МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫЙ ИНЖИНИРИНГ

Курсовая работа

По предмету: Программирование на языке C++
На тему: Создание игры «Викторина» в Borland Builder C++

Выполнил: студент 2-в курса
телекоммуникационных
технологий Ерназаров Батыр
Принял: _____

НУКУС 2016г.

Содержание

Введение

Borland C++ — среда программирования (IDE), разработанная фирмой Borland для создания программ на языках программирования Си и C++. Каждая версия среды включает компилятор, поддерживающий свой стандарт языка программирования. Первоначально среда программирования использовалась для создания программ под DOS, но с появлением и распространением Windows и Windows NT были предложены средства для разработки приложений для этих операционных систем.

Borland C++ исторически восходит к Turbo C, но, в отличие от Turbo C, поддерживает объектно-ориентированное программирование. За время своего развития среда разработки дополнялась специализированными библиотеками, предназначенными для быстрой разработки приложений. В частности, примером применения объектно-ориентированного подхода для создания приложений под DOS стала библиотека Turbo Vision, в то время как аналогичным примером применения объектно-ориентированного подхода для создания приложений под Windows стала библиотека Object Windows Library.

Первая доступная версия Borland C++, имевшая номер 2.0, вышла в одна тысяча девятьсот девяностом году под DOS (для OS/2 данная версия вышла в 1992 году). В 1991 году вышла версия 3.0 с поддержкой сборки Windows-приложений. Спустя год вышло обновление 3.1, в котором был реализован оконный IDE и шаблоны приложений OWL 1.0 и Turbo Vision 1.0.

Начиная с версии 4.0 (1993 год) в продукте удалена поддержка сборки MS-DOS приложений, и IDE стал полностью Windows-ориентированным. В 1995 году вышла 4.52 с поддержкой Windows 95 и OWL 2.5. В марте 1996 года выходит 5.0, которая стала поддерживать Windows NT 3.51 (Windows NT 4.0 ещё находился в разработке).

В 1997 году выходит последний релиз Borland C++ IDE (впоследствии замещённой серией Borland C++ Builder), который ещё поддерживал генерацию кода дляреального режима и MS-DOS. Этот релиз имел номер 5.02.

Последняя версия Borland C++ 5.5, содержащая только компилятор, без IDE, доступна для бесплатного скачивания с сайта Embarcadero.[1][2]

Эволюция Borland C++:

Turbo C → **Borland C++** → Borland C++ Builder → Codegear C++ Builder (и Codegear Turbo C++) → Embarcadero C++ Builder (и Embarcadero Turbo C++)

Глава 1: Теоретическая часть

В этой главе будут представлены с какими компонентами мы будем работать в Borland Builder C++.

1.1 Компонент Button

Простейшей и, пожалуй, наиболее часто используемой кнопкой является кнопка **Button**, расположенная на странице библиотеки Standard. Реже используется кнопка **BitBtn**, отличающаяся, прежде всего, возможностью отобразить на ее поверхности изображение. Большинство свойств, методов и событий у этих видов кнопок одинаковы.

Основное с точки зрения внешнего вида свойство кнопки - *Caption* (надпись). В надписях кнопок можно предусматривать использование клавиш ускоренного доступа, выделяя для этого один из символов надписи. Перед символом, который должен соответствовать клавише ускоренного доступа, ставится символ амперсанда "&". Этот символ не появляется в надписи, а следующий за ним символ оказывается подчеркнутым. Тогда пользователь может вместо щелчка на кнопке нажать в любой момент клавишу Alt совместно с клавишей выделенного символа.

Например, если в вашем приложении имеется кнопка выполнения какой-то операции, вы можете задать ее свойство *Caption* равным «&Start». На кнопке эта надпись будет иметь вид «Start». И если пользователь нажмет клавиши Alt+S, то это будет эквивалентно щелчку на кнопке.

Основное событие любой кнопки - *OnClick*, возникающее при щелчке на ней. Именно в обработчике этого события записываются операторы, которые должны выполняться при щелчке пользователя на кнопке. Помимо этого есть еще ряд событий, связанных с различными манипуляциями клавишами и кнопками мыши. Писать обработчик события *OnClick* надо, если вы не связываете свойством *Action* данную кнопку с каким-то действием.

Свойство *Cancel*, если установить его в true, определяет, что нажатие пользователем клавиши Esc будет эквивалентно щелчку на данной кнопке. Это свойство целесообразно задавать равным true для кнопок Отменить в различных диалоговых окнах, чтобы можно было выйти из диалога, нажав на эту кнопку или нажав клавишу Esc.

Свойство *Default*, если его установить в true, определяет, что нажатие пользователем клавиши ввода Enter будет эквивалентно нажатию на данную кнопку, даже если данная кнопка в этот момент не находится в фокусе. Правда, это сработает, если в фокусе находится какой-то оконный компонент. Если же в момент нажатия Enter в фокусе находится другая

кнопка, то все-таки сработает именно кнопка в фокусе. Если у нескольких кнопок на форме свойство *Default* задано равным true, то при нажатии Enter сработает та из них, которая находится раньше в последовательности табуляции. Из методов, присущих кнопкам, имеет смысл отметить один - *Click*. Выполнение этого метода эквивалентно щелчку на кнопке, т.е. вызывает событие кнопки *OnClick*. Этим можно воспользоваться, чтобы продублировать какими-то другими действиями пользователя щелчок на кнопке. Пусть, например, вы хотите, чтобы при нажатии пользователем клавиши с символом "Б" или "б" в любой момент работы с приложением выполнялись операции, предусмотренные в обработчике события *OnClick* кнопки Button1. Поскольку неизвестно, какой компонент будет нажиматься в фокусе в момент этого события, надо перехватить его на уровне формы. Такой перехват осуществляется, если установить свойство формы KeyPreview = true. Тогда в обработчике события формы *OnKeyPresss* можно написать оператор: if((Key=='Б')||(Key=='б')) Button1->Click();
Если пользователь ввел символ "Б" или "б", то в результате будет выполнен обработчик щелчка кнопки Button1.

1.2 Компонент Label

Для отображения различных надписей на форме используются в основном компоненты **Label**, **StaticText** и **Panel**. Первые два из этих компонентов — метки, специально предназначенные для отображения текстов. Основное назначение панели **Panel** - компоновка компонентов в окне формы, однако можно использовать и для вывода текстов.

Тексты, отображаемые в компонентах, определяются значением их свойства *Caption*. Его можно устанавливать в процессе проектирования или задавать и изменять программно во время выполнения приложения. Например:

```
Label1->Caption = "Ерназаров Батыр";
```

Свойство *Caption* имеет тип строки AnsiString. При присваивании этому типу числовой информации происходит ее автоматическое преобразование в строку. Поэтому вы можете непосредственно осуществлять подобные присваивания. Например:

```
Label1->Caption = 5.1;
```

Оператор приведет к появлению в метке надписи «5,1». Но если вы хотите занести в метку смешанную информацию, состоящую из строк символов и чисел, вы должны воспользоваться функциями *FloatToStr* и*IntToStr*, переводящими соответственно числа с плавающей запятой и целые в строку. Для формирования текста, состоящего из нескольких фрагментов, можно использовать операцию "+", которая для строк означает их склеивание (конкатенацию). Например, если в программе имеется целая переменная I,

отображающая число студентов ТАТУ, то вывести в метку Label1 информацию об этом можно оператором:

```
Label1->Caption = "Число студентов ТАТУ: "+IntToStr(I);
```

Во всех компонентах цвет фона определяется свойством *Color*, а цвет надписи подсвойством *Color* свойства *Font*. Если цвет специально не задавать, то цвет фона обычно сливается с цветом контейнера, содержащего метку, так что фон просто не заметен.

Для метки **Label** цвет и шрифт - единственно доступные элементы оформления надписи. Компоненты **StaticText** и **Panel** имеют кроме того свойство *Border - Style*, определяющее рамку текста - бордюр. При стиле *sbsNone* метка **StaticText** по виду не отличается от метки **Label**. Вероятно, если уж использовать бордюр, то наиболее приятный стиль *sbsSunken*.
Компонент **Panel** кроме свойства *BorderStyle* имеет еще свойства *Bevellnner, BevelOuter, BevelWidth, BorderWidth*, которые предоставляют богатые возможности оформления надписи. Таким образом, с точки зрения оформления выводимого текста максимальные возможности дает **Panel** и минимальные **Label**.

Размещение всех текстовых компонентов на форме определяется, в частности, свойствами *Top, Left, Height, Width, Aline*, общими для всех оконных компонентов. Эти свойства, определяющие координаты компонента, его размеры и их изменение при изменении пользователем размеров родительского компонента.

Размер меток **Label** и **StaticText** определяется также свойством *AutoSize*. Если это свойство установлено в true, то вертикальный и горизонтальный размеры компонента определяются размером надписи. Если же*AutoSize* равно false, то выравнивание текста внутри компонента определяется свойством *Alignment*, которое позволяет выравнивать текст по левому краю, правому краю или центру клиентской области метки. В панели**Panel** также имеется свойство *AutoSize*, но оно не относится к размерам надписи *Caption*. Однако свойство выравнивания Alignment работает и для панели.

В метке **Label** имеется свойство *WordWrap* - допустимость переноса слов длинной надписи, превышающей длину компонента, на новую строчку. Чтобы такой перенос мог осуществляться, надо установить свойство*WordWrap* в true, свойство *AutoSize* в false (чтобы размер компонента не определялся размером надписи) и сделать высоту компонента такой, чтобы в нем могло поместиться несколько строк. Если *WordWrap* не установлено в true при *AutoSize* равном false, то длинный текст, не помещающийся в рамке метки, просто обрезается**.**

1.3 Компонент StringGrid

StringGrid - таблица строк. Компонент **StringGrid** представляет собой таблицу, содержащую строки. Данные таблицы могут быть только для чтения или редактирования. Таблица может иметь полосы прокрутки, причем заданное число первых строк и столбцов может быть фиксированным и не прокручиваться. Таким образом, можно задать заголовки столбцов и строк, постоянно присутствующие в окне компонента. Каждой ячейке таблицы может быть поставлен в соответствие некоторый объект.
Компонент **StringGrid** предназначен в первую очередь для отображения таблиц текстовой информации. Однако компонент может отображать и графическую информацию.

Свойства *ColCount* и *RowCount* определяют соответственно число столбцов и строк, свойства *FixedCols* и*FixedRows* - число фиксированных, не прокручиваемых столбцов и строк. Цвет фона фиксированных ячеек определяется свойством *FixedColor*.
Свойства *LeftCol* и *TopRow* определяют соответственно индексы первого видимого на экране в данный момент прокручиваемого столбца и первой видимой прокручиваемой строки.
Свойство *ScrollBars* определяет наличие в таблице полос прокрутки. Причем полосы прокрутки появляются и исчезают автоматически в зависимости от того, помещается таблица в соответствующий размер или нет.
Свойство *Options* является множеством, определяющим многие свойства таблицы: наличие разделительных вертикальных и горизонтальных линий в фиксированных (*goFixedVertLine* и *goFixedHorzLine*) и не фиксированных (*goVertLine* и *goHorzLine*) ячейках, возможность для пользователя изменять с помощью мыши размеры столбцов и строк (*goCoISizing* и *goRowSizing*), перемещать столбцы и строки (*goColMoving* и*goRowMoving*) и многое другое. Важным элементом в свойстве *Options* является *goEditing* - возможность редактировать содержимое таблицы.

В основном компонент **StringGrid** используется для выбора пользователем каких-то значений, отображенных в ячейках.
Свойства *Col* и *Row* показывают индексы столбца и строки выделенной ячейки. Возможно также выделение пользователем множества ячеек, строк и столбцов.
Среди множества событий компонента **StringGrid** следует отметить событие *OnSelectCell*, возникающее в момент выбора пользователем ячейки. В обработчик этого события передаются целые параметры *ACol* и*ARow* - столбец и строка выделенной ячейки, и булев параметр *CanSelect* - допустимость выбора.

Параметр *CanSelect* можно использовать для запрета выделения ячейки, задав его значение false. А параметры *ACol* и *ARow* могут использоваться для какой-то реакции программы на выделение пользователя.

1.4 Компонент Image

Image - управление изображений. Нередко возникает потребность украсить свое приложение какими-то картинками. Это может быть графическая заставка, являющаяся логотипом вашего приложения или же это могут быть фотографии при разработке приложения, работающего с базой данных сотрудников некоего учреждения. В первом случае потребуется компонент **Image**, расположенный на странице Additionol библиотеки компонентов, во втором его аналог **DBImage**, связанный с данными и расположенный на странице Data Controls.

Откройте новое приложение и перенесите на форму компонент **Image**. Его свойство, которое может содержать картинку - *Picture*. Нажмите на кнопку с многоточием около этого свойства или просто сделайте двойной щелчок на Image, и перед вами откроется окно Picture Editor, позволяющее загрузить в свойство*Picture* какой-нибудь графический файл (кнопка Load), а также сохранить открытый файл под новым именем или в новом каталоге. Щелкните на Load, чтобы загрузить графический файл. Перед вами откроется окно Load Picture. По мере перемещения курсора в списке по графическим файлам в правом окне отображаются содержащиеся в них изображения. Вы можете найти графические файлы в каталоге Images. Он обычно расположен в каталоге C:\Program files\Common Files\Borland Shared.

В окне загрузки графического файла вы можете не только просмотреть изображение, хранящееся в выбираемом файле, но и увидеть размер изображения - цифры в скобках справа вверху.

После загрузки файла щелкните на ОК и в вашем компоненте **Image** отобразится выбранная вами картинка. Когда вы в процессе проектирования загрузили картинку из файла в компонент **Image**, он не просто отображает ее, но и сохраняет в приложении. Это дает вам возможность поставлять ваше приложение без отдельного графического файла. Впрочем, в **Image** можно загружать и внешние графические файлы в процессе выполнения приложения.

Если установить свойство *AutoSize* в true, то размер компонента **Image** будет автоматически подгоняться под размер помещенной в него картинки. Если же свойство *AutoSize* установлено в false, то изображение может не поместиться в компонент или, наоборот, площадь компонента может

оказаться много больше площади изображения. Свойство *Stretch* позволяет подгонять не компонент под размер рисунка, а рисунок под размер компонента. Установите *AutoSize* в false, растяните или сожмите размер компонента **Image** и установите *Stretch* в true. Вы увидите, что рисунок займет всю площадь компонента, но поскольку вряд ли реально установить размеры **Image** точно пропорциональными размеру рисунка, то изображение исказится.
Устанавливать *Stretch* в true может иметь смысл только для каких-то узоров, но не для картинок. Свойство*Stretch* не действует на изображения пиктограмм, которые не могут изменять своих размеров. Свойство *Center*, установленное в true, центрирует изображение на площади **Image**, если размер компонента больше размера рисунка.

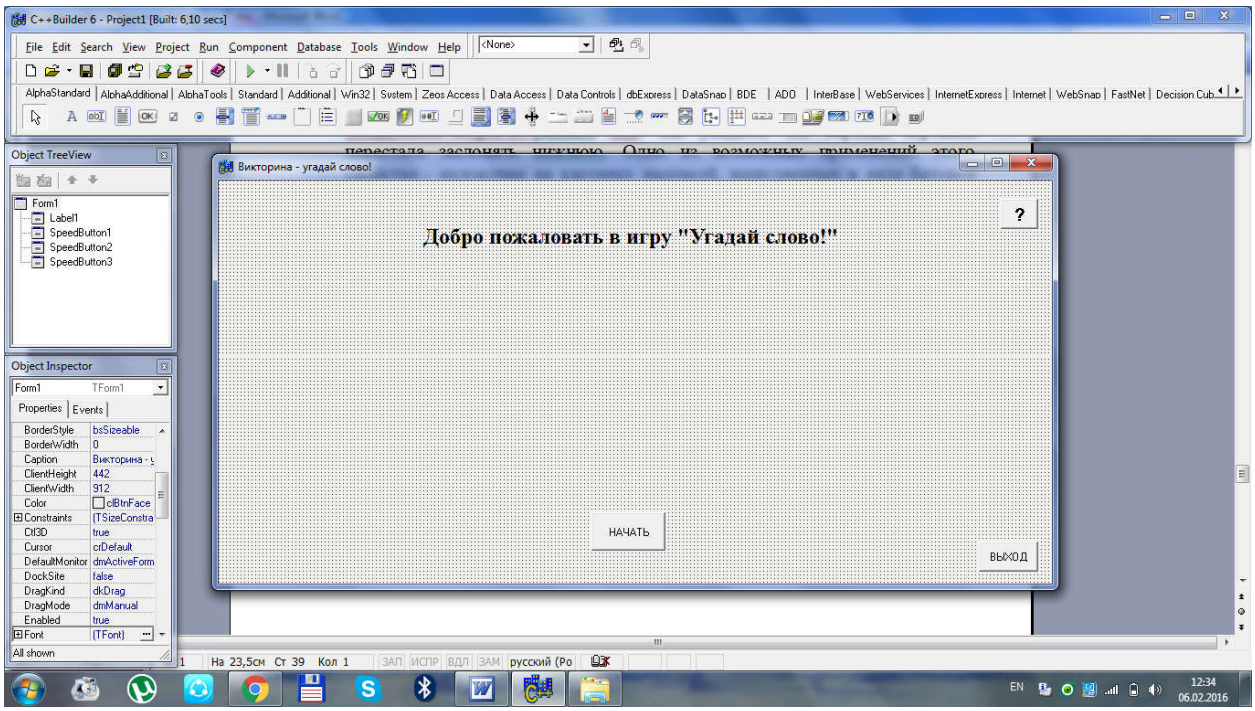
Свойство *Transparent* (прозрачность). Если *Transparent* равно true, то изображение в **Image** становится прозрачным. Это можно использовать для наложения изображений друг на друга. Поместите на форму второй компонент **Image** и загрузите в него другую картинку. Только постарайтесь взять какую-нибудь мало заполненную, контурную картинку. Можете, например, взять картинку из числа помещаемых обычно на кнопки, например, стрелку (...\program files\common files\borland shared\images\buttons\orrowlr.bmp). Передвиньте ваши **Image** так, чтобы они перекрывали друг друга, и в верхнем компоненте установите *Transparent* равным true. Вы увидите, что верхняя картинка перестала заслонять нижнюю. Одно из возможных применений этого свойства - наложение на картинку надписей, выполненных в виде битовой матрицы. Эти надписи можно сделать с помощью встроенной в C++Builder программы Image Editor.
Учтите, что свойство *Transparent* действует только на битовые матрицы.

Глава 2: Практическая часть

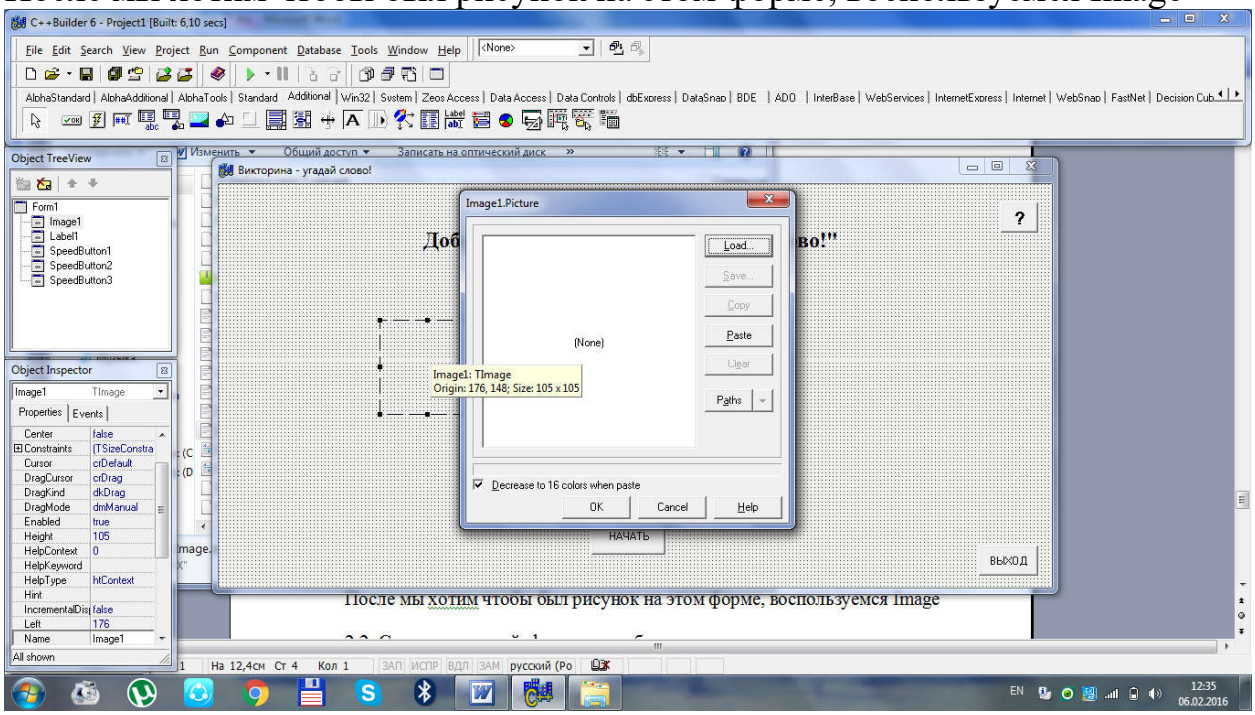
В этой главе будут показаны последовательность действий для создании игры «Викторина».

2.1 Добавление компонентов на форму

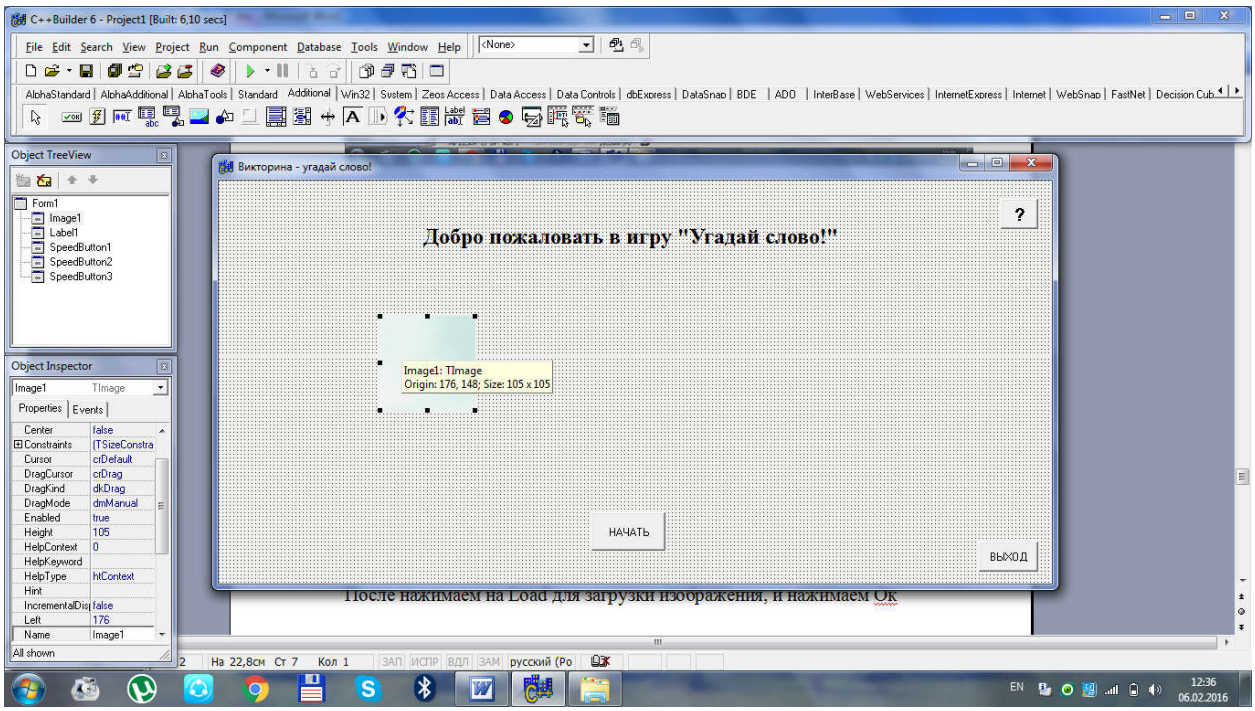
Для начало открываем Borland Builder C++. Сохраняем проект, и на форму добавляем Label, и 3 Button. Переименовав их по схеме, получаем следующий вид:



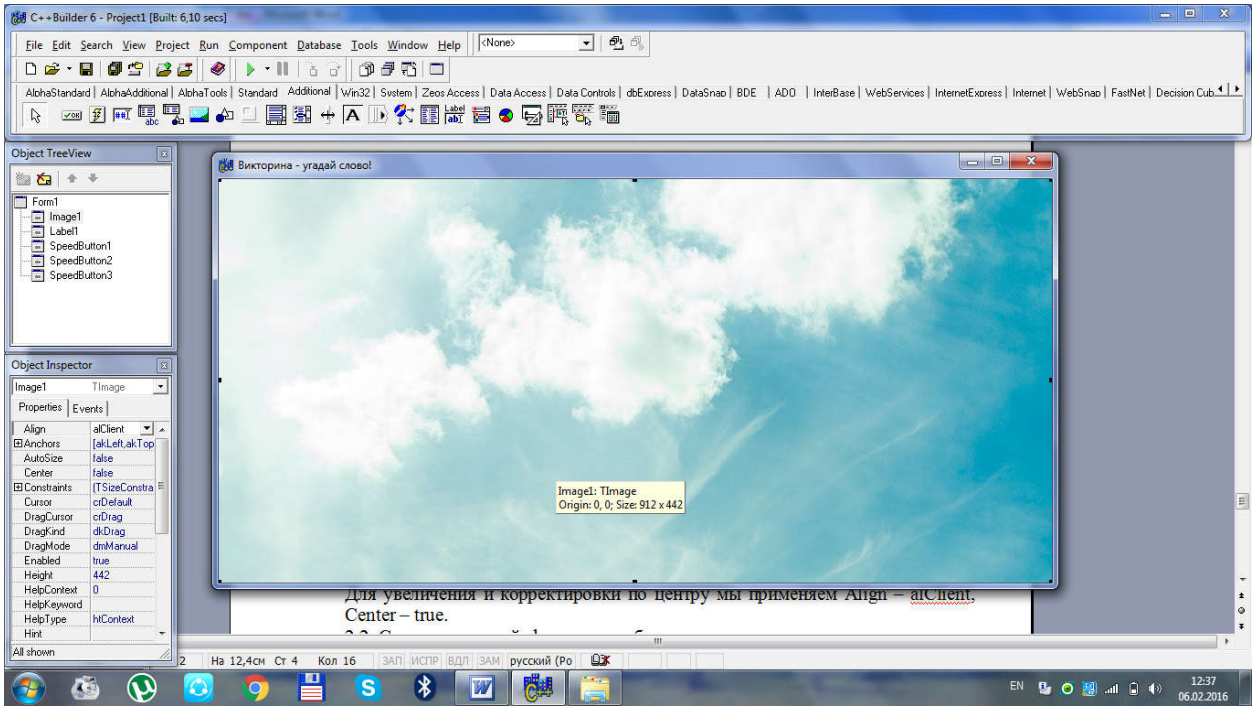
После мы хотим чтобы был рисунок на этом форме, воспользуемся Image



После нажимаем на Load для загрузки изображения, и нажимаем Ок



Для увеличения и корректировки по центру мы применяем Align – alClient, Center – true.



Здесь все вышло, но мы потеряли других компонентов, т.е. они на заднем плане Image. Чтоб исправить это нажимаем дважды на Form1 и пишем следующий код:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Image1->SendToBack();
}
```

Теперь все исправлено! Но до этого мы должны написать на компонентах следующие коды:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->Hide();
Form2->Show();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
Form1->Hide();
Form3->Show();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form1->Close();
}
//---------------------------------------------------------------------------


void __fastcall TForm1::SpeedButton1Click(TObject *Sender)
{
Form1->Hide();
```

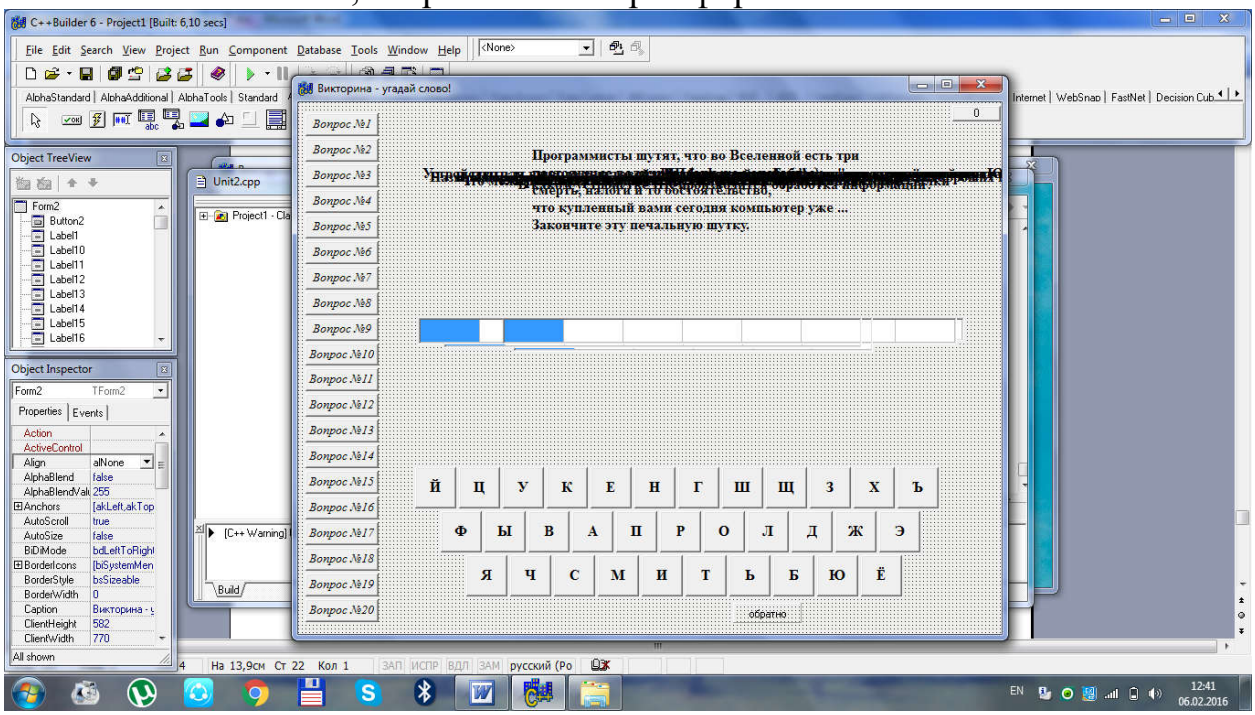
```
Form2->Show();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::SpeedButton3Click(TObject *Sender)
{
Form1->Close();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::SpeedButton2Click(TObject *Sender)
{
Form1->Hide();
Form3->Show();
}
```

2.2 Создание второй формы и добавление элементов, соответственно для игры

Нажав на «НАЧАТЬ», открывается вторая форма



Мы тут создали 20 кнопок для вопросов, и 20 Label для текста вопроса, 20 StringGrid для таблицы ответа, и 33 кнопок (Button) для букв на русском языке, и 1 Button для очистки StringGrid.
В каждом вопросе имеется ответ переменной, т.е.:

```
void __fastcall TForm2::question1Click(TObject *Sender)
{
```

```
Button2Click(Form2);
answer = "ДЖОЙСТИК";
StringGrid1->Show();
StringGrid2->Hide();
StringGrid3->Hide();
StringGrid4->Hide();
StringGrid5->Hide();
StringGrid6->Hide();
StringGrid7->Hide();
StringGrid8->Hide();
StringGrid9->Hide();
StringGrid10->Hide();
StringGrid11->Hide();
StringGrid12->Hide();
StringGrid13->Hide();
StringGrid14->Hide();
StringGrid15->Hide();
StringGrid16->Hide();
StringGrid17->Hide();
StringGrid18->Hide();
StringGrid19->Hide();
StringGrid20->Hide();
Label1->Show();
Label2->Hide();
Label3->Hide();
Label4->Hide();
Label5->Hide();
Label6->Hide();
Label7->Hide();
Label8->Hide();
Label9->Hide();
Label10->Hide();
Label11->Hide();
Label12->Hide();
Label13->Hide();
Label14->Hide();
Label15->Hide();
Label16->Hide();
Label17->Hide();
Label18->Hide();
Label19->Hide();
Label20->Hide();
}
```

Скрыв все ненужные таблицы и текста мы соответственно выбиваем и отвечаем. Для StringGrid мы только изменяем количество столбцов.
А для Button буквы код выглядит следующим образом:

```
void __fastcall TForm2::letterAClick(TObject *Sender)
{
int l = answer.Length();
for (int i=1;i<=l;i++)
{
  if (answer[i] == 'A'){
    StringGrid1->Cells[i-1][0] = "A";
    StringGrid2->Cells[i-1][0] = "A";
    StringGrid3->Cells[i-1][0] = "A";
    StringGrid4->Cells[i-1][0] = "A";
    StringGrid5->Cells[i-1][0] = "A";
    StringGrid6->Cells[i-1][0] = "A";
    StringGrid7->Cells[i-1][0] = "A";
    StringGrid8->Cells[i-1][0] = "A";
    StringGrid9->Cells[i-1][0] = "A";
    StringGrid10->Cells[i-1][0] = "A";
    StringGrid11->Cells[i-1][0] = "A";
    StringGrid12->Cells[i-1][0] = "A";
    StringGrid13->Cells[i-1][0] = "A";
    StringGrid14->Cells[i-1][0] = "A";
    StringGrid15->Cells[i-1][0] = "A";
    StringGrid16->Cells[i-1][0] = "A";
    StringGrid17->Cells[i-1][0] = "A";
    StringGrid18->Cells[i-1][0] = "A";
    StringGrid19->Cells[i-1][0] = "A";
    StringGrid20->Cells[i-1][0] = "A";
  }
}
}
```

Но надо учитывать что нам нужен переменная ответа:

```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;
```

String answer = "";

И для в углу Button мы пишем код очистки StringGrid:

```
void __fastcall TForm2::Button2Click(TObject *Sender)
{
for (int i=0;i<=StringGrid1->ColCount-1;i++){
    StringGrid1->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid2->ColCount-1;i++){
    StringGrid2->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid3->ColCount-1;i++){
    StringGrid3->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid4->ColCount-1;i++){
    StringGrid4->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid5->ColCount-1;i++){
    StringGrid5->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid6->ColCount-1;i++){
    StringGrid6->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid7->ColCount-1;i++){
    StringGrid7->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid8->ColCount-1;i++){
    StringGrid8->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid9->ColCount-1;i++){
    StringGrid9->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid10->ColCount-1;i++){
    StringGrid10->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid11->ColCount-1;i++){
    StringGrid11->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid12->ColCount-1;i++){
    StringGrid12->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid13->ColCount-1;i++){
    StringGrid13->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid14->ColCount-1;i++){
```

```
    StringGrid14->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid15->ColCount-1;i++){
    StringGrid15->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid16->ColCount-1;i++){
    StringGrid16->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid17->ColCount-1;i++){
    StringGrid17->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid18->ColCount-1;i++){
    StringGrid18->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid19->ColCount-1;i++){
    StringGrid19->Cells[i][0] = " ";
}
for (int i=0;i<=StringGrid20->ColCount-1;i++){
    StringGrid20->Cells[i][0] = " ";
}
}
```

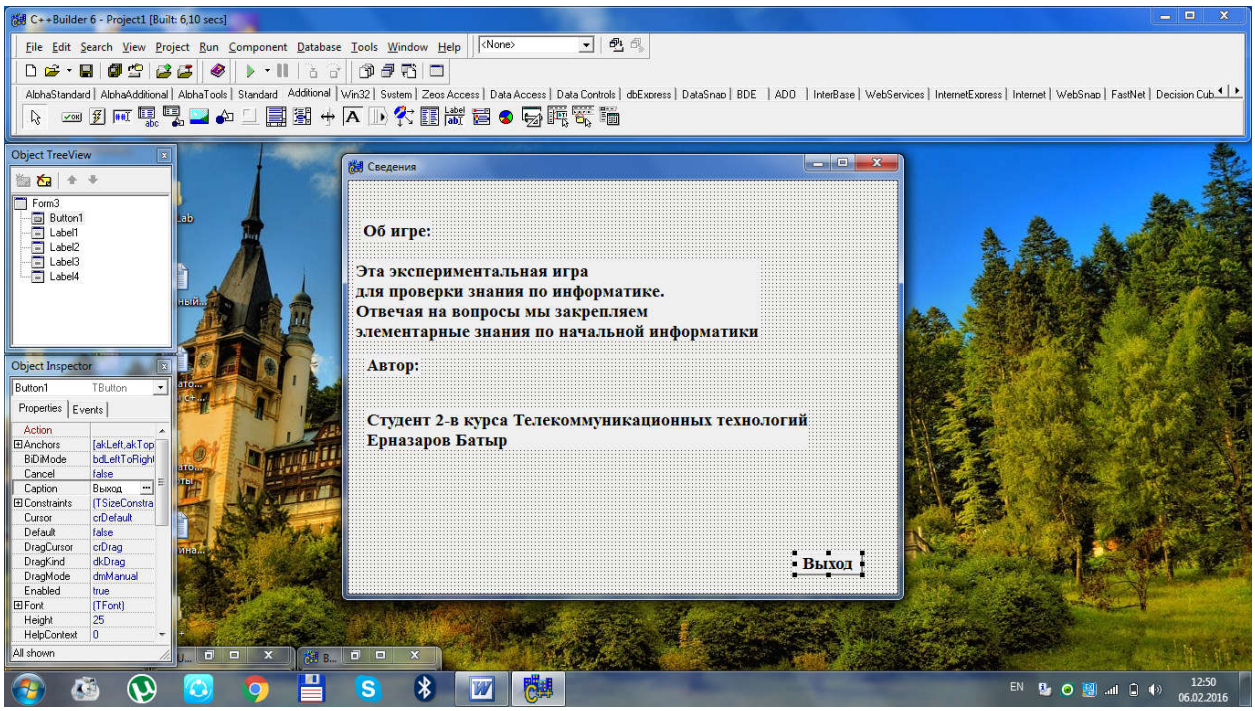
И последнюю кнопку для возврата на Form1:

```
void __fastcall TForm2::SpeedButton1Click(TObject *Sender)
{
Form2->Close();
Form1->Show();
}
```

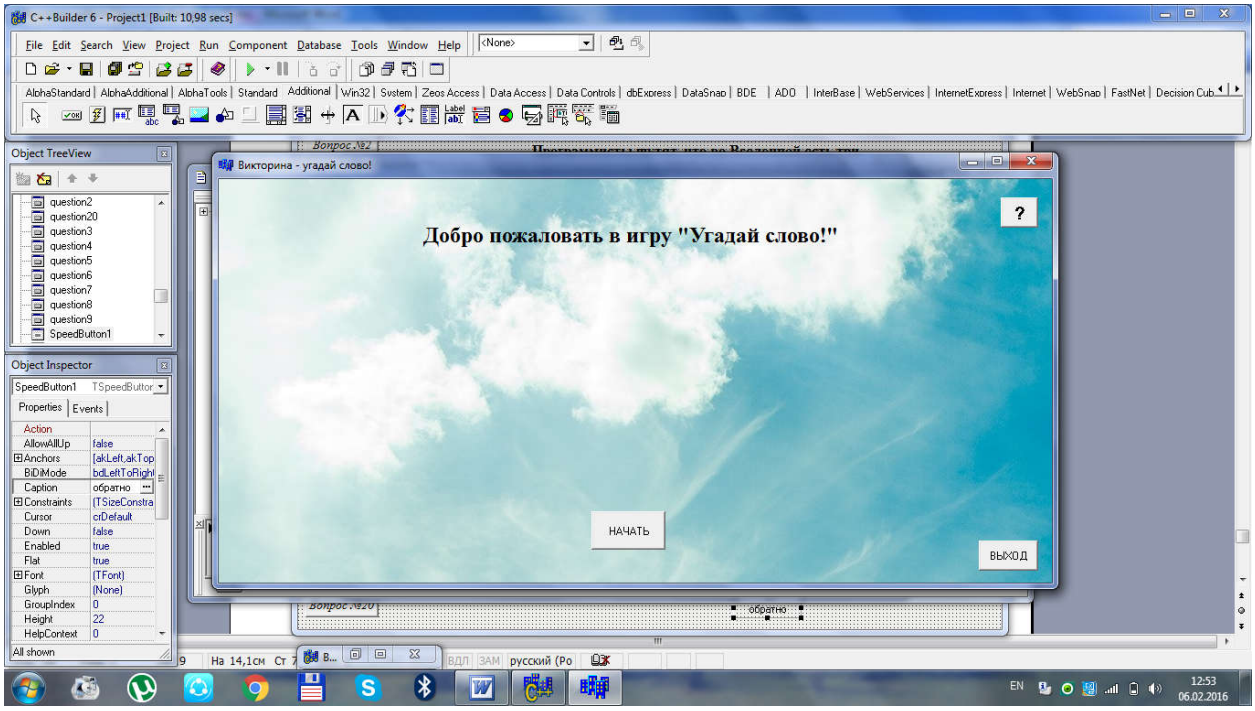
2.3 Создание третий формы для информации об игре и об авторе

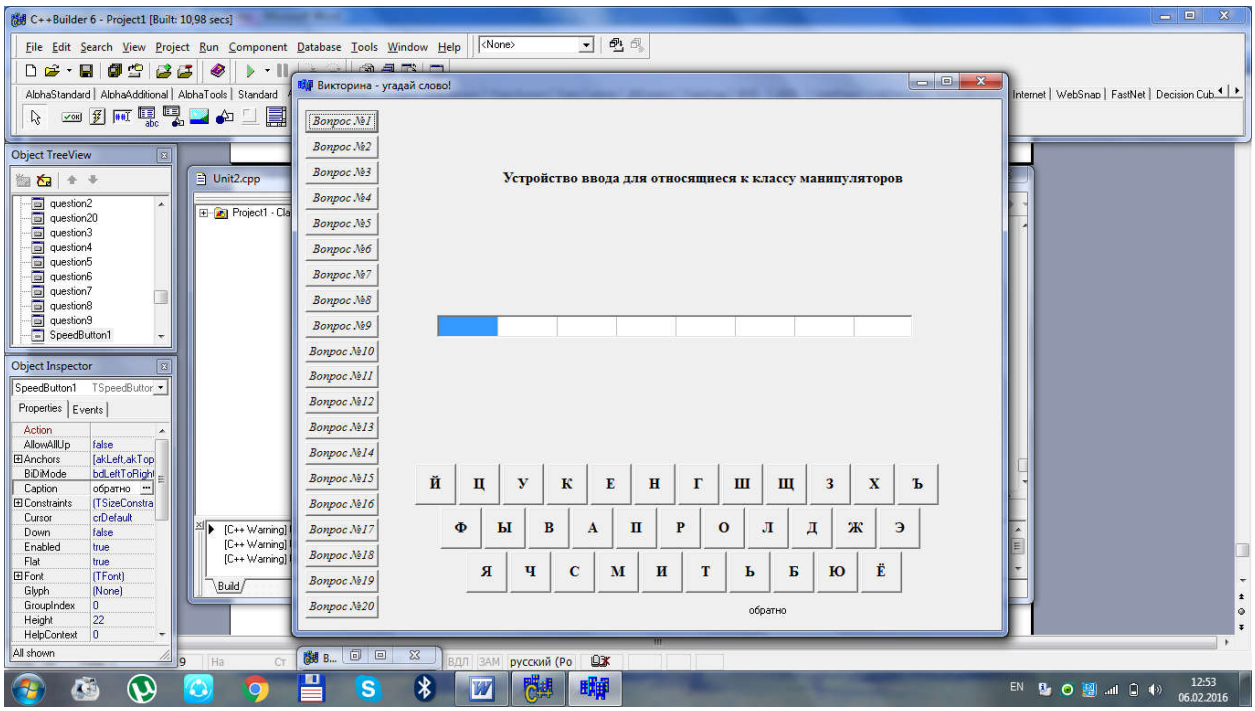
И наконец для последней формы мы добавляем только 4 Label и 1 Button.
В тексте Label я написал кратко о себе и об игре, и Button для возврата на Form1:

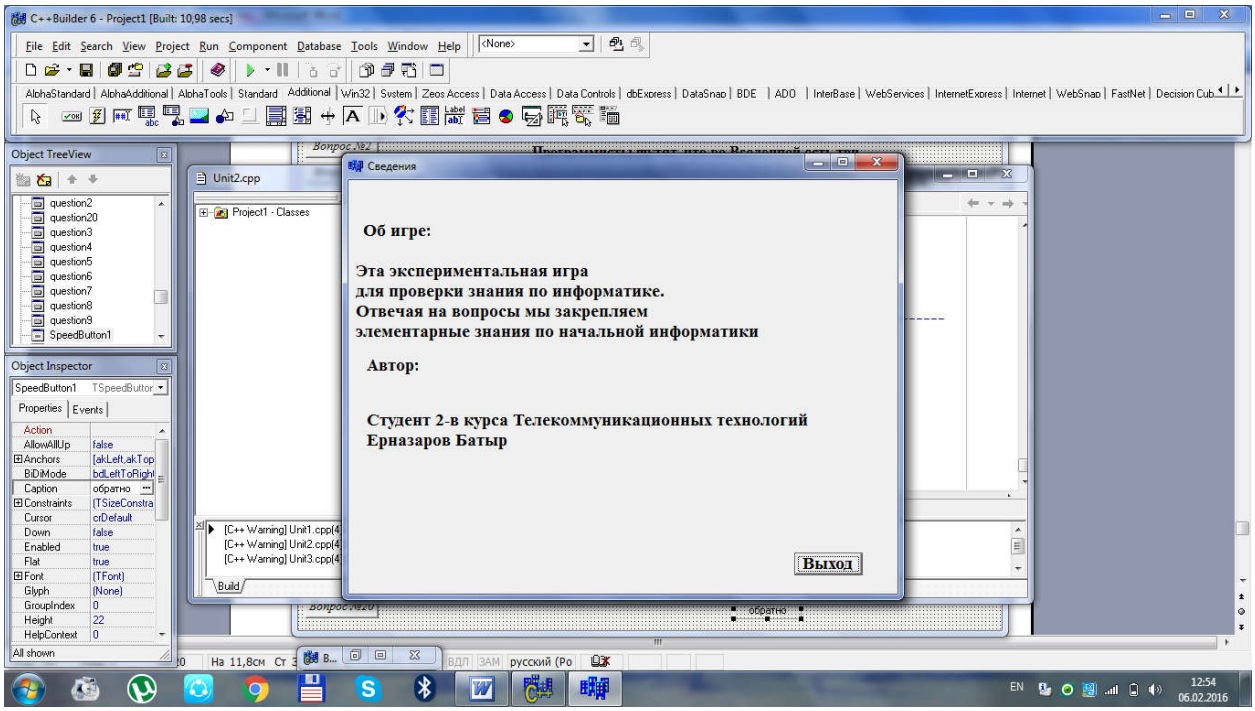
```
void __fastcall TForm3::Button1Click(TObject *Sender)
{
Form3->Close();
Form1->Show();
}
```



И в конце мы получаем готовую программу игры «Викторина»:




C++Builder 6 - Project1 [Built: 10,98 secs]
Викторина - угадай слово!
Вопрос №1
Вопрос №2
Вопрос №3
Вопрос №4
Вопрос №5
Вопрос №6
Вопрос №7
Вопрос №8
Вопрос №9
Вопрос №10
Вопрос №11
Вопрос №12
Вопрос №13
Вопрос №14
Вопрос №15
Вопрос №16
Вопрос №17
Вопрос №18
Вопрос №19
Вопрос №20
Устройство ввода для относящиеся к классу манипуляторов
Й Ц У К Е Н Г Ш Щ З Х Ъ
Ф Ы В А П Р О Л Д Ж Э
Я Ч С М И Т Ь Б Ю Ё
обратно
Object TreeView
Object Inspector
Unit2.cpp


C++Builder 6 - Project1 [Built: 10,98 secs]
Сведения
Об игре:
Эта экспериментальная игра
для проверки знания по информатике.
Отвечая на вопросы мы закрепляем
элементарные знания по начальной информатики
Автор:
Студент 2-в курса Телекоммуникационных технологий
Ерназаров Батыр
Выход
Object TreeView
Object Inspector
Unit2.cpp

Заключение

C++ Builder — программный продукт, инструмент быстрой разработки программного обеспечения на языке программирования C++. Системы быстрой разработки основываются на технологии визуального проектирования и событийного программирования. Программирование в подобных системах сводится в основном к наглядному строительству приложения из готовых компонентов, которые содержит сама среда и к написанию функций обработки тех или иных событий, на которые способна реагировать Windows. Такой подход сильно уменьшает время написания программы, так-как большую часть кода наша среда автоматически генерирует самостоятельно.

У нас огромное количество возможностей использование этой программы. Не только для игры, а также создание неких программ, например для использования в быту либо в определенных профессиях

Список используемой литературы

1. Неформальное введение в C++ и Turbo Vision / Под ред. И. И. Дериева. — Ленинград: Галерея «Петрополь», 1992. — 384 с.
2. *Касаткин А.И., Вальвачев А.Н.* Профессиональное программирование на языке Си: От Turbo C++ к Borland C++ / Под общей ред. А. И. Касаткина. — Минск: «Вышэйшая школа», 1992. — 240 с. — ISBN 5-339-00807-X.
3. Справочник по классам C++ 3.1/4.0 / Под ред. И. И. Дериева. — Киев: Диалектика, 1994. — 256 с. — ISBN 5-7707-6293-4.
4. Справочник по библиотеке Object Wiindows 2.0 для C++ / Под ред. И. И. Дериева. — 2-е изд. — Киев: Диалектика, 1995. — 494 с. — ISBN 5-777-6294-2.
5. *Киммел П.* Borland C++ 5 = Using Borland C++ 5 / Под ред. И. И. Дериева. — СПб.: BHV, 1997. — 976 с. — ISBN 5-7791-0053-5.