РУКОВОДСТВО

СОДЕРЖАНИЕ

◆

ОСНОВНЫЕ ВОЗМОЖНОСТИ
СОЗДАНИЕ ПРОЕКТОВ

◆

ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ
НАПИСАНИЕ ПЛАГИНОВ

◆

◆

Авторы:

Kruptar: Джинни
Руководство: FoX (Chime[RUS])

2016

1. ОСНОВНЫЕ ЭЛЕМЕНТЫ ПРОГРАММЫ И ГЛАВНОЕ МЕНЮ

Рассмотрим **три версии данной программы**: **7.0.0.85**, **7.1.1.10** и **7.1.1.17** (поскольку есть нюансы).

Различия будут выделяться цветом: **только** для версии **7.0.0.85** — зелёный, для версий **7.1.1.10** и **7.1.1.17** будет синий цвет, красный будет в том случае, когда возможность реализована **только** в **7.1.1.17**.

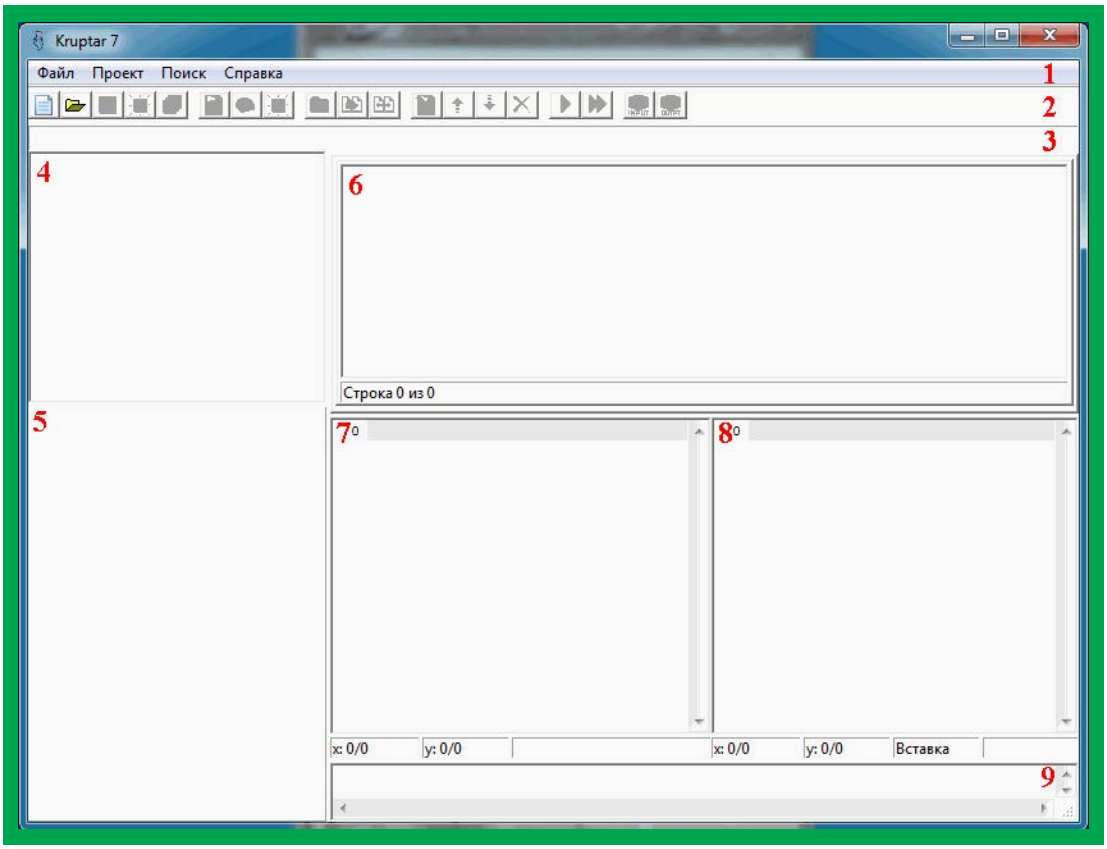


Рисунок 1.1. Внешний вид программы Kruptar **версии 7.0**

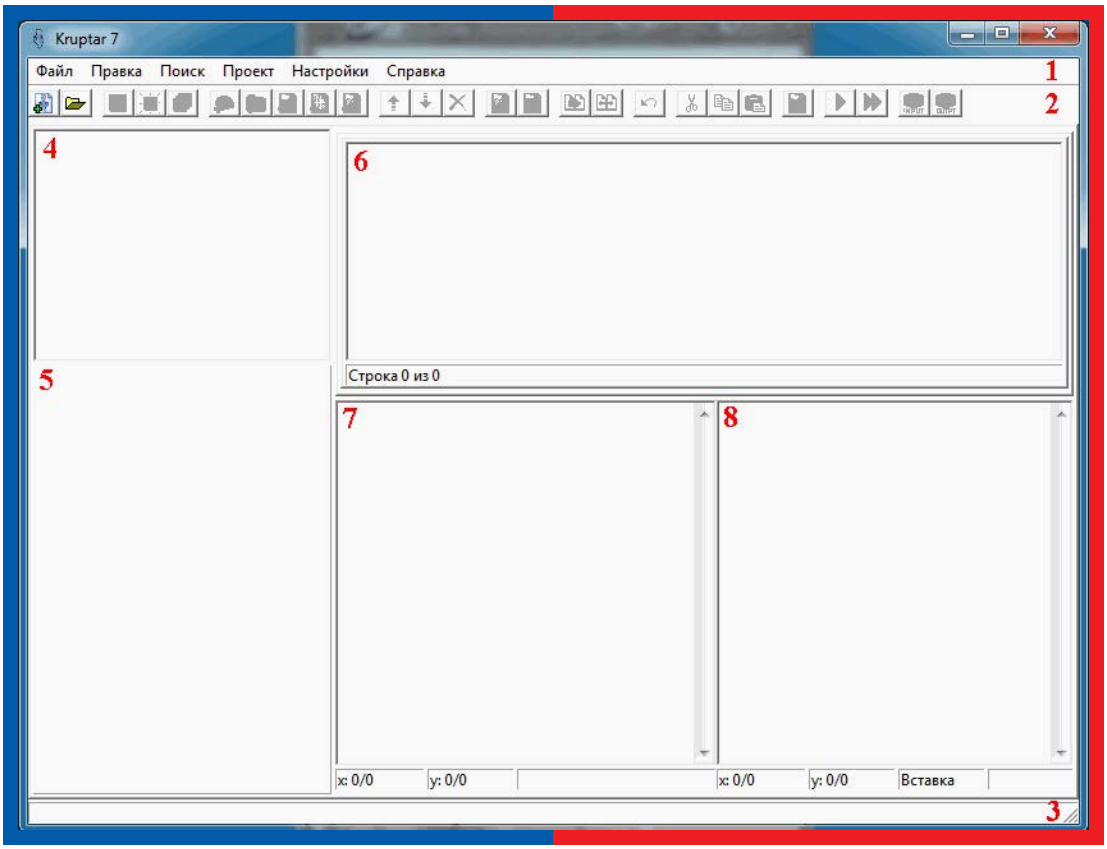


Рисунок 1.2. Внешний вид программы Kruptar **версии 7.1**

Основные элементы: 1. Главное меню; 2. Панель инструментов; 3. Панель состояния; 4. Элементы проекта; 5. Параметры проекта; 6. Список текстовых элементов; 7. Оригинальный текст; 8. Редактируемый текст; 9. Окно сообщений (скрыто).

2. ГОРЯЧИЕ КЛАВИШИ

Ctrl+N – создать проект.
Ctrl+O – открыть проект.
Ctrl+S – сохранить проект.
Ctrl+Alt+S – сохранить всё (сохранить как).
Ctrl+Alt+X – закрыть.
Ctrl+Alt+A – добавить элемент.
Ctrl+T – добавить таблицу.
Ctrl+Alt+T – загрузить таблицу из файла (переустановить ссылки таблиц).
Ctrl+Q – импорт указателей из файла.
Ctrl+G – добавить группу.
Ctrl+F7 – перезагрузить текст.
Ctrl+F8 – вернуть исходный текст.
Ctrl+Alt+D – параметры словаря.
Ctrl+Alt+G – генерировать словарь.
Shift+Ctrl+F6 – словарь в таблицу.
Ctrl+Alt+P – добавить указатели.
Ctrl+Del – очистить.
Ctrl+Alt+W – вернуть таблицы списка.
Ctrl+Alt+V – переменные.
Ctrl+U – переместить вверх.
Ctrl+H – переместить вниз.
Ctrl+D – удалить элемент.
F9 – пересчитать и вставить указатели и текст из текущей группы в ROM.
Ctrl+F9 – пересчитать и вставить указатели и текст из каждой группы в ROM.
F11 – запустить исходный ROM в эмуляторе.
F12 – запустить выходной ROM в эмуляторе.
Ctrl+F – найти.
Ctrl+R – заменить.
F3 – найти далее.

* Можно увидеть во вкладке «Правка главного меню»
* Только в контекстном меню окна «Редактируемый текст»

Ctrl+Z – отменить.
Ctrl+Alt+Z – повторить.
Ctrl+X – вырезать.
Ctrl+C – копировать.
Ctrl+V – вставить.
Ctrl+A – выделить всё.
F2 – переименовать.
Ctrl+Alt+O – добавить из файла.
Ctrl+Alt+L – загрузить из файла
Ctrl+F2 – сохранить в файл.
Ctrl+Alt+E – добавить/удалить свойства для вставки.
Ctrl+Alt+F – исправить позиции указателей.

Ctrl+F1 – о программе.

Некоторые пункты главного меню не имеют горячих клавиш.

Файл:
Сохранить всё.
Сохранить как.
Закрыть всё.
Выход.

Проект:
Выбрать эмулятор.

Таблицы:
Сохранить в файл.
Заполнить ANSI-символами.

Группы:
Загрузить список адресов для вставки текста...

Списки пойнтеров:
Сортировать по возрастанию.
Сортировать по убыванию.

Настройки:
Язык.
Шрифт.

Справка:
Библиотеки (Плагины).

3. ПОДГОТОВКА К СОЗДАНИЮ ПРОЕКТА

Рассмотрим создание проекта на примере игры **The Legend of Zelda: Links Awakening DX (GBC).**

Здесь показан только один из множества возможных путей решения задачи по нахождению всех нужных циферок для создания проекта. Шрифты можно увидеть при помощи понравившегося тайлового редактора, а тексты при помощи понравившегося HEX-редактора. Они тут не запакованы, и объёмы ROM-а не столь велики, чтобы чего-то не найти. Как видим, текст читается и имеет кодировку, похожую на ASCII.

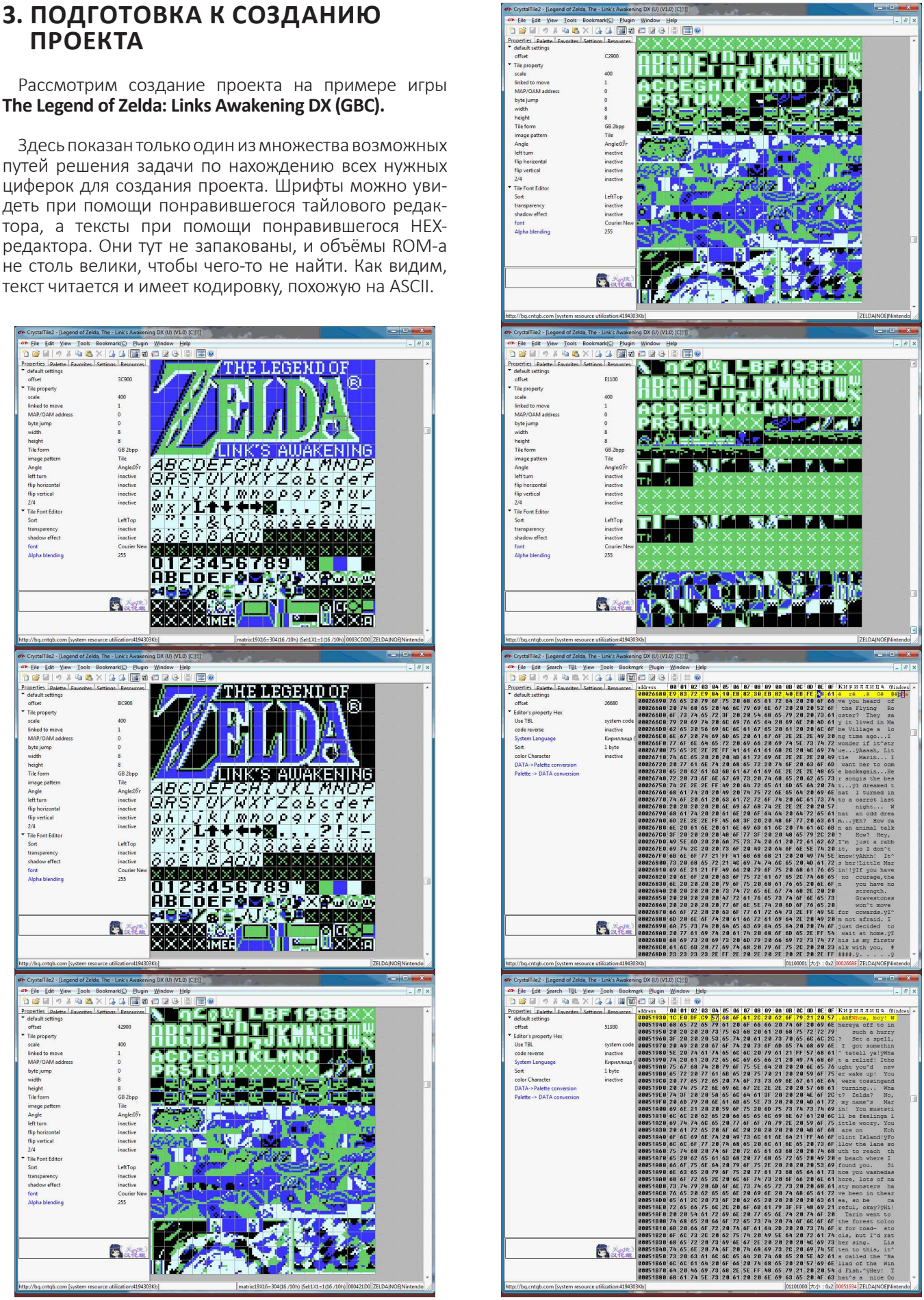


Рисунок 2. Предварительное исследование. Поиск шрифтов и текстов в CrystalTile 2

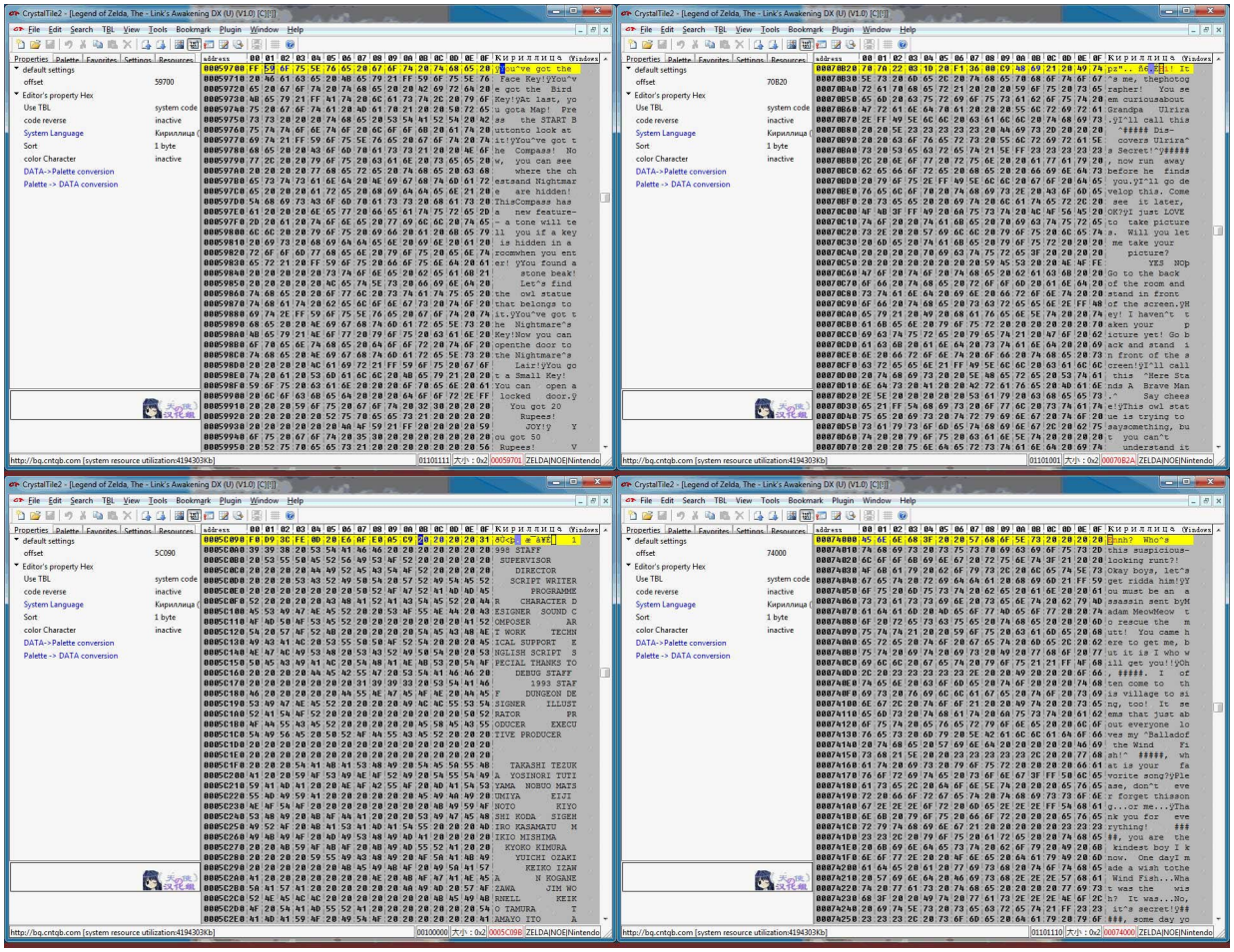


Рисунок 2 (продолжение). Предварительное исследование. Поиск текстов в CrystalTile2

О том, как находить указатели на эту платформу (Game Boy / Game Boy Color), можно прочитать здесь:

chief-net.ru/index.php?option=com_content&task=view&id=109&Itemid=33

Мне удобнее разбирать ROM на блоки (либо файлы), выбрать те, где есть текст и указатели и работать уже конкретно с ними. Здесь, в связи с особенностями архитектуры консоли, будут блоки по **16 килобайт**, они получены при помощи простой и понятной программки *Cracker*'а и небольшого батника: zelda64rus.ucoz.ru/_fr/5/Cut_und_analyze.7z

Теперь можно записать куда-нибудь адреса начала и окончания текстов (нули, идущие после текста, в данном случае можно считать текстом).

Мне удобно делать это при помощи Hex-редактора (здесь *Hex Workshop*) и электронных таблиц (здесь *Excel*).

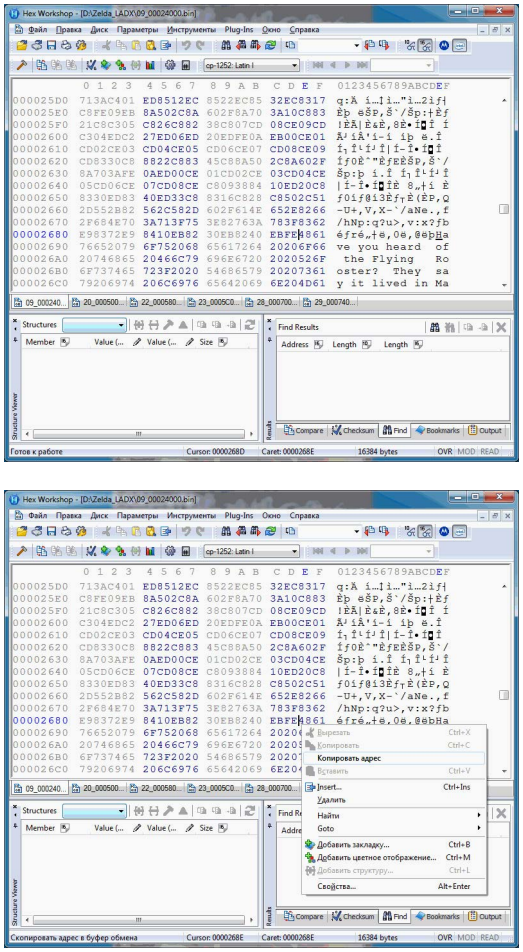


Рисунок 3. Копируем адреса начала текстов из каждого блока

В блоке **28_00070000.bin**, начиная с адреса **0x0001** и заканчивая **0x0560**, содержатся указатели. Копируем их в отдельный файл и удалим первый байт: **01**. Нам он не нужен, т.к. указателем на текст явно не является, а для чего он был нужен разработчикам, меня не интересовало.

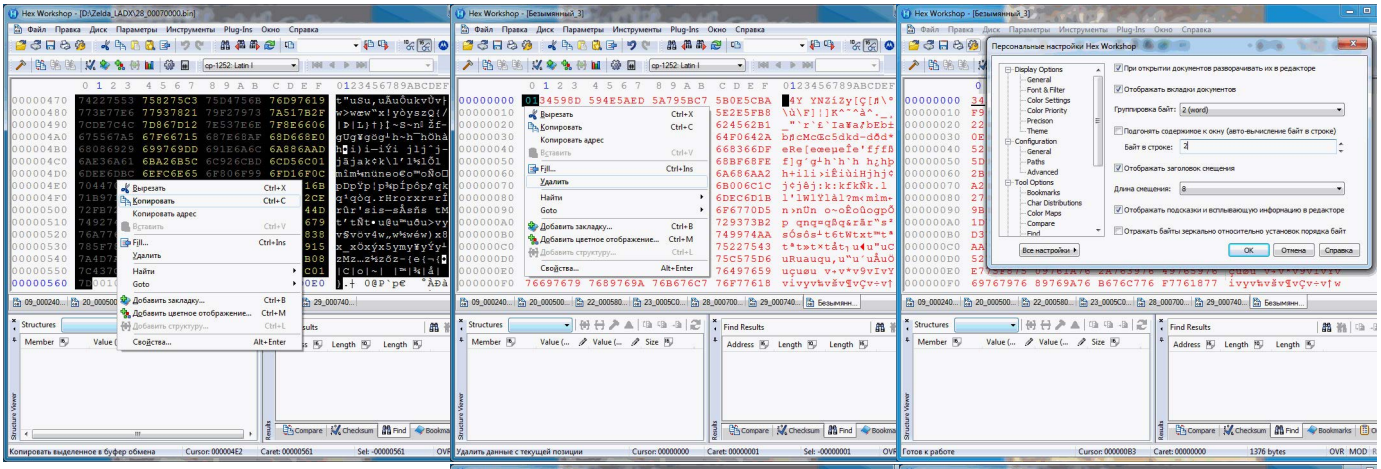
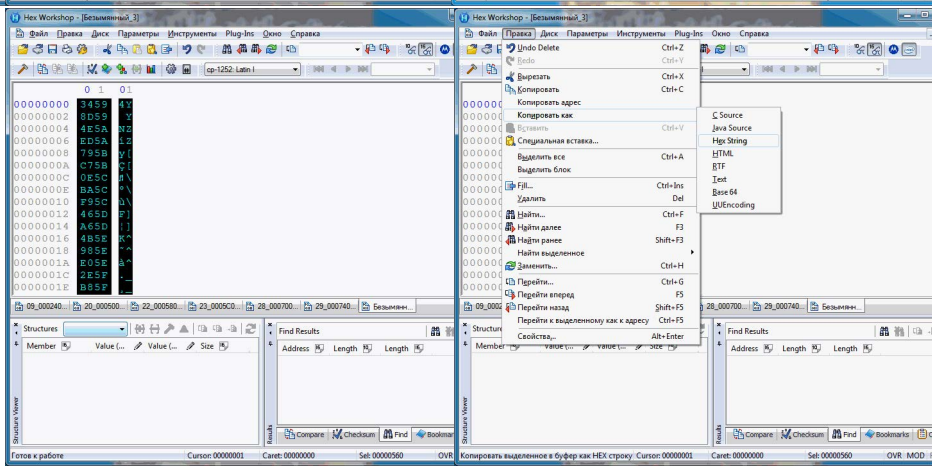


Рисунок 4. Копируем указатели

Приведём к удобному виду и скопируем указатели в текстовый редактор (здесь *AkelPad*). Удалим лишние пробелы и увидим, сколько же у нас всего указателей и предположительное количество строк.



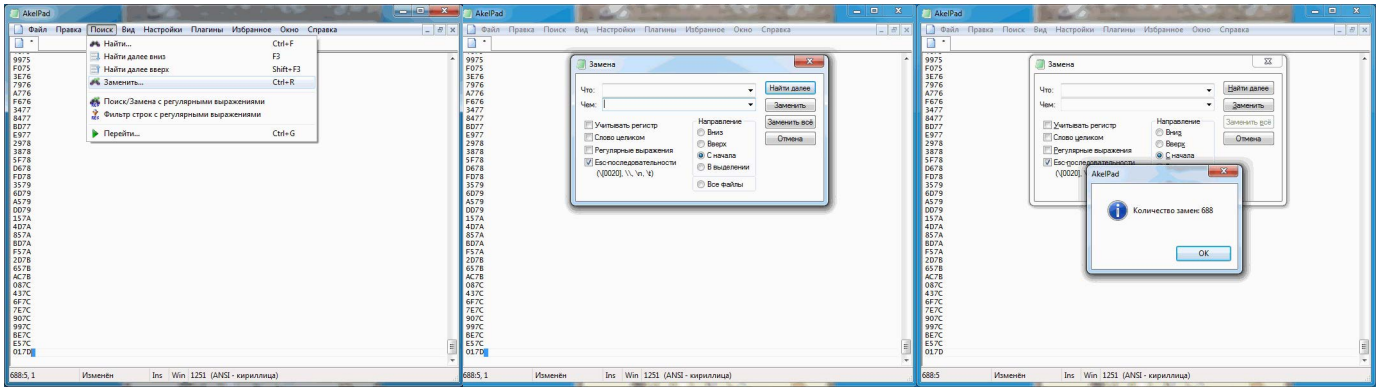


Рисунок 5. Удаляем лишнее

Выделяем всё и опять копируем. В *Excel* выделяем столбец, выбираем текстовый формат ячеек и вставляем сюда полученные указатели. Далее можно их перевернуть (**LE** -> **BE**) и сравнить с адресами начала текстов (**+ 0x4000**).

Теперь и указатели можно разбить на 5 групп, т.к. для титров тут указателей не нашлось. Получим начальный и конечный адреса для каждой группы указателей. Всё это, конечно, можно найти и посчитать при помощи Hex-редактора и калькулятора, но мне лень.

Таблицы для перекодировки текста тоже удобно хранить подобным образом. А лишние знаки табуляции удалять в текстовом редакторе, непосредственно перед сохранением в файл (***.tbl**).

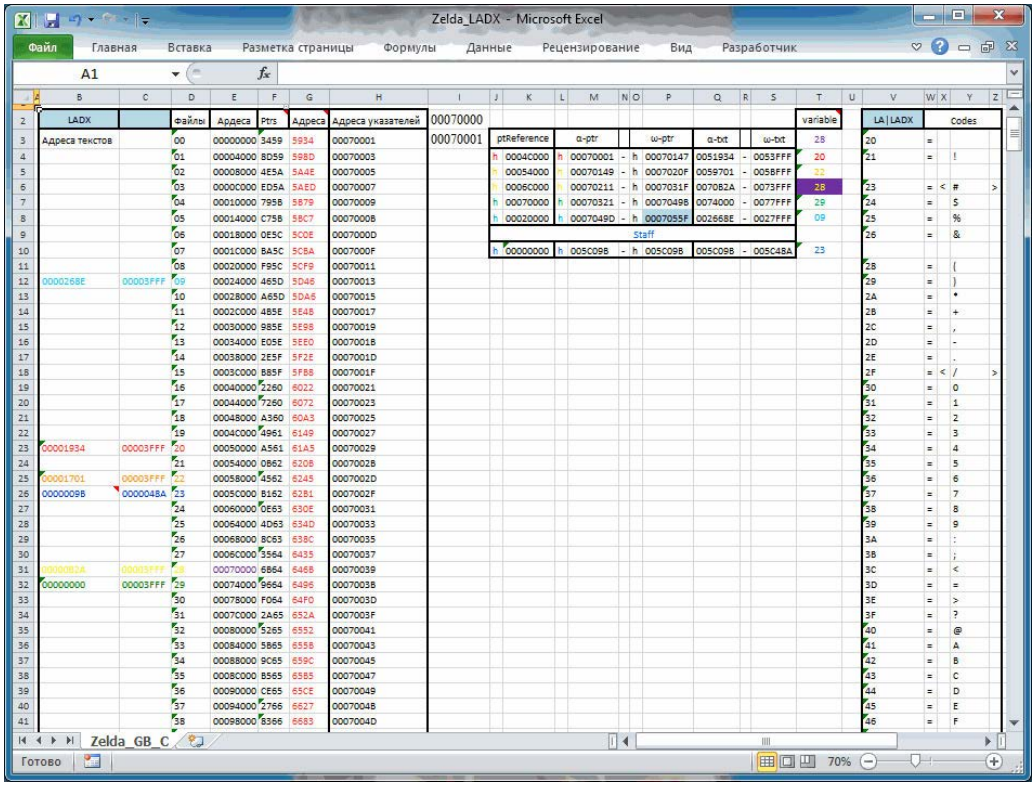


Рисунок 6. Все циферки, необходимые для создания проекта, в одном месте

Таблицу символов можно получить различными способами. В данном случае карта символов шрифта не была найдена. Чтобы составить полную таблицу символов, все тексты, кроме титров, копировались в один отдельный файл.

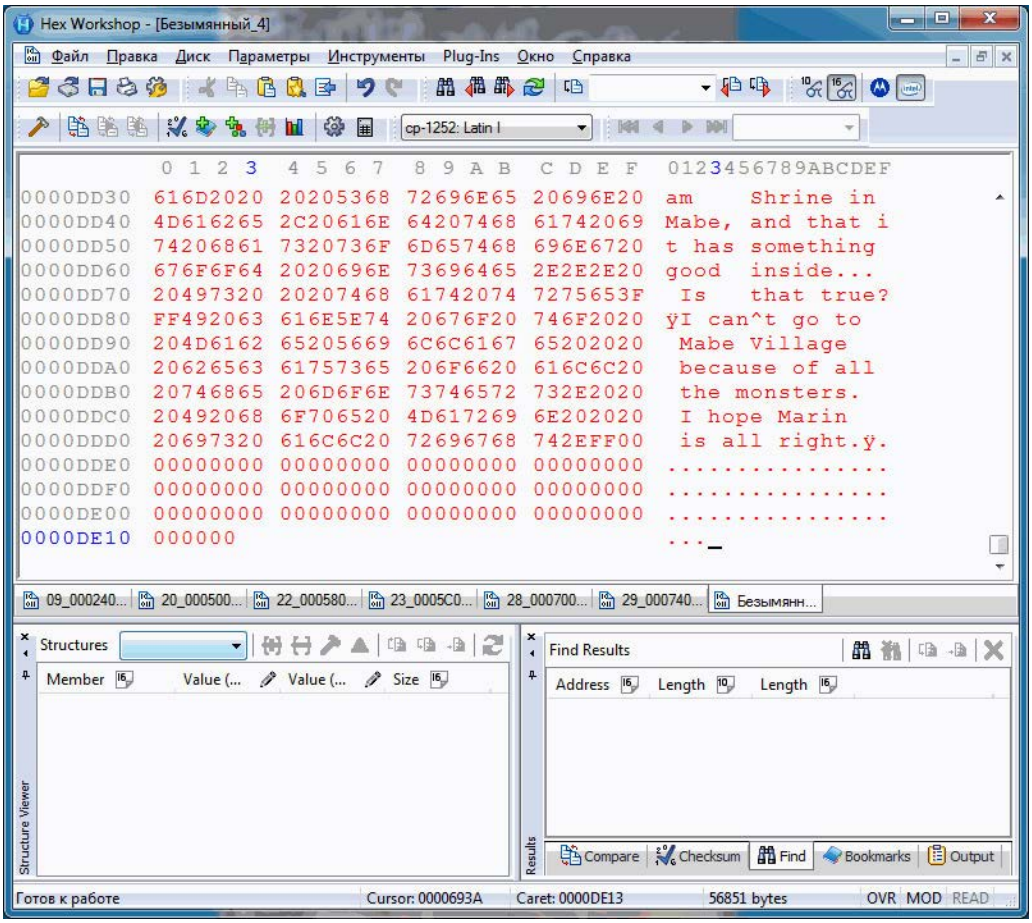


Рисунок 7. Тексты из всех блоков (кроме титров) скопированы в один файл

Когда весь диалоговый текст находится в одном файле, то можно определить все коды символов и специальные коды, которые в нём содержатся. Сразу же видим **FF** в конце текста, — предположим, что это символ конца строки и тут же это проверим (**Рисунок 8**).

Количество символов получилось меньше, чем количество указателей, что может говорить как о наличии нескольких указателей на одну строку, так и о наличии других символов конца строки (здесь ещё можно обратить внимание на то, что адреса всех найденных **FF** можно скопировать в блокнот и опять же использовать для проверки, сравнив с указателями). Проверим теперь все возможные коды с **F?**. Пролистаем немножко и найдём ещё один код, похожий на конец строки: **FE**. И ещё коды **F0**, **F1**, **F2** и **F3**, которые впоследствии окажутся стрелочками (**Рисунок 9**)..

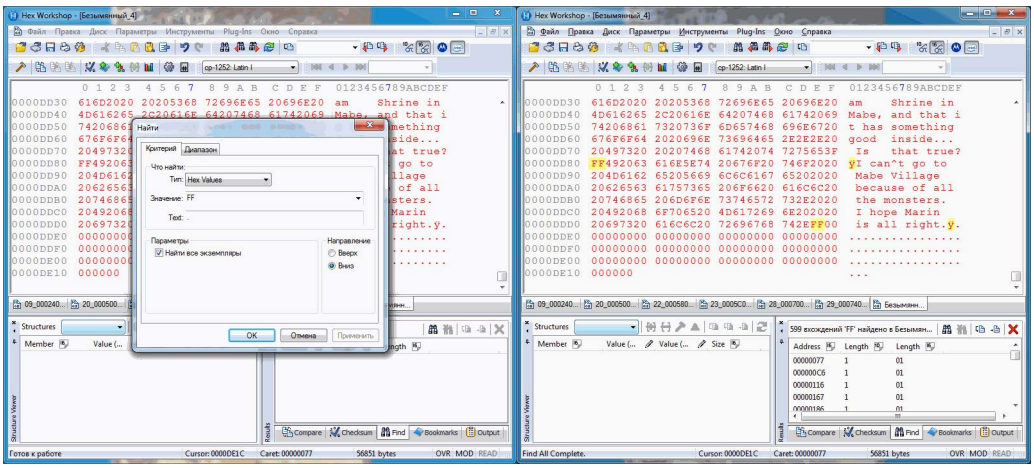


Рисунок 8. Проверяем «FF»

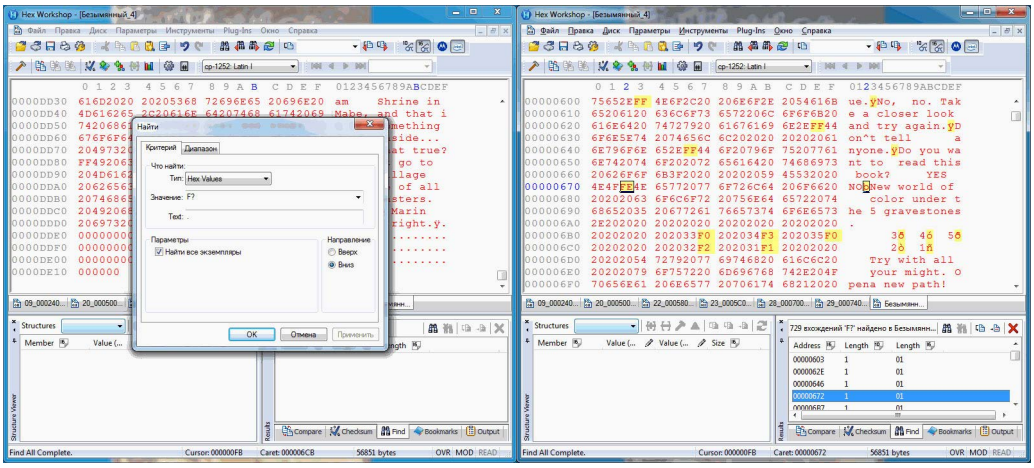


Рисунок 9. Проверяем «F?»

Скопируем эти коды в текстовый редактор, а здесь заменим на **00**, чтобы больше к ним не возвращаться. Повторяем процедуру до нахождения всех нужных кодов. Коды букв известны, т.к. текст читается, а вот апостроф явно имеет код, отличный от стандартного.

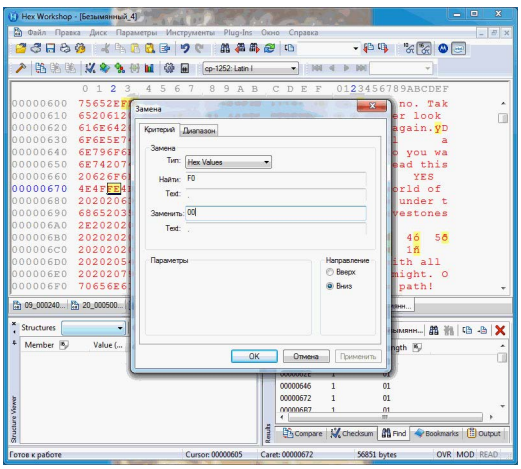


Рисунок 10.
Заменяем все «F?», переходим к следующей группе кодов «E?» и т.д.

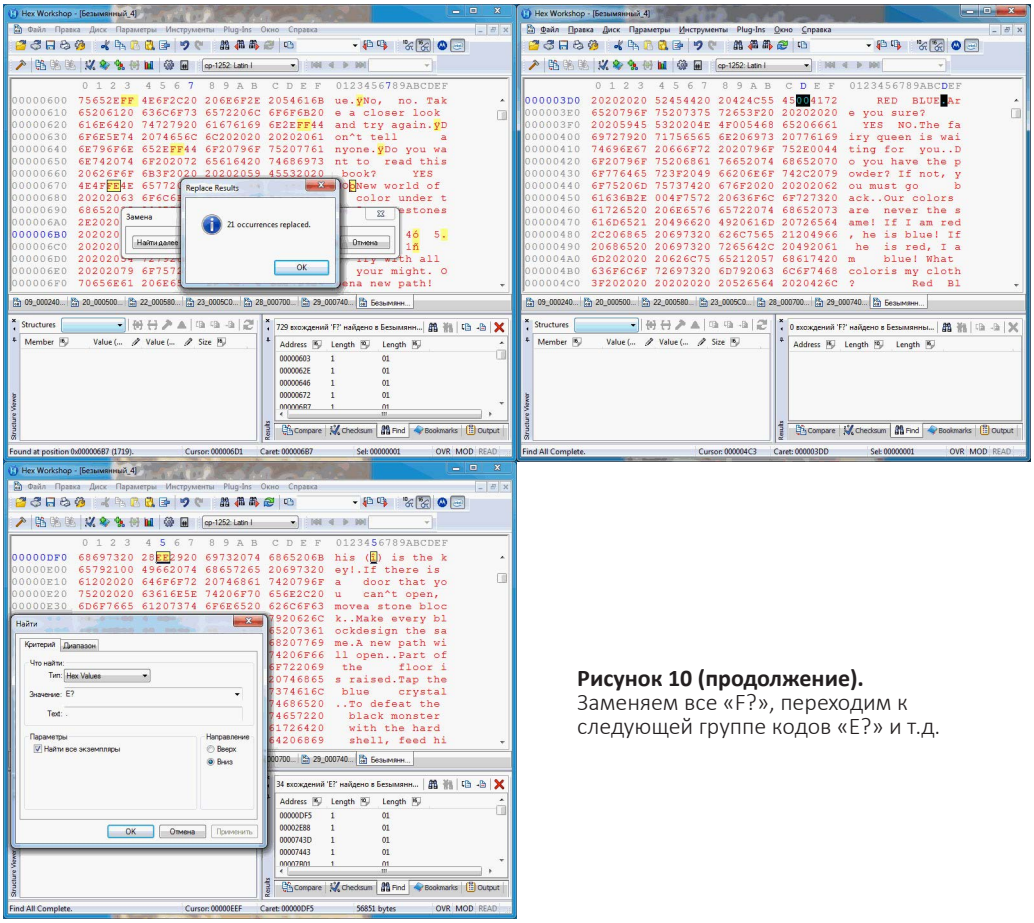


Рисунок 10 (продолжение).
Заменяем все «F?», переходим к следующей группе кодов «E?» и т.д.

Коды оставшихся символов алфавита (европейские символы, среди ромхакеров иногда называемые «умляуты»), которые есть в шрифте, но в этом тексте не используются, можно узнать уже после составления проекта с помощью Kruptar и эмулятора.

Для того, чтобы читались все символы, в том числе японские, какие-либо значки или европейские «умляуты», всегда рекомендую сохранять таблицу в кодировке **Unicode (**здесь **UTF-16 LE)**, коды, отличные от символов, заключать в скобки (тут **[квадратные]**).

Если известно, что код обозначает — именовать, если нет — нумеровать (либо писать сам код, если он короткий).

После **ends** идут символы конца строки. Здесь таковых получилось два.

Теперь мы знаем абсолютные адреса текстов и указателей. Есть таблица с оригинальным алфавитом и служебными кодами (за ними бывают скрыты иконки, звуки, имена и прочие функции).

Можно возвращаться к созданию проекта в Kruptar.

4. СОЗДАНИЕ ПРОЕКТА И КОНТЕКСТНОЕ МЕНЮ

ПРОЕКТ

Нажимаем **Ctrl+N**, и в окне «**Элементы проекта**» появляется древовидная структура пустого проекта и его элементов:

Без имени 1 (**Project1**)

Вкладка «**Параметры проекта**» при выборе элемента **Без имени 1** (**Project1**) примет вид:

kpInputRom (**kpSourceROM**)

Здесь нужно выбрать файл, откуда будет загружаться скрипт (ROM либо бинарный файл).

kpOutputRom (**kpDestinationROM**)

Здесь нужно выбрать файл, куда скрипт будет записываться (ROM либо бинарный файл).

Удобнее, когда каждый файл находится в своей папке.
Например: исходный в **Original**
редактируемый в **Russian**

kpCoding (**kpCodePage**)

Кодировка таблиц, используемых в проекте, определяется здесь. По умолчанию выбрана кодировка ANSI (1251 (ANSI — кириллица)). Мы выбираем UNICODE (1200 (UTF-16 LE)), т.к. таблица у нас сохранена именно в нём.

Есть ещё SHIFT-JIS (пригодится для японского языка), а в версии **7.1** есть ещё больше кодировок, но для чего они все нужны, я не знаю.

kpFlags

Пункт для хранения различных флагов настроек. Имеет один флаг, равный **1** (**h01**), который устанавливает первый в списке переносов строки код в таблице, как невидимый. Если кодов переноса строки много, то это выглядит не очень здорово. Возможно, здесь предполагалось расширение гибкости программы посредством системы таких флагов.

КОНТЕКСТНОЕ МЕНЮ

Так как проектов может быть много, в версии **7.0** несколько проектов идут списком в окне «**Элементы проекта**», а в **7.1** создаются **вкладки** и, соответственно, перемещать позиции вверх/вниз здесь уже избыточно.

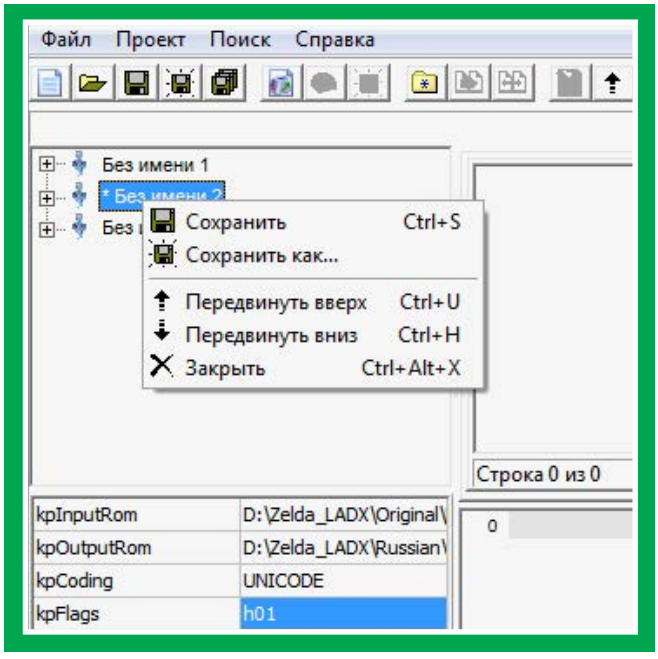


Рисунок 11.1. Новый проект в версии **7.0**

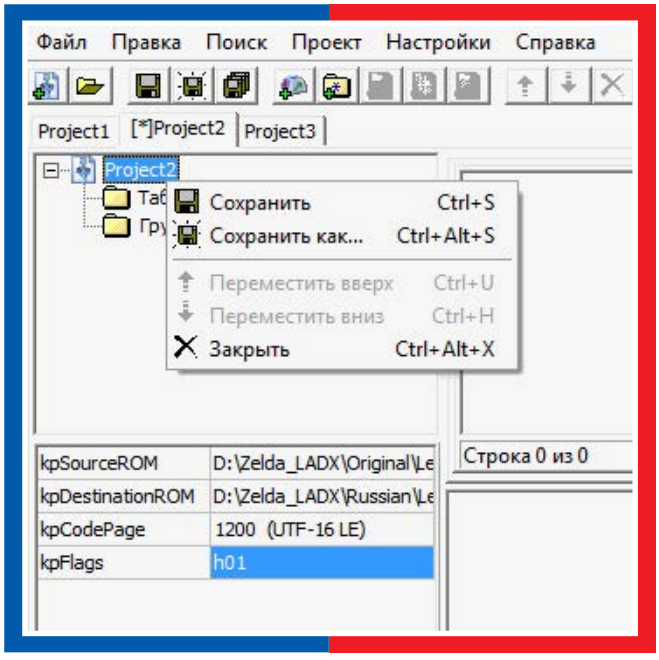


Рисунок 11.2. Новый проект в версиях **7.1**

ТАБЛИЦЫ

Нажимаем **Ctrl+T**, и появляется пустая таблица с такими элементами:

Table1

В версии **7.0** при загрузке таблицы из файла (**Ctrl+Alt+T**) меняется имя таблицы (**Table1**) на имя файла, в версии **7.1** такой эффект достигается, если таблицу добавлять **не** через **Ctrl+T**, а через **Ctrl+Alt+O**, при загрузке из файла (**Ctrl+Alt+L**) имя таблицы остаётся **Table1**. Есть возможность переименовать этот элемент: выделить его и нажать правую кнопку мыши или F2. Можно создать много таблиц: добавить как уже готовую, так и пустую, или составлять её прямо в Kruptar.

> А ещё можно сохранить таблицу в той кодировке, которая была указана ранее в проекте: **Сохранить в файл** (**Ctrl+F2**).

Рекомендую добавлять две таблицы: **для чтения текста** – **свою**, **для вставки текста** – **свою**, даже если они одинаковые, т.к. если понадобится одну из них заменить, это сделать окажется проще, чем в случае с общей таблицей. Если двум разным символам в таблице соответствует один код, то читаться и записываться в скрипт будет символ, который находится в таблице выше. Это следует учитывать, если понадобится читать английские символы, а записывать русские, которые в шрифте нарисованы поверх английских, а также при поиске ошибок во время проверки правильности составления проекта.

Коды окончания строки

Здесь можно добавить нулевой элемент при помощи **Ctrl+Alt+A** и переименовать его в код окончания строки либо прочитать, если таблица уже была добавлена.

Коды разрыва строки

Здесь можно добавить нулевой элемент при помощи **Ctrl+Alt+A** и можно переименовать его в код разрыва строки либо прочитать, если таблица уже была добавлена.

> В данном скрипте (Zelda DX) кодов переноса (разрыва) строки не нашлось, но для удобства восприятия текста такой код будет добавлен при помощи плагина.

Символы

Здесь можно добавить нулевой элемент при помощи **Ctrl+Alt+A** (в версии **7.1.1.10** есть баг, в **7.1.1.17** он исправлен) либо прочитать (переименовать), если таблица уже была добавлена. Либо заполнить таблицу ANSI-символами.

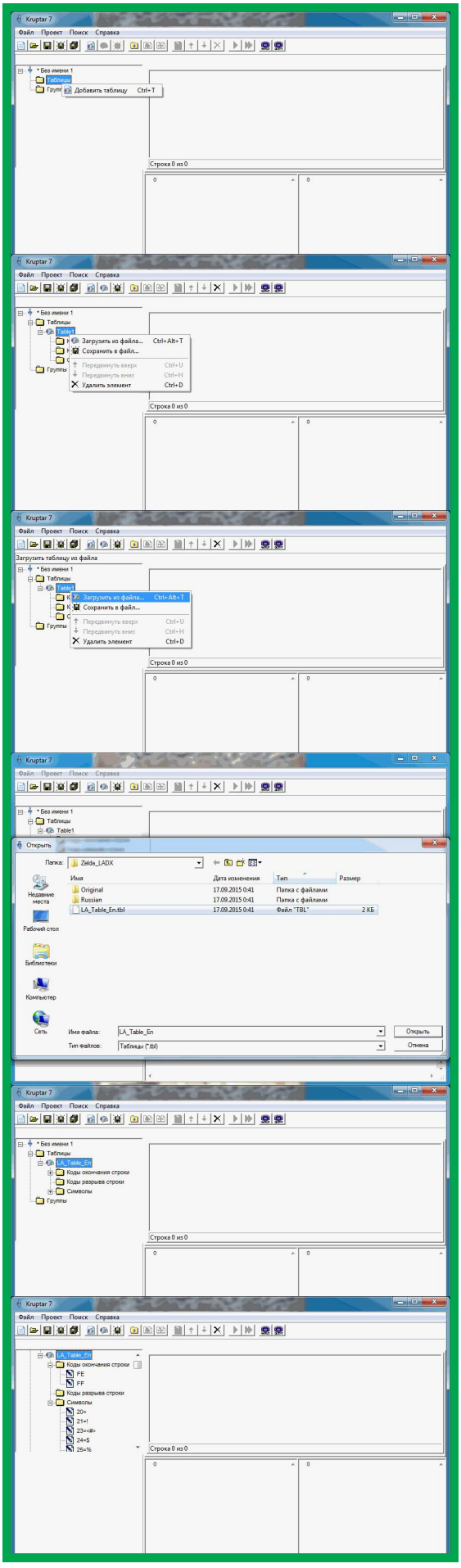


Рисунок 12. Загрузка таблицы из файла в версии **7.0**

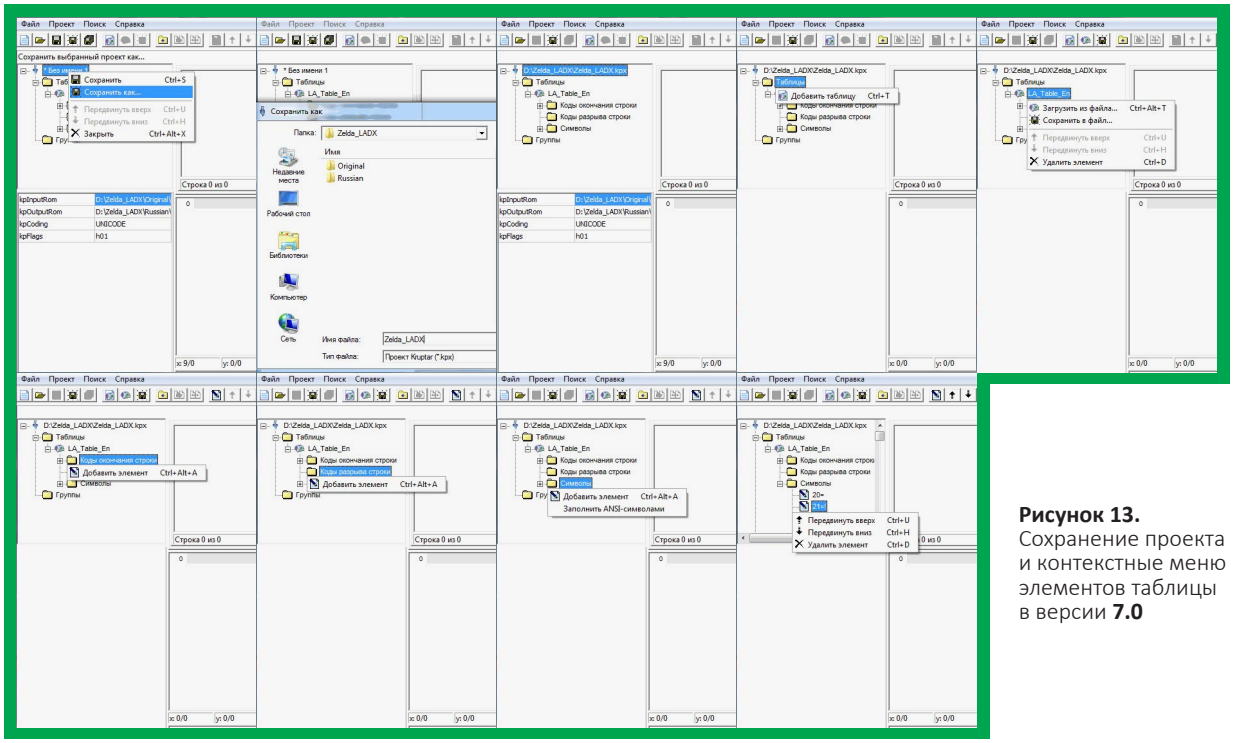


Рисунок 13. Сохранение проекта и контекстные меню элементов таблицы в версии **7.0**

КОНТЕКСТНОЕ МЕНЮ

Функционал контекстного меню элементов таблицы в версии **7.1** значительно расширен и это следует рассмотреть отдельно.

Проект был сохранён в версии **7.0.0.85** и открыт в **7.1.1.17**, что тоже является одной из важных особенностей.

Обратной совместимости нет: создав либо сохранив проект в версии **7.1**, в **7.0** его уже НЕ ОТКРЫТЬ, т.к. код загрузки и сохранения проектов был переписан и оптимизирован.

Появилась возможность сохранять (**Ctrl+F2**) не только саму таблицу, но и её элементы:

LA_Table_En_TRMN.tbl
LA_Table_En_CLRF.tbl
LA_Table_En_CHR.tbl

К появившимся в этой версии стандартным операциям редактирования добавилась работа с файлами:

Ctrl+Alt+O — добавляет к уже существующим элементам те, что содержатся в выбранном файле;

Ctrl+Alt+L — загружает элементы из файла, удаляя существующие, если они есть.

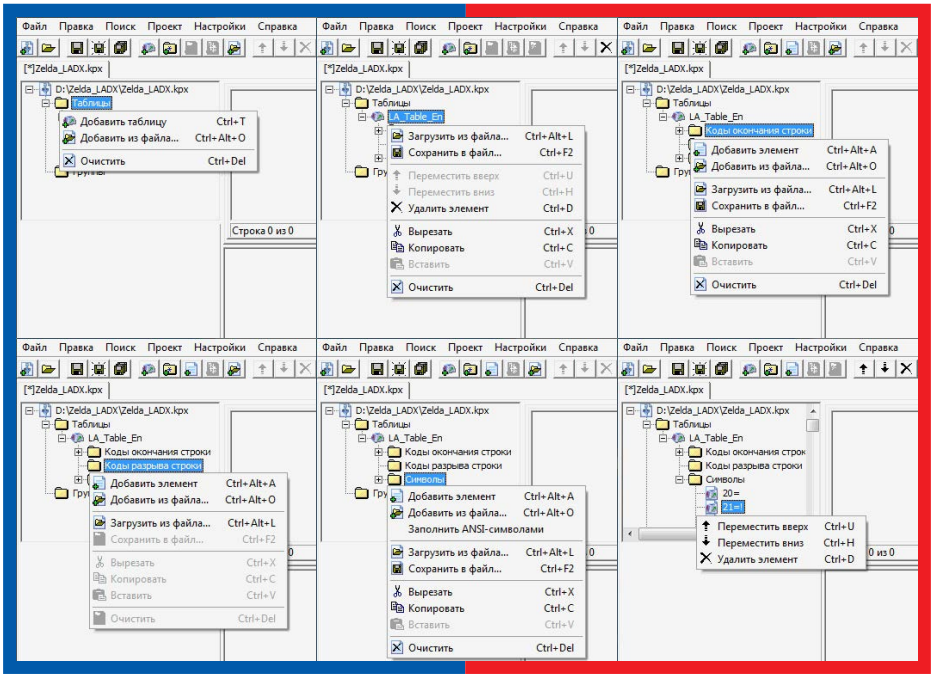


Рисунок 14. Контекстные меню элементов таблицы в версиях **7.1**

ГРУППЫ

Нажимаем **Ctrl+G**, и появляется пустая группа с такими элементами:

Group1

Вкладка «**Параметры проекта**» при выборе элемента **Group1** примет вид:

grLibrary (**grPlugin**)

Т.к. Kruptar всегда использует плагины для извлечения и вставки скрипта здесь нужно выбрать один из существующих плагинов, либо, если они не подходят, выбрать свой, предварительно написав его и положив в папку **Lib**.

Плагин **Standard.kpl** подходит для скриптов, в которых строки оканчиваются символом конца строки, по умолчанию используется именно он.

Плагин **NULL.kpl** создан для скриптов в которых символ конца строки = **00**, тогда он в проекте становится невидимым (вернее, не считывается при чтении символов, но записывается при записи).

Описание, если, конечно, писавший плагин потрудился его написать, можно почитать в **Справке**.

> Тема написания плагинов освещена отдельно далее в руководстве, в дополнение к существующей:
>
> magicteam.net/index.php?page=documents&-show=Написание плагинов для Kruptar 7

grDictionary (**grIsDictionary**)

Этот параметр по умолчанию равен *FALSE* и означает, что группа не является словарём. Если же в тексте присутствует DTE/MTE оптимизация, одна из групп обязательно должна им являться. Когда этот параметр равен *TRUE*, в контекстном меню элемента **Group1** становятся доступны следующие опции:

Ctrl+Alt+D — Параметры словаря (Словарь DTE/MTE),
Ctrl+Alt+G — Генерировать словарь.

Но поскольку в этом проекте нет словаря, то и рассматривать данный момент не будем. Хотя в версии **7.1** появилось важное отличие: возможность использовать в таблицах символ разрыва строки **\n**, что позволяет добавлять в словарь слова или части слов, разделённые CLRF.

Возможно, будет приложение, и там частично рассмотрены ещё несколько проектов, в том числе и с MTE. Также, контекстное меню данного элемента содержит пункты:

Ctrl+F7 — Перезагрузить оригинальный текст (Перезагрузить текст). Обновляет окно с оригинальным текстом.

Ctrl+F8 — Вернуть оригинальный текст (Вернуть исходный текст). Загружает текст из окна «**Оригинальный текст**» в окно «**Редактируемый текст**». Весьма полезная опция.

Shift+Ctrl+F6 — Словарь в таблицу. Добавляет словарь в выбранную таблицу. Опция активна только когда настроены «**Параметры словаря**».

Загрузить список адресов для вставки текста... — в последующих версиях этот пункт переименован и перемещён в контекстное меню элемента «**Диапазоны адресов**». Загружает адреса из файла.

F9 — Пересчитать пойнтеры и вставить текст в ROM (Пересчитать и вставить). В версиях **7.0.0.85** и **7.1.1.10** есть баг (когда кнопка активна, а указателей уже нет), в **7.1.1.17** он частично исправлен (уже нет зависания, но всё же, если указать диапазон адресов для записи текста, то в выходном файле этот диапазон будет заполнен нулями).

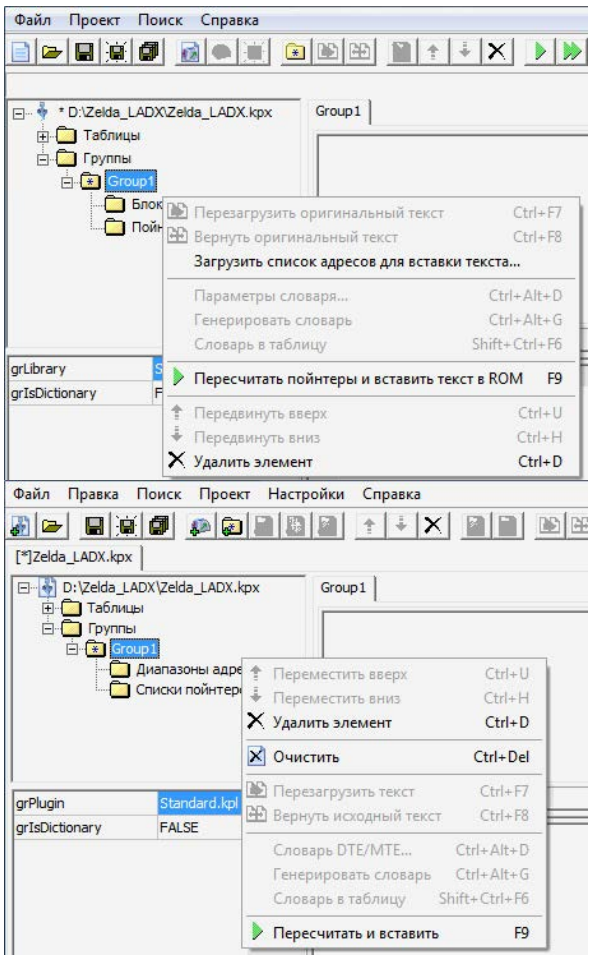


Рисунок 15. Контекстное меню элемента «Group1»

«**Список текстовых элементов**» реагирует на создание новой группы появлением вкладки с её названием. В версии **7.0** есть только подсветка, а вот в **7.1** появилось удобное контекстное меню, но подсветка исчезла. Групп, также как проектов и таблиц, может быть много, соответственно, версия **7.1** в плане работы с группами удобнее. В данном проекте групп будет **6**, т.к. тексты, включая титры, разбиты на **6 частей** (см. **Рисунок 6**).

Блоки для текста (**Диапазоны адресов**)

Тут задаются границы, в пределах которых будет находиться записываемый текст для данной группы. Подобных диапазонов может быть несколько. Обычно текст пишется в тот же диапазон что и оригинальный, но бывает необходимость перенести переведённый текст, например, в конец рома, если позволяют указатели. Здесь, казалось бы, ничего сложного нет, но может возникнуть ошибка с неправильно введённым значением.

Возьмём пример, приведённый автором в версии **7.1** (для **7.0** всё аналогично): **002A3C00-002AFFFF**. Если попытаться ввести этот диапазон (**Ctrl+Alt+A** и вставить), то увидим ошибку, в которой данный пример и показан. Если этот же диапазон записать в файл и «Загрузить список адресов для вставки текста...» (**Ctrl+Alt+O** (Добавить из файла) или Ctrl+Alt+L (Загрузить из файла) — здесь разницы в командах нет), то диапазон будет добавлен как **02A3C00-02AFFFF**. Т.е. для ввода в программе нужно использовать 7 разрядов на число для записи диапазона, а когда программа берёт диапазон из файла, работает и с 8. В данном проекте в каждой группе будет по 1 диапазону.

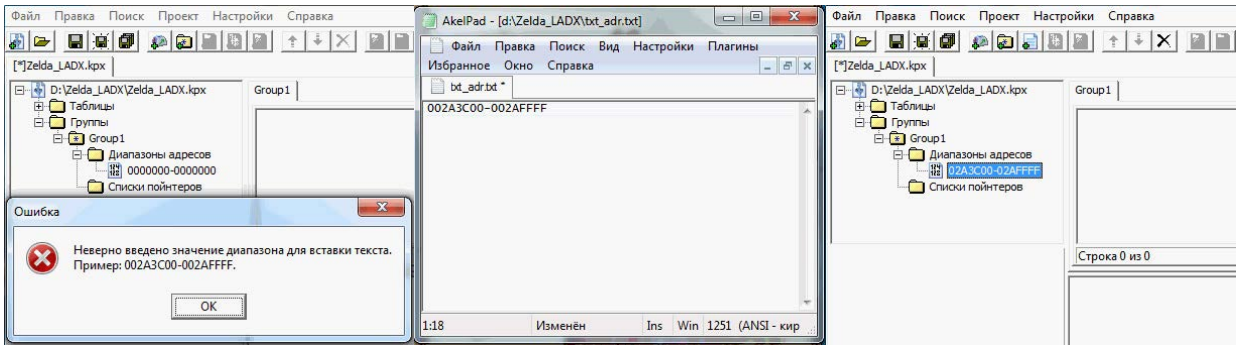


Рисунок 16. Добавление элемента

ПОЙНТЕРЫ (указатели)

Пойнтеры (**Списки пойнтеров**)

Нажимаем **Ctrl+Alt+A**, и начинается самое интересное.

List1

Самое запутанное, неоднозначное и непонятное — тоже тут. Вкладка «**Параметры проекта**» при выборе элемента **List1** отличается для каждой рассматриваемой версии количеством, именами и расположением элементов. Контекстное меню **List1** довольно большое, и тоже отличается в каждой версии.

Правда, системный подход есть и здесь. Все параметры можно разделить на три группы: со списком выбора, числовые и логические.

ptInputTable (**ptSrcTable**)

Выбираем таблицу, которую ранее добавили. По ней будем **извлекать** текст в этом листе.

ptOutputTable (**ptDestTable**)

Выбираем таблицу, которую ранее добавили. По ней будем **вставлять** текст в этом листе.

ptDictionary

Здесь можно выбрать группу, которая является словарём, когда таковая имеется.

ptPointerSize

Тут выбираем размер указателей (в байтах): 0, 1, 2, 3, 4. В версии **7.1** появилась ещё пара типов: **1(signed)** и **2(signed)**. Значения таких пойнтеров могут быть отрицательными.

Числовые параметры могут быть записаны в *десятичной* либо *шестнадцатеричной* системе счисления, во втором случае нужно писать префикс **h**. В версии **7.0** имеются недоработки по переводу одних значений в другие: не всё переводится автоматически. В **7.1** это исправлено.

ptReference (ОЧЕНЬ ВАЖНЫЙ ПАРАМЕТР!)

Это число, которое устанавливает однозначное соответствие указателя (значение указателя) и текста (фактический адрес текста). Т.к. указатели могут быть относительными, этот параметр позволяет считать любые указатели как абсолютные. Такая точка отсчёта или **разница смещений** — кому какая терминология ближе.

Записывается в ***шестнадцатеричном*** виде. Если параметр не задан или задан неверно, нормально отобразить текст в проекте вряд ли получится! В некоторых играх (например, Fragile Dreams) каждый указатель такой точкой отсчёта считает адрес идентификатора строки (4 байта перед самим указателем). Т.е. даже это не панацея...

ptInterval

Когда указатели идут блоком, но после каждого из них есть дополнительные байты, идентифицирующие строку или указывающие её размер, задействуется этот элемент. В версии **7.1.1.17** могут быть отрицательные интервалы. Записывается в ***десятичном*** виде.

ptShiftLeft

Сдвиг, может быть в диапазоне от **0** до **31**. Видимо, бывают указатели, сдвинутые на какое-то количество бит. Записывается в ***десятичном*** виде.

ptMultiply (**ptAlignment**)

Этот параметр позволяет показать, что указатели выровнены, т.е. кратны какому либо числу. В тексте, в таком случае, после знака окончания строки добавляются нули. Здесь диапазон от **1** до **8**. В версии **7.1** записывается в ***десятичном*** виде.

ptAlign (**ptCharSize**)

Этот параметр один из самых забываемых. Когда в игре используется двухбайтовая кодировка текста, нужно поставить **2**. Вспоминаю только в случае выявления ошибок. Диапазон значений от **1** до **8**. В версии **7.1** записывается в ***десятичном*** виде.

ptStringLength

Здесь можно задать длину строки, например, когда нужно вытащить текст без использования указателей. Диапазон от **0** до **32768**. Записывается в ***десятичном*** виде.

Почти все логические параметры по умолчанию не активны.

ptMotorola (**ptBIG_ENDIAN**)

Порядок байтов. Вечный спор тупоконечников с остроконечниками.

ptSigned (**ptAutoPtrStart**)

Эта опция создана для автоматического дополнения **ptReference** для каждого пойнтера, когда точка отсчёта является началом одного пойнтера. Часто встречается в играх на Sega Mega Drive.

Формула:
СМЕЩЕНИЕ СТРОКИ В РОМЕ = ptReference + ЗНАЧЕНИЕ ПОЙНТЕРА + СМЕЩЕНИЕ ПОЙНТЕРА В РОМЕ

ptPunType (**ptSplittedPtrs**)

Когда указатель разбит на части. В версии **7.1.1.17** изменено поведение этого свойства в зависимости от размера указателя.

ptAutoStart (**ptAutoReference**)

Можно вместо указания **ptReference** активировать эту опцию, когда адрес начала блока указателей (первого указателя) и **ptReference** совпадают.

ptSeekSame

Оптимизация такая. Когда в тексте несколько одинаковых строк, то при пересчёте они будут вставляться в одно и то же место, т.е. останется одна. По умолчанию эта опция включена, но иногда её требуется отключать. Например, я отключаю при проверке правильности составления проектов.

ptPtrToPtr

Указатели, указывающие на указатели, которые указывают на текст. Я пытался сделать проект на GBA с такими указателями, но не получилось, т.к. там была группа указателей, каждый из которых указывал на другую группу указателей. Т.е. у меня здесь лыжи явно не едут. Ежели здесь можно указывать только на один указатель, указывающий на текст, то непонятно, где это может использоваться.

ptNotInSource

Не извлекать текст из исходного файла.

ptSNESlorom

Формат указателей SNES Lo-ROM.

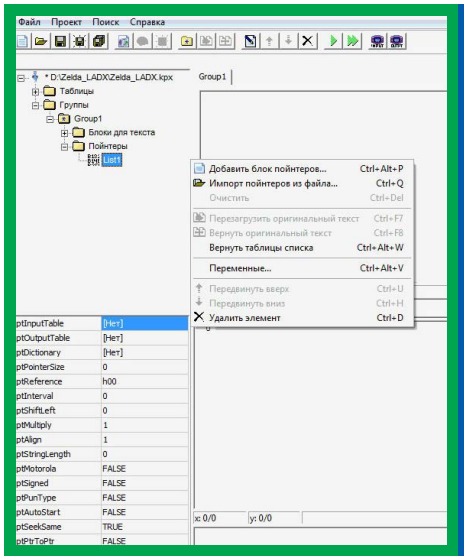


Рисунок 17.1.
«List1» в версии **7.0.0.85**

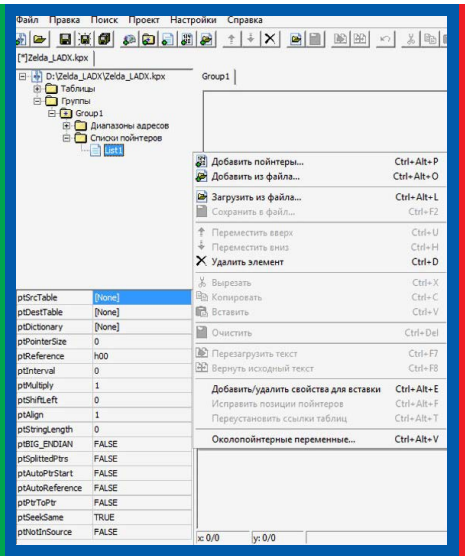


Рисунок 17.2.
«List1» в версии **7.1.1.10**

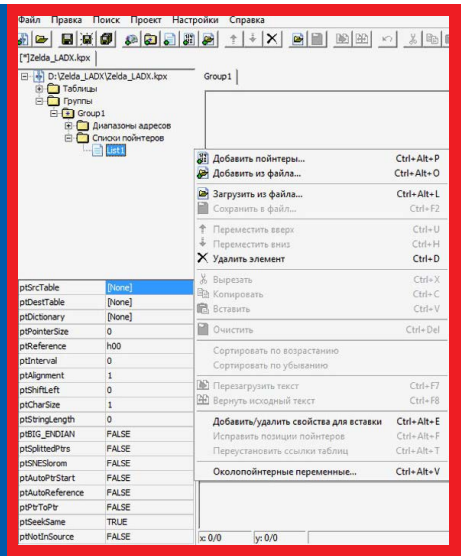


Рисунок 17.3.
«List1» в версии **7.1.1.17**

КОНТЕКСТНОЕ МЕНЮ

Добавить блок пойнтеров... (**Добавить пойнтеры...**) (**Ctrl+Alt+P**).

Если в **ptInputTable** (**ptSrcTable**) не выбрана таблица, будет ошибка.

Если **ptPointerSize=0**, то будет предоставлена возможность указать адреса на блок текста, который необходимо извлечь без применения указателей.

Если **ptPointerSize** отличен от **0**, то нужно указать адрес первого и адрес последнего указателя. Писать эти значения нужно с префиксом **h** в *шестнадцатеричной* системе счисления, либо в *десятичной без всяких префиксов*.

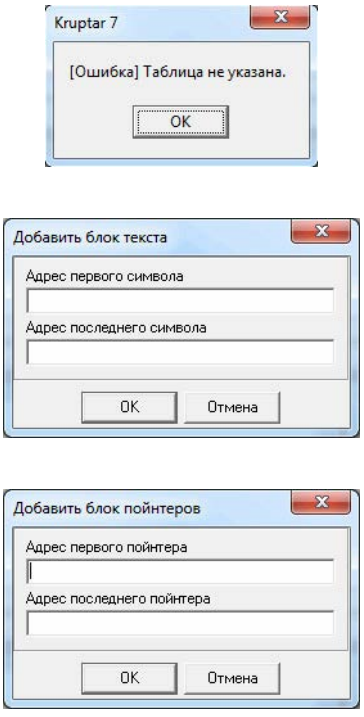


Рисунок 18.
Три варианта сообщения в **7.0.0.85**

Импорт пойнтеров из файла... (**Ctrl+Q**)

Позволяет загружать адреса указателей из текстового файла. Это бывает нужно в том случае, когда они расположены не таблицей, либо когда в таблице есть указатели-пустышки, и нам не нужно работать с ними.

На **Рисунке 6** в столбце **H** считаются адреса указателей. Впоследствии этот столбец можно просто скопировать в текстовый файл и использовать для импорта.

В версиях **7.1** эта опция была разделена на две:

Добавить из файла... (**Ctrl+Alt+L**)

Добавляет из файла адреса указателей, которых нет в списке. Одинаковые не добавляются.

Загрузить из файла... (**Ctrl+Alt+O**)

Загружает список из файла. Также появилась возможность сохранять список указателей в файл (чего нет в **7.0**) и стандартные команды редактирования:

Сохранить в файл... (**Ctrl+F2**)

Очистить (**Ctrl+Del**)

Удаляет все элементы текущего списка *List*.

Перезагрузить оригинальный текст (**Перезагрузить текст**) (**Ctrl+F7**)

Обновляет окно «**Оригинальный текст**» с использованием текущей таблицы.

Вернуть оригинальный текст (**Вернуть исходный текст**) (**Ctrl+F8**)

Копирует содержимое окна «**Оригинальный текст**» в окно «**Редактируемый текст**». Т.е. в окне с переводом мы вновь увидим оригинальный текст. Оригиналом, естественно, может быть и русский...

Если проект при этом не сохранять, ***.txt** файл в его архиве останется таким же, как до применения этой функции. От такой же команды в *Group* отличается лишь тем, что изменения коснутся только текущего листа, а не всей группы, которая может содержать их большое количество.

Вернуть таблицы списка (**Ctrl+Alt+W**)

Каждый элемент списка *List*, т.е. каждая строка, может быть извлечена (вставлена) по своей таблице. При изменении таблицы для всего *List* таблицы в каждой строке остаются такими же, как были при первом извлечении, либо установлены впоследствии. Чтобы применить это глобальное изменение к каждой строке, и существует эта команда.

В моём понимании программы, таблицы нужно менять только в элементе **Таблицы**, поэтому я создаю их избыточное количество, если проект того требует, но зато, если что-то меняется, то меняется сразу и везде. Возможно, это связано с непониманием некоторых тонкостей работы с каждым указателем-строкой.

В **7.1** эта опция была переименована и сгруппирована с функционально похожими собратьями для указателей:

Добавить/удалить свойства для вставки (**Ctrl+Alt+E**)

Добавляет/удаляет на панель «**Параметры проекта**» довольно много свойств указателей, по которым текст можно вставлять:

ptDestPtrSize
ptDestReference
ptDestStrLen
ptDestMultiple (ptDestAlignment)
ptDestAlign (ptDestCharSize)
ptDestShiftLeft
ptDestAutoPtrStart
ptDestSplittedPtrs
ptDestSNESlorom

То есть, имеется возможность очень многое изменить относительно оригинала.

Переменные (**Околопоинтерные переменные**) (**Ctrl+Alt+V**)

Работает только в случае ненулевого значения **ptInterval**, т.е. в проекте, где адреса указателей импортируются из файла, видимо, работать не будет. Добавляет возможность автоматически пересчитывать, как минимум, длину строки. Не скажу, что работает идеально, но добиться правильного результата пересчёта всё-таки можно.

В некоторых играх длина строки может учитывать символы окончания, может не учитывать, ещё бывает выравнивание окончаний до чётной длины. Это уже можно сделать плагинами.

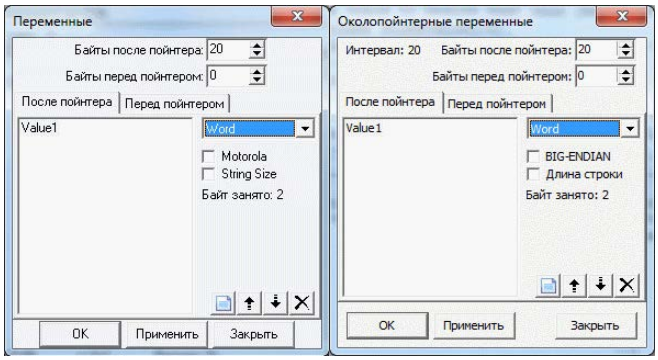


Рисунок 19. Переменные

Есть ещё пара функций сортировки указателей в элементе *List*, они появились в версии **7.1.1.17**:

Сортировать по возрастанию
Сортировать по убыванию

Концепция их применения для меня остаётся загадкой. Т.к. в конечном итоге в этой версии присутствует специальная сортировка текста (видимо, по размеру строк), что лишает возможности проверить правильность составления проекта путём сравнения выходного файла с оригинальным и сводит на нет ВСЕ её плюсы.

СТРОКИ И УКАЗАТЕЛИ. ЧТЕНИЕ И ПРАВКА

Настало время посмотреть, где все элементы сходятся.

На **Рисунке 20** показана завершающая фаза создания проекта со стандартным плагином (**Standard.kpl**) и отключённым **ptSeekSame** (мне это необходимо для проверки правильности составления проекта).

Group1 переименовываем в **Block_20**, выбираем стандартный плагин, указываем диапазон адресов: **0051934-0053FFF**, выбираем размер пойнтеров **ptPtrSize=2**, указываем **ptReference=h4C000**, устанавливаем **ptSeekSame=False**, добавляем адреса пойнтеров: **h00070001** и **h00070147**.

Видим, что «**Список текстовых элементов**» содержит **164** строки, окна «**Оригинальный текст**» и «**Редактируемый текст**» содержат выделенную в списке строку **000001**, а «**Параметры проекта**» — свойства последней строки: исходную и выходную таблицу для этой строки, адрес указателя на эту строку, **ptReference** (в **7.1** добавлены выходные адрес указателя и ptReference) и последний логический параметр, **pnFixed**. Что он делает в версиях **7.0.0.85** и **7.1.1.10**, для меня так и осталось загадкой, но в текстовом файле для **7.1.1.10** написано: *«Исправлена вставка при pnFixed=True…»* Если это исключение для **ptSeekSame**, то оно не работает.

Ещё можно заметить, что окна «**Оригинальный текст**» и «**Редактируемый текст**» имеют строчку с информацией:

1. Позиция курсора и размер строки (количество символов). В **7.1** показывает ширину окна, если вдруг нет символа переноса (как раз наш случай), что не есть хорошо. А вот в **7.0** именно количество символов в строке, плюс тут **есть столбец с нумерацией строк, где хорошо видно, закончилась ли строка или просто не поместилась в окне**, и вообще это очень удобная вещь — компонент SynEdit (в принципе, только из-за неё и живёт версия **7.0.0.85**). ;=)

2. Номер строки и количество строк (номер начинается с нуля и никогда не сравняется с числом строк).

3. Размер текста (количество байт). Размер первой строки: **89 байт**.

Ещё можно заметить подсветку кодов и панелей **List1** и **Block_20** в версии **7.0**.

В версии **7.1** нет подсветки кодов, но панели **List1** и **Block_20** (бывшая **Group1**) имеют свои контекстные меню, что, видимо, делает работу с ними несколько более удобной.

По **F9** можно увидеть сообщение о том, что весь текст не уместился, после чего сравнить оригинальный и русифицируемый файл (вставляется всё в данный момент по оригинальной таблице) и выявить ошибки, допущенные при составлении проекта. Они могут быть самыми разными, оставленные разработчиками в том числе, но в данном случае это отключённый **ptSeekSame**. Я отключал его именно для того, чтобы сравнить оригинальный и изменённый файлы, т.к. довольно часто разработчики не пользуются подобным способом оптимизации. Но в данном случае можно увидеть, что после строки о покупке лука у нас идёт её повтор, чего нет в оригинале.

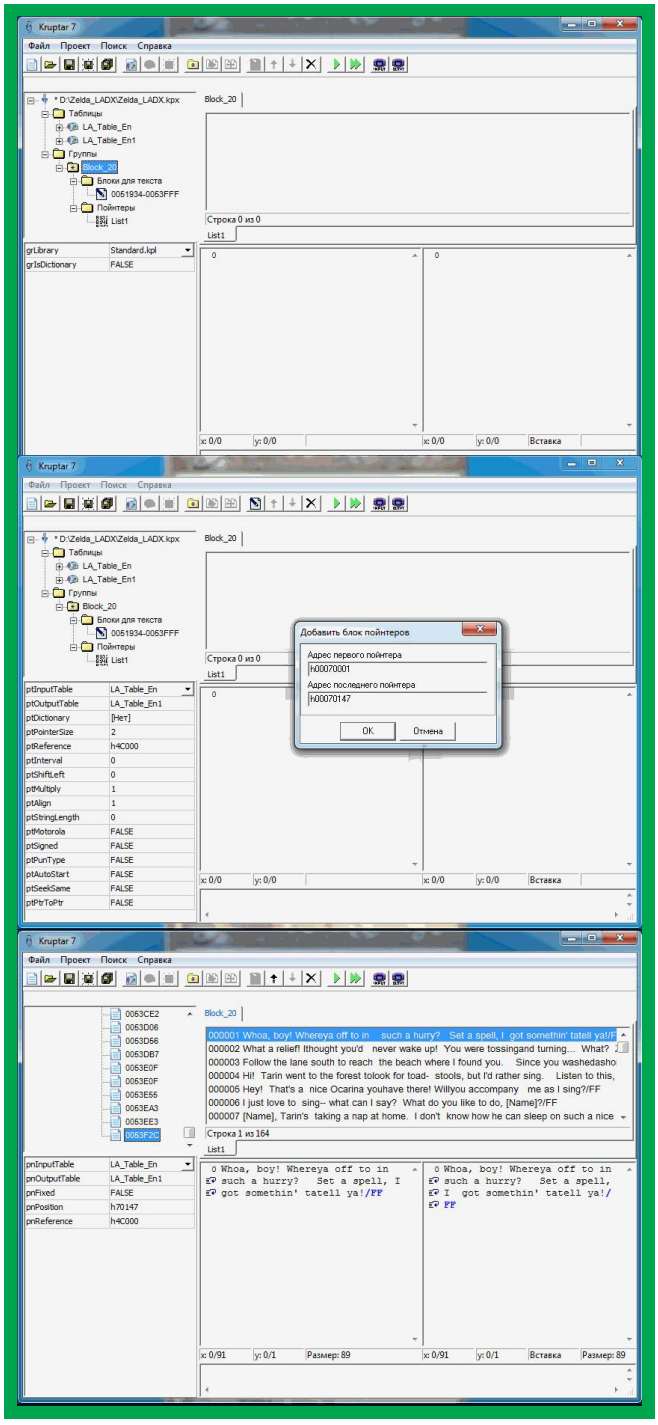


Рисунок 20. Первый блин комом

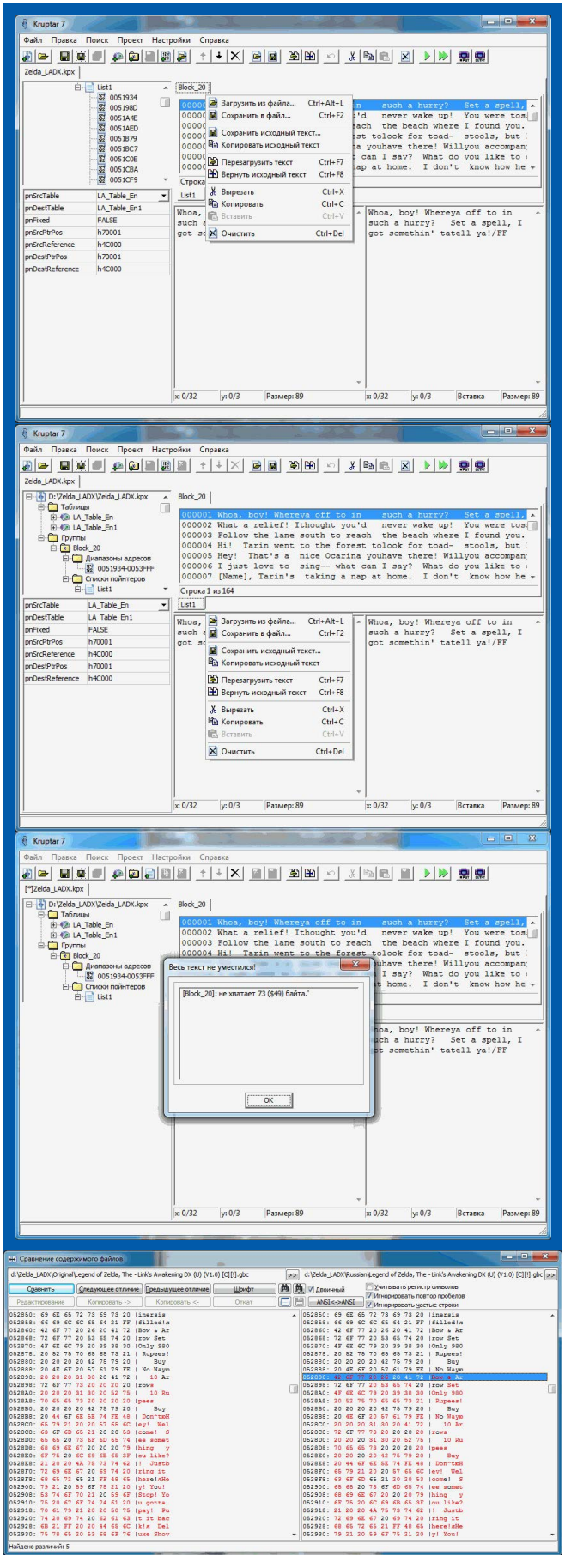


Рисунок 20 (продолжение). Первый блин комом

Ставим **ptSeekSame=True** и повторяем процедуру записи и сравнения.

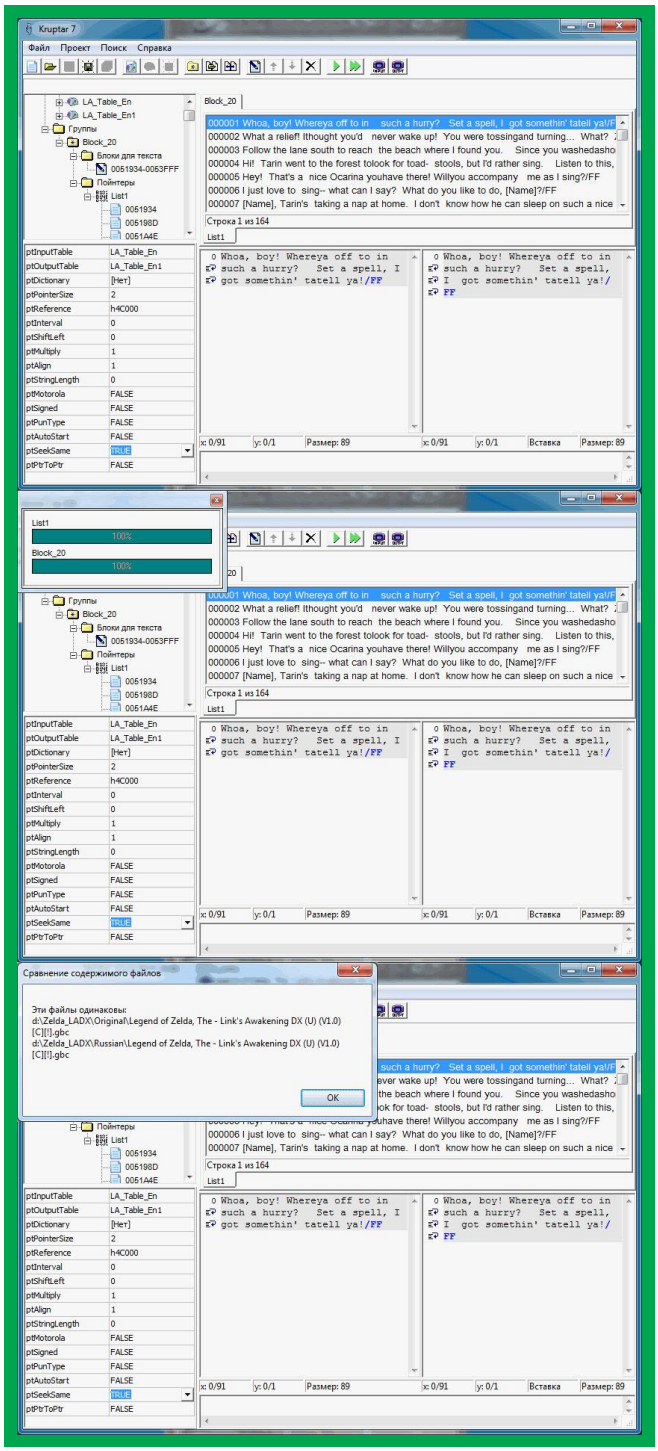


Рисунок 21. Проект составлен правильно и даже работает

Ах, да, если мы попытаемся проделать подобное сравнение в версии **7.1.1.17**, получится такая картинка:

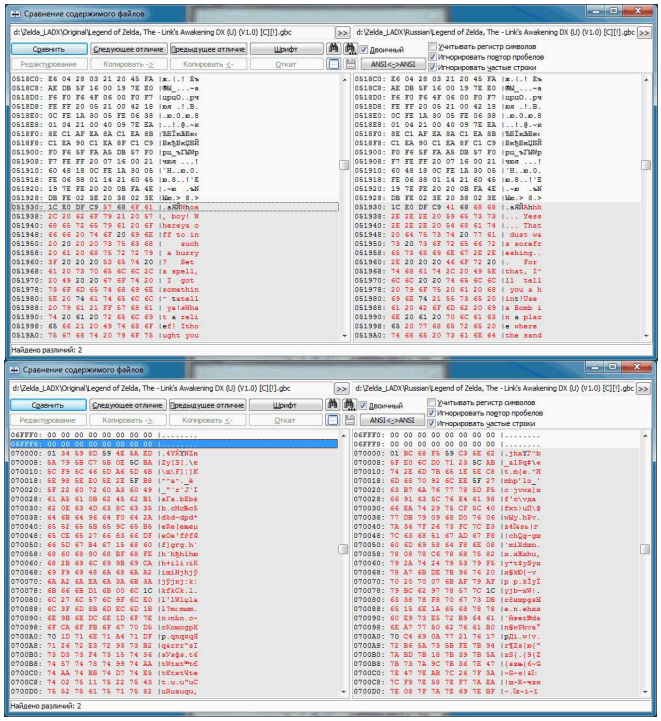


Рисунок 22. Вставка того же блока в версии **7.1.1.17**

То есть, тут остаётся только верить автору и надеяться, что всё сложилось правильно. Как в этой версии находить свои ошибки, я не знаю...

Теперь можно уже было бы приступить непосредственно к процессу перевода, но текст без переноса строки править весьма неудобно. Поэтому здесь чуть забежим вперёд и покажем, что это довольно просто исправить, используя свой плагин вместо стандартного. В нём правилась только функция **GetStrings**.

Сначала очищаем **List1** от того, что получили и выбираем в списке плагинов **Zelda_GB.kpl**. Тут я немножко воспользовался послезнанием. Так как нельзя сменить плагин, не очистив старый лист от старого текста, но нельзя и загрузить только что написанный плагин в уже запущенный Kruptar.

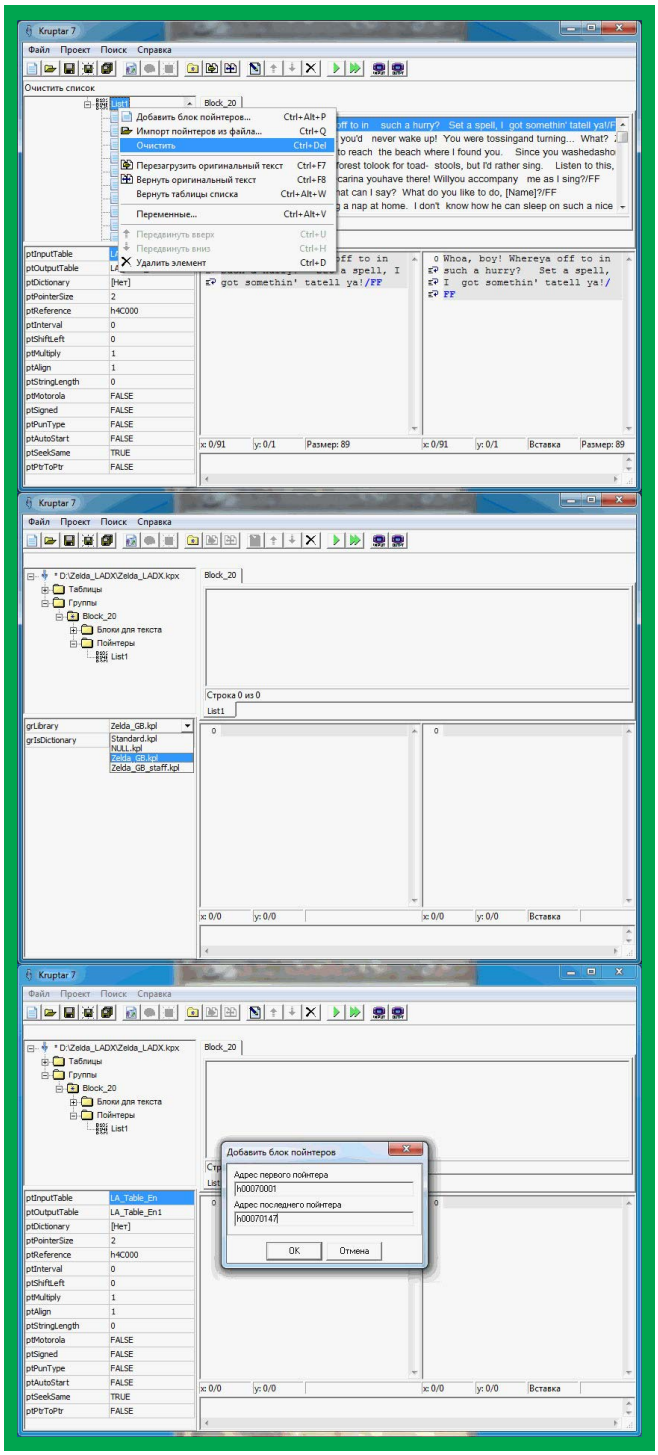
То есть, требуется положить в папку **Lib** плагин **Zelda_GB.kpl** и перезапустить Kruptar. Я же написал плагин до того, как решил написать эту доку, и он уже был в папке **Lib**. Т.е. мне осталось только очистить **List** и выбрать его. Далее опять **Ctrl+Alt+P**, и вот тут нужно было бы писать адреса начального и конечного указателей заново, но они сохранились, т.к. я всего лишь правлю проект и Kruptar не перезагружал.

Видим, что появился нераспознанный код **/01**. Его добавляет плагин в качестве символа разрыва строки. Вносим его в таблицу, которая используется для извлечения скрипта. В таблице для вставки текста **обратно** в POM этот код не нужен, т.к. его не было в оригинале.

```
library Zelda_GB;

uses
  ShareMem,
  Needs;

{$E .kpl}

resourcestring
 SKPLDescription = 'The Legend of Zelda:' + #13#10 +
                   'Link's Awakening';

var
 ROM: PBytes = NIL;
 RomSize: Integer = 0;
 EndsRoot: PTableItem = NIL;
 MaxCodes: Integer = 0;
 Align: Integer = 1;

function Description: AnsiString; stdcall;
begin
 Result := SKPLDescription
end;

function NeedEnd: Boolean; stdcall;
begin
 Result := False
end;

function GetMethod: TMethod; stdcall;
begin
 Result := tmNone
end;

procedure SetVariables(X: PBytes; Sz: Integer; ER: PTableItem; MC, AL: Integer); stdcall;
begin
 ROM := X;
 RomSize := Sz;
 EndsRoot := ER;
 MaxCodes := MC;
 Align := AL
end;

function GetData(TextStrings: PTextStrings): AnsiString; stdcall;
var                              //Функция писателя
 R: PTextString;
begin
 Result := '';
 If TextStrings = NIL then Exit;
 With TextStrings^ do
 begin
  R := Root;
  While R <> NIL do
  begin
    Result := Result + R^.Str;
    R := R^.Next
  end
 end
end;

function GetStrings(X, Sz: Integer): PTextStrings; stdcall;
var                              //Функция читателя
 Cnt: Word; P: PByte; PC: PChar absolute P;
begin
 New(Result, Init);
 with Result.Add^ do
 begin
  P := Addr(ROM[X]);
  Cnt:=0;                        //Счётчик символов. Тип byte мало, т.к. строка
                                 //может быть и длиннее 255, потому Word
  while P^ <> 0 do
    if (P^ = $FE) or (P^ = $FF) then
                   begin Str := Str + PC^; break end     //Тут просто лень
 else begin
     Cnt:=Cnt + 1;
     Str := Str + PC^;
     Inc(P);
if Cnt mod 16 = 0 then Str := Str + #1  //Строку разрываем, когда она достигает
     end                               //длины 16 байт, 01 добавляя в таблицу
 end
end;

exports
 GetMethod,
 SetVariables,
 GetData,
 GetStrings,
 DisposeStrings,
 NeedEnd,
 Description;

end.
```

Код 1. Свой плагин для функции переноса строки

Поэтому просто возвращаем оригинальный текст (**Ctrl+F8**). Обратите внимание, что размер строки в окне «**Оригинального текста**» и «**Редактируемого текста**» отличается: **94** вынимаем и **89** вставляем обратно (т.е. столько же, сколько и без использования этого плагина, см. **Рисунок 20**). Но редактировать текст теперь значительно удобней.



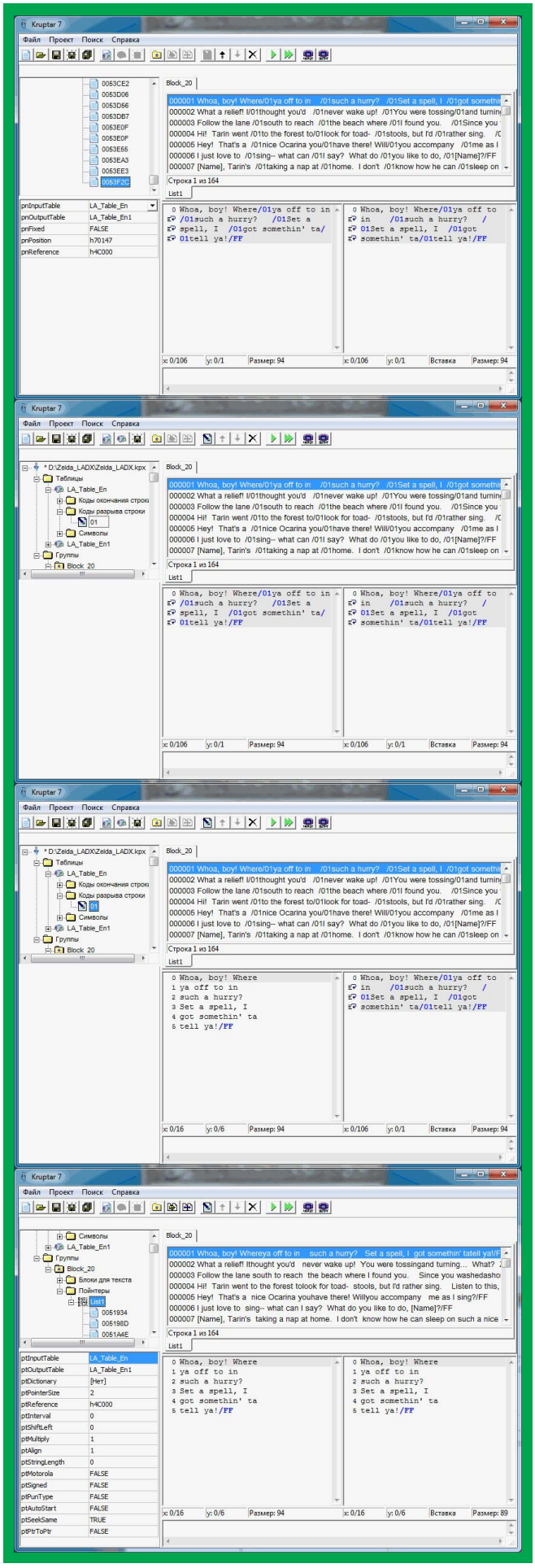


Рисунок 23. Плагин добавляет символ разрыва строки

С остальными блоками всё аналогично, кроме **Block_23_staff**. Но об этом чуть позже.

Рассмотрим ещё раз коды символов. С помощью Нех-редактора были найдены все символы, встречающиеся в скрипте, но не в шрифте. Вот эту разницу и настало время обнаружить.

Для этого была составлена тестовая таблица от **00** до **FD** и по ней по порядку заменён один из первых текстов, встречающихся в игре.

Таким образом были получены все коды и соответствующие им тайлы в шрифте (крестики пришлось нумеровать). Наверняка есть более продвинутые способы поиска, но мне был интересен самый простой.

По наиболее очевидной дельте, находится карта тайлов шрифта (т.к. тут ничего не пожато) и появляется возможность использовать все неиспользуемые символы в целях перевода.

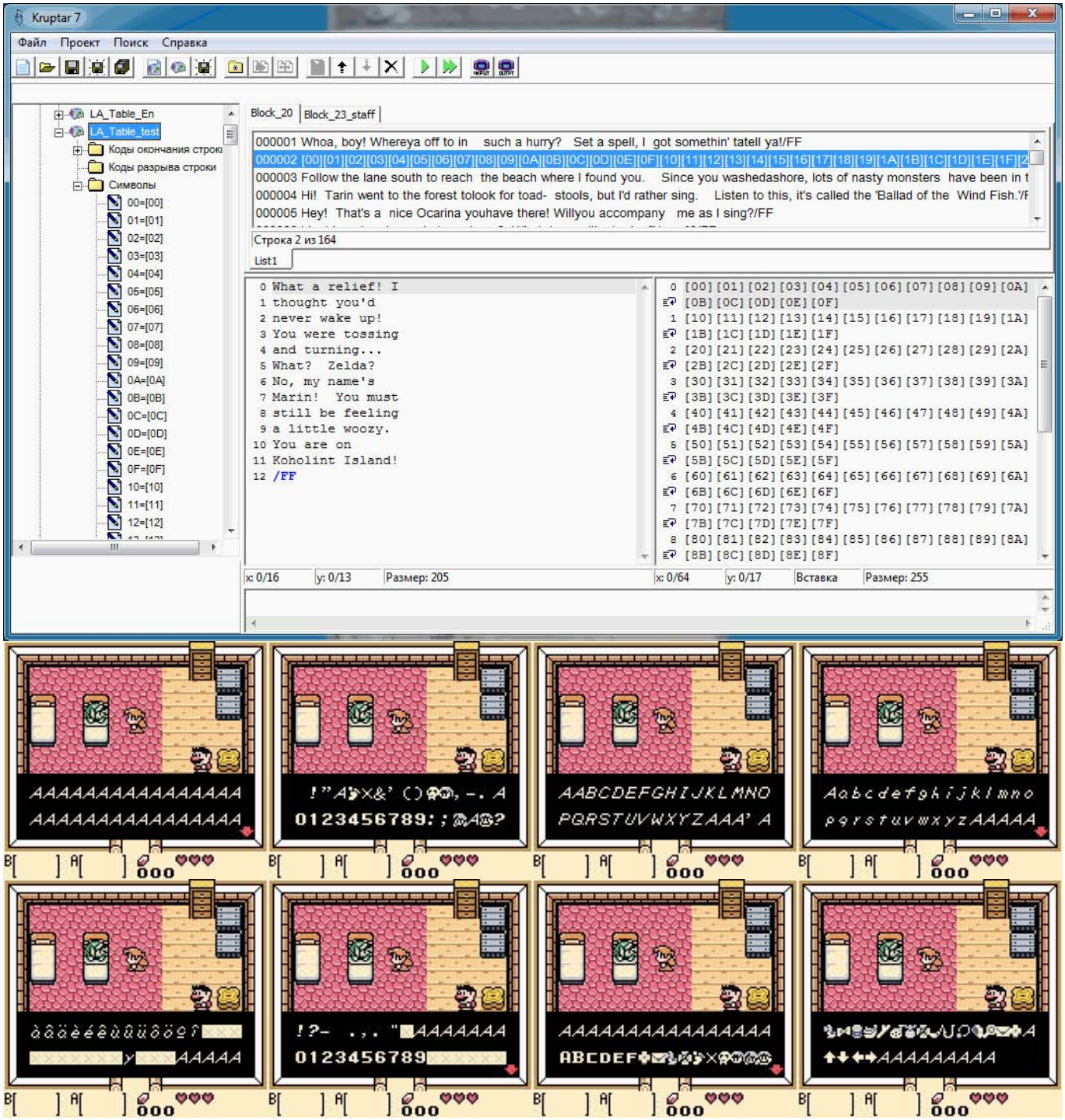


Рисунок 24. Символы и их коды

ЧТЕНИЕ И ПРАВКА БЕЗ УКАЗАТЕЛЕЙ. ТИТРЫ

Для титров тоже использовался плагин, но т.к. в них не было плашки для текста и использовалась вся ширина экрана, строка кратна **18**.

Вся разница в плагине:

```
library Zelda_GB_staff;
...
function GetStrings(X, Sz: Integer): PTextStrings; stdcall;
var                                  //Функция читателя.
 Cnt: Word; P: PByte; PC: PChar absolute P;
begin
 New(Result, Init);
 with Result.Add^ do
 begin
  P := Addr(ROM[X]);
  Cnt:=0;
  while P^ <> $FA do
  begin
   Cnt:=Cnt + 1;
   Str := Str + PC^;
   Inc(P);
   if Cnt mod 18 = 0 then Str := Str + #1
  end
 end
end;
...
```

Код 2. Фрагмент плагина (для титров)

А вот, если счётчик взять размером в **1 байт** и просто ужесточить условие:

```
library Zelda_GB_staff;
...
function GetStrings(X, Sz: Integer): PTextStrings; stdcall;
var                                  //Функция читателя.
 Cnt: Byte; P: PByte; PC: PChar absolute P;
begin
 New(Result, Init);
 with Result.Add^ do
 begin
  P := Addr(ROM[X]);
  Cnt:=0;
  while P^ <> $FA do
begin
   Cnt:=Cnt + 1;
   Str := Str + PC^;
   Inc(P);
   if Cnt mod 18 = 0 then begin Str := Str + #1; Cnt:=0 end
  end
 end
end;
...
```

Код 3. Фрагмент плагина (для титров) 2

Тут символ окончания строки **FA**, но мы до него всё равно не дойдём... Здесь на двух примерах просто показана разница, как можно его задать в плагине (можно не указывать в таблице). Возможно, это кому-то пригодится.

Отличие данной записи от той, что находится в плагине **Zelda_GB.kpl**:

```
...
if (P^ = $FE) or (P^ = $FF) then
begin Str := Str + PC^;
break
end
...
```

Код 4. Отличия

Отличие состоит в том, что текст просто перестанет читаться, встретив этот символ, а сам он записан не будет. Если необходимо записать символ конца строки, а это нужно практически всегда, правится функция **GetData**. В случае с **FA** это могло выглядеть так:

```
...
function GetData(TextStrings: PTextStrings): AnsiString; stdcall;
var                                  //Функция писателя.
 R: PTextString;
begin
 Result := '';
 If TextStrings = NIL then Exit;
 With TextStrings^ do
 begin
  R := Root;
  While R <> NIL do
  begin
   Result := Result + R^.Str + #250;          //Вся правка!
   R := R^.Next
  end
 end
end;
...
```

Код 5. Правки

Тогда знака конца строки в проекте даже видно не будет. Что, собственно, и реализовано в плагине **NULL.kpl** для символа **00**.

Собственно, сам блок титров вынимается, как и предыдущий, за исключением того, что **ptPointerSize=0**, а вместо адресов первого и последнего указателей записываются адреса первого и последнего символов текста (см. диапазон адресов для блока текста). В версии **7.0** нужно обязательно указать **ptStringLength** (что не всегда удобно). Если этого не сделать (т.е. оставить «0»), Kruptar примется искать ответ на ***«Самый Главный Вопрос Жизни, Вселенной и Вообще»*** (попросту зависнет). Если же **ptStringLength** меньше разницы между концом и началом текста, то строку порежет на кусочки.

В данном случае символ окончания строки был сознательно проигнорирован в таблице, но он есть в плагине, т. е. можно выбрать больший диапазон для текста и увидеть, где строка закончится. На самом деле, в этом режиме строки можно вынимать и с учётом символа-терминатора и без него, регулируя это границами диапазона.

В версии **7.1** баг («деление на 0»?) исправлен.

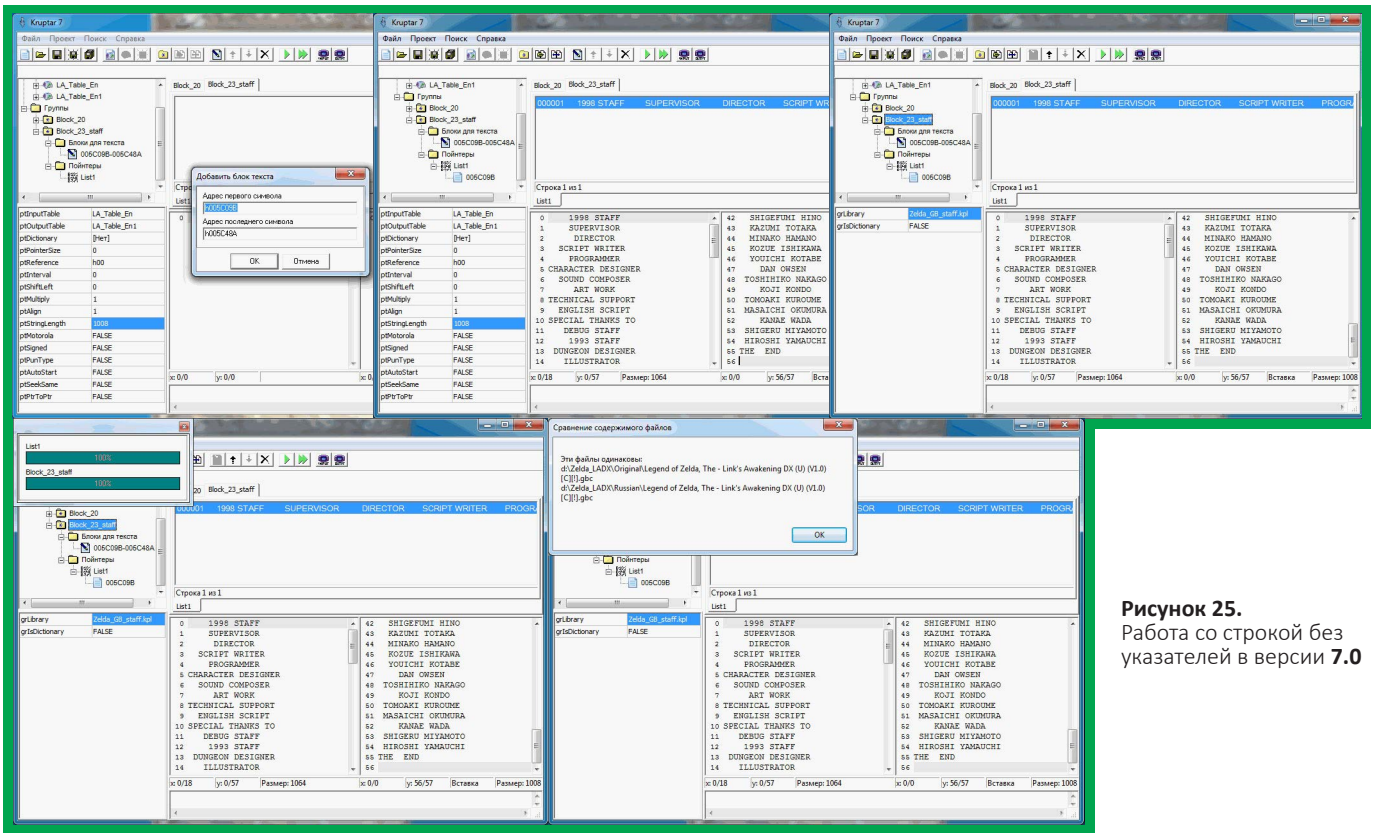


Рисунок 25.
Работа со строкой без указателей в версии **7.0**

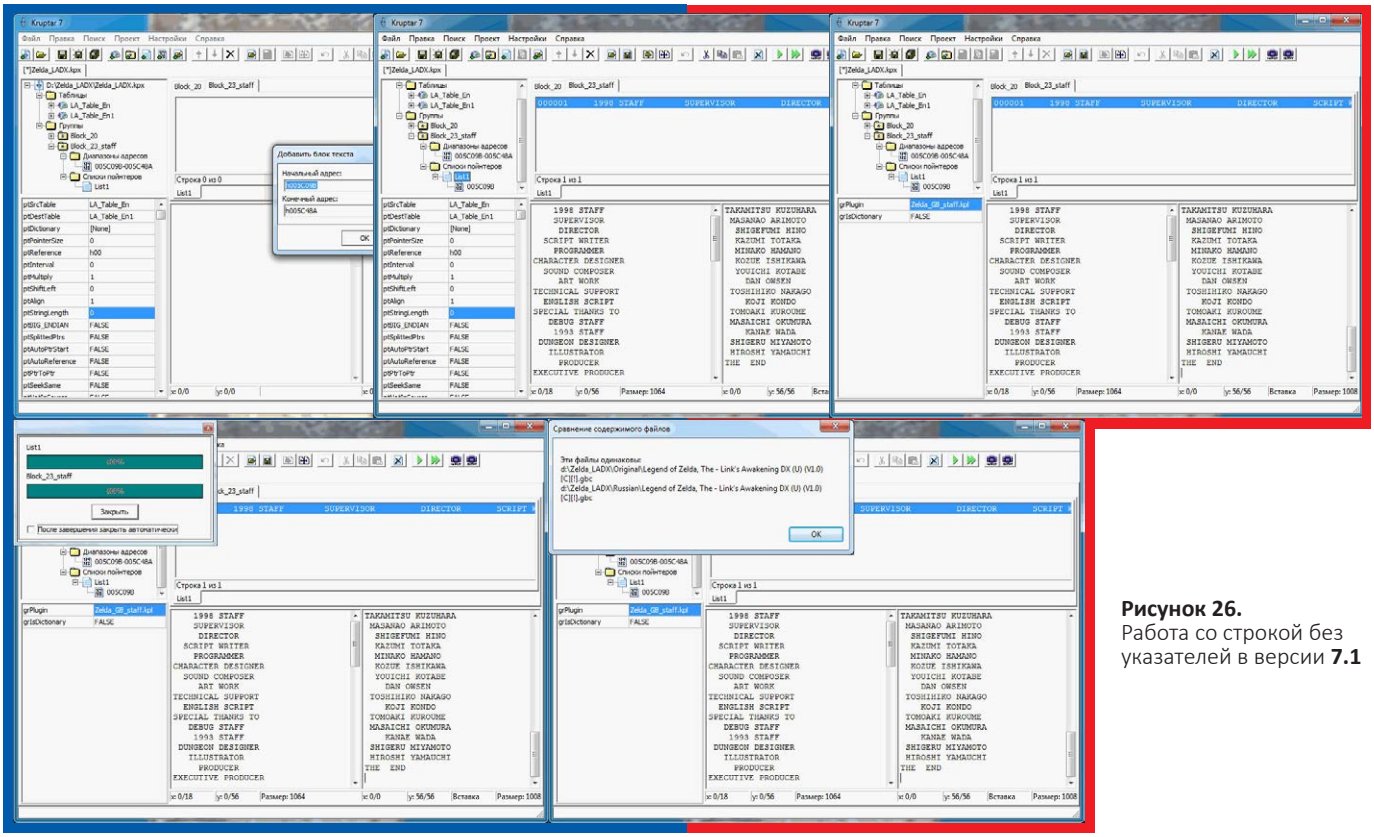


Рисунок 26.
Работа со строкой без указателей в версии **7.1**

ПРОВЕРКА ПРОЕКТА

Я обычно делаю проверки, сравнивая оригинальный и изменённый файлы, дважды.

1. Для первого блока в самом начале, выявляя все возможные свои мелкие, но неприятные ошибки, коими могут быть неверно понятые значения в разбираемом скрипте, либо обычная невнимательность: невыставленное выравнивание указателей до какой-либо кратности (в текст добавляются нули), вкл.\выкл. **ptSeekSame**, неправильно определённые диапазоны адресов для вставки текста, разница в таблицах.

2. В итоге, когда уже извлечён весь скрипт. Опять же, проверяется внимательность, иногда находятся некоторые моменты, о которых до этого информации не имелось. Например, здесь в одной строке разработчики, видимо, удалили лишнюю точку, заменив ее терминатором **FF**. А строка *«IN SOIL SLEEPS SECRETS, BENEATH YOUR SOLES...»* по версии разработчиков имела иммунитет к **ptSeekSame**. В Kruptar я такой возможности не обнаружил и пришёл к использованию замены одного байта в подобных случаях. Это даёт возможность закончить проверку с минимальным количеством различий. (см. **Рисунок. 27**).

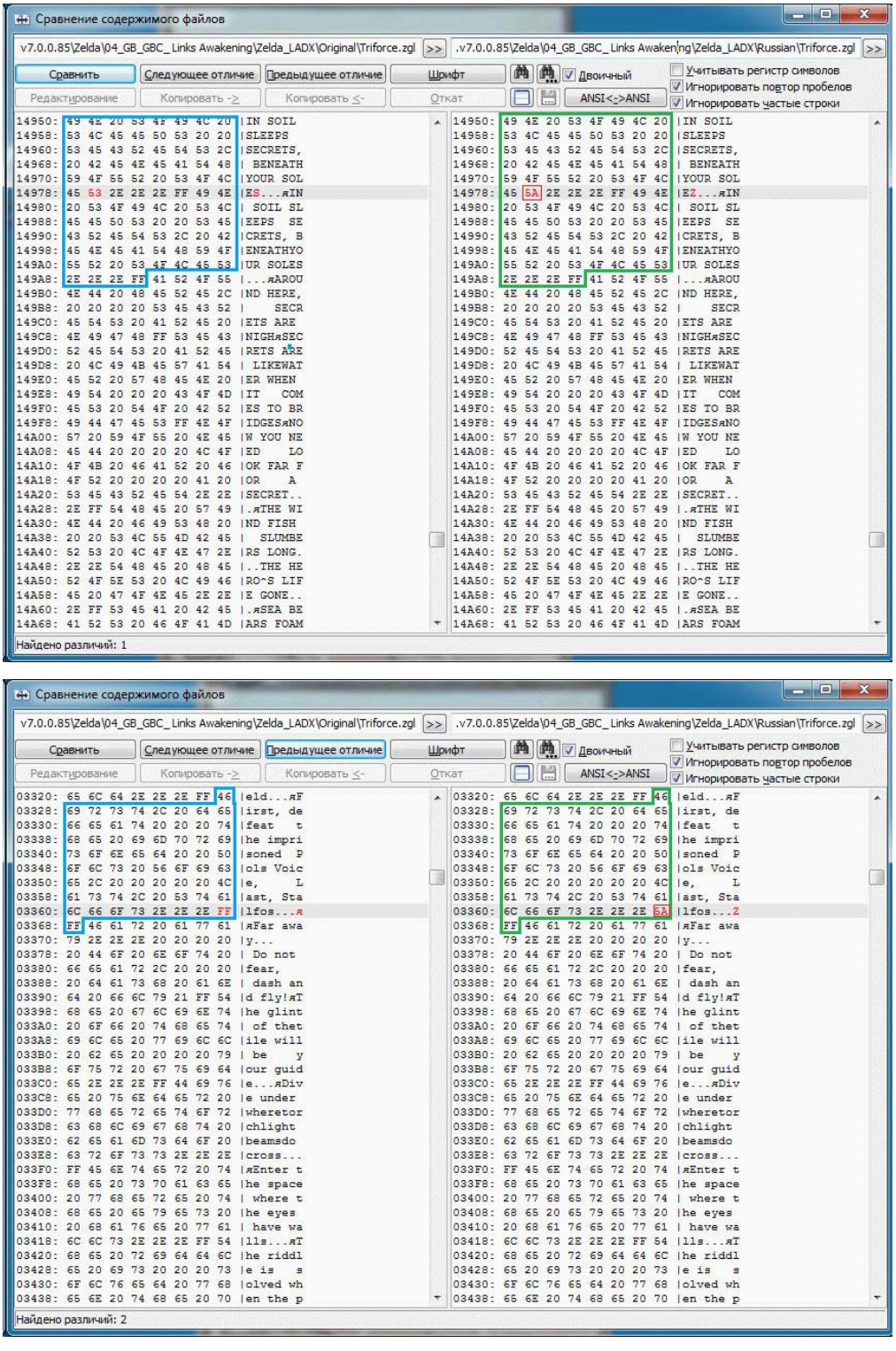


Рисунок 27. Последняя проверка правильности составления проекта

Этот рисунок старый, и делался тогда, когда про данный документ мыслей ещё не было. Проект составлялся для вырезанных текстовых блоков (ну, и указателей, конечно), соединённых в один файл, а не для POMa. Такой способ применяется мной довольно часто во время исследования, а чтобы потом заново не пересчитывать всё, если понадобится сделать проект для POMa, необходимые циферки считаются и хранятся в электронных таблицах.

Подумал немножко и решил проверить, можно ли сделать иммунитет к **ptSeekSame**, разделив лист на два. Оказалось, что нет... Но появилась картинка с различиями до замены «SOLES» на «SOLEZ».

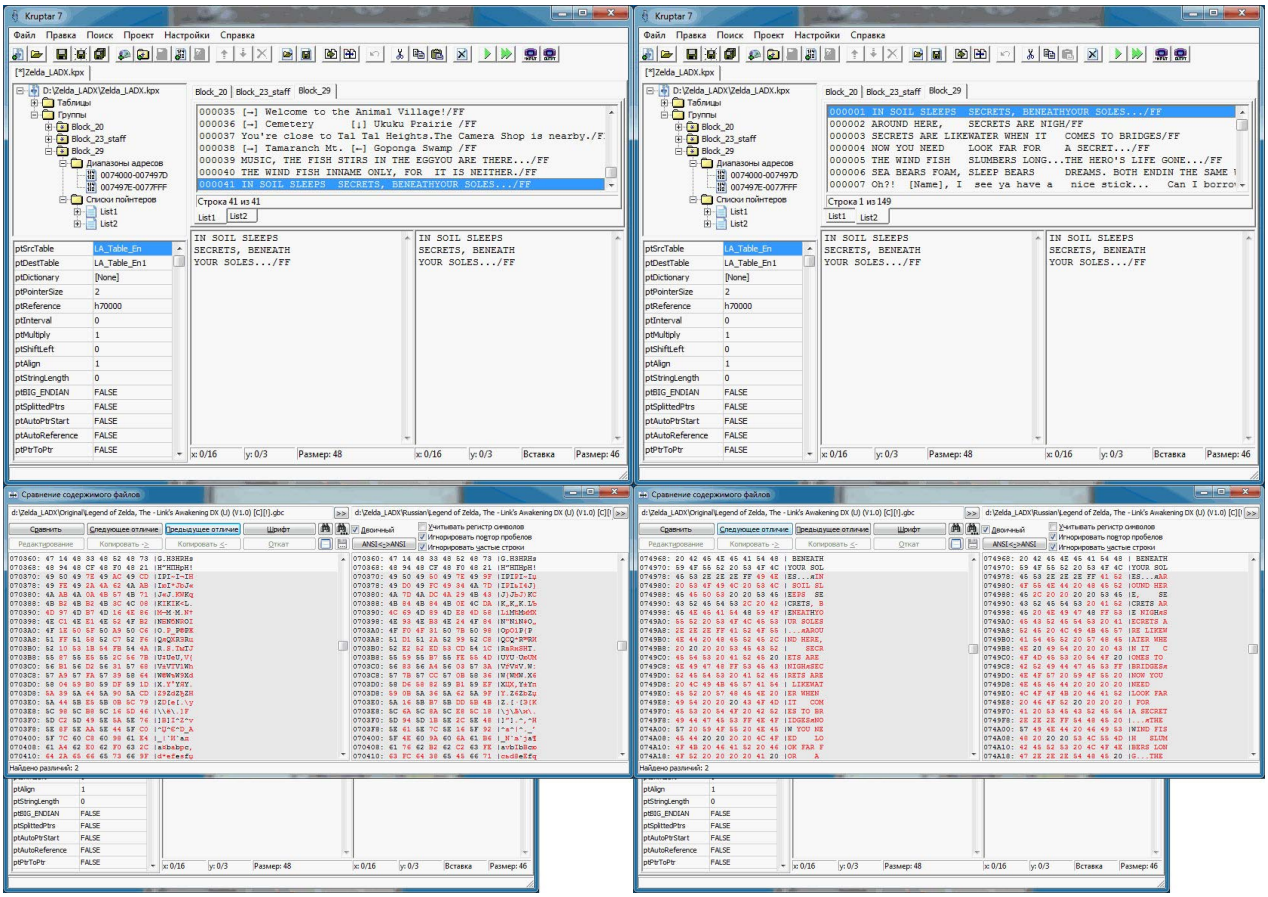


Рисунок 28. Те же «секреты», но до правки «SOLES» --> «SOLEZ».

На этом, наверное, данный раздел руководства (основные возможности и баги программы) можно считать более-менее рассмотренным. Документ и так получился довольно обширный.

Дополнительные возможности (плагины), надеюсь, будут лаконичнее (читайте о них далее в разделе «**ПЛАГИНЫ**».

Полезные ссылки (на версии программы в том числе) можно увидеть в конце данного руководства в разделе «**ССЫЛКИ**».

5. ПЛАГИНЫ. ОБЩАЯ ИНФОРМАЦИЯ И ПРИМЕРЫ

Документация, которую написал ***Griever***, была для конкретного случая запаковки. Но прочитать её всё равно полезно:

magicteam.net/index.php?page=documents&show=Написание плагинов для Kruptar 7

Плагины же бывают разными. Я попытаюсь систематизировать разновидность одного типа. Он не сложный, но очень неприятно, когда проект не работает из-за такой мелочи, как неверно определённый символ разрыва строки. Такое случается довольно часто.

Специальные коды в играх иногда содержат довольно много символов, и если среди них есть тот, что в таблице обозначен как конец строки, Kruptar обрывает извлекаемую строку на этом коде, и часть данных теряется. Чтобы этого избежать, люди придумали вносить подобные коды в исключения при чтении скрипта (см. плагин ***Griever***'а для Zelda: Twilight Pprincess):

magicteam.net/forum/index.php?topic=298.msg12854#msg12854)

На самом деле, гораздо проще и правильнее не каждый такой код вносить в исключения, а вычитать из обработки сразу целые семейства кодов. Этому посвящены две темы на форуме группы MagicTeam:

magicteam.net/forum/index.php?PHPSESSID=220825bf73399ee4a29f55f095c895d6&topic=21.0

magicteam.net/forum/index.php?topic=298.0

Необходимый навык, правильное распознавание контрольных кодов, приходит самостоятельно вместе с опытом: работа с WildCard в Hex-редакторе, например *(в разных редакторах это реализовано по-разному; наибольшую свободу действий даёт редактор, в котором можно заменять на символ WildCard (Joker) ниббл (полубайт), а не байт целиком (в HexWorkshop это «?»))*, или более продвинутые методы.

При составлении таблицы (способ описан в предыдущем разделе) выясняется, нужен ли Kruptar'у плагин или нет. Если он-таки нужен, то понадобится Delphi (Lazarus не пожелал обрабатывать ассемблерные вставки из Needs.pas). Но если очень хочется, можно и на Си писать:

romhack.github.io/doc/kruptarPlugins

Я стараюсь вносить в плагин все найденные типы кодов, вне зависимости от того, содержат они символ разрыва или нет. Это позволяет исключить ошибки невнимательности, т.е. если что-то было пропущено, мы это увидим в проекте, но проект в любом случае будет работать верно.

Рассматривать будем только функции **GetData** и **GetStrings**, т.к. меняются только они и название плагина.

Вот пример плагина для однобайтовой кодировки (игра **Zelda: Ocarina of Time (N64)**) и фрагмент таблицы:

```
0540=[/C]
0541=[Red]
0542=[Green]
0543=[Blue]
0544=[l_Blue]
0545=[Pink]
0546=[Yellow]
0547=[Black]
0602=[sp_01]
0603=[sp_02]
077072=[07_200]
0C0A=[delay_1]
0C14=[delay_2]
1100A0=[11_223]
123880=[snd_04]
1300=[icon_01]
1301=[icon_02]
1302=[icon_03]
1303=[icon_04]
1400=[/slow]
1401=[slow_01]
1402=[slow_02]
1403=[slow_03]
15000110=[BG1]
15002000=[BG2]
1E00=[score_HAR]
1E01=[score_Poe]
1E02=[score_fish]
1E03=[score_Epona]

01
04
ends
02
```

Рисунок 29.
Фрагмент таблицы

```
function GetData(TextStrings: PTextStrings): AnsiString; stdcall;
var
 R: PTextString;
begin
 Result := '';
 If TextStrings = NIL then Exit;
 With TextStrings^ do
 begin
  R := Root;
  While R <> NIL do
  begin
   Result := Result + R^.Str + #2;  // код конца строки: 02, можно его и не отображать в проекте
   R := R^.Next
  end
 end
end;

Function GetStrings(X, Sz: Integer): PTextStrings; stdcall;
var
 P: PByte; PC: PChar absolute P;
 Len: Byte; Code: Byte;
begin
 New(Result, Init);
 with Result.Add^ do
 begin
  P := Addr(ROM[X]);
  while P^ <> 0 do
  begin
  Len := 0;  //Обозначаем переменную длины кода, а то вдруг их там нет
   if (P^ = $05) or (P^ = $06) or (P^ = $07)  //Здесь идёт перечисление всех первых байтов контрольных кодов (берутся из таблицы)
   or (P^ = $0C) or (P^ = $0E) or (P^ = $11)  //Если встречается кто-то из них, то нужно прочитать контольный код
   or (P^ = $12) or (P^ = $13) or (P^ = $14)
   or (P^ = $15) or (P^ = $1E)
   then begin
    Code := P^;  //Запоминаем этот первый байт
    Str := Str + PC^;  //Записываем его в извлечённый текст (Вот он 1 байт, который не будем учитываеть в Len)
    Inc(P);  //Переходим к следующему
    case Code of  //Т.к. зависимость длины кода от каких-то байт в нём обнаружена не была, просто выбираем
     $05,
     $06,
     $0C,
     $0E,
     $13,
     $14,
     $1E: Len := 1;  //Если код начинается с $05, $06, $0C, $0E, $13, $14 или $1E, то его длина 2 байта, осталось 1 байт считать
     $07,
     $11,
     $12: Len := 2;  //Если код начинается с $07, $11, $12, то длина кода 3, следовательно, нужно прочитать ещё 2 байта
     $15: Len := 3  //Если код начинается с $15, то длина его 4, т.е. осталось считать 3 байта
    end;
    X := Length(Str);
    SetLength(Str, X + Len);
    Move(P^, Str[X + 1], Len);
    Inc(P, Len);  //Здесь считываем нужное количество байт
   end else
    if P^ = $02 then Break  //Это можно написать и здесь: while P^ <> $02 do, но мне обычно лень, т.к. ноль не мешает
   else
   begin
    Str := Str + PC^;  //Обычное чтение, когда никакие контрольные коды не встретились
    Inc(P)
   end
  end
 end
end;
```

Код 6. Пример плагина для однобайтовой кодировки в игре Zelda: Ocarina of Time (N64)

Вот пример чуть более хитрого плагина для однобайтовой кодировки и фрагмент таблицы (игра **Zelda: The Minish Cap (GBA)**):

```
Function GetData(TextStrings: PTextStrings): AnsiString; stdcall;
var
 R: PTextString;
begin
 Result := '';
 If TextStrings = NIL then Exit;
 With TextStrings^ do
 begin
  R := Root;
  While R <> NIL do
  begin
   Result := Result + R^.Str + #0;  //Код конца строки: 00, можно его и не отображать в проекте
   R := R^.Next
  end
 end
end;

Function GetStrings(X, Sz: Integer): PTextStrings; stdcall;
var
 P: PByte; PC: PChar absolute P;
 Len: Byte; Code: Byte;
begin
 New(Result, Init);
 with Result.Add^ do
 begin
  P := Addr(ROM[X]);
  while P^ <> 0 do
  begin
  Len := 0;  //Обозначаем переменную длины кода, а то вдруг их там нет
   if (P^ = $01) or (P^ = $02) or (P^ = $03)  //Опять идёт перечисление всех первых байтов контрольных кодов
   or (P^ = $04) or (P^ = $05) or (P^ = $06)  //Если встречается кто-то из них, то нужно прочитать контольный код
   or (P^ = $07) or (P^ = $08) or (P^ = $09)
   or (P^ = $0C) or (P^ = $0F)
   then begin
    Code := P^;  //Запоминаем первый байт
    Str := Str + PC^; //Записываем его в извлечённый текст
    Inc(P);  //Переходим к следующему
    case Code of  //Т.к. зависимость длины кода от каких-то байт в нём обнаружена не была, просто выбираем
     $01,
     $02,
     $06,
     $08,
     $09,
     $0C,
     $0F: Len := 1;  //Длина этих кодов 2 байта, 1 уже был считан
     $03,
     $07: Len := 2;  //Длина этих кодов 3 байта, 1 уже был считан
     $04: case P^ of  //А вот если код начинается с $04, то тут не всё так однозначно
           $10: Len := 2;  //Если следом за $04 идёт $10 длина такого кода будет 3
           else Len := 1   //В любом другом случае 2
          end;
     $05: case P^ of  //Аналогично поступаем и в этом случае
           $FF: Len := 1;  //Если следом за $05 идёт $FF, длина такого кода будет 2
           else Len := 2   //В любом другом случае: 3
          end
    end;
    X := Length(Str);
    SetLength(Str, X + Len);
    Move(P^, Str[X + 1], Len);
    Inc(P, Len);  //Здесь считываем нужное количество байт
   end else
   begin
    Str := Str + PC^;  //Обычное чтение, когда никакие контрольные коды не встретились
    Inc(P)
   end
  end
 end
end;
```

Код 7. Пример плагина для однобайтовой кодировки в игре Zelda: The Minish Cap (GBA)

```
0101=[01_001]
0200=[/C]
0201=[Red]
0202=[Green]
0203=[Blue]
0300DB=[03_045]
0301FE=[03_077]
0301FF=[03_078]
030200=[03_079]
030201=[03_080]

041000=[04_094]
041006=[04_095]
041007=[04_096]
04100C=[04_097]
04100E=[04_098]
0412=[04_099]
0413=[04_100]
0414=[04_101]
0415=[04_102]

050309=[Sw_News_1p1]
05465F=[05_189]
05FF=[05_190]

0600=[Name]
0601=[Choice_1]
0602=[Choice_2]
0603=[Choice_3]
070800=[07_201]
071005=[07_202]
08FF=[08_235]
0900=[09_236]
0978=[09_237]
0C00=[A]
0C01=[B]
0C02=[L]
0C03=[R]

0A
ends
00
```

Рисунок 30.
Фрагмент таблицы

Есть ещё более хитрый подход: байт окончания строки: **00**. Системные коды могут не просто его содержать, а с него начинаться (!), но там можно зацепиться за 3-й байт (0E, в таблице указан отдельно).

```
Function GetData(TextStrings: PTextStrings): AnsiString; stdcall;
var
 R: PTextString;
begin
 Result := '';
 If TextStrings = NIL then Exit;
 With TextStrings^ do
 begin
  R := Root;
  While R <> NIL do
  begin
   Result := Result + R^.Str + #0;  //Код конца строки: 00. Не отображается в проекте
   R := R^.Next
  end
 end
end;

Function GetStrings(X, Sz: Integer): PTextStrings; stdcall;
var
 P: PByte; PC: PChar absolute P;
 Len: Integer; Code: Byte;
begin
 New(Result, Init);
 with Result.Add^ do
 begin
  P := Addr(ROM[X]);
  while 2+2=4 do  //Рыбу оставляем, но шунтируем
  begin
  Len := 0;
   if (P^ = $01) or (P^ = $02) or (P^ = $03)
   then begin  //Считать 1 байт в строку
    Code := P^;
    Str := Str + PC^;
    Inc(P);
    case Code of
     $01,
     $02,
     $03: Len := 1;
    end;
    X := Length(Str);
    SetLength(Str, X + Len);
    Move(P^, Str[X + 1], Len);
    Inc(P, Len);
   end else
   if P^ = $00  //Проверяем этот код отдельно
   then begin
       Inc(P, 2);
       Code := P^;
       dec(P, 2);
       if Code = $0E  //Если второй символ после него 0E, то это хитрый код, и его нужно записать
       then begin
       Str := Str + PC^; Inc(P);
       Str := Str + PC^; Inc(P)
           end else break  //Если нет — конец строки без записи и выход
       end else
   begin
    Str := Str + PC^;
    Inc(P)
   end
  end
 end
end;
```

Код 8. Пример плагина для однобайтовой кодировки. Системные коды начинаются с байта **00**

```
0E=[0E]
01 00=[0_1]
02 00=[0_2]
03 00=[0_3]
0101=[1_1]
0201=[1_2]
0301=[1_3]
00 02=[2_0]
0102=[2_1]
0202=[2_2]
0302=[2_3]
8102=[2_81]
8202=[2_82]
8302=[2_83]
8402=[2_84]
8502=[2_85]

ends
00
```

Рисунок 31.
Фрагмент таблицы

Тот же тип плагина, но с той лишь разницей, что здесь всегда будем писать **00**, т.к. он является кодом пробела, а раз так, то и код окончания строки, **0000**, будет в проекте отображаться.

```
Function GetStrings(X, Sz: Integer): PTextStrings; stdcall;
var
 P: PByte; PC: PChar absolute P;
 Len: Integer; Code: Byte;
begin
 New(Result, Init);
 with Result.Add^ do
 begin
  P := Addr(ROM[X]);
  while  2+2=4 do  //Рыбу оставляем, но шунтируем
  begin
  Len := 0;
   if  (P^ = $00) or (P^ = $E5) or (P^ = $E7) or (P^ = $E9)
   or (P^ = $EC) or (P^ = $EE) or (P^ = $ED) or (P^ = $F2)
   or (P^ = $F4) or (P^ = $F5) or (P^ = $F8) or (P^ = $FB)
   or (P^ = $FC) or (P^ = $FF)
   then begin  //Считать 1 байт в строку
    Code := P^;
    Str := Str + PC^;
    Inc(P);
    case Code of
     $E5,
     $EE,
     $F2,
     $F5,
     $F8,
     $FC: Len := 1;
     $E7,
     $E9,
     $EC,
     $FF: Len := 2;
     $ED,
     $FB: Len := 3;
     $F4: Len := 5;
     $00: case P^ of
           $00: begin Str := Str + PC^; break  //Конец строки пишем и выходим
              end;
           else Len := 0  //Т.к. 00, т.е. пробел уже записан
         end;
    end;
    X := Length(Str);
    SetLength(Str, X + Len);
    Move(P^, Str[X + 1], Len);
    Inc(P, Len);
   end else
   begin
    Str := Str + PC^;
    Inc(P)
   end
  end
 end
end;
```

Код 9. Пример плагина для однобайтовой кодировки.
Код пробела: **00**, код окончания строки: **0000**

```
00=
01=0
02=1
...
E500=!
E501=!!
E502=?
...
E70000=[E700]
E70100=[E701]
E70500=[E705]
E90000=[E900]
F200=[F200]
F404121219FF=[F401]
F500=[F500]
EC0000=[EC00]
ED002200=[ED00]
FB040050=[FB04]
FF0000=[FF00]

E8
EA
EB
ends
0000
```

Рисунок 32.
Фрагмент таблицы

Вот пример плагина для однобайтовой кодировки с кодами, содержащими в себе поле размера (игры **Zelda** на **GC**, **Wii** и **WiiU**, в которых используются **bmg** контейнеры для хранения текста: **The Wind Waker**, **Four Swords Adventures**, **Twilight Princess**, **Twilight Princess HD**.

```
Function GetData(TextStrings: PTextStrings): String; stdcall;
Var
 R: PTextString;
begin
 Result := '';
 If TextStrings = NIL then Exit;
 With TextStrings^ do
 begin
  R := Root;
  While R <> NIL do
  begin
   Result := Result + R^.Str + #0;  //Т.к. код конца строки всего один,
можно его и не отображать в проекте
   R := R^.Next
  end
 end
end;

Function GetStrings(X, Sz: Integer): PTextStrings; stdcall;
var
P: PByte; PC: PChar absolute P;
Len: Byte; Code: Byte;
begin
 New(Result, Init);
 with Result.Add^ do
 begin
  P := Addr(ROM[X]);
  while P^ <> 0 do
  begin
   if P^ = $1A then  //Более продвинутая система кодов: все начинаются с $1A и содержат в себе размер
   begin
    Str := Str + PC^;  //Читаем $1A
    Inc(P);  //Переходим к следующему байту
    Code := P^;  //Запоминаем байт идущий после $1A, т.к. он и является размером кода
    Str := Str + PC^;  //Читаем байт размера в проект
    Inc(P);  //Переходим к следующему
    Len := Code - 2;  //Можно прочитать оставшиеся байты в строку (т.к. два байта уже прочитаны,
от размера отнимаем 2)
    X := Length(Str);
    SetLength(Str, X + Len);
    Move(P^, Str[X + 1], Len);
    Inc(P, Len)  //Здесь считываем нужное количество байт
   end else
    begin
     Str := Str + PC^;  //Обычное чтение, когда никакие контрольные коды не встретились
     Inc(P)
    end
  end
 end
end;
```

Код 10. Пример плагина для однобайтовой кодировки с кодами, содержащими размер. Контейнеры для текста: bmg.

```
1A05000000=[Name]
1A0500000D=[L]
1A0500000E=[R]
1A0500000F=[X]
1A05000010=[Y]
1A05000011=[Z]
1A06FF000000=[/c]
1A06FF000001=[Red]
1A06FF000002=[Green]
1A06FF000003=[Blue]
1A06FF000004=[Yellow]
1A06FF000005=[l_Blue]
1A06FF000006=[Purple]
1A06FF000007=[Silver]
1A06FF000008=[Orange]
1A070200060100=[07_101]
1A08020024000101=[08_120]
1A0902000500000166=[09_135]
1A0A0200090000037401=[0A_232]
1A0B020026000000140A01=[0B_312]
1A0C02000D00000032010000=[0C_313]
1A0D02002C000001400000013C=[0D_319]
1A0E020007000000015C0000015C=[0E_327]
1A0F02001002000001C3000001C400=[0F_379]
1A1002002300000000000002EF000002EE=[10_423]
1A1102001D0000003200000039C0000039B=[11_532]
1A150200180000013A0000013C0000013C0000013D=[15_535]

0A
ends
00
```

Рисунок 33.
Фрагмент таблицы

Вот пример плагина для двубайтовой кодировки с кодами, содержащими в себе поле размера (игры **Zelda** на **NDS**: **Phantom Hourglass** и **Spirit Tracks**, в которых используются **bmg** контейнеры для хранения текста, но текст в **Unicode**).

```
function GetData(TextStrings: PTextStrings): AnsiString; stdcall;
var
 R: PTextString;
begin
 Result := '';
 If TextStrings = NIL then Exit;
 With TextStrings^ do
 begin
  R := Root;
  While R <> NIL do
  begin
   Result := Result + R^.Str + #0 + #0; //Код конца строки: 0000, можно его и не отображать в проекте
   R := R^.Next
  end
 end
end;

function GetStrings(X, Sz: Integer): PTextStrings; stdcall;
type
 TWordRec = packed record
  A, B: Char
 end;
var
 PW: PWord; PB: PByte absolute PW; PC: ^TWordRec absolute PW;
 Len: Word; Code: Byte;
begin
 New(Result, Init);
 with Result.Add^ do
 begin
  PW := Addr(ROM[X]);
  while PW^ <> 0 do
  begin
   if PW^ = $001A then  //Система кодов с $1A в двубайтовом варианте (здесь Little Endian)
   begin
    Str := Str + PC.A + PC.B;  //Читаем 2 байта: $001A
    Inc(PW);  //Переходим к следующему слову.
    Code := PB^;  //Запоминаем размер, кодов больше 255 не встречал, поэтому размер здесь определён только 1 байтом.
    Str := Str + PC.A + PC.B;  //Читаем 2 байта размера в проект
    Inc(PW);  //Переходим к следующему слову.
    Len :=  Code - 4;  //Можно прочитать оставшиеся байты в строку
    X := Length(Str);
    SetLength(Str, X + Len);
    Move(PB^, Str[X + 1], Len);
    Inc(PB, Len)  //Здесь считываем нужное количество байт
   end else
   begin
    Str := Str + PC.A + PC.B;  //Обычное чтение, когда никакие контрольные коды не встретились
    Inc(PW)
   end
  end
 end;
end;
```

Код 11. Пример плагина для двубайтовой кодировки с кодами, содержащими размер.
Контейнеры для текста: bmg. Кодировка: Unicode

```
1A0006000000=[Ch1]
1A0006FE0000=[Name]
1A0008000A000000=[08_012]
1A0008FF00000000=[/c]
1A0008FF00000100=[Red]
1A0008FF00000200=[Green]
1A0008FF00000300=[Blue]
1A000A000E0014000000=[0A_128]
1A000CFF02000159306A3000=[0C_191]
1A000EFF020000443061306930 00=[0E_192]
1A0010FF020001553093304B304F3000=[10_193]
1A00220401000900770069006E00730000007700690006E000000770069006E007300=[Wins]
1A00280401000000740069006D00650073000000740069006D00650000007400690 06D0065007300=[Times]
1A002E040100000052007500700065006500730000005200750070006500650050000052007500700065006500 7300=[Rupees]
1A00340401000100730065006300 6F006E00640073000000730065006300 6F006E00640000007300650063006F006E0064007300=[Seconds]
1A00400401000000070006F00730074006300610072006400730000007000 6F00730074006300610072006400000070006F00730074006300610072006400 7300=[Postcards]
1A004C0401000000730075006200 6D006900730073006900 6F006E00730000007300750062006D006900730073006900 6F006E00000073007500620 06D006900730073006900 6F006E007300=[Submissions]

0A00
ends
0000
```

Рисунок 34. Фрагмент таблицы

Вот пример плагина для двубайтовой кодировки с кодами, содержащими в себе поле размера (игры **Zelda: Skyward Sword** (**Wii**) и **The Wind Waker HD** (**WiiU**) (UTF-16 Big Endian), **A Link Between Worlds** (**3DS**) и **Triforce Heroes** (**3DS**) (UTF-16 Little Endian), в которых используются **msbt** контейнеры для хранения текста).

```
function GetData(TextStrings: PTextStrings): AnsiString; stdcall;
var
 R: PTextString;
begin
 Result := '';
 If TextStrings = NIL then Exit;
 With TextStrings^ do
 begin
  R := Root;
  While R <> NIL do
  begin
   Result := Result + R^.Str + #0 + #0;
   R := R^.Next
  end
 end
end;

function SwapMe2(V: Word): Word;
  asm xchg al,ah end;

function GetStrings(X, Sz: Integer): PTextStrings; stdcall;
type
 TWordRec = packed record
  A, B: Char
 end;
var
 PW: PWord; PB: PByte absolute PW; PC: ^TWordRec absolute PW;
 Len: Word; Code: Word;
begin
 New(Result, Init);
 with Result.Add^ do
 begin
  PW := Addr(ROM[X]);
  while PW^ <> 0 do
  begin
   if (PW^ = $000E) or (PW^ = $0E00) then  //Двухбайтовая система
кодов с $0E (Здесь LE и BE)
   begin
   if PW^ = $0E00 then Len:=1 else Len:=0;  //Используем переменную
Len не по назначению: для проверки Endiannes.
    Str := Str + PC.A + PC.B;  //Читаем 2 байта: $000E или $0E00
    Inc(PW);  //Переходим к следующему слову
    Str := Str + PC.A + PC.B;  //Читаем 2 байта: они нам не интересны
    Inc(PW);  //Переходим к следующему слову
    Str := Str + PC.A + PC.B;  //Читаем 2 байта: они нам тоже не интересны
    Inc(PW);  //Переходим к следующему слову
    Str := Str + PC.A + PC.B;  //Читаем 2 байта: добрались до размера.
Он показывает сколько байт в коде ещё осталось прочитать
    Code := PW^;  //Запоминаем размер, кодов больше 255 не встречал, но
код взял словом (может так правильнее)
    Inc(PW);  //Переходим к следующему слову
    if Len=1 then  Len := SwapMe2(Code) else Len := Code;  //Сохраняем
размер в Len (переводим в LE если нужно)
    X := Length(Str);
    SetLength(Str, X + Len);
    Move(PB^, Str[X + 1], Len);
    Inc(PB, Len)  //Здесь считываем нужное количество байт
   end else
   begin
    Str := Str + PC.A + PC.B;  //Обычное чтение, когда никакие кон-
трольные коды не встретились
    Inc(PW)
   end
  end
 end
end;
```

Код 12. Пример плагина для двубайтовой кодировки с кодами, содержащими размер. Котнейнеры для текста: msbt. Кодировка: UTF-16 (LE и BE)

```
000E0000000300020000=[Red1]
000E0000000300020001=[Red2]
000E0000000300020002=[Yellow1]
000E0000000300020003=[Blue1]
000E0000000300020004=[Green1]
000E0000000300020005=[Yellow2]
000E0000000300020006=[Violet]
000E0000000300020007=[Green2]
000E0000000300020008=[Blue2]
000E0000000300020009=[Red4]
000E000000030002000A=[Silver]
000E000000030002000B=[Gold]
000E000000030002000C=[Black]
000E000000030002FFFF=[/C]
000E0001000000002FFFF=[But1]
000E0001000100020000=[But2]
000E0001000400020001=[Delay01]
000E0001000500040050000=[01_0039]
000E0001000600020CD=[01_0067]
000E0001000700004FF000000=[01_0084]
000E0001000800020CD=[01_0085]
000E0001000900040000000=[01_0090]
000E0001000900040000FF05=[01_0253]
000E0001000900040FFFFFF=[01_1118]
000E0001000D00020500=[01_1464]
000E0001000F0000=[01_1467]
000E0001001100020CD=[01_1468]
000E0001001200040000001=[01_1477]
000E0002000000000=[Link]
000E0002000100020001=[Item001]
000E0002000200040000000=[Var0]
000E0002000300060000000000CD=[Count01]
000E0002000400020CD=[A]
000E0002000400020CD=[B]
000E0003000100000=[03_1677]
000E0003000400020CD=[Bug01]
000E0003000500040000000=[01_003E]
000E0003001000080000045700002EEF=[StaFF]

000A
ends
0000
```

Рисунок 35.
Фрагмент таблицы (Big Endian)

```
0E0000000200020005A00=[00_001]
0E0000000300020009000=[Blue]
0E0000000300020002A00=[Red]
0E0000000300020002B00=[Green]
0E0000000300020FFFF=[/C]
0E0001000000000={Link}
0E0001000100000={Player1}
0E0001000200000={Player2}
0E0001000300000={Player3}
0E0001000500060000000000CD=[01_014]
0E000100050006000000FFFF00CD={count01}
0E0001000600020002CD={choice01}
0E0001000600020003CD={choice02}
0E0001000600020004CD={choice03}
0E0001000700020001000=[01_031]
0E0001001100040014002800=[01_061]
0E0002000000020000000={-Zelda0-}
0E0002000100040000000CD={-Eastern Palace1-}
0E0002000100040022000CD={-Lost Woods1-}
0E0002000200040058000 1CD={-Hammer6-}

0A00
ends
0000
```

Рисунок 36.
Фрагмент таблицы (Little Endian)

Единственным недостатком подобных плагинов является то, что поле размера в файлах **bmg** или **msbt** не изменяется при пересборке, но это не относится к правильному извлечению текста. =)

Если у вас сложилось впечатление, что Kruptar подходит исключительно для перевода игр серии Zelda, то это неверное впечатление:

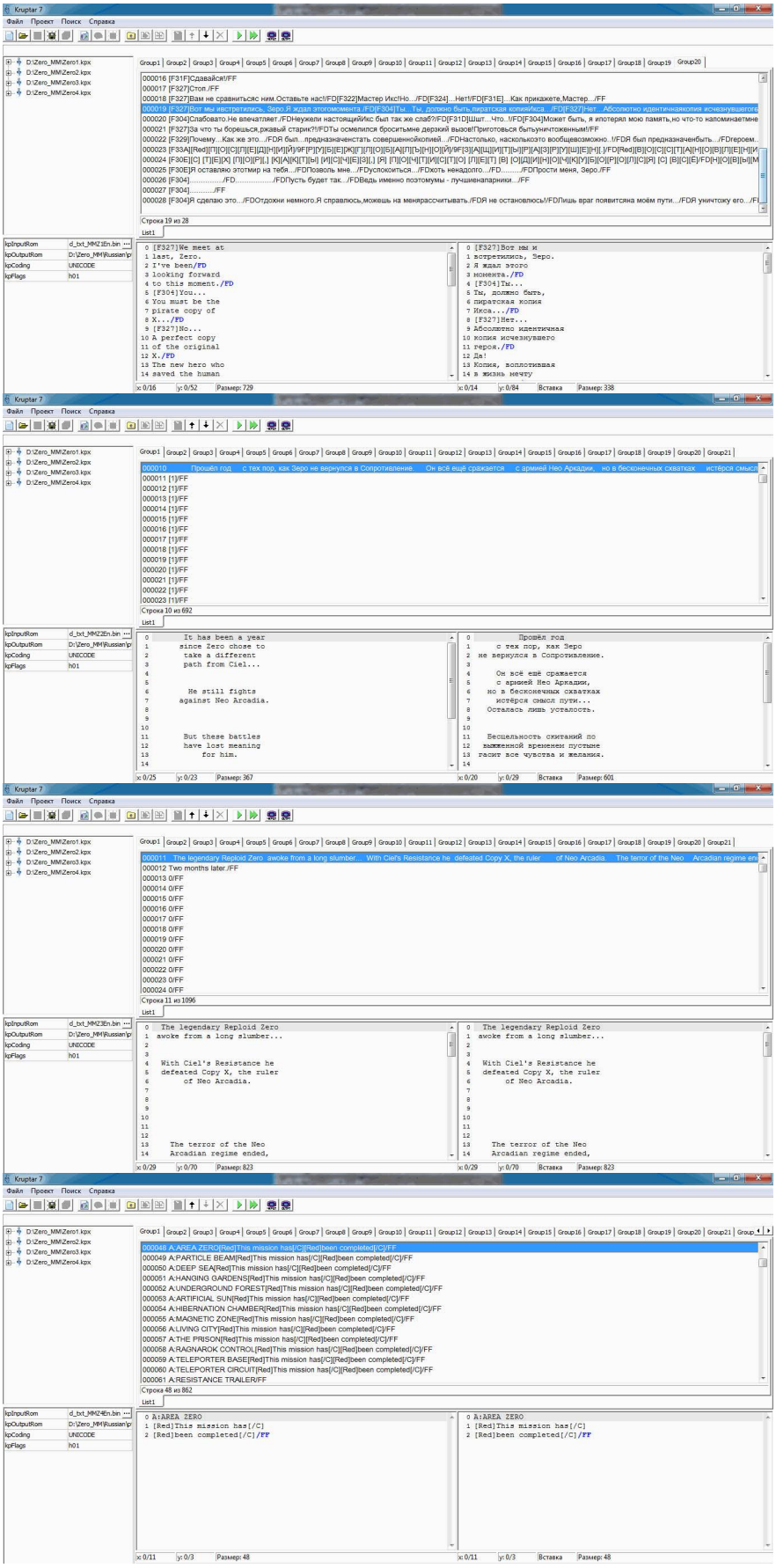


Рисунок 37.
Проекты к переводу игр Mega Man Zero 1—4 (GBA)

6. ССЫЛКИ

Раздел с данным руководством на форуме группы переводов «Шедевр»:
http://shedevr.org.ru/forum/viewforum.php?f=19&sid=cadd2a3887b56555e7b4710e80c169e5

О Kruptar в сети:
chiefnet.1bb.ru/viewtopic.php?id=614 (по-русски)
www.romhacking.net/forum/index.php?topic=16496.0 (не по-русски)

Сайт автора программы (Джинни, группа переводов Magic Team):
www.magicteam.net

Kruptar для чайников (Kruptar версий 6.x):
www.magicteam.net/index.php?page=documents&show=Kruptar для чайников

Написание плагинов для Kruptar 7:
www.magicteam.net/index.php?page=documents&show=Написание плагинов для Kruptar 7 (по-русски)
romhack.github.io/doc/kruptarPlugins (не по-русски)

Темы на форуме группы Magic Team, посвященные программе и помощи в работе с ней:
magicteam.net/forum/index.php?PHPSESSID=2eecf33ffafa2228a2f62f08e04df737&topic=298.0
magicteam.net/forum/index.php?topic=368.0
magicteam.net/forum/index.php?topic=21.0
magicteam.net/forum/index.php?topic=139.0
magicteam.mybb.ru/viewtopic.php?id=45
magicteam.net/forum/index.php?topic=260.0

Поиск хекс-значений по маске, пригодится при составлении таблиц:
magicteam.net/forum/index.php?topic=361.0

Скрипторий (игры серии Zelda):
zelda64rus.ucoz.ru/forum/23-614-1

ВЕРСИИ ПРОГРАММЫ, РАССМОТРЕННЫЕ В ДАННОМ РУКОВОДСТВЕ, МОЖНО СКАЧАТЬ ПО ССЫЛКАМ:

Kruptar v7.0.0.85 (версия Kruptar с «нумерацией строк») (интерфейс: Ru):
shedevr.org.ru/zelda64rus/tools/Kruptar.v7.0.0.85.rar

Kruptar v7.1.1.10 (версия Kruptar без «нумерации строк» и «особой сортировки») (интерфейс: Ru, En, Pt-BR):
shedevr.org.ru/zelda64rus/tools/Kruptar.v7.1.1.10.rar

Kruptar v7.1.1.17 (версия Kruptar без «нумерации строк» с «особой сортировкой» (интерфейс: Ru, En, Pt-BR):
www.magicteam.net/download.php?id=1&file=programs/kruptar7/Kruptar.v7.1.1.17.rar

КОНЕЦ...