Лабораторная работа № 11
Работа с таймером и анимация объектов

Цель работы: изучить компонент ***Timer*** и его применение при создании приложений, которые требуют регулярного выполнения некоторых действий с определенным временным интервалом.

Часть 1

Задание: Разработать программу моделирования движения точки по окружности заданного радиуса с постоянной скоростью. Предусмотреть возможность плавного изменения скорости ее движения.

Теоретические сведения

Компонент ***Timer***, размещенный на странице ***Additional***, относится к не визуальным компонентам. Он позволяет вводить необходимые задержки между выполнением тех или иных действий. Компонент ***Timer*** имеет два свойства, которые дают возможность управлять его работой:

- ***Enabled*** – «включение-выключение» таймера. Это свойство отвечает за работу таймера. Если его значение *True*, то таймер работает (активен), в противном случае (*False*) он будет находиться в неактивном состоянии.
- ***Interval*** – интервал времени в миллисекундах между последовательными событиями ***OnTimer***, генерируемыми таймером. Если задать нулевое значение данного параметра, то таймер перестанет работать.

Единственное событие, поддерживаемое компонентом ***Timer***, – ***OnTimer***. Оно происходит всякий раз по истечению промежутка времени, заданного в свойстве ***Interval***, если таймер активен. По умолчанию свойство ***Enabled*** установлено в значение *True*, т.е. после запуска программы таймер активен.

Компонент ***Timer*** не отличается высокой точностью измерения промежутков времени. Его точность порядка 50 миллисекунд.

Для плавного изменения скорости движения точки будем использовать компонент ***TrackBar***, находящийся на странице ***Win32***. Он представляет собой визуальный элемент управления в виде ползунка, который можно перемещать клавишами или мышью во время выполнения программы. Это позволяет наглядно управлять какими-либо параметрами работы программы, например, размером изображения или скоростью движения.

Рассмотрим основные свойства компонента ***TrackBar***:

- ***Position*** – значение целого типа, которое характеризует позицию ползунка и изменяется в пределах, задаваемых свойствами ***Min*** и ***Max***. По умолчанию они равны, соответственно, нулю и десяти. Изменяя значения свойств ***Min*** и ***Max***, можно увеличивать или уменьшать количество возможных значений свойства ***Position***.
- ***Orientation*** – горизонтальное (*trHorizontal*) или вертикальное (*trVertical*) перемещение ползунка.

- ***TickMarks*** – размещение шкалы относительно компонента: сверху при горизонтальном расположении или слева при вертикальном расположении (*tmTopLeft*); аналогично снизу или справа (*tmBottomRight*); с обоих сторон (*tmBoth*).

При перемещении ползунка генерируется событие ***OnChange***, при этом свойство ***Position*** характеризует позицию, в которой он находится.

Практическая часть

Создаем новый проект и изменяем у формы свойство ***Caption*** на название лабораторной работы. Сохраняем проект в рабочую папку командой ***File → Save Project as***.

Для моделирования движения точки используется свойство ***Canvas*** формы, т.е. рисовать будем непосредственно на форме. Компоненты управления разместим на панели в нижней части формы, «прижав» ее к нижней границе формы (***Align = alBottom***).

Радиус окружность задается в компоненте ***LabeledEdit*** (страница ***Additional***). Для запуска и остановки движения будем использовать одну кнопку. В начале работы программы при нажатии на нее точка начинает движение, а при повторном нажатии останавливается. Компонент ***TrackBar*** будет использоваться для плавного изменения скорости движения точки, т.е. значения свойства ***Interval*** компонента ***Timer***. Единицей измерения скорости выберем градусы в секунду, о чем напишем в метке, расположенной слева от компонента ***TrackBar***. Текущее значение скорости движения также будем выводить в метке, расположенной у левой верхней границы компонента ***TrackBar***.

Определим имена и некоторые параметры основных компонентов:

- ***LEditRad*** – радиус окружности (***Text*** *= '100'* – значение радиуса по умолчанию);
- ***LabelDS*** – информация о единицах измерения скорости движения точки (***Caption*** *= 'Скорость (град./сек)',* ***WordWrap*** *= true* – для того чтобы сообщение выводилось в две строки);
- ***ButtonMove*** – запуск/остановка движения (***Caption*** *= 'Запустить'*)
- ***TrackBarSpeed*** – регулятор скорости движения точки (***Min*** *= 1,* ***Max*** *= 90,* ***Position*** *= 45* – по умолчанию выбираем скорость движения точки 45 градусов в секунду);
- ***LabelSpeed*** – скорость движения точки (***Caption*** *= '45');*
- ***Timer1*** – таймер (***Enabled*** *= false* – таймер не активен, ***Interval*** = 22 – примерный интервал генерации прерываний, соответствующий угловому перемещению точки на 45 градусов за одну секунду, рассчитанный по формуле *Round(1000 * StepPoint / 45)*, где *StepPoint* – угол, на который поворачивается радиус-вектор точки за одно прерывание).

Интерфейс программы приведен на рис. 12.1.

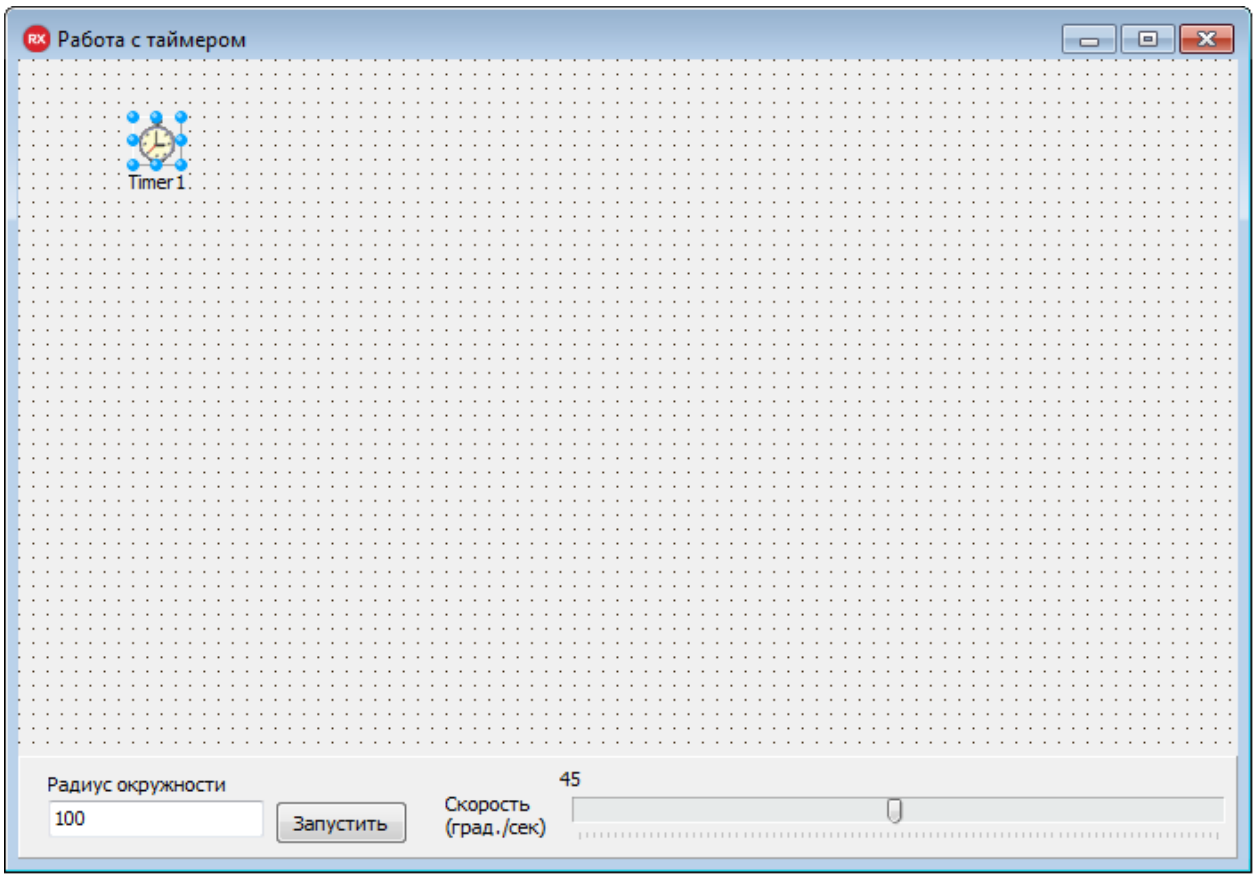

Рис. 12.1. Интерфейс программы

В секции ***Implementation*** опишем несколько переменных:

- ***RadCircle*** (***integer***) – радиус окружности, по которой вращается точка;
- ***CenterX, CenterY*** (***integer***) – координаты центра окружности вращения, которые будет совпадать с координатами середины компонента ***Image***;
- ***PointX, PointY*** (***integer***)– текущие координаты точки;
- ***Angle*** (***real***) – текущий угол в градусах, определяющий положение точки;
- ***RadPoint*** (***integer***) – «радиус» точки;
- ***StepPoint*** (***real***)– угловой шаг поворота радиус-вектора точки в градусах (от значения этого параметра будет зависеть «плавность» движения точки);
- ***PointColor*** (***TColor***) – цвет точки.

Для инициализации переменных создадим обработчик события ***OnCreate*** формы, в котором определим начальный угол поворота, радиус окружности вращения, «радиус» точки, шаг изменения угла поворота радиус-вектора, координаты центра области отображения формы (часть формы занимает панель с элементами управления), начальные координаты точки, соответствующие нулевому значению угла поворота, и цвет точки, а также нарисуем точку в ее начальном положении:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
  Angle := 0;
 RadCircle := 100;
 RadPoint := 5;
 StepPoint := 1;
 CenterX := Form1.Width div 2;
 CenterY := (Form1.Height - Panel1.Height) div 2;
 PointX := CenterX + RadCircle;
 PointY := CenterY;
 PointColor := clRed;
 Canvas.Pen.Color := PointColor;
 Canvas.Brush.Color := PointColor;
 Canvas.Ellipse(PointX - radPoint,PointY - RadPoint,
          PointX + RadPoint, PointY + RadPoint);
end;
```

 *Заметим, что указывать имя **Form1** при обращении к свойствам и методам формы в обработчиках необязательно. В данном случае оно указано исключительно для наглядности.*

При нажатии на кнопку ***ButtonMove*** необходимо сделать таймер активным, а название кнопки заменить на слово *Остановить*. При этом следующее нажатие должно сделать таймер снова не активным и вернуть первоначальное название кнопки. Другими словами, свойство ***Enabled*** компонента ***Timer1*** должно при каждом нажатии меняться на противоположенное. Кроме того, запретим изменять радиус окружности вращения во время движения точки. Создаем обработчик события ***OnClick*** кнопки ***ButtonMove*** и заносим в него следующий код:

```
if Timer1.Enabled then
 begin
  Button1.Caption := ' Запустить';
  LEditRad.Enabled := true;
 end
else
  begin
   RadCircle := StrToInt(LEditRad.Text);
   LEditRad.Enabled := false;
   Button1.Caption := ' Остановить';
 end;
Timer1.Enabled := not Timer1.Enabled;
```

Алгоритм движения точки по окружности включает в себя четыре шага:

- «стирание» точки в ее текущем положении;
- увеличение угла поворота радиус-вектора на заданный шаг;
- расчет новых координат точки после поворота радиус-вектора;
- рисование точки в новом месте.

Эти действия должны выполняться в обработчике события ***OnTimer*** компонента ***Timer1***. Для расчета новых координат точки воспользуемся полярной системой координат, в которой координаты задаются радиус-вектором ***R*** и углом ***φ***. Их преобразование в декартовы координаты производится по формулам:

$$X = R * \cos(\varphi)$$
$$Y = R * \sin(\varphi).$$

В рассматриваемом случае необходимо учесть три фактора: целочисленность координат, расчет угла поворота в градусах и смещение начала системы координат в центр области отображения формы. С учетом этого и введенных обозначений расчет координат точки будет производиться по формулам:

```
PointX := Trunc(RadCircle * cos(DegToRad(Angle)) + CenterX);
PointY := Trunc(RadCircle * sin(DegToRad(Angle)) + CenterY);.
```

Функция ***DegToRad(const Degs : Extended) : Extended*** *– это математическая функция, преобразовывающая значение градусов в радианы. Она находится в библиотеке* ***Math****.*

Создаем обработчик события ***OnTimer*** компонента ***Timer1***. Для стирания точки нарисуем ее цветом формы:

```
Canvas.Pen.Color := Form1.Color;
Canvas.Brush.Color := Form1.Color;
Canvas.Ellipse(PointX - RadPoint, PointY - RadPoint,
               PointX + RadPoint, PointY + RadPoint);
```

Далее увеличиваем значение угла поворота на значение шага и выполняем проверку на полный угол поворота:

```
Angle := Angle + StepPoint;
if Angle >= 360  then Angle := 0;
```

После этого рассчитываем новое положение точки, задаем цвет и рисуем точку в ее новом положении.

Запускаем программу и проверяем ее работоспособность при разных значениях радиуса окружности вращения.

Добавьте в программу код, запрещающий вводить в компонент ***LEditRad*** *любую нечисловую информацию. Организуйте проверку, позволяющую вводить только такие значения радиуса вращения, при которых точка не будет выходить за границы области отображения.*

Реализуем возможность изменения скорости вращения точки. Для этого создадим обработчик события ***OnChange*** компонента ***TrackBar1***. В нем необходимо по значению ***Position*** рассчитать новое значения интервала генерации прерываний для компонента ***Timer1*** (соответствующая формула приведена выше при описании задания для него начальных параметров):

Timer1.Interval := Trunc(1000 * StepPoint / TrackBarSpeed.Position);

Значение угловой скорости движения точки выведем в соответствующую метку:

LabelSpeed.Caption:= IntToStr(TrackBarSpeed.Position);

Запускаем программу и проверяем ее работоспособность при разных скоростях вращения.

Доработаем программу таким образом, чтобы при изменении размера формы точка по-прежнему вращалась вокруг середины компонента ***Image1***. Для этого необходимо пересчитать значения переменных ***CenterX*** и ***CenterY***. При изменении размеров формы происходит событие ***OnResize***. Создаем обработчик соответствующего события и заносим в него код пересчета координат центра вращения:

CenterX := Form1.Width div 2;

CenterY := (Form1.Height - Panel1.Height) div 2;.

Запускаем программу и убеждаемся в ее работоспособности.

Дополните программу возможностью регулирования «плавности» вращения точки, т.е. изменения значения переменной ***StepPoint****, по аналогии с регулированием скорости движения. Добавьте еще один компонент* ***TrackBar****, разрешив менять в нем значения от 1 (по умолчанию) до 20. При перемещении ползунка необходимо изменять значение переменной* ***StepPoint*** *и пересчитывать скорость вращения.*

Варианты заданий для самостоятельной работы

Задание: Выполнить моделирование заданного вида движения с возможностью остановки и повторного запуска, а также регулирования скорости движения.

Вариант	Задание
1	Точка, движущаяся по сторонам квадрата заданного размера
2	Точка, движущаяся случайным образом, не выходя за границы области рисования
3	Точка, движущаяся по одной из диагоналей области рисования, «отталкиваясь» от ее вершин
4	Точка, движущаяся по прямой от верхней границы области рисования до нижней границы и обратно
5	Точка, движущаяся по прямой от левой границы области рисования до правой границы и обратно
6	Отрезок заданной длины, вращающийся вокруг своей середины
7	Отрезок заданной длины, вращающийся вокруг одной из своих крайних точек
9	Отрезок заданной длины, вращающийся вокруг точки, делящей его в отношении 1:3
10	Отрезок заданной длины, вращающийся вокруг точки, делящей его в отношении 1:4
11	Окружность, «расширяющаяся» от заданного минимального радиуса до заданного максимального радиуса и обратно
12	Квадрат, «расширяющийся» от заданной минимальной длины стороны до заданной максимальной длины стороны и обратно
13	Два диаметра окружности заданного радиуса, расположенные взаимно перпендикулярно и вращающиеся вокруг ее центра
14	Два радиуса окружности заданного радиуса, расположенные взаимно перпендикулярно и вращающиеся вокруг ее центра
15	Прямоугольник с заданными сторонами, «расширяющийся» по горизонтали до границы области рисования и возвращающийся в исходное состояние
16	Прямоугольник с заданными сторонами, «расширяющийся» по вертикали до границы области рисования и возвращающийся в исходное состояние
17	Ромб с заданными диагоналями, «пульсирующий» по правилу: длины обоих диагоналей увеличиваются на фиксированное значение заданное количество раз, после чего он возвращается в исходное состояние
18	Метроном (отрезок, на верхнем конце которого окружность), «качающийся» на заданный угол относительно вертикальной оси
19	Маятник (отрезок, на нижнем конце которого окружность), «качающийся» на заданный угол относительно вертикальной оси
20	Закрашенный круг, двигающийся по границам области рисования

21	Шарик заданного радиуса, «стучащийся» в правый край области рисования. В начальный момент он находится на заданном расстоянии от края, затем движется до него, ударяется и возвращается в начальное положение
22	Шарик заданного радиуса, «стучащийся» в нижний край области рисования. В начальный момент он находится на заданном расстоянии от края, затем движется до него, ударяется и возвращается в начальное положение
23	Два шарика заданного радиуса находятся у противоположенных краев области рисования на одном уровне по вертикали. Они двигаются навстречу друг другу, сталкиваются в середине области рисования, отталкиваются и двигаются обратно
24	Два шарика заданного радиуса находятся у противоположенных краев области рисования на одном уровне по горизонтали. Они двигаются навстречу друг другу, сталкиваются в середине области рисования, отталкиваются и двигаются обратно
25	Две точки двигаются по окружности заданного радиуса навстречу друг другу с одинаковой скоростью. При встрече они свободно проходят друг через друга
26	Квадрат, «пульсирующий» случайным образом в диапазоне от минимальной до максимальной длины стороны
27	Окружность, «пульсирующая» случайным образом в диапазоне от минимального до максимального значения радиуса
28	Прямоугольник, «пульсирующий» случайным образом в диапазоне от минимальной до максимальной длины диагонали
29	Точка, движущаяся по левой полуокружности заданного радиуса, отталкиваясь от ее крайних точек
30	Точка, движущаяся по верхней полуокружности заданного радиуса, отталкиваясь от ее крайних точек

Часть 2

Задание: Разработать игровую программу «Попади в мишень», реализующую следующие правила игры:

- На форме в случайном месте через определенный промежуток времени появляется объект (мишень) – квадрат случайного размера, а рядом с ним написано некоторое слово. Имеются два списка слов – «черный» и «белый», в каждом из которых находятся слова по определенной теме, например, «черный» список – это слова по теме «математика», а «белый» – по теме «фрукты». Слова из этих списков случайным образом выводятся рядом с мишенью.

- Мишень видна некоторое время, или до того момента, как по ней мышью кликнет пользователь, затем исчезает и появляется в другом месте экрана с новым словом.
- Будем считать, что время показа мишени и время между последовательными ее появлениями совпадают.

Задача пользователя: за сеанс игры «попасть» (кликнуть мышью) в максимальное количество мишеней, рядом с которыми выведены слова из «белого» списка. Попадание в мишень со словом из «черного» списка считается промахом. Количество попаданий, количество промахов и оставшееся до конца игры время отображаются в верхней части панели управления, расположенной в правой части формы. Помимо этого, на ней располагаются компоненты для выполнения следующих действий перед началом каждого сеанса:

- Выбор уровня от 1 до 6. Каждый следующий уровень в два раза уменьшает время, в течение которого видна мишень.
- Выбор длительности сеанса игры в минутах с набором стандартных сеансов от 1 минуты до 10 минут.
- Кнопка начала и принудительной остановки игры.

При любом окончании игры выдается время игры, количество попаданий и промахов, а также показатель эффективности, который вычисляется по формуле:

$$K_{\text{эф}} = \frac{K^+ - K^-}{T}$$

где $K_{эф}$ – показатель эффективности;

K^+ – количество попаданий;

K^- – количество промахов;

T – время игры.

Подготовка к выполнению работы

Создаем новый проект и изменяем у формы свойство ***Caption*** на название лабораторной работы. Сохраняем проект в рабочую папку командой ***File → Save Project as***.

Выбираем тему игры и создаем текстовый файл *Белый_список.txt*, в который заносим название выбранной темы (первой строкой) и 10 слов, относящихся к ней. В файл *Черный_список.txt* заносим 10 любых слов, не относящихся к выбранной теме игры. Оба файла сохраняем в рабочем каталоге.

Формирование интерфейса программы

Для разделения окна программы на область игры и панель управления используем компонент ***Panel1***, который «прижимаем» к правой границе формы (***Align***=*alRight*). С целью отображения текущих параметров игры размещаем на панели управления четыре компонента ***Label***:

- ***LabelSec1*** (***Caption*** = *'Оставшееся время'*, ***Visible*** = *false*);
- ***LabelSec*** – обратный секундомер, который показывает время до окончания игры в минутах и секундах (***Caption*** = *'0.0'*, ***Font.Color***=*clBlue*, ***Font.Size***=*30*, ***Visible*** = *false*);
- ***LabelYesNo*** (***Caption*** = *'Попадания Промахи'*, ***Visible*** = *false*);
- ***LabelYes*** – количество попаданий (***Caption*** = *'0'*, ***Font.Color***=*clBlue*, ***Font.Size***=*30*, ***Visible*** = *false*);
- ***LabelNo*** – количество промахов (***Caption*** = *'0'*, ***Font.Color***=*clBlack*, ***Font.Size***=*30*, ***Visible*** = *false*).

Установку начальных параметров игры будем осуществлять компонентами:

- ***LabelTime*** (***Caption*** = *'Задайте время (мин)'*).
- ***ComboBoxTime*** – задание времени игры в минутах одним из двух способов: выбором значения из выпадающего списка, содержащего числа от 1 до 10, и непосредственным вводом времени в минутах. Для этого вызовем редактор списка ***Items*** и введем соответствующие значения (рис. 12.1). По умолчанию выберем время игры равное одной минуте (***ItemIndex***=0).
- ***LabelLevel*** (***Caption*** = *'Выберите уровень'*).
- ***ComboBoxTime*** – задание уровня игры. В отличие от задания времени игры уровень можно только выбирать из выпадающего списка (***Style*** = *csDropDownList*). Аналогично предыдущему случаю в редакторе списка ***Items*** введем значения от 1 до 6 и выберем по умолчанию первый уровень.
- ***ButtonPlay*** (***Caption*** = *'Играть*).

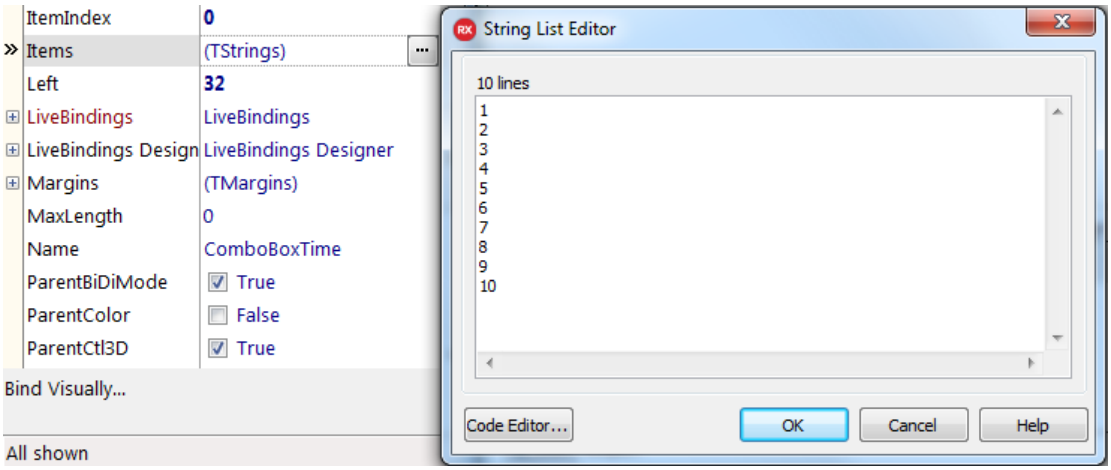

Рис. 12.1. Редактор списка компонента ComboBox

Интерфейс программы показан на рис. 12.2.

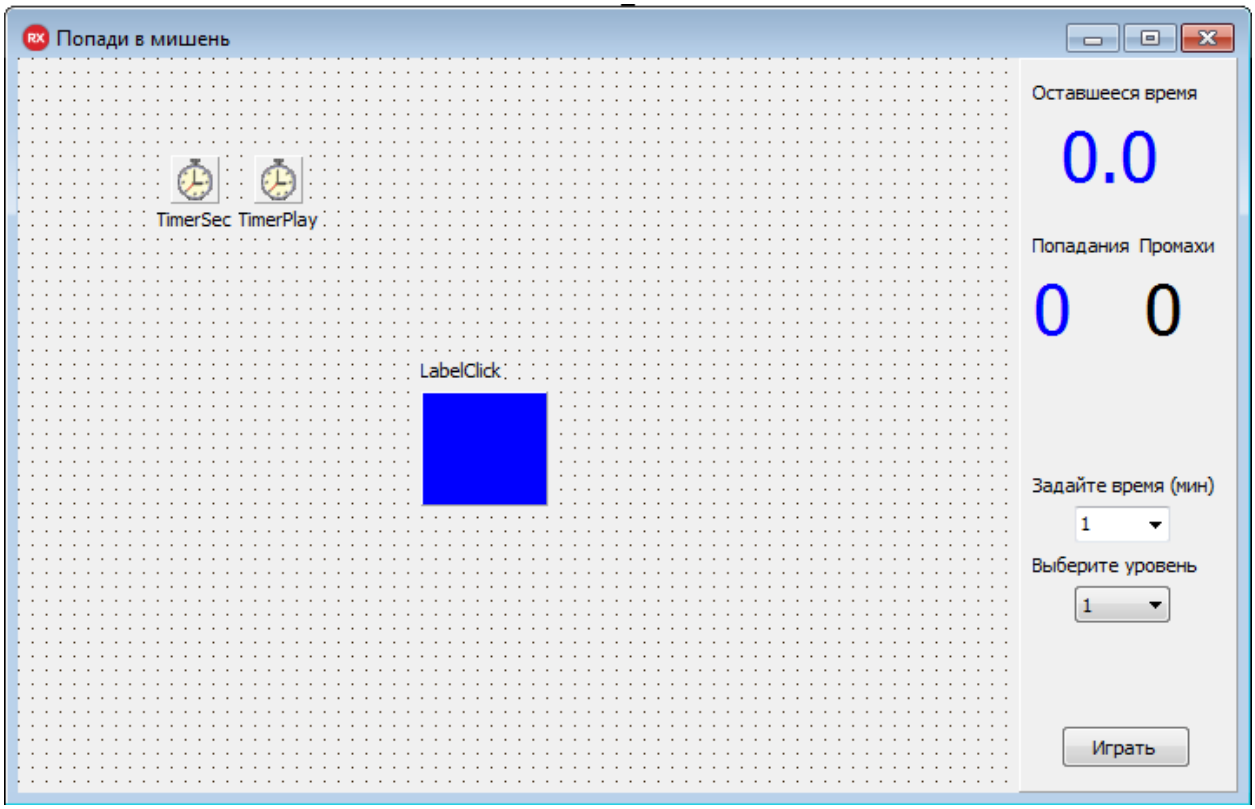

Рис. 12.2. Интерфейс программы

Разработка программы. Предварительные действия

Определим несколько констант после слова ***implementation***:

- ***MaxTime*** *= 2.0* – время показа мишени (в секундах) для первого уровня игры;
- ***MinDim*** *= 20* – минимальный размер мишени;
- ***MaxDim*** *= 100* – максимальный размер мишени.

Здесь же опишем четыре переменные целого типа и две переменные типа список строк (***TStringList)***:

- ***PlayTime*** – время в секундах, оставшееся до конца игры;
- ***NumAll*** – количество выпавших мишеней со словами из «белого» списка;
- ***NumGetWhite*** – количество попаданий;
- ***NumGetBlack*** – количество промахов;
- ***ListWhite*** – «белый» список слов;
- ***ListBlack*** – «черный» список слов.

Класс ***TStringList*** представляет собой список (массив) строк, доступ к каждой из которых осуществляется по номеру, начиная с нуля. Количество элементов списка содержится в свойстве ***Count***, поэтому строки нумеруются от *0* до *Count – 1*.

Перед началом игры необходимо заполнить «белый» и «черный» списки словами из соответствующих файлов. Это можно сделать в обработчике события создания формы ***FormCreate***. Создадим его и определим в нем переменную строкового типа ***Dir***, в которую занесем полный путь к файлам.

Первой операцией при работе с классами является их создание (выделение необходимой памяти). Это производится конструктором ***Create***:

ListWhite := TStringList.Create;
ListBlack := TStringList.Create;

Для определения каталога, в котором находятся файлы со словами, воспользуемся функцией ***SelectDirectory***, диалоговое окно которой показано на рис. 12.3:

function SelectDirectory (const Caption:string; const StartDir:string;
out Dir: string) : Boolean,

где ***Caption*** – комментарий к выбору;
StartDir – каталог, устанавливаемый по умолчанию;
Dir – выбранный каталог.

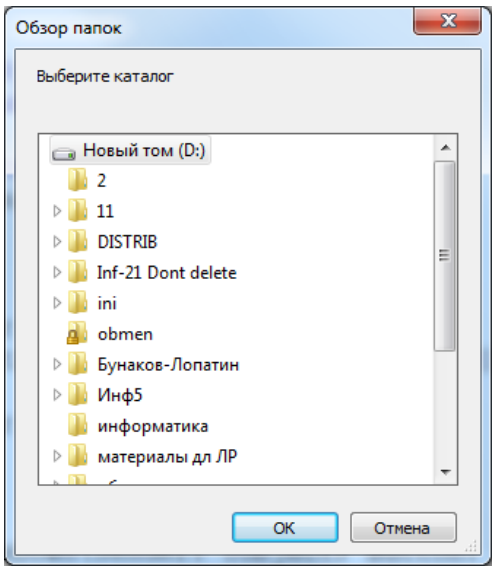

Рис. 12.3. Окно выбора каталога

Данная функция выводит диалоговое окно, в котором пользователь может выбрать нужный каталог. В качестве результата она возвращает значение ***true***, если пользователь выбрал каталог и нажал кнопку ***OK***, и значение ***false*** в случае нажатия кнопки ***Отмена*** или закрытия диалогового окна без выбора каталога. Другими словами, структура программы обработчика будет следующей:

if SelectDirectory('Выберите каталог', 'D:\', Dir) then
<считывание слов из файлов>.

Отметим одну особенность работы функции ***SelectDirectory***. При выборе диска она возвращает его название с обратной наклонной чертой в конце, например, *'D:\'*. Если же выбирается какой-либо каталог, то наклонная черта в конце строки отсутствует, например, *'D:\Игра'*. Для открытия файла необходимо добавить его имя (*Белый_список.txt* или *Черный_список.txt*) к выбранному каталогу ***Dir***, т.е. наклонная черта всегда должна быть последним символом в имени каталога. Этого можно добиться с помощью следующего условного оператора:

```
if Copy(Dir, Length(Dir), 1) <> '\' then
  Dir := Dir + '\'.
```

 Функция ***Copy(Source : string; StartChar, Count : Integer) : string*** возвращает строку, которая содержит символы строки ***Source***, начиная с символа под номером ***StartChar*** в количестве ***Count***. Если указанное количество символов меньше длины строки, то возвращаются все символы, до конца строки. Нумерация символов начинается с единицы.

Далее проверим наличие нужных файлов в указанном каталоге и в случае отсутствия хотя бы одного из них сделаем недоступной кнопку начала игры:

```
if Not FileExists(Dir + 'Белый_список.txt') or
   Not FileExists(Dir + 'Черный_список.txt') then
  begin
    ShowMessage('Не найдены списки');
    ButtonPlay.Enabled:= false;
  end
else
  <чтение данных из файлов>.
```

 Функция ***FileExists (const FileName : string) : boolean*** проверяет существование файла с именем ***FileName*** и возвращает значение *true*, если такой файл существует.

Для чтения данных из файлов и заполнения списков воспользуемся методом ***LoadFromFile*** класса ***TStringList***, который заносит в список последовательно все строки из файла с указанным именем:

```
ListWhite.LoadFromFile(Dir + 'Белый_список.txt');
ListBlack.LoadFromFile(Dir + 'Черный_список.txt');
```

В «белом» списке первой строкой записано название темы игры, которое надо вывести в заголовке формы, а затем удалить из списка:

```
Form1.Caption := Form1.Caption + 'Тема: ' + ListWhite[0];
ListWhite.Delete(0);
```

Начало и принудительная остановка игры

Для начала игры необходимо уровень игры и время сеанса, а также «запустить» таймеры. Поскольку для остановки используется та же кнопка, будем определять текущее состояние игры по состоянию таймера ***TimerPlay***: если он включен (***Enabled*** = *true*), то пользователь хочет остановить игу, а если выключен(***Enabled*** = *false*), – то начать ее. Таким образом, структура программы обработки нажатия кнопки ***ButtonPlay*** будет следующей (переменные ***n*** и ***t*** – локальные рабочие переменные):

```
procedure TForm1.ButtonPlayClick(Sender: TObject);
var n: integer;
    t: real;
begin
 if TimerPlay.Enabled then
   <остановка игры>
 else
   <начало игры>
end.
```

Игра может быть остановлена двумя способами: принудительно (при нажатии на кнопку ***ButtonPlay***) и по истечении времени сеанса. В обоих случаях необходимо выполнить одинаковые действия, поэтому напишем отдельную процедуру остановки игры ***StopGame***, которую в данном обработчике вызовем, если таймер ***TimerPlay*** включен. В этой процедуре необходимо обращаться к компонентам, расположенным на форме, поэтому опишем ее в качестве метода класса ***TForm1***, т.е. в раздел ***private*** (процедура будет вызываться только методами данного класса) данного класса добавим строчку:

```
procedure StopGame;
```

Для остановки игры необходимо выполнить пять блоков действий:

1. Остановить оба таймера:
```
TimerSec.Enabled := false;
TimerPlay.Enabled := false;
```
2. Сделать доступными компоненты выбора уровня игры и времени сеанса (после начала игры изменять ее условия запрещается), а также изменить название кнопки:
```
LabelLevel.Enabled := true;
ComboBoxLevel.Enabled := true;
LabelTime.Enabled := true;
ComboBoxTime.Enabled := true;
ButtonPlay.Caption := 'Играть';
```
3. Скрыть мишень:
```
PanelClick.Visible := false;
LabelClick.Visible := false;
```
4. Рассчитать показатель эффективности, т.е. вычесть из количества попаданий количество промахов и разделить это значение на время игры:

```
sp := (NumGetWhite - NumGetBlack) / StrToFloat(ComboBoxTime.Text);
if sp < 0 then sp:= 0.0;
```

5. Вывести результаты игры:

```
ShowMessage('Ваш результат: ' + chr(10) +
   'За ' + ComboBoxTime.Text + '  минут у вас ' + chr(10) +
   IntToStr(NumGetWhite) + ' попаданий и ' +
   IntToStr(NumGetBlack) + ' промахов из ' +
   IntToStr(NumAll) + ' возможных.' + chr(10) +
  'Ваш показатель - ' + FloatToStr(sp) + ' попаданий в минуту.');
```

Указанный параметр процедуры ***ShowMessage*** обеспечит вывод результатов игры в виде, показанном на рис. 12.4.

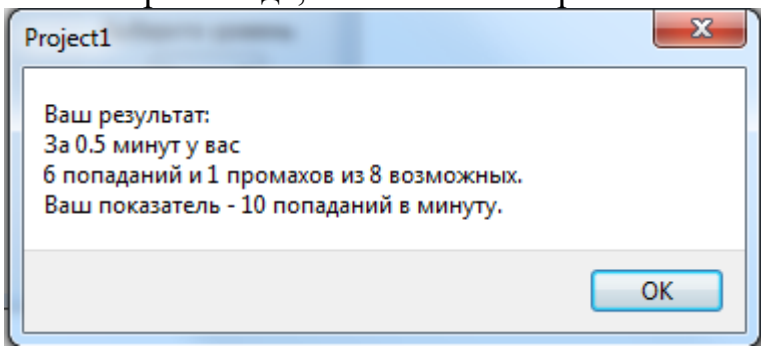

Рис. 12.4. Сообщение о результатах игры

Функция ***chr*** преобразует указанное целое число в символ в соответствии с таблицей ASCII. Обращение ***chr (10)*** возвращает символ перевода строки.

На этом процедура остановки игры заканчивается. Переходим к определению параметров игры и ее началу.

Первым делом заносим в переменные ***t*** и ***n*** время игры в минутах и ее уровень:

```
t := StrToFloat(ComboBoxTime.Text);
n := StrToInt(ComboBoxLevel.Text);
```

Организуйте запрет ввода в компонент ***ComboBoxTime*** *нечисловых данных. Учтите при этом, что время игры задается в минутах и может быть вещественным числом.*

Переводим время игры в секунды:

```
PlayTime:= Round(t * 60.0).
```

Вводим запрет на изменение уровня и времени игры во время сеанса:

```
LabelLevel.Enabled := false;
```

ComboBoxLevel.Enabled := false;
LabelTime.Enabled := false;
ComboBoxTime.Enabled := false;

Обнуляем количество выпасших мишеней со словами из «белого» списка, количество попаданий и количество промахов, а также отображаем эти значения в соответствующих метках, делая их видимыми:

NumAll:= 0;
NumGetWhite:= 0;
NumGetBlack:= 0;
LabelYes.Visible:= true;
LabelNo.Visible:= true;
LabelYesNo.Visible:= true;
LabelYes.Caption:= IntToStr(NumGetWhite);
LabelNo .Caption:= IntToStr(NumGetBlack);
LabelSec.Caption:= IntToStr(trunc(t))+ '.' + IntToStr(Round(frac(t)*60));
LabelSec.Visible:= true;
LabelSec1.Visible:= true;

По правилам игры на первом уровне интервал между последовательными появлениями мишеней и время их видимости равен заданному максимальному значению. Переход на каждый следующий уровень уменьшает его в два раза, что можно выразить формулой:

$$t_n = \frac{t_{\max}}{2^{n-1}},$$

где t_n – время для n-го уровня;

t_{max} – максимальное время (время для первого уровня);

n – уровень игры.

Запишем эту формулу на языке программирования:

TimerPlay.Interval:= Round(MaxTime / power(2, n-1) * 1000);

Изменяем название кнопки и «запускаем» таймеры:

ButtonPlay.Caption:= 'Остановить';
TimerSec.Enabled:= true;
TimerPlay.Enabled:= true;

Инициализируем генератор случайных чисел:

Randomize;

Процедура ***Randomize*** инициализирует («запускает») генератор случайных чисел. Если не обращаться к ней, то при каждом запуске программы будет выдаваться одна и та же последовательность чисел. По этой причине процедуру ***Randomize*** желательно вызвать хотя бы один раз при загрузке формы.

Отображение мишени

В начале игры (после нажатия кнопки «Играть») мишень не видна. По прошествии заданного интервала времени (***TimerPlay.Interval***) она «появляется» на экране и, если пользователь не щелкнул на ней мышью, остается видимой в течение времени, заданного тем же значением интервала. После этого все повторяется.

Создадим обработчик события ***OnTimer*** компонента ***TimerPlay*** и опишем в нем переменную ***n*** целого типа. Работу соответствующей программы можно описать следующим образом:

1. Если мишень невидима, то рассчитать для нее случайные координаты и размер, взять случайным образом слово из какого-либо списка и сделать ее видимой.
2. Если мишень видима, то просто сделать ее невидимой.

Таким образом, структура программы будет следующей:

```
if not PanelClick.Visible then
  begin
    <расчет параметров и выбор слова>
  end;
PanelClick.Visible := not PanelClick.Visible;
LabelClick.Visible := PanelClick.Visible;
```

Поскольку мишень квадратная, определяем случайное значение высоты в заданном диапазоне и присваиваем его параметру ширины мишени:

```
PanelClick.Height := RandomRange(MinDim, MaxDim);
PanelClick.Width := PanelClick.Height.
```

Слово всегда выводится сверху от мишени, поэтому вертикальная координата будет изменяться от значения, равного высоте надписи в метке ***LabelClick.Height*** с небольшим зазором (например, 10 пикселей) до значения высоты области отображения за вычетом высоты мишени. Горизонтальная координата изменяется от нуля до значения ширины области отображения за вычетом ширины мишени и ширины панели управления:

```
PanelClick.Top := RandomRange(LabelClick.Height + 10,
                  Form1.ClientHeight - PanelClick.Height);
PanelClick.Left := RandomRange(0, Form1.ClientWidth - Panel1.Width –
                   PanelClick.Width);
```

Случайный характер выбора слов из «белого» и «черного» списков реализуем следующим образом. Генерируем случайное число в диапазоне от 1 до 11, и если оно меньше или равно пяти, то выбираем «белый» список, в противном случае – «черный» список. При этом считаем количество вариантов появления слов из «белого» списка.

```
if RandomRange(1, 11) <= 5 then
  begin
    n := RandomRange(0, ListWhite.Count);
    LabelClick.Caption:= ListWhite[n];
    inc(NumAll);
  end
```

```
    else
      begin
       n := RandomRange(0, ListBlack.Count);
       LabelClick.Caption:= ListBlack[n];
      end;

     LabelClick.Left:= PanelClick.Left;
    LabelClick.Top:= PanelClick.Top - LabelClick.Height - 3;
```

Подсчет попаданий и промахов

Для подсчета количества попаданий и промахов создадим обработчик события OnMouseDown компонента PanelClick (мишени) и опишем в нем переменную ***i***:

```
procedure TForm1.PanelClickMouseDown(Sender: TObject; Button:
            TMouseButton;   Shift: TShiftState; X, Y: Integer);
var i: integer;
begin
end;
```

При попадании мишень становится невидимой:

```
  PanelClick.Visible:= false;
  LabelClick.Visible:= false;
```

Теперь осталось определить, является ли данный щелчок промахом, или это попадание, увеличить значение соответствующего счетчика и отобразить его:

```
  for i := 0 to ListBlack.Count - 1 do
    if LabelClick.Caption = ListBlack[i] then
      begin
       inc(NumGetBlack);
       LabelNo.Caption := IntToStr(NumGetBlack);
       exit;
      end;

  for i := 0 to ListWhite.Count - 1 do
    if LabelClick.Caption = ListWhite[i] then
      begin
       inc(NumGetWhite);
       LabelYes.Caption := IntToStr(NumGetWhite);
      end;
```

Отсчет времени и остановка игры

Время, оставшееся до конца игры в секундах, хранится в переменной ***PlayTime***. Интервал, заданный для таймера ***TimerSec***, составляет одну секунда. Следовательно, при каждом прерывании от данного таймера необходимо уменьшить на единицу значение ***PlayTime***, вывести оставшееся до конца игры время, переведя его в минуты и секунды, и остановить игру при достижении нулевого значения:

```
procedure TForm1.TimerSecTimer(Sender: TObject);
begin
  Dec(PlayTime);
 LabelSec.Caption := IntToStr(PlayTime div 60)+ '.' +
                     IntToStr(PlayTime mod 60);
 if PlayTime = 0 then  StopGame;
end;
```

Закрытие программы

При закрытии программы надо освободить динамическую память, которая отводилась для работы. Это выполняется при помощи метода ***Free***. Создаем обработчик события ***OnClose*** и заносим в него два строчки:

```
ListWhite.Free;
ListBlack.Free;
```

Запускаем игру и проверяем ее работу на разных уровнях.

 Функция ***MessageDlg*** позволяет вывести на экран некоторое сообщение и несколько кнопок (*Yes*, *No, OK* и т. д.) для запроса дальнейших действий:

MessageDlg(const Msg: String; Type: TMsgDlgType; Butt: TMsgButtons; HelpCtx: Longint): Word,

где ***Msg*** – строка, которая будет показываться в сообщении.

Type – вид иконки в окне сообщения и строки в заголовке. Наиболее часто используемые значения данного параметра: ***mtConfirmation*** (Подтверждение), *mtInformation* (Сообщение), *mtWarning* (Предупреждение) и *mtError* (Ошибка).

Butt – набор кнопок, выводимых в окне сообщения. Некоторые возможные значения: *mbYes* (кнопка *Yes*), *mbNo* (кнопка *No*), *mbCancel* (кнопка *Cancel*), *mbOK* (кнопка *OK*). Если кнопок несколько, то они заключаются в квадратные скобки. Например, для вывода двух кнопок *Yes* и *No* надо написать [*mbYes*, *mbNo*]).

HelpCtx – это идентификатор контекстной справки. Его можно задать нулевым.

Результатом выполнения функции *MessageDlg* является значение, содержащее информацию о том, какую кнопку нажал пользователь.

Возможные значения отличаются от названия кнопок только префиксом: *mrYes, mrNO, mrCancel, mrOK*.

Пример использования функции:

if MessageDlg('Завершить', mtConfirmation, [mbYes, mbNo], 0) = mrYes then exit;

Варианты заданий для самостоятельной работы

Задание: изменить правила игры или дополнить ее новыми возможностями.

Вариант	Задание 1	Задание 2
1	Считать штрафом не только щелчок по мишени со словом из «черного» списка, но и щелчок по форме	Создать два «белых» списка по разным темам и добавить компонент ***ComboBox***, в который занести их темы. Перед началом игры пользователь может выбрать любую из тем
2	Менять цвет мишени случайным образом, выбирая его из пяти возможных цветов	По аналогии с кнопкой *Играть/Остановить* добавить кнопку *Пауза/Продолжить*
3	Менять цвет выводимого слова случайным образом, выбирая его из пяти возможных цветов	После вывода результатов игры автоматически записать их в текстовый файл с именем *Результат.txt*
4	Добавить возможность расширения множества слов в «черном» и «белом» списках перед началом игры с помощью одного компонента ***Edit*** и двух компонентов ***Button*** (*В черный список*, *В белый список*)	За щелчок по панели управления начислять два штрафных балла
5	После вывода результатов игры автоматически записать показатель эффективности в файл *Рекорд.txt*, если такого файла нет, или значение в нем меньше достигнутого показателя. Перед началом игры, если такой файл существует, считать значение	Слова из «белого» списка выводить шрифтом размером 10 пунктов, а слова из «черного» списка – размеров 8 пунктов

	из него и вывести на панель управления со словом *Рекорд*.	
6	При нажатии на кнопку *Играть* выводить запрос *Вы уверены, что хотите играть на уровне **N**?*, где значение ***N*** соответствует выбранному уровню. Дальнейшие действия зависят от ответа	Сделать так, чтобы на каждом уровне игры мишень имела свой цвет
7	Слова из «белого» списка выводить синим цветом, а слова из «черного» списка – красным	Добавить возможность расширения множества слов в «черном» и «белом» списках перед началом игры с помощью одного компонента ***Memo*** и двух компонентов ***Button*** (*В черный список*, *В белый список*)
8	В качестве мишени использовать компонент ***GroupBox***, а слова писать в его заголовке	Добавить метку вывода бонусных баллов, которые начисляются за каждые три попадания подряд. При расчете показателя эффективности бонусные баллы вычитать из штрафных баллов. Если количество бонусных баллов превышает количество штрафных баллов, то последнее значение обнулить.
9	Менять размер выводимого слова случайным образом в интервале от 8 до 11 пунктов	Реализовать «моральную поддержку» игрока: после пятого попадания на панели управления написать слово *Молодец*, после десятого – *Отлично*, а после пятнадцатого – *Вы вне конкуренции*.
10	Оставить только счетчик попаданий, а промахи вычитать из количества попаданий.	Добавить возможность расширения множества слов в «черном» и «белом» списках перед началом игры с помощью одного двух компонентов ***Memo*** (*Черный список*, *Белый список*) и одного компонента ***Button***
11	Сменить цвет секундомера после того, как половина времени игры прошло	После вывода результатов игры и нажатия кнопки ***OK*** выводить запрос *Закончить игру?*. При положительном ответе игра

		заканчивается, а при отрицательном продолжается.
12	Если задано время игры более трех минут, то при нажатии на кнопку *Играть* выводить запрос *Вы сможете играть так долго?*. Дальнейшие действия зависят от ответа	Считать штрафом не только щелчок по мишени со словом из «черного» списка, но и щелчок по панели управления
13	При первом попадании выводить на панель управление сообщение *Попал*. При следующих попаданиях, если они идут подряд, – сообщение *Опять попал*. При промахе это сообщение убирать	На панель управления добавить компонент, дублирующий выводимое слово
14	Создать два «белых» списка по разным темам и добавить компонент ***ComboBox***, в который занести их темы. Перед началом игры пользователь может выбрать любую из тем	Сделать так, чтобы секундомер показывал не оставшееся до конца игры время, а время игры.
15	Реализовать систему бонусных баллов: если игрок три раза подряд попал по мишени, списывать с него один штрафной балл. Если количество бонусных баллов превышает количество штрафных баллов, то последнее значение обнулить.	Сделать так, чтобы на каждом уровне игры секундомер имел свой цвет
16	Сделать выбор времени игры в интервале от одной до 10 минут с помощью ползунка ***TrackBar***	Если прошло половина времени игры, то остановить игру, выдать предупреждение об этом, после чего продолжить игру
17	При попадании цвет мишени становится красным, и она не исчезает с экрана до окончания времени видимости	Добавить возможность расширения множества слов в «черном» и «белом» списках перед началом игры с помощью одного двух компонентов ***Edit*** (*Черный список*, *Белый список*) и одного компонента ***Button***
18	Реализовать информирование	За 10 секунд до окончания игры

	игрока о штрафах: после третьего промаха на панели управления написать слово *Внимательнее*, после пятого – *Соберитесь*, а после седьмого – *Вы очень рассеяны*.	сделать секундомер мигающим
19	Добавить кнопки *Редактировать черный список* и *Редактировать белый список*, при нажатии на которые в компонент ***Memo*** выводится соответствующий список и появляется кнопка *Сохранить*. Она позволяет изменить редактируемый список	Выводить слова не выше, а ниже мишени
20	Если до конца игры осталось менее 30 секунд, остановить игру, выдать предупреждение об этом, после чего продолжить игру	Каждое пятое попадание отмечать сообщением на панели управления
21	Выводить на панель управление сообщение *Попадание* или *Промах* после каждого щелчка	За 30 секунд до окончания игры изменить цвет секундомера на зеленый, а за 10 секунд – на красный
22	В начале игры указывать не каталог, в котором находятся файлы с «белым» и «черным» списками, а имена файлов	В качестве мишени использовать компонент ***Image***, а слова писать внутри него
23	Выводить на панель управления через каждые 30 секунд сообщение о том, что очередной интервал времени прошел	Добавить кнопку *Редактировать списки*, при нажатии на которую в компоненты ***Memo*** выводятся оба списка и появляется кнопка *Сохранить*. Она позволяет изменить содержание списков
24	При первом промахе выводить на панель управление сообщение *Промах*. При следующих промахах, если они идут подряд, – сообщение *Опять промах*. При попадании это сообщение убирать	Сделать выбор уровня игры с помощью ползунка ***TrackBar***

25	Создать два «белых» списка по разным темам и добавить две кнопки, названия которых соответствуют их темам. Перед началом игры пользователь нажатием кнопки выбирает тему игры	Сделать выбор уровня игры с помощью нажатия одной из пяти кнопок
26	Заканчивать игру, если количество штрафных баллов превышает количество попаданий на три с соответствующим сообщением	Создать два «белых» списка по разным темам и добавить компонент ***RadioGroup***, кнопкам которого присвоить названия их тем. Перед началом игры пользователь может выбрать любую из тем
27	Создать два «белых» списка по разным темам и добавить два компонента ***CheckBox***, заголовкам которых присвоить названия тем. Перед началом игры пользователь может выбрать любую из тем. Если он выбрал обе темы, то создать объединенный «белый» список	За щелчок по форме начислять два штрафных балла
28	Между метками, отображающими количество попаданий и промахов вставить метку, со значением «:» (двоеточие), которая мигает с некоторой частотой	Каждое пятый промах отмечать сообщением на панели управления
29	Создать два «белых» списка по разным темам и добавить два компонента ***CheckBox***, заголовкам которых присвоить названия тем. Перед началом игры пользователь может выбрать любую из тем. Запретить одновременный выбор обоих тем	За 30 секунд до окончания игры секундомер переходит в мигающий режим: его цвет меняется с синего на красный и наоборот каждую секунду
30	Заканчивать игру, если количество штрафных баллов превышает пять с соответствующим	Между метками, отображающими количество попаданий и промахов вставить метку, принимающую одно из

	сообщением	трех значений: «=» (равно), «<» (меньше) или «>» (больше). Текущее значение метки выбирается, исходя из соотношения количества попаданий и промахов

Варианты заданий повышенной сложности

1. Организуйте систему регистрации пользователей и хранения личных рекордов для каждого уровня игры. Перед началом работы вводится логин. Если данный пользователь раньше уже играл, то находится и отображается его личный рекорд для выбранного уровня, а если не играл, то он его логин автоматически запоминается. Помимо этого находится и отображается абсолютный рекорд на данном уровне среди всех пользователей. Все достижения в процессе игры оперативно отображаются на панели управления. Количество пользователей не ограничено.
2. Замените «белый» и «черный» списки слов соответствующими каталогами рисунков и отображайте их на мишени вместо слов. Добавьте «бонусные» мишени со значками одного и двух сердечек, которые будут изредка и ненадолго появляться в случайном месте игрового поля. Щелчок на одно сердечко увеличивает количество попаданий на два, а щелчок на два сердечка обнуляет количество промахов.
3. Организуйте медленное хаотичное движение по игровому полю мерцающей звездочки. Если мишень закрывает ее, то попадание или промах по данной мишени удваиваются. При этом мишень обесцвечивается (становится прозрачной) в течение 3 секунд, затем исчезает с игрового поля.
4. Добавьте ещё один «белый» список из другой предметной области и вторую мишень, которая будет появляться на 30% реже, но время ее видимости соответствует уровню игры. Распределение слов из «белых» и «черного» списков по мишеням случайное. При попадании по дополнительной мишени она делится на две части (если размеры этих частей больше минимально возможного размера), которые разлетаются в случайном направлении на случайные расстояния, не выходя при этом за границы игрового поля. Слова у разлетевшихся мишеней меняются случайным образом, и они остаются видимыми то время, которое соответствует уровню игры. При попадании на разлетевшиеся мишени они будут снова разделяться до тех пор, пока их размер будет больше минимального. Основная мишень при этом не меняет характер своего поведения. Игрок должен попадать по всем мишеням, содержащим слова из «белых» списков.

5. Добавьте возможность одновременного появления до пяти мишеней. Условие их появления следующее. При щелчке по пустой области программа переходит в режим одновременной выдачи от 2 до 5 мишеней, слова по которым распределяются случайным образом. Одновременно появляется мишень, позволяющая вернуться в нормальный режим игры при щелчке по ней. Она появляется реже, чем основные мишени в 3-7 раз. Конкретное время ее появления случайно, а время видимости в 2 раза меньше, чем у основных мишеней. С переходом в режим выдачи нескольких мишеней происходит «смена суток» на игровом поле, т.е. меняется вся цветовая гамма.

1. Назначения, свойства и методы компонента ***Timer***.
2. Опишите алгоритм анимации объектов.
3. Свойство ***Align***, его назначение и возможности.
4. Класс ***TStringList***, его основные свойства и методы.
5. Как в программе можно выбрать рабочий каталог.
6. Функция ***MessageDlg***, ее назначение, тип и формальные параметры.