Практическая работа №11, Угадай слово

Постановка задачи

Разработать программу, которая реализует логическую игру «Угадай слово», известную каждому с детства.

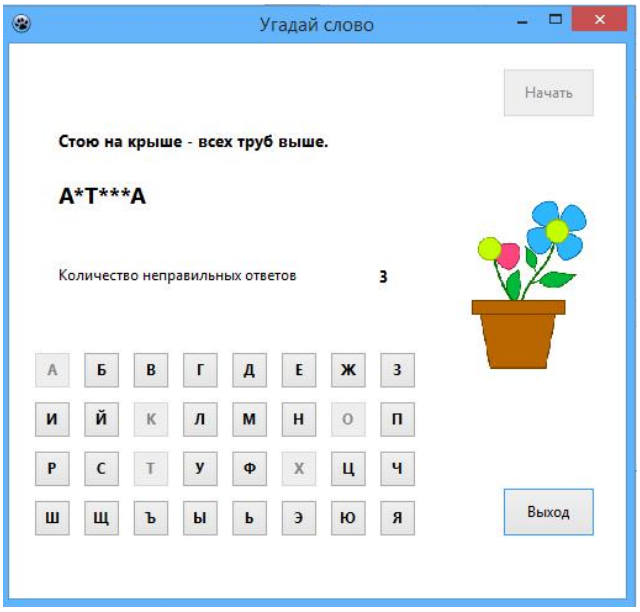

Рис.11.1

Правила игры

Ведущий загадывает слово, которое надо угадать, и записывает черточки, количество которых равно количеству букв в загаданном слове.
(Для подсказки можно использовать вопрос, ответом на который является загаданное слово.) Игроку предоставляется право выбора букв по одной, при этом он может допустить только десять ошибок (попыток).
Игрок называет букву, которая на его взгляд может присутствовать в слове. Если выбранная игроком буква есть в загаданном слове, то ведущий записывает ее вместо черточки на своем месте. Если буква встречается в слове несколько раз, то ведущий «открывает» все эти буквы. Если буква, названная игроком, отсутствует в слове, то уменьшается количество предоставленных попыток.
Игра заканчивается, если слово угадано, или если исчерпаны все предоставленные попытки.

Новым в этой работе являются:

- создание компонент во время выполнения программы и обработка их событий,
- вывод иллюстраций.

Информационная постановка задачи

1. Информация о вопросах и ответах хранится в текстовом файле (**`Questions.txt`**). На каждый вопрос-ответ отводится две строки: в первой строке – вопрос, а во второй – ответ.
2. Информация из текстовых файлов в начале программы считывается в массив (**`AQ`**), из которого затем случайным образом выбираются тексты вопросов.

3. Выбранный вопрос выводится на экран, а ответ записывается в переменную **POISK_WORD**.
4. Формируется угадываемое слово в виде черточек (переменная **POISK_WORD _NEW**). Количество черточек соответствует количеству букв (например, **POISK_WORD _NEW ='– – – – – –'**).
5. Для выбора отгадываемых букв слова используются кнопки-буквы, которые создаются в ходе программы.
6. При нажатии на кнопку с буквой, которая есть в слове, эта буква появляется на нужном месте, т.е. меняется переменная **POISK_WORD _NEW** (например, **POISK_WORD _NEW ='–A–A– –'**). Если такой буквы нет, то уменьшается количество попыток (переменная **ATTEMPT**). Кнопка, содержащая нажатую букву, в дальнейшем становится недоступной.
7. При угадывании слова (**POISK_WORD = POISK_WORD _NEW**) игроку предоставляется возможность начать новую игру или выйти из игры.
8. При использовании всех попыток (**ATTEMPT=0**) на экране появляется не угаданное слово и предоставляется возможность начать новую игру или выйти из игры.

План разработки программы

Построение программы будем выполнять поэтапно.

Первый этап. Формирование формы экрана

1. Откройте новый проект.

2. Разместите в форме экземпляры компонент в соответствии с рис.14.2.

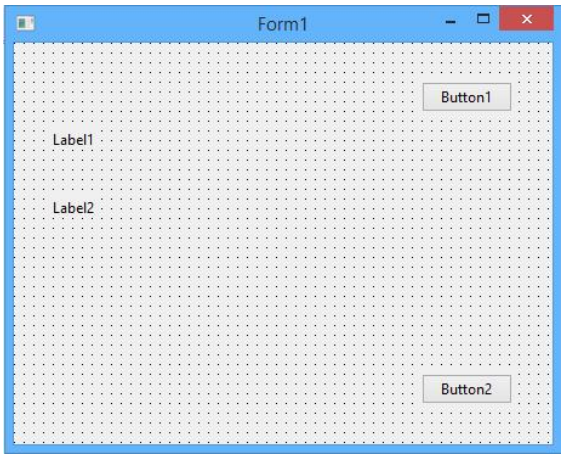

Рис.11.2

3. Выделите объект **Form1**, перейдите на вкладку **Events (События) Инспектора объектов** (**Object Inspector**), найдите событие **OnCreate**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий текст:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
// Формирование элементов формы
  With Form1 do
  begin
   Caption:='Угадай слово';
   Height:=450;
   Width:=500;
   Color:=clWhite;
```

```
  end;
  With Button1 do
  begin
    Caption:='Начать';
    Height:=40;
    Width:=75;
    Top:=20;
    Left:=400;
    Font.Size := 9;
  end;
  With Button2 do
  begin
    Caption:='Выход';
    Height:=40;
    Width:=75;
    Top:=360;
    Left:=400;
    Font.Size := 9;
  end;
  With Label1 do
  begin
    Caption:='';
    Height:=40;
    Width:=250;
    Top:=70;
    Left:=40;
    Font.Style:=[fsBold];
    Font.Size := 10;
  end;
  With Label2 do
  begin
    Caption:='';
    Height:=40;
    Width:=75;
    Top:=110;
    Left:=40;
    Font.Style:=[fsBold];
    Font.Size := 12;
  end;
 Read_File; //Процедура чтения информации из файла
end;
```

Пояснение

Для компонент **Form1**, **Button1**, **Button2**, **Label1**, **Label2** в тексте программы определены все необходимые свойства (размеры, местоположение и т.п.), поэтому при размещении компонент нет необходимости устанавливать свойства компонент с помощью вкладки **Properties (Свойства) Инспектора объектов** (**Object Inspector**).

4. Разместите в блоке реализации после слова **implementation** описание констант, типов и переменных, которые будут использоваться в программе:

```
Const
   M=100;                    //Максимальное количество записей в массиве
TYPE
   T_R = Record             //Структура записи массива
          Que:string[250];
          Ans:string[30];
         end;
   R=array[1..N_Z]of T_R;
Var
   AQ:R;                    //Массив Вопросов и Ответов
   Questions_F:TextFile; //Файловая переменная
   POISK_WORD,POISK_WORD_NEW:String; //Отгадываемое слово
   KOL_QUE:Integer;         //Количество вопросов в файле
   Size_WORD: Integer;      //Длина отгадываемого слова
```

5. Разместите после блока описания переменных процедуру чтения информации из файла и формирование массива вопросов **AQ**. К этой процедуре происходит обращение в конце процедуры **TForm1.FormCreate** (см.п.3).

```
procedure Read_File;
//Чтение информации из файла и запись в массив
Var KOL, I:Integer;
begin
 Assignfile(Questions_F,'Questions.txt');
 Reset(Questions_F);
 KOL:=1; I:=1;
 While not Eof(Questions_F) do
  begin
   if (KOL mod 2)=1 then Readln (Questions_F, AQ[I].QUE)
                    else
                    begin
                     Readln(Questions_F, AQ[I].ANS);
                     Inc(I);
                    end;
   Inc(KOL);
  end;
 KOL_QUE:=I;
 closefile(Questions_F);
end;
```

6. Создадим процедуру, которая обрабатывает ситуацию нажатия кнопки **Button1** (Начать игру). Для этого выделите объект **Button1**, перейдите на **вкладку Events Инспектора объектов** (**Object Inspector**), найдите событие **OnClick**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий текст:

```
procedure TForm1.Button1Click(Sender: TObject);
Var I,NUMBER:integer;
begin
 Randomize;
 NUMBER:=Random(KOL_QUE-1)+1;          //Случайное число - номер записи
 Label1.Caption:=AQ[NUMBER].QUE;
 POISK_WORD:=AQ[NUMBER].ANS;
 POISK_WORD_NEW:='';
```

```
 Size_WORD:= Length(POISK_WORD);      //Длина отгадываемого слова
 POISK_WORD_NEW:='';
 For I:=1 to Size_WORD do
   POISK_WORD_NEW:=POISK_WORD_NEW+'*';
 Label2.Caption:=POISK_WORD_NEW;
 Form1.Button1.Enabled:=False;
end;
```

Пояснение

Начало игры означает, что нужно выбрать вопрос, который затем выводится на экран (**Label1**), и сформировать зашифрованное слово (**Label2**, **POISK_WORD_NEW**), которое будет отгадывать игрок. Выбор вопроса из массива производится с помощью функции Random.

7. Самостоятельно добавьте событие обработки кнопки **Button2** (Выход).

8. Создайте текстовый файл **Questions.txt** с помощью программы *Блокнот*. Файл запишите в ту же папку, где находится и программа, при этом **используя кодировку UTF-8**.

Текст файла может выглядеть так:

```
Стою на крыше - всех труб выше.
АНТЕННА
Всех на свете обшивает, а сама не надевает.
ИГОЛКА
Твой хвостик я в руке держал, ты полетел, я побежал.
ШАРИК
Золотое решето, черных домиков полно.
ПОДСОЛНУХ
Ах, не трогайте меня. Обожгу и без огня!
КРАПИВА
```

9. Сохраните проект, запустите и протестируйте его. После запуска программы на экране видны две кнопки - **Начать** и **Выход**. Если нажать кнопку **Начать**, то появляется текст вопроса и зашифрованное слово ответа, при этом кнопка **Начать** становится недоступной. Проверьте, чтобы при каждом новом запуске программы у вас всегда выбирался новый текст ответа.

10. Если вы были внимательными, то обратили внимание, что при выводе ответа в зашифрованном виде (звездочки) количество символов удвоено. Например, как на рисунке 11.3 вместо 7 звездочек выведено 14. В чем причина?

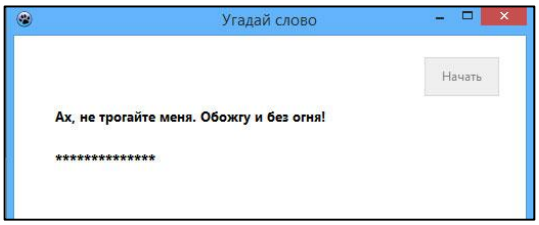

Рис.11.3

Немного теории

Для рядового пользователя эти понятия «символ» и «строка» весьма абстрактны. Когда он вводит с клавиатуры «А», то считает это символом, буквой. Когда вводит «4», то это - цифра. А про пробелы, знаки препинания или арифметические знаки он и вовсе не думает. Но программист должен понимать, как все эти знаки воспринимаются компьютером.

На самом деле, все, что мы вводим с клавиатуры - это символы. Буква «z», цифра «3», *пробел*, знак умножить, знак процента и т.д. – все это символы. *Компьютер* же может оперировать только цифрами, причем двоичными – такими, которые содержат лишь 0 или 1. Все эти буквы, десятичные цифры и прочие знаки для него не значат ровным счетом, ничего. И для того, чтобы мы могли как-то обрабатывать текст, цифры и прочую информацию, нужно было придумать специальную систему для перевода информации в понятную компьютеру, и обратно. Так появились **кодовые страницы.**

Кодовая страница *(англ. code page) - специальная таблица, сопоставляющая каждому значению байта некоторый символ.*

На заре развития компьютеров была разработана кодовая страница **ASCII** (англ. *American Standard Code for Information Interchange* - Американский кодовый стандарт для обмена информацией). Первая версия этого стандарта появилась в 1963 г. Эта страница содержала 7-ми битные символы, в каждом байте один *бит* был не задействован. Минимальное двоичное число, которое могло храниться в 7-ми битах - это ноль. Максимальное – 1111111 или 127 в десятичной системе счисления. Поэтому 128 символов содержалось в первой *ASCII* таблице. Помимо латинских букв, *таблица* содержала и другие символы – цифры, арифметические знаки, знаки препинания, символы пробела, скобки и т.п. Каждому символу соответствовал собственный номер в таблице. Если мы вводили английскую букву «А», то в *компьютер* попадал номер этого символа в таблице – 65. Или, в двоичном виде, 100 0001. Таким образом, символы можно было сравнивать между собой. Английское «В» находилось под номером 66 и, следовательно, было больше, чем «А». Строчные буквы имели другие номера, например, «а» шла в таблице под номером 97 и считалась большей, чем «А».

Если мы вводили число «65», то для компьютера это не было числом 65. Символу «6» соответствует номер 54 кодовой таблицы, а символу «5» – номер 53.

127 символов было явно недостаточно, чтобы можно было вводить текст и на других языках. Поэтому *таблица ASCII* развивалась, из года в год появлялись новые стандарты. Каждый символ стал уже 8-ми битным. Посмотрите на калькуляторе – восемь *бит* могут содержать максимальное число 1111 1111, при переводе в десятичную систему мы получим 255 символов. Первая половина таблицы оставалась неизменной, зато вторую половину таблицы можно было задействовать для символов других языков и псевдографики, с помощью которой программисты времен *MS-DOS* рисовали окошки, панельки, таблицы и *меню*. Однако и этого было слишком мало, чтобы закодировать символы всех языков на Земле. Для каждого языка приходилось разрабатывать свой стандарт, несовместимый с другими. Причем, для одного языка могло быть разработано несколько стандартов. Для русского языка, например, имеются стандарты CP866 (*Code Page* 866), *KOI8-R,* KOI8-U, ISO-8859-5, и это только самые распространенные. На смену *ASCII* пришла *кодировка* **ANSI** (англ. *American National Standards Institute* - Американский Национальный Институт Стандартов). Так, в MS *Windows* кодовая страница *ANSI*, содержащая кириллицу - это *Windows*-1251 (или CP1251), которая появилась в 1990-1991 гг.

Однако и этого было недостаточно, ведь для каждого языка по-прежнему требовалась собственная *кодировка*, а языков на Земле много. Назрела необходимость переходить к «широким» стандартам, в которых символ занимает более одного байта. Так, в 1991 г. был предложен стандарт **Юникод** (англ. *Unicode*) - универсальная система кодирования символов, представляющая знаки практически всех языков. В этом стандарте в одном документе можно использовать символы кириллицы, китайские или японские иероглифы, знаки математических формул, музыкальные знаки и т.п.

Первая версия Юникода имела фиксированный размер символов 2 байта (16 *бит*). В одной кодировке уже можно было использовать 65 535 символов. Однако оказалось, что и этого недостаточно. Юникод получил дальнейшее развитие, и из года в год стали появляться новые версии и стандарты, основанные на Юникоде. Имеются такие стандарты, как **UTF-8** (англ. *Unicode Transformation Format*, 8 *bit* - 8-ми битный формат преобразования Юникода), **UTF-16**, **UTF-32**.

В Lazarus, в основном, используется формат *UTF-8*.

UTF-8 появился 2 сентября 1992 года. Основное его отличие от первоначального Unicode в том, что в *UTF-8* символы имеют **не фиксированный** размер. Если используются символы с номером меньшим, чем 128, то они занимают 1 *байт*, как обычный *ASCII*-текст. Символы с номером от 128 и больше могут занимать от 2 до 6 *байт* (реально максимальный размер символа - 4 байта, т.к. в Юникоде нет символов с большим номером). Символы кириллицы занимают, например, по 2 байта. Так что *цепочка символов* в 5 *байт* в *UTF-8* не всегда означает строку из пяти символов.

Такой подход делает *UTF-8* самой экономичной кодировкой для совместимости со старыми стандартами, однако есть и минусы. К сожалению, для русскоязычных (и вообще для всех не англоязычных) пользователей *Windows* в Lazarus придется столкнуться с некоторыми проблемами применения различных кодировок: в Lazarus используется *кодировка UTF-8*, ОС *Windows* использует *UTF-16*, а в консольных приложениях *Windows* используется системная *кодировка* CP866 (то есть, стандарт *ANSI*). Ее же используют некоторые функции компилятора FPC. Так что в некоторых случаях нам придется пользоваться функциями преобразования кодировок, например, UTF8ToANSI(), ANSIToUTF8() и т.п.

В LAZARUS имеется тип **TUTF8Char**, который позволяет работать с **любыми** отдельными символами, в том числе и русскими. Имеет смысл всегда использовать его вместо стандартного **Char**. Но для этого нам нужно будет подключить *модуль* **LCLType**, где этот тип описан.

По материалам лекций «Программирование на Lazarus» (ИНТУИТ), автор: Вячеслав Ачкасов

11. Перейдите в Редактор кода и пролистайте код модуля вверх. Там вы увидите раздел uses (англ. *use* – использовать), где перечислены подключенные модули. Нам нужно поставить после последнего модуля запятую и добавить наш *подключаемый модуль*:

```
uses
Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs,
StdCtrls, LCLType;
```

12. Внесите изменения в процедуру TForm1.Button1Click:

Первоначальный текст	Откорректированный текст

`procedure TForm1.Button1Click(Sender: TObject);`
`Var I,NUMBER:integer;`
`begin`
` Randomize;`
` NUMBER:=Random(KOL_QUE-1)+1;`
` Label1.Caption:=AQ[NUMBER].QUE;`
` POISK_WORD:=AQ[NUMBER].ANS;`
` POISK_WORD_NEW:='';`
` Size_WORD:= Length(POISK_WORD);`
` POISK_WORD_NEW:='';`
` For I:=1 to Size_WORD do`
`  POISK_WORD_NEW:=POISK_WORD_NEW+'*';`
` Label2.Caption:=POISK_WORD_NEW;`
` Form1.Button1.Enabled:=False;`
`end;`
	`procedure TForm1.Button1Click(Sender: TObject);`
	`Var I,NUMBER:integer;`
	`begin`
	` Randomize;`
	` NUMBER:=Random(KOL_QUE-1)+1;`
	` Label1.Caption:=AQ[NUMBER].QUE;`
	` POISK_WORD:=AQ[NUMBER].ANS;`
	` POISK_WORD_NEW:='';`
	` **POISK_WORD:=UTF8ToAnsi(POISK_WORD);**`
	` Size_WORD:= Length(POISK_WORD);`
	` POISK_WORD_NEW:='';`
	` For I:=1 to Size_WORD do`
	`  POISK_WORD_NEW:=POISK_WORD_NEW+'*';`
	` Label2.Caption:=POISK_WORD_NEW;`
	` Form1.Button1.Enabled:=False;`
	`end;`

Проверьте правильность работы программы.

Второй этап. Создание компонент во время выполнения программы и обработка их событий

Для дальнейшей работы над проектом нам необходимо создать 33 кнопки, которые будут принимать значения букв русского алфавита, и обрабатывать ситуацию поиска выбранной буквы (нажатой кнопки). Эта задача не сложная, но очень утомительная – 33 раза повторить одни и те же операции. Как упростить этот процесс покажем на отдельном примере, для этого создадим **отдельный проект** (в отдельной папке).

В данном примере по нажатию кнопки **Button1** создаются 32 кнопки, которые располагаются в три ряда. Свойства **Caption** этих кнопок принимают значения букв русского алфавита от «А» до «Я» (кроме буквы «Ё»). По нажатию каждой кнопки идет обращение к процедуре **TForm1.BtnClick**, которая определяет какая буква русского алфавита соответствует нажатой кнопке. Дальше идет обращение к процедуре **POISK**, и нажатая кнопка становится недоступной.

Процедура **POISK** ищет встречается ли выбранная буква (нажатая кнопка) в заданном слове **POISK_WORD**, и если да, то заменяется в **POISK_WORD_NEW** соответствующий символ на заданную букву. Но здесь возникает трудность при сравнение отдельных символов из текстового файла (формат ANSI) и названия созданных клавиш (формат UTF-8). Для упрощения работы введем массив символов **STR_N** типа TUTF8Char.

1. Откройте **новый** проект.

2. Разместите в форме экземпляры компонентов **Label1** и **Button1**.

3. Разместите в блоке реализации после слова implementation описание констант и переменных:

```
Const N=32;              //Количество букв-кнопок алфавита
 Type STR_30=Array[1..30] of TUTF8Char;
 Var Btn: TButton;        // Переменной для создания кнопок
     STR_N :STR_30;
     CHAR_:TUTF8Char;     // Имя нажатой клавиши-буквы
     Size_WORD:Integer;   // Размер отгадываемого слова
     POISK_WORD,POISK_WORD_NEW:String;   // Отгадываемое слово
```

4. Для события **OnClick** компоненты **Button1** напишите такой текст:

```
procedure TForm1.Button1Click(Sender: TObject);
// Процедура создания кнопок
var
   I: integer;
   SYMBOL: Array[1..N] of TUTF8Char;
begin
// Таблица для кодирования создаваемых кнопок
SYMBOL[1]:='А'; SYMBOL[2]:='Б'; SYMBOL[3]:='В'; SYMBOL[4]:='Г'; SYMBOL[5]:='Д';
SYMBOL[6]:='Е'; SYMBOL[7]:='Ж'; SYMBOL[8]:='З'; SYMBOL[9]:='И'; SYMBOL[10]:='Й';
SYMBOL[11]:='К';SYMBOL[12]:='Л';SYMBOL[13]:='М';SYMBOL[14]:='Н';SYMBOL[15]:='О';
SYMBOL[16]:='П';SYMBOL[17]:='Р';SYMBOL[18]:='С';SYMBOL[19]:='Т';SYMBOL[20]:='У';
SYMBOL[21]:='Ф';SYMBOL[22]:='Х';SYMBOL[23]:='Ц';SYMBOL[24]:='Ч';SYMBOL[25]:='Ш';
SYMBOL[26]:='Щ';SYMBOL[27]:='Ъ';SYMBOL[28]:='Ы';SYMBOL[29]:='Ь';SYMBOL[30]:='Э';
SYMBOL[31]:='Ю';SYMBOL[32]:='Я';

for I:=1 to N do            // Цикл по созданию кнопок на форме
  begin
    // Создаем кнопку на Form1
```

```
    Btn:=TButton.Create(Form1);
    with Btn do
     begin
      Parent:= Form1; // Определение родителя (Form1), т.е. где создаются кнопки
      // Определение свойств создаваемых кнопок
      Height := 30;
      Width := 30;
      // Определяем место положение кнопки на форме
      Top := 200+((I-1) div 11)*40;
      Left := ((I-1) mod 11 ) * 40 + 30;
      Font.Size := 9;
      Font.Style:=[fsBold];
      // Определение буквы, которая будет изображена на кнопке
      Caption := SYMBOL[I];
      // Определение имени процедуры, обрабатывающей событие нажатия кнопки
      OnClick := @FORM1.BtnClick;
    end;
 end;

  POISK_WORD:='РОССИЯ';  // Пример слова
  POISK_WORD:=UTF8ToAnsi(POISK_WORD); //Преобразование в ANSII-код
  Size_WORD:= Length(POISK_WORD);      //Длина отгадываемого слова

  POISK_WORD_NEW:='';
  For I:=1 to Size_WORD do
  begin
  STR_N[I]:='*';
  POISK_WORD_NEW:=POISK_WORD_NEW+'*';
  end;
  Label1.Caption:=POISK_WORD_NEW;
  Form1.Button1.Enabled:=False;
 end;
```

5. Процедуру **`TForm1.BtnClick`** (обработка события нажатия созданных кнопок) мы создаем сами. Для этого вставьте текст этой процедуры в общий текст программы перед тем, как осуществляется обращение к процедуре **`TForm1.BtnClick`**.

```
procedure TForm1.BtnClick(Sender: TObject);
// Процедура обработки события нажатия созданных кнопок
begin
// Переменная Sender содержит имя объекта, которому соответствует данное событие
  CHAR_:=(Sender as TButton).Caption;
  POISK; //Обращение процедуре поиска буквы
  (Sender as TButton).Enabled:=False; //Кнопка буквы недоступна
end;
```

6. В раздел описания процедур (перед **`private`**) добавьте строку описания заголовка процедуры:

```
procedure BtnClick(Sender: TObject);
```

7. Для того, чтобы наша программа заработала необходимо внести объявление переменных и описание процедуры P**OISK**.

```
 Procedure POISK;
//Поиск буквы в слове и «открытие» ее
Var I:Integer;
      Flag:Boolean;      // Флаг найдена ли в слове нажатая буква
  begin
```

```
  Flag:=False;
  For I:=1 to Size_WORD do  // Поиск буквы в слове
   If AnsitoUTF8(POISK_WORD[I])=CHAR_ then
     begin
      STR_N[I]:=CHAR_;
      Flag:=True;
     end;

  If Flag=True Then // Изменение выводимой информации на экран
    begin
     POISK_WORD_NEW:='';
     For I:=1 to Size_WORD do
      POISK_WORD_NEW:=POISK_WORD_NEW+STR_N[I];
     Form1.Label1.Caption:=POISK_WORD_NEW;
    end;
  end;
```

8. Сохраните проект, запустите и протестируйте его. После запуска программы на экране видна только одна кнопка **Button1** и закодированное слово в виде звездочек. После нажатия на эту кнопку на экране появляется три ряда кнопок с буквами (см.рис.11.4) и кнопка **Button1** становится недоступной.

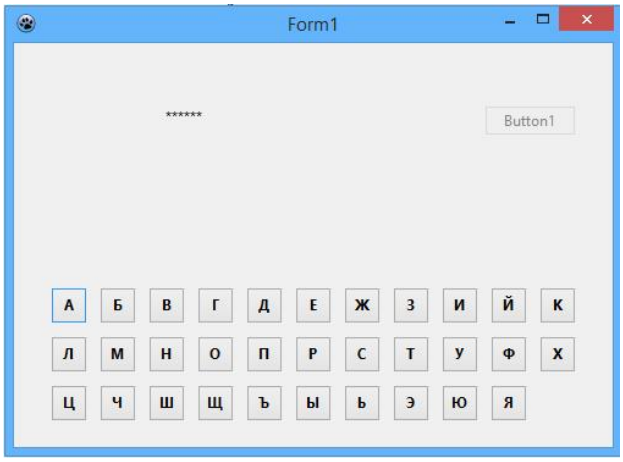

Рис.11.4

Нам известно слово, которое зашифровано – это слово «РОССИЯ». Протестируйте программу, введя буквы этого слова, и проверьте, чтобы они у вас «открылись» в зашифрованном слове.

Третий этап. Создание кнопок перед началом игры

Выполните этот этап самостоятельно, внеся изменения в основной текст программы, созданный на втором этапе. При создании кнопок расположите их не в три ряда, а в четыре (см.рис.11.5). Подумайте, как это сделать.

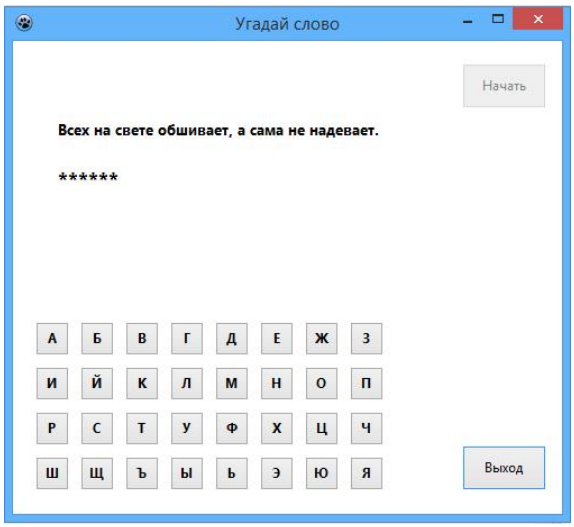

Рис.11.5

Внесите изменения в раздел описания констант, типов и переменных, которые будут использоваться в дальнейшем в программе (просто замените уже существующее описание):

```
Const
     M=100;                    //Максимальное количество записей в массиве
     N=32;                     //Количество букв-кнопок алфавита
     Attempt=10;               //Максимальное количество попыток
TYPE
     T_R = record              //Структура записи массива
        Que:string[250];
        Ans:string[30];
       end;
     R=array[1..M]of T_R;
     STR_30=Array[1..30] of TUTF8Char;
Var  AQ:R;                      //Массив Вопросов и Ответов
     Quetions_F:TextFile;       //Файловая переменная
     KOL_QUE:Integer;           //Количество вопросов в файле
     Btn: TButton;              // Переменной для создания кнопок
     STR_N:STR_30;
     CHAR_:TUTF8Char;           // Нажатая клавиша
     Size_WORD:Integer;         // Размер отгадываемого слова

     // Массив букв для кодирования кнопок
     POISK_WORD,POISK_WORD_NEW:String;   // Отгадываемое слово
     N_Attempt:Integer;         //Количество ошибок
     iw,ih: integer;            // Первоначальный размер компонента Image
     AFile: String;             // Имя картинки
```

Четвертый этап. Анализ состояния игры

Необходимо внести изменения в программу, для определения закончена ли игра, т.е. угаданы все буквы в зашифрованном слове. Вместе с тем установим количество допустимых ошибок игрока.

1. Разместите в блоке реализации описание дополнительных констант и переменных:

```
Const
  Attempt=10; //Максимальное количество попыток
Var N_Attempt:Integer;  //Количество ошибок
```

2. Разместите в форме экземпляры компонентов **Label3, Label4** (сообщение о количестве допущенных ошибок) и **Panel1** (сообщение о завершении игры).

3. Дополните текст процедуры **TForm1.FormCreate** описанием новых компонент:

```
With Label3 do
  begin
    Caption:='';
    Height:=40;
    Width:=75;
    Top:=180;
    Left:=40;
    Font.Size := 9;
  end;
  With Label4 do
  begin
    Caption:='';
    Height:=40;
    Width:=75;
    Top:=180;
    Left:=300;
    Font.Style:=[fsBold];
    Font.Size := 10;
  end;
  With Panel1 do
  begin
    Caption:='';
    Height:=45;
    Width:=185;
    Top:=15;
    Left:=110;
    Font.Style:=[fsBold];
    Font.Size:= 10;
    Visible:=False;
  end;
```

4. Вначале процедуры **TForm1.Button1Click** добавьте следующий текст для вывода сообщений о количестве неправильных ответов:

```
 Form1.Panel1.Visible:=False;
 Label3.Caption:='Количество неправильных ответов';
 N_Attempt:=0;
 Label4.Caption:=IntToStr(N_Attempt);
```

5. Внесите изменения в текст процедуры поиска **POISK** для анализа результата. Процедура может выглядеть следующим образом:

```
Procedure POISK ;
//Поиск буквы в слове и «открытие» ее
Var I:Integer;
      Flag:Boolean;          // Флаг - найдена ли в слове нажатая буква
  begin
  Flag:=False;
```

```
  For I:=1 to Size_WORD do  // Поиск буквы в слове
    If AnsitoUTF8(POISK_WORD[I])=CHAR_ then    // Буква найдена
     begin
      STR_N[I]:=CHAR_;     // Вставка найденной буквы в закодированное слово
      Flag:=True;
     end;
  If Flag=False Then        // Буква не найдена
    begin
      N_Attempt:=N_Attempt+1;
      Form1.Label4.Caption:=IntToStr(N_Attempt);
    end
 Else
   // Изменение выводимой информации на экран
    begin
     POISK_WORD_NEW:='';
     For I:=1 to Size_WORD do
      POISK_WORD_NEW:=POISK_WORD_NEW+STR_N[I];
     Form1.Label2.Caption:=POISK_WORD_NEW;
    end;
  If UTF8ToAnsi(POISK_WORD_NEW)=POISK_WORD then
  //Игра окончена - отгадано все слово
   begin
    With Form1.Panel1 do
     begin
      Visible:=True;
      Font.Color:=clBlack;
      Caption:='Вы выиграли!';
     end;
   Form1.Button1.Enabled:=True;
  end;
 If N_Attempt=Attempt then
//Игра окончена - количество ошибок равно количеству попыток
 begin
  With Form1.Panel1 do
   begin
    Visible:=True;
    Font.Color:=clRed;
    Caption:='Вы проиграли...';
   end;
  Form1.Button1.Enabled:=True;
  end;
end;
```

6. Сохраните проект, запустите и протестируйте его.

Пятый этап. Вывод иллюстраций

Для того, чтобы программа была более привлекательной, добавим вывод измененных изображений в зависимости от количества допущенных ошибок. Ряд изменения изображений может выглядеть так:




0 ошибок 1 ошибка 2 ошибки 3 ошибки и т.д.

Осталось только добавить эти изображения в программу, но сначала создадим самостоятельную программу, которая будет выводить изображения в зависимости от нажатия кнопки.

Немного теории

Для вывода иллюстрации используется компонент **Image**, который находится на вкладке **Additional** палитры компонентов.

В таблице приведены основные свойства компонента **Image**.

Имя свойства	Значение
Name	Имя компонента
Picture	Свойство, являющееся объектом типа **Tbitmap**. Определяет выводимую картинку
Left	Расстояние от левого края формы до левой границы области картинки
Top	Расстояние от верхней границы формы до верхней границы области картинки
Height	Высота картинки
Width	Ширина картинки
Stretch	Признак автоматического сжатия или растяжения картинки таким образом, чтобы она была видна полностью в области, размер которой задан свойствами **Width** и **Height**
AutoSize	Признак автоматического изменения размера компонента в соответствии с реальным размером картинки. Если значение свойства **AutoSize** равно **True**, то при изменении значения свойства Picture автоматически меняется размер области вывода иллюстрации так, чтобы была видна вся картинка. Если значение свойства **AutoSize** равно **False**, а размер картинки превышает размер области, то отображается только часть картинки

Картинку, отображаемую в области **Image**, можно задать во время создания формы или во время работы программы:

- во время создания формы картинка задается установкой значения свойства **Picture**.
- во время работы программы – применением метода **LoadFromFile**.

Например, для вывода иллюстрации, находящейся в файле **dog.bmp**:

```
Image1.Picture.LoadFromFile('dog.bmp');
```

При проектировании формы можно жестко задать предельный размер иллюстрации. И если реальный размер иллюстрации превышает размер области, выделенной для ее вывода, то необходимо вычислить коэффициент масштабирования и установить максимально возможные, пропорциональные ширине и высоте иллюстрации, значения свойств **Width** и **Height** области вывода иллюстрации. А если размер иллюстрации меньше области вывода, то можно пропорционально увеличить картинку.

Реальные размеры иллюстрации, загруженной в область **Image**, можно получить из свойств:

Image1.Picture.Bitmap.Width

Image1.Picture.Bitmap.Height.

Пример программы вывода картинок

Программа позволяет последовательно выводить на экран при нажатии на кнопку **Button1** три картинки, находящиеся в файлах **1.bmp**, **2.bmp**, **3.bmp**. После последней картинки будет выведена опять первая - **1.bmp**. При выводе на экран размер картинки масштабируется в зависимости от заданного размера компонента **Image.**

Откройте новый проект. Разместите на форме экземпляры компонентов **Image** и **Button**.

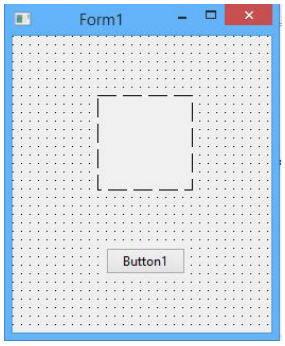

Рис.11.6. Исходная форма

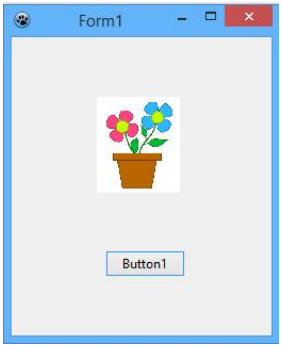

Рис.11.7. Выполнение программы

```
unit Unit1;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Image1: TImage;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
```

```
implementation
Var
   iw,ih: integer;    // Первоначальный размер компонента Image
   N_Image: integer; // Номер выводимой картинки
   AFile: String;     // Имя картинки

{$R *.dfm}
// изменение размера области вывода иллюстрации пропорционально
// размеру иллюстрации
Procedure ScaleImage;
// Масштабирование изображения
var
    pw, ph : integer;       // Размер иллюстрации
    scaleX, scaleY : real; // Масштаб по X и Y
    scale : real;           // Масштаб
begin
      // Иллюстрация уже загружена, получаем ее размеры
      pw := Form1.Image1.Picture.Width;
      ph := Form1.Image1.Picture.Height;
      scaleX := iw/pw;
      scaleY := ih/ph;

      // Выбераем наименьший коэффициент
      if  scaleX < scaleY
          then scale := scaleX
          else scale := scaleY;

      // Изменяем размер области вывода иллюстрации
Form1.Image1.Height := Round(Form1.Image1.Picture.Height*scale);
Form1.Image1.Width := Round(Form1.Image1.Picture.Width*scale);
// так как Strech = True и размер области пропорционален
// размеру картинки, то картинка масштабируется без искажений
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 If N_Image = 4 Then N_Image:=1;
 // Формирование имени картинки
 AFile:=IntToStr(N_Image)+ '.bmp';
 //Установка значения свойства Picture для вывода картинки
 // во время работы программы
 Form1.Image1.Picture.LoadFromFile(AFile);
 // Масштабирование картинки
 ScaleImage;
 N_Image:=N_Image+1;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 Image1.AutoSize := False;
 Image1.Stretch := True;      // Разрешим масштабирование
   // Запомним первоначальный размер области вывода иллюстрации
   iw := Image1.Width;
   ih := image1.Height;
   N_Image:=1;  // Начальное значение номера выводимой картинки
end;
end.
```

Шестой этап. Заключительный

Самостоятельно внесите изменения в программу, добавив вывод картинок в зависимости от количества допущенных ошибок.

Протестируйте полученную программу.

Программа имеет еще один недостаток - после окончания игры (вывод панели с сообщением об окончании игры), если нажимать кнопки с буквами, то программа может аварийно завершиться. Чтобы этого не было в начале процедуры Poisk выполните проверку:

```
If (N_Attempt<Attempt) and (STR_N<>STR_R) then
```

Т.е. выполнять поиск нажатой буквы только в случае, если не исчерпаны все попытки и не отгадано все слово.