы с вами научим наш виртуальный прибор работать не только с низкочастотной частью диапазона и аудиоустройствами, но и с высокоскоростной платой сбора данных и для удобства принимать команды управления основными режимами программы с пульта... Данная статья рассчитана прежде всего в помощь программистам и инженерам-разработчикам в области цифровой обработки сигналов (DSP).

Сергей Бадло

by raхр www.programmersforum.ru

DSP – (Digital Signal Processing)
преобразование сигналов,
представленных в цифровой форме

Зоны Найквиста – зеркальные отображения спектра при частотах выше половины частоты дискретизации

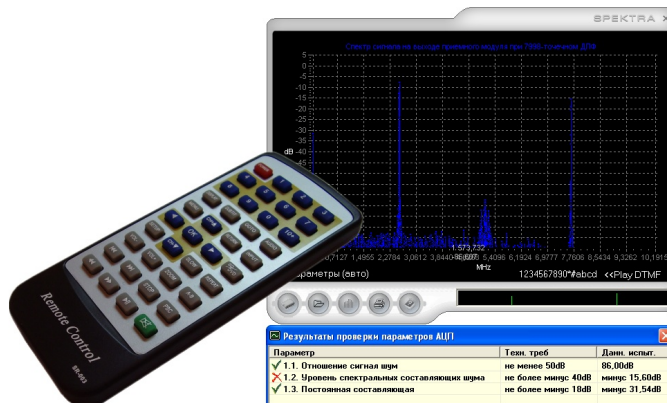


Рис. 1. Виртуальный прибор

Зачем-же все это нужно? Как мы уже знаем, виртуальные приборы (ВП) благодаря гибкости в их построении все больше вытесняют дорогостоящие автономные аппаратные решения, таких как осциллографы, спектроанализаторы и др. При этом пользователь не ограничен в выборе средств для анализа и обработки информации, что сводится лишь к изменению программного обеспечения. Приведем пример: вы купили «имиджевый» осциллограф, используете его до поры до времени... и вот настает момент, когда меняется задача и вам понадобился анализ спектра АЦП. Снова затраты? Рассмотрим подробнее...

Краткий экскурс...

В настоящее время основополагающим принципом цифровой обработки сигналов (ЦОС) является преобразование аналогового сигнала в цифровой на промежуточной частоте (перенос спектра). Это позволяет исключить такие недостатки аналогового способа формирования квадратурных сигналов, как: невысокие стабильность и нелинейность, неидентичность каналов, смещение фазы и трудности последующей фильтрации. Кроме того, это несколько снижает жесткие требования к элементной базе по частотным характеристикам. Но правило остается, чем выше быстродействие аппаратной логики, тем больший диапазон наблюдения можно охватить, не прибегая к различного рода

программным ухищрениям и ограничениям при выборе частот преобразования по зонам Найквиста, дискретности, тактовой частоты и даже питания и многое-многое другое.

Промышленные платы Hammerhead от Bittware с успехом справляются с этими условиями. Аппаратную часть виртуального оборудования подключают к промышленному компьютеру, как правило, через шины USB или PCI. Первый вариант не требует вскрытия компьютера, а второй дает обмен на порядок быстрее (спецификация USB3.0 увы пока редко встречается). Задача верхнего уровня сводится к окончательному анализу полученных данных с сочетанием сервисного удобства персонального компьютера (ПК). Кроме того, можно выделить лишь небольшое количество фирм, производящих комплекты для создания виртуального оборудования для работы с DSP в области ВЧ/СВЧ. Наиболее крупные из них – Analog Devices, Bittware, Kontron, National Instruments и Texas Instruments.

Аппаратная часть. Краткое описание объекта

Аппаратной основой (см. рис.2-3) виртуального прибора служит 8-ми слотовое шасси VD3U от компании Kontron [2] с промышленным контроллером CP-306, с установленными периферийными cPCI (Compact PCI) платами Hammerhead фирмы Bittware формата 3U (1U – принятая высота корпуса 44 мм) с четырьмя сигнальными процессорами SHARC ADSP-21160 от Analog Devices. Платы обеспечивают прием и обработку в реальном времени (единицы микросекунд) данных от 4-х каналов АЦП с частотой выборки 32 МГц каждый. Питание обеспечивает встроенный в шасси источник +3.3V/+5V/+12V/-12V. Обмен данными осуществляется по PCI шине. Программная оболочка виртуального прибора в комплексе с аппаратной служит для оценки работоспособности, уровня шумов, джиттера, динамического диапазона модулей АЦП.



Рис. 2. Тестовое шасси от Kontron

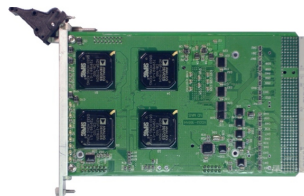


Рис. 3. Промышленная DSP плата Hammerhead

Предпосылки реализации ПО

Как же получить данные с DSP платы? Все достаточно просто. Для доступа к периферии производитель предоставляет набор драйверов и библиотеку <Hil.dll> или <Hil32v60.dll> в зависимости от версии, подробное описание API которой вы можете узнать в документации [2, 3], а полный список экспортируемых функций вы можете просмотреть в модуле <dsp.pas> (см. ресурсы к статье). Нам-же понадобятся следующие:

- | | |
|-----------------------|--|
| . dsp21k_open_by_id() | – массив указателей на область памяти каждого процессора |
| . dsp21k_reset_bd() | – реинициализация программы в процессоре |
| . dsp21k_load_exe() | – загрузка прошивки в процессор |
| . dsp21k_start() | – старт программы в процессоре |
| . dsp21k_dl_int() | – запись значения в область памяти процессора |
| . dsp21k_ul_int() | – считывание значения с области памяти процессора |
| . dsp21k_ul_32s() | – считывание значения с области памяти процессора (32 бит) |

В основу работы программы положен все тот-же алгоритм быстрого преобразования

Фурье (БПФ), применяемый к полученным отсчетам сигнала на нижнем уровне и переданными для обработки наверх.

Таким образом, уже можем определить основные требования к нашему виртуальному прибору:

- . возможность загрузки прошивки в сигнальные процессоры платы Hammerhead
- . визуализация первичных отсчетов, полученных промышленной платой с модуля АЦП
- . построение спектра (БПФ) в реальном времени и с возможностью сохранения дампов
- . возможность распечатки отображаемых (сохраненных) данных
- . управление основными режимами с пульта ДУ

Практика. Разработка ПО и средства отладки

Итак, приступим к основной задаче. Для работы нам понадобится следующее:

- . IDE среда разработки Borland Delphi 5-7 (для разработки ПО верхнего уровня)
- . промышленная плата Hammerhead и модулем АЦП
- . генератор сигналов типа Г4-301 или любой другой до 100 МГц
- . USB приемник дистанционного управления из материала [4] (см. рис.4)



Рис. 4. USB-IR приемник

- . любой ИК пульт дистанционного управления для тестирования

Работа с DSP платой

Внешний вид программы реализован через скин-систему, более подробно (см. исходники). Рассмотрим основные ключевые моменты по доступу к данным... Прежде всего, необходимо подключить и проинициализировать библиотеку для работы с драйвером (см. листинг 1):

получаем доступ к драйверу

ЛИСТИНГ-1

```
...
function LoadMyDll: bool;
var HLib: THandle;
    NewLib: Boolean;
begin
    result:= false;

    HLib:= LoadLibrary('Hil.dll'); // осуществляем динамическое подключение-
    NewLib:= (HLib<>0);
    if not NewLib then HLib:= LoadLibrary('Hil32v60.dll');
    if HLib = 0 then exit;
    dsp21k_open_by_id      := GetProcAddress(HLib, 'dsp21k_open_by_id');
```

```

dsp21k_close      := GetProcAddress(HLib, 'dsp21k_close');

if NewLib then begin
  dsp21k_load_exe  := GetProcAddress(HLib, 'dsp21k_load_exe');
  dsp21k_dl_exe    := dsp21k_load_exe;
end else begin
  dsp21k_load_exe  := GetProcAddress(HLib, 'dsp21k_dl_exe');
  dsp21k_dl_exe    := dsp21k_load_exe;
end;

dsp21k_start      := GetProcAddress(HLib, 'dsp21k_start');
dsp21k_reset_bd   := GetProcAddress(HLib, 'dsp21k_reset_bd');
dsp21k_ul_32s     := GetProcAddress(HLib, 'dsp21k_ul_32s');
dsp21k_dl_int     := GetProcAddress(HLib, 'dsp21k_dl_int');
dsp21k_ul_int     := GetProcAddress(HLib, 'dsp21k_ul_int');
dsp21k_proc_running := GetProcAddress(HLib, 'dsp21k_proc_running');

if @dsp21k_open_by_id = nil then exit; // обрабатываем исключения-
if @dsp21k_close = nil then exit;
if @dsp21k_dl_exe = nil then exit;
if @dsp21k_start = nil then exit;
if @dsp21k_reset_bd = nil then exit;
if @dsp21k_dl_32s = nil then exit;
if @dsp21k_dl_int = nil then exit;
if @dsp21k_loaded_file = nil then exit;
if @dsp21k_ul_int = nil then exit;

result:= true
end;

```

После чего, заведем массивы для хранения отсчетов комплексных и квадратурных составляющих сигнала `dim[]`, `qcos[]`, `qsin[]` и расчетных значений спектра `bpf[]`. И инициализируем процессорную плату (см. листинг 2):

инициализируем процессорную плату

ЛИСТИНГ–2

```

...
Dim    : array [0..10000] of real;
qcos   : array [0..10000] of extended;
qsin   : array [0..10000] of extended;
bpf    : array [0..20000] of extended;

procedure TForm1.init_adc;
var i,j: integer;
    p : pointer;
begin
  adr_en_data:= $50000; // адрес флага разрешения передачи для нижнего уровня
  adr_val_data := $52713; // адрес области памяти с данными
  cnt_data     := 1000; // кол-во отсчетов

```

```

id_mod      := 0;           // 0 – слот для платы сбора данных
id_proc     := 1;           // 1 – процессор на плате

try
  SetLength(ArrData, cnt_data);    // задаем массив для отсчетов

  // =====
  // так как мы заранее не знаем, в каком слоте и в каком процессоре загружена
  // прошивка, то “пробегаемся” по всем слотам и процессорам и считываем
  // возвращаемый указатель (хэндл)
  // в двумерный массив–матрицу indata[i,j], после чего осуществляем сброс,
  // загрузку прошивки и старт процессора на плате Hammerhead
  // =====
  for i:=0 to 63 do //
  for j:=1 to 4 do begin
    p:= dsp21k_open_by_id(i, j);
    if p<>nil then indata[i,j].PHandle:= p // просто двумерный
  end;
  dsp21k_reset_bd(indata[id_mod, id_proc].PHandle); // сброс прошивки
  dsp21k_load_exe(indata[id_mod, id_proc].PHandle, 'data\21160.dxe');
  dsp21k_start(indata[id_mod, id_proc].PHandle); // старт прошивки
  sleep(10) // делаем на всяк пожарный задержку–
except show_tn(2, 'Ошибка доступа. Нет модуля или связи...','SPEKTRA');
  gl_adc:= false end
end;

```

Получение квадратур сигнала позволяет судить о фазовых неравномерностях в каналах и позволяет производить более тонкую подстройку модулей АЦП. Теория их формирования выходит за рамки данного материала и вряд-ли заинтересует читателя, поэтому приведем лишь сам код (см. листинг 3):

вычисление квадратур сигнала

ЛИСТИНГ–3

```

...
procedure quadr(auto:boolean;m:integer;var qcos,qsin:array of extended;
               var dim:array of real; var quad:integer);
var k1, k2, k3, k4, k5, k6, k7, k8, kcos, ksin,
    a, aa, b, bb, c, cc, d, dd, f, vcos, vsin: real; i: integer;
begin
  k1:= 0.1169777; // коэффициенты окна–
  k2:= 0.4131758;
  k3:= 0.7499999;
  k4:= 0.9698463;
  k5:= 0.9698463;
  k6:= 0.7499999;
  k7:= 0.4131758;
  k8:= 0.1169777;

  kcos:= 4*(k1 – k3 + k5 – k7 – k2 + k4 – k6 + k8);
  ksin:= 4*(k1 – k3 + k5 – k7 + k2 – k4 + k6 – k8);

```

```

randomize;

if auto then // формирование псевдопоследовательности
  for i:=0 to m do dim[i]:= trunc((512 * sin((sink * i * pi/2) + pi/4) +
    random(4)) / 4);

m:= ceil(m/32);           // отсчет = 32 бита
quad:= m;                 // кол-во отсчетов
for i:=1 to m do begin
  a:= (dim[i*32+1]        + dim[i*32+8] + dim[i*32+17] + dim[i*32+25])*k1;
  b:= (dim[i*32+1+1]      + dim[i*32+8+1] + dim[i*32+17+1] + dim[i*32+25+1])*k2;
  c:= (dim[i*32+1+2]      + dim[i*32+8+2] + dim[i*32+17+2] + dim[i*32+25+2])*k3;
  d:= (dim[i*32+1+3]      + dim[i*32+8+3] + dim[i*32+17+3] + dim[i*32+25+3])*k4;

  aa:= (dim[i*32+1+4]     + dim[i*32+8+4] + dim[i*32+17+4] + dim[i*32+25+4])*k1;
  bb:= (dim[i*32+1+1+4]   + dim[i*32+8+1+4] + dim[i*32+17+1+4] +
    dim[i*32+25+1+4])*k5;
  cc:= (dim[i*32+1+2+4]   + dim[i*32+8+2+4] + dim[i*32+17+2+4] +
    dim[i*32+25+2+4])*k6;
  dd:= (dim[i*32+1+3+4]   + dim[i*32+8+3+4] + dim[i*32+17+3+4] +
    dim[i*32+25+3+4])*k7;
  f      := a - c + aa - cc;
  vcos   := f - b + d - bb + dd;
  vsin   := f + b - d + bb - dd;
  qcos[i]:= vcos + kcos * 10;      // Re – действительная часть
  qsin[i]:= vsin + ksin * 10;      // Im – мнимая часть
end
end;

```

И собственно то, ради чего все задумывалось, выборка данных с нижнего уровня, передача отсчетов в уже знакомую нам процедуру БПФ, поиск максимума и определение среднего уровня шума в полученном спектре (см. листинг 4):

флаг разрешения на нижний уровень и синхронное чтение данных

ЛИСТИНГ–4

```

...
procedure TForm1.load_adc;
var a : array [1..20000] of double;
    b : array [1..20000] of double;
    st: array [0..10000] of real;
    exp, mant, lab: string;
    i, nf, nn, ns, n: integer;
    signoise, re, im, l, m, druc, dmax: extended;
begin
  series2.Clear; series3.Clear; inwav1.Clear; // очищаем при каждой выборке
  dsp21k_dl_int(indata[id_mod, id_proc].PHandle, adr_en_data, 1); //set bit
  while dsp21k_ul_int(indata[id_mod, id_proc].PHandle, adr_en_data)<>0 do ;

  // передаем на нижний уровень указатель на массив ArrData для заполнения–

```

```

dsp21k_ul_32s(indata[id_mod, id_proc].PHandle, adr_val_data, cnt_data,
               @ArrData[0]);

for i:= 0 to length(ArrData) - 1 do begin
    inwav1.add(ArrData[i]);
    dim[i]:= ArrData[i]; // заносим значения отсчетов в массив dim[] для
                        // расчета квадратур
end;

if not gl_viborka then begin // выбор режима визуализации – отсчеты / спектр
    series2.Assign(inwav1);
    series2.SeriesColor:= clred // fix – цвет серии пропадает
end else begin
    // _____
    quadr(false, 999, qcos, qsin, dim, quad);

    signoise:= -1000; // fix – задаем минимальный уровень шума
    nf:= 999;
    ch.Title.Text.Text:= 'Спектр сигнала на выходе приемного модуля при ' +
                        inttostr(nf-1) + '-точечном БПФ';
    nn:= ceil(20000000/nf);
    for i:=0 to nf do begin
        a[i]:= dim[i-1];
        b[i]:= 0
    end;
    re:=0; im:=0;

    // =====
    // получение спектра
    // =====
    fft(a, b, nf, 4, 1);

    for i:=1 to nf-1 do begin
        st[i]:=sqrt(a[i]*a[i]+b[i]*b[i]); // получаем модуль
        if st[i]=0 then st[i]:= 1e-100;
        bpf[i]:= 20 * log10(st[i] / nf) // переводим в дБ
    end;

    // =====
    // для нормирования по уровню необходимо определить максимум, для этого
    // сканируем отсчеты и определяем уровень больший заданного signoise = -1000
    // =====
    signoise:=-1000;
    for i:= 10 to nf div 2-5 do begin
        if bpf[i] > signoise then begin
            signoise:= bpf[i];
            k:= i
        end
    end;
end;

```



```
// =====
// поиск среднего шума
// =====
nn:= 0;
re := 0;
druc:= -1000;
for i:= 10 to nf div 2-5 do begin
  if i < k-10 then begin
    re:= re + bpf[i] * bpf[i];
    inc(nn);
    if druc < bpf[i] then druc:= bpf[i]
  end;
  if i > k+10 then begin
    re:= re + bpf[i] * bpf[i];
    inc(nn);
    if druc < bpf[i] then druc:= bpf[i]
  end
end;
re:= -sqrt(re/nn);
// =====
// отрисовываем
// =====
for i:=0 to ceil(nf/2-1) do series3.Addy(re, '', clred);
end;
```

Скомпилировав проект, запустим его на выполнение. Подключив генератор сигналов на вход приемного модуля, подадим тестовый синусоидальный сигнал. В результате уже можем наблюдать отсчеты и при необходимости сами квадратуры (чередующиеся мнимые и действительные составляющие) сигнала (см. рис.5 и 6):

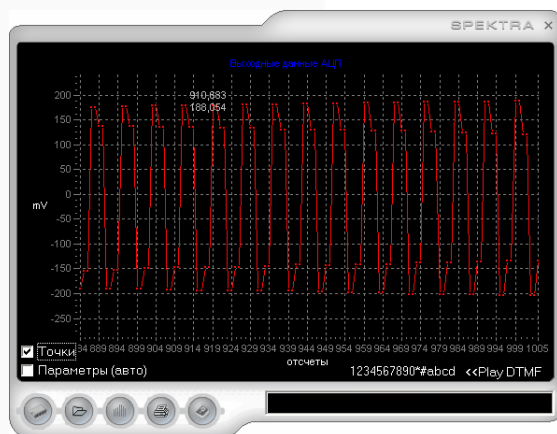


Рис. 5. Отсчеты с АЦП. REAL-TIME

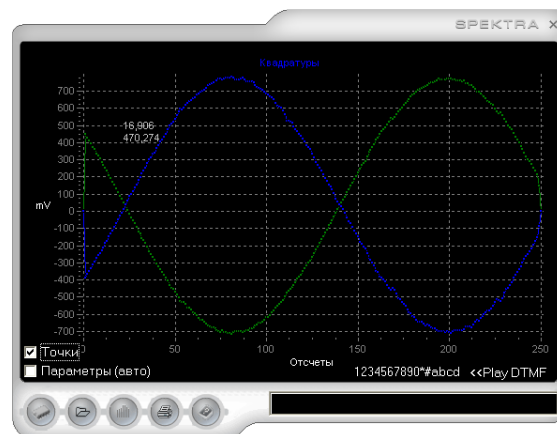


Рис. 6. Квадратуры сигнала с АЦП

Работа с пультом ДУ

Несмотря на то, что данная «фича» не является основной функцией в такого рода программах, а в некоторых случаях и вредной :), но обойти ее стороной никак не могу. Что для этого нужно? Да всего ничего, либо собрать приемник на COM порт и управлять через WinLirc, либо использовать USB.IR приемник, что даст гораздо более стабильные результаты. Было реализовано оба варианта.

Чтобы не увеличивать код, сигнатуры нескольких кнопок с ИК пульта были сняты заранее и введены в виде констант. Сама конструкция USB.IR приемника, программа и алгоритм декодирования пакетов были подробно рассмотрены в статье [4], поэтому тут заострять внимание на них не будем и перейдем сразу к коду (см. листинг 5):

управление режимами с пульта через приемник USB

ЛИСТИНГ–5

```
...
procedure TForm1.ic(i: integer);
begin
  case i of // нажатия на кнопки
    1: image1.OnClick(nil);      // чтение с текущего устройства–
    2: image2.OnClick(nil);      // загрузить файл WAV/MP3 или потоковые данные–
    3: image3.OnClick(nil);      // переключение режимов "спектр / отсчеты"–
    4: image4.OnClick(nil);      // вывод на печать–
    5: image5.OnClick(nil);      // информация о программе–
    6: image6.OnClick(nil)      // закрыть программу–
  end
end;

// =====
// сигнатуры кнопок 1, 2, 3, 4, 5, 6 пульта SR-003
// =====
var remote: array[0..5] of string = (
  '12-08-13-07-07-07-13-08-12-08-06-07-06-07-06-08-09-05-07-14-06-14-07-12',
  '13-07-14-07-13-08-05-08-13-07-06-08-13-07-06-07-13-08-06-07-06-07-14-07-06-
    07-13-08-06-07-12',
  '13-08-13-07-13-08-13-07-13-07-07-07-06-07-06-08-05-08-06-07-06-07-06-08-06-
    07-13-08-12-08-11',
  '13-08-12-08-13-07-13-08-13-08-06-07-13-08-05-08-06-07-06-07-06-07-06-08-05-
    08-13-07-06-08-14',
  '13-08-12-07-06-07-06-07-14-08-05-08-05-08-05-08-13-07-06-08-13-07-13-08-06-
    08-12-07-14-07-12',
  '14-07-13-07-06-07-06-08-06-07-13-08-13-07-06-08-05-08-06-07-14-07-13-07-13-
    08-06-07-06-07-12');

procedure TForm1.tuTimer(Sender: TObject);
var i: integer; s: string;
    p: boolean;
    j,k,z: smallint;
begin
  // =====
  // получение кода кнопки с USB приемника (сигнатуры)
  // =====
  if (GetInfraCode('ra_usb') = 1) then Exit;
  if DataLength = 0 then Exit;
  for i:= 0 to DataLength-1 do begin
    s:= s + inttohex(InputInfraData[i], 2);
    if i<> DataLength-1 then s:= s + '-'
  end;
end;
```

```
// =====
// проверка сигнатуры с учетом интервала доверия и передача на
// интерпретатор команд
// =====
for i:=1 to 6 do
  for z:= 0 to DataLength-1 do
    if (strtoint('$'+copy(s,(z*3)+1,2))-dover >=
      strtoint('$'+copy(remote[i],(z*3)+1,2)))or
      (strtoint('$'+copy(remote[i],(z*3)+1,2)) <=
      strtoint('$'+copy(s,(z*3)+1,2))+dover) then begin
      p:= true;
      k:= z
    end else begin p:= false; break end;
  if p then ic(k)
end;
```

Осталось проверить работоспособность управления по ИК. Для этого, подключив USB.IR приемник, нажимаем кнопку «2» на пульте, отвечающую за смену режима отображения «отсчеты / спектр» (см. рис.7 и 8):

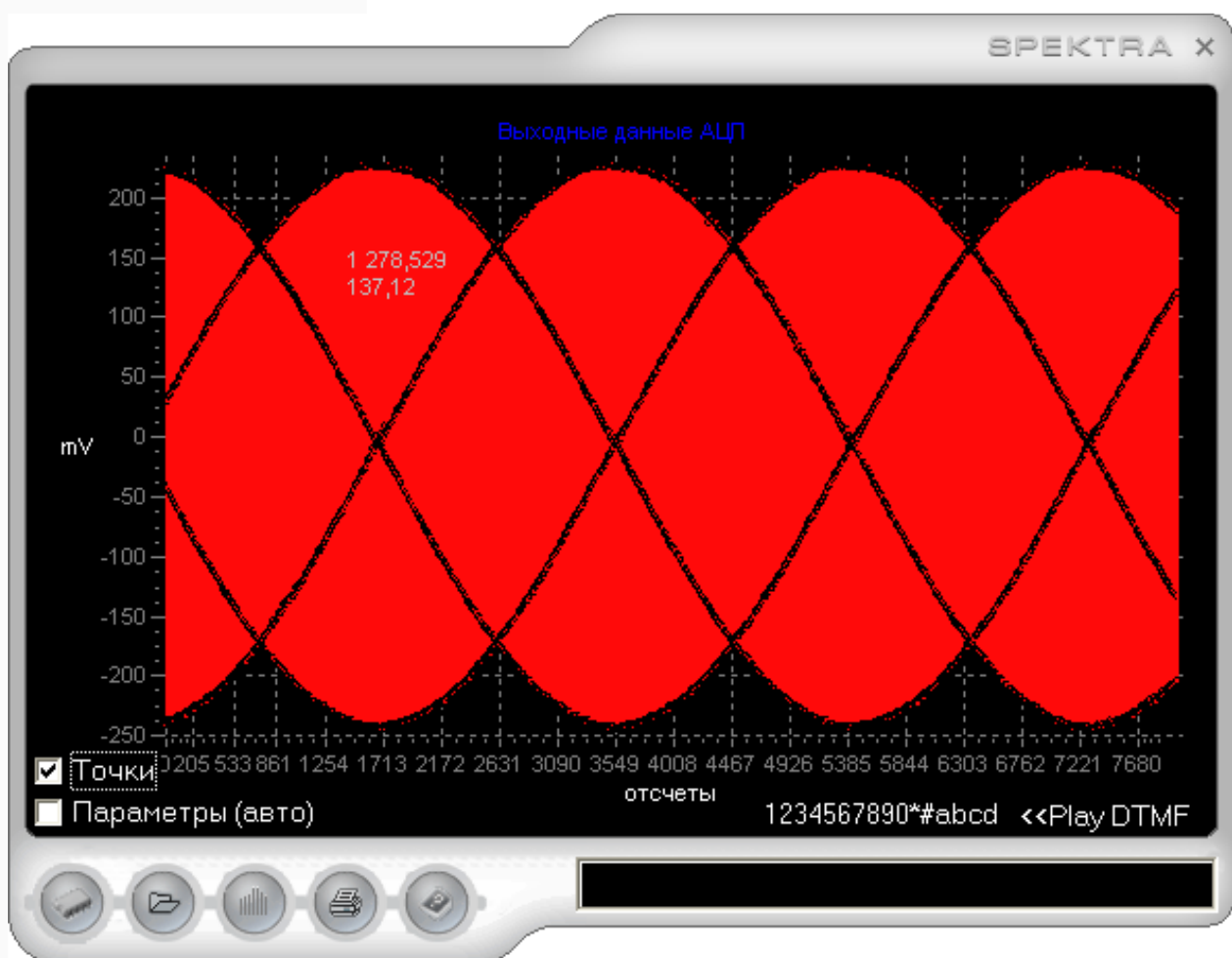


Рис. 7. Отсчеты сигнала с АЦП. REAL-TIME. Переключение режимов с пульта

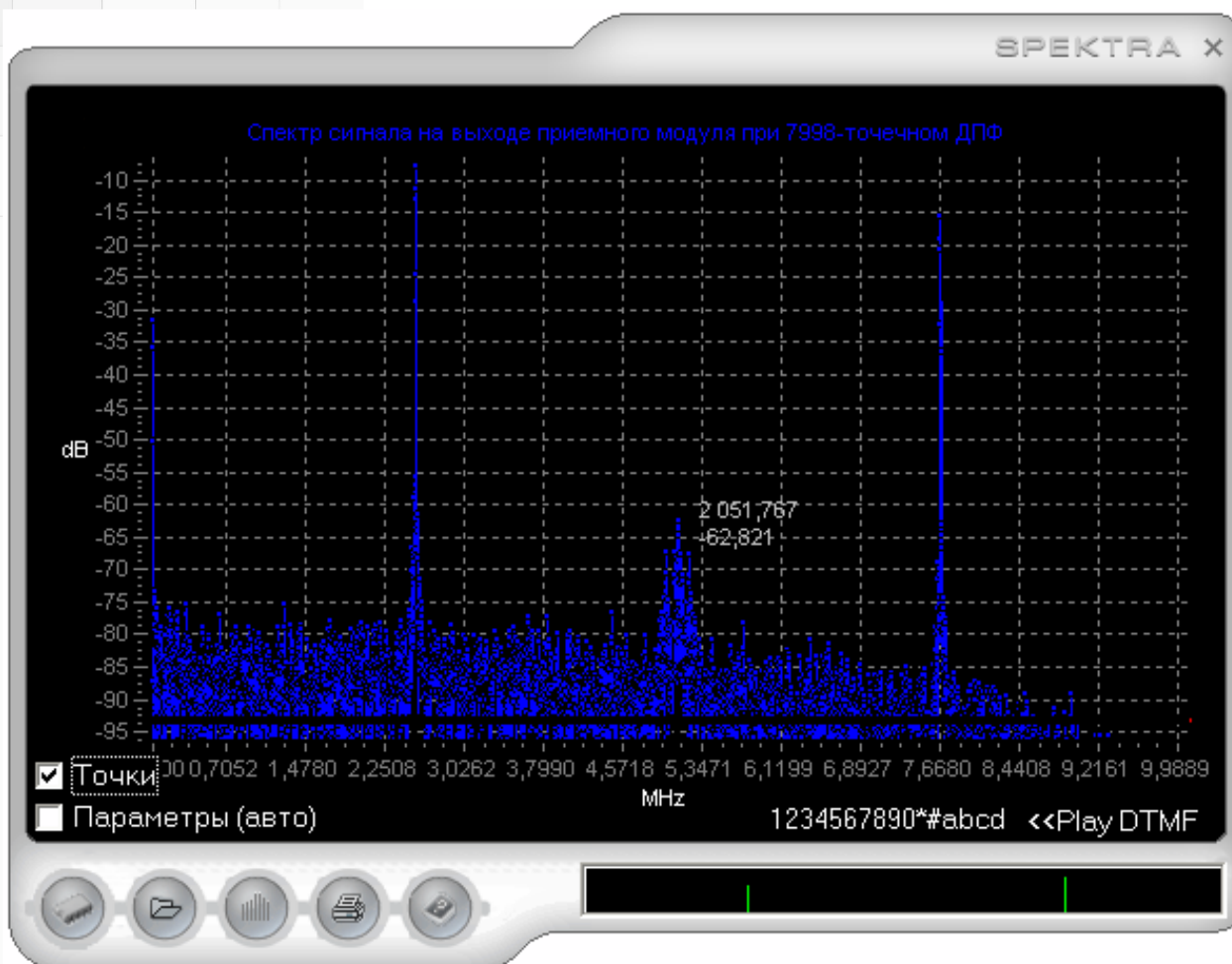


Рис. 8. Спектр сигнала с АЦП. REAL-TIME. Переключение режимов с пульта

Заключение

Рассмотренный виртуальный прибор позволяет сэкономить время и снизить стоимость любой системы сбора и анализа данных с модулей АЦП без привлечения дорогой специализированной аппаратуры, за счет применения гибкого ПО в сочетании с производительностью DSP.

Полные исходные тексты и компиляцию виртуального спектроанализатора SPEKTRA (файл [fft2.zip](#)) вы можете загрузить на форуме клуба программистов (раздел «Журнал клуба программистов. Первый выпуск»). Если тема представляет для вас интерес – пишите, задавайте вопросы на форуме <http://www.programmersforum.ru>

Ресурсы

- . С.Бадло. Быстрое преобразование Фурье. Практика использования. – Блог клуба программистов, 05.02.2010 <http://pblog.ru/?p=658>
- . Data Sheet Kontron Modular Computers GmbH, 2003, ID26799, rev.01
- . Practical Design Techniques for Sensor Signal Conditioning, Analog Devices, 1998
- . Е.Бадло, С.Бадло. USB термометр и дистанционка в одном флаконе. Часть 2. – Радиолобитель, 2010, №1, с.48 <http://raxp.radioliga.com/cnt/s.php?p=us2.pdf>
- . Ресурсы и компиляция проекта SPEKTRA <http://raxp.radioliga.com/cnt/s.php?p=fft2.zip>

Игра Fortress. Конкурс на создание лучшего бота. Итоги

Здравствуйте читатели [блога программистов](#). 18 января на форуме программистов стартовал конкурс на создание лучшего бота для игры в Fortress...

Аблязов Руслан

by гpy3uH <http://pblog.ru/?author=8>

Более подробно правила описаны в этой теме [1], а обсуждение конкурса приведено тут [2]. Также там есть вся необходимая информация для создания бота. Бот представляет собой библиотеку (DLL) с тремя экспортируемыми функциями.

Краткий экскурс...

Игра рассчитана на двух игроков. У каждого есть база. У базы есть щит. Есть также три типа ресурсов: энергия (En), металл (Me) и электроэлементы (El). Есть набор проектов, которые игроки могут реализовывать. Они бывают разных типов: атака чужой базы, ремонт своей базы, развитие своей базы и т.д. Всего проектов 30, перед началом игры игрок должен выбрать только 15 из них на своё усмотрение (согласно выбранной стратегии). Каждый проект стоит некоторое количество ресурсов. Игроки делают ходы (выбирают проекты) по очереди. Если у игрока не хватает ресурсов ни на один проект, он пропускает ход. Задача игрока уничтожить базу противника (уменьшить броню базы до нуля).

Конкурсная горячка. Чего добились

На создание бота участникам было отведено более двух месяцев. В итоге, было, написано пять ботов:

- . Бот «Разрушитель-1» от Somebody [3]
- . Бот «FTBotPro (Neeter)» от Neeter [4]
- . Бот «VovanZ ExBot8» от VovanZ [5]
- . Бот «Crusader v1.4» от гpy3uH [6]
- . Бот «Alar BOT v 1.19» от Alar [7]

Для начала узнаем, как-же они хороши играя против SimpleBot v1.0, так как именно он был доступен всем и поставлялся вместе с самой игрой (см. таблицу 1):

Таблица 1. Результаты игр со встроенным ботом SimpleBot v1.0

Параметр / Бот	«Разрушитель-1»	«FTBotPro (Neeter)»	«VovanZ ExBot8»	«Crusader v1.4»	«Alar BOT v1.19»
Количество игр	10000	10000	10000	10000	10000
Количество побед встр-го	17	8	1534	337	3290
Количество побед противника	9197	9990	8275	9440	6710
Количество «ничьих»	786	2	191	223	0
% побед	91	99	82	94	67
% «ничьих»	7	~0	19	2	0
Занятые места	3	1	4	2	5

Теперь самое интересное – боты играют друг с другом. Играли они по круговой системе: каждый с каждым, у кого больше побед тот и выиграл. Игры производились в стандартном режиме игры (см. таблицу 2):

Таблица 2. Результаты игр по круговой системе. Раунд-1

Параметр / Бот с ботом	Alar BOT v 1.19 vs VovanZ ExBot8	Alar BOT v 1.19 vs Разру- шитель- 1	Alar BOT v 1.19 vs Crusader v1.4	Alar BOT v 1.19 vs FTBotPro (Neeter)	VovanZ ExBot8 vs Разру- шитель- 1	VovanZ ExBot8 vs Crusader v1.4	VovanZ ExBot8 vs FTBotPro (Neeter)
Количество игр	10000	10000	10000	10000	10000	10000	10000
Кол-во побед / кол-во ошибок первого бота	1226 / 0	0 / 0	2542 / 1258	0 / 0	0 / 0	2213 / 0	354 / 0
Кол-во побед / кол-во ошибок второго бота	8774 / 0	10000 / 0	7458 / 0	9996 / 0	8824 / 0	7685 / 0	9168 / 0
Количество «ничьих»	0	0	0	4	1176	102	478
Кол-во очков первого бота / второго бота	420680 / 1779320	200000 / 2000000	657560 / 1542440	200000 / 1999280	200000 / 1788320	598340 / 1583300	263720 / 1850240
Победитель	VovanZ ExBot8	Разру- шитель- 1	Crusader v1.4	FTBotPro (Neeter)	Разру- шитель- 1	Crusader v1.4	FTBotPro
Процент побед, %	87	100	74	100	100	76	91

Прошло 7 сражений и остается еще 3 битвы, и уже можно подвести некоторые итоги:

- . «FTBotPro (Neeter)» – 2
- . «Crusader v1.4» – 2
- . «Разрушитель-1» – 2
- . «VovanZ ExBot8» – 1
- . «Alar BOT v 1.19» – 0

Можно уже сказать, что «VovanZ ExBot8» получит предпоследнее место, так как уже ни с кем играть не будет. Посмотрев его статистику можно сделать вывод, что его проигрыш обусловлен тем, что он не использует шпионаж вообще. «Alar BOT v 1.19» делает упор только на мелкие атаки, полностью игнорирует проекты по починке базы и почти не развивает базу (увеличивает только количество батарей) и никакого шпионажа. Шпионаж в этой игре один из ключевых моментов. Переходим к следующему кругу (см. таблицу 3):

Таблица 3. Результаты игр по круговой системе. Раунд-2

Параметр / Бот с ботом	Разрушитель-1 vs Crusader v1.4	Разрушитель-1 vs FTBotPro (Neeter)
Количество игр	10000	10000
Кол-во побед / кол-во ошибок первого бота	3972 / 0	1437 / 0
Кол-во побед / кол-во ошибок второго бота	5712 / 0	7942 / 0
Количество «ничьих»	316	621
Кол-во очков первого бота / второго бота	914960 / 1228160	458660 / 1629560
Победитель	Crusader v1.4	FTBotPro (Neeter)
Процент побед, %	57	79

Перед последним сражением сложилась следующая ситуация:

- . «FTBotPro (Neeter)» – 3
- . «Crusader v1.4» – 3

- . «Разрушитель-1» - 2
- . «VovanZ ExBot8» - 1
- . «Alar BOT v 1.19» - 0

Опять-же, проигрыш «Разрушитель-1» обусловлен тем, что он из шпионажа использовал только 26 и 28 проекты (похищение металла и уничтожение рудника). Снова видно, что встретились два достойных соперника, но «Crusader v1.4» оказался сильнее. Победы «крестоносца» - 54 %, ничьи - 10%. Из-за Минимального перевеса «Crusader v1.4» необходимо было провести битву между ними в расширенном режиме игры, при лимите ходов в 1000 (см. таблицу 4):

Таблица 4. Результаты игр по круговой системе. Стандартный и расширенный режим

Параметр / Бот с ботом	Crusader v1.4 vs FTBotPro (Neeter)	Crusader v1.4 vs FTBotPro (Neeter)
	стандартный режим	расширенный режим
Количество игр	10000	10000
Кол-во побед / кол-во ошибок первого бота	5442 / 0	4907 / 0
Кол-во побед / кол-во ошибок второго бота	3494 / 0	4708 / 0
Количество «ничьих»	1064	385
Кол-во очков первого бота / второго бота	1179560 / 828920	1083260 / 1047440
Победитель	Crusader v1.4	Crusader v1.4
Процент побед, %	54	49

Снова видно, что встретились два достойных соперника, но «Crusader v1.4» оказался сильнее. Победы «крестоносца» - 49 %, ничьи ~ 4%. Битва в расширенном режиме показала минимальный перевес «Crusader v1.4». По итогам игр в стандартном режиме игры, сложилась следующая ситуация:

- . «Crusader v1.4» - 4
- . «FTBotPro (Neeter)» - 3
- . «Разрушитель-1» - 2
- . «VovanZ ExBot8» - 1
- . «Alar BOT v 1.19» - 0

Заключение

Итак, места распределились следующим образом:

- . Первое место - «Crusader v1.4»
- . Второе место - «FTBotPro (Neeter) »
- . Третье место - «Разрушитель-1»
- . Четвертое место - «VovanZ ExBot8»
- . Пятое место - «Alar BOT v 1.19»

Также можно сделать вывод о том, что чем больше используется шпионаж, тем больше шансов на победу. «Разрушитель-1» почти не использовал шпионаж, а «VovanZ ExBot8» вообще пренебрег шпионскими проектами. «Alar BOT v 1.19», вообще выбивается из общей колеи - он «спамил» мелкими атаками. А теперь каждый может [скачать игру](#) с ботами и проверить их.

Ресурсы

- . Правила игры <http://programmersforum.ru/showthread.php?t=79260>

- . Обсуждение конкурса <http://programmersforum.ru/showthread.php?t=81296>
- . Бот (исходники) «Разрушитель-1» от Somebody
<http://programmersforum.ru/attachment.php?attachmentid=23203&d=1269771063>
- . Бот (исходники) «FTBotPro (Neeter)» от Neeter
<http://programmersforum.ru/attachment.php?attachmentid=23199&d=1269767005>
- . Бот (исходники) «VovanZ ExBot8» от VovanZ
<http://programmersforum.ru/attachment.php?attachmentid=23193&d=1269718900>
- . Бот (исходники) «Crusader v1.4» от ppy3uH
<http://programmersforum.ru/attachment.php?attachmentid=23175&d=1269684979>
- . Бот «Alar BOT v 1.19» от Alar <http://programmersforum.ru/member.php?u=1>
- . Игра Fortress и боты <http://pblog.ru/wp-content/uploads/Fortress-competition-results.zip>
- . Ознакомительная версия игры Fortress 2
<http://programmersforum.ru/showthread.php?p=488246>