Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему:

ИГРОВОЕ ПРОГРАММНОЕ СРЕДСТВО «SOKOBAN»

Студент: гр. 751001
Красковский Д.А

Руководитель: Данилова Г.В.

Минск 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компьютерные игры становятся все более красочными и реалистичными. Количество любителей расслабиться с их помощью постоянно растет. Однако это не означает, что классические, любимые всеми логические игры и головоломки, перестали пользоваться спросом. Ярким примером такой игры, известной почти каждому, является Sokoban.

Sokoban – логическая игра-головоломка, в которой игрок передвигает ящики по лабиринту, показанному в виде плана, с целью поставить все ящики на заданные конечные позиции. Только один ящик может быть передвинут за раз, причём герой игры – «кладовщик» – может только толкать ящики, но не тянуть их.

Самый первый Sokoban появился в 1981 году, вместе с культовым персональным компьютером IBM PC, но и по сей день над этой, простой с виду, игрой ломают головы миллионы людей по всему миру. Игра заинтересовала не только геймеров, но и математиков. Было доказано, что задача решения уровня Sokoban NP – трудная. Сложность решения уровней Sokoban вызвана как сильным ветвлением дерева решений (сопоставимым с шахматами), так и большой его глубиной – для решения некоторых уровней требуется больше 1000 толканий ящиков.

На сегодняшний день большинство реализаций Sokoban существуют в качестве приложений в сети Интернет, хороших офлайн-приложений на ПК очень мало. Главный недостаток онлайн-приложений: сложность создания своих уровней и обмен уровнями с другими пользователями. Наиболее удобно хранить уровни в обычном текстовом файле, а наиболее эффективно работу с файлами можно организовать в офлайн-приложении на ПК. Кроме того, онлайн-приложения требуют постоянного доступа к сети Интернет, который не всегда присутствует, а даже если и присутствует, то пользователь скорее всего будет смотреть видео на YouTube, а не разгадывать головоломки.

Таким образом, Sokoban является по-настоящему культовой игрой, развивающей логическое мышление.

Целью данного курсового проекта является создание полноценной игровой среды Sokoban, предназначенной для организации досуга пользователя.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

В настоящее время существует огромное количество реализаций игры Sokoban, представленных как в виде онлайн-приложений, так и офлайн. В этом разделе будут рассмотрены наиболее популярные из них.

1.1.1 «Sokoban»

«Sokoban» – самая первая версия игры. Была создана в 1981 году Хироюки Имабаяси, и издана в 1982 году японской компанией Thinking Rabbit. Была доступна на платформах NEC PC-8801, Commodore 64, IBM PC, Apple II.

Учитывая возраст игры, неудивительно, что оригинальный Sokoban недоступен на данный момент на современных операционных системах Windows и Linux. Для того чтобы поиграть в данную игру, необходимо или приобрести старый компьютер, или установить эмулятор, например, DOSBox. Но и в данном случае если уж не качество текстур, то разрешение экрана не смогут оставить довольным пользователя. Кроме того, фатальным недостатком данной игры является отсутствие редактора уровней. Конечно, можно было создавать уровни вручную в простом текстовом файле, однако данный подход далёк от привычного на сегодняшний день и понятного для пользователя графического интерфейса. Разработчики игры понимали, насколько важны дополнительные уровни для Sokoban, поэтому было выпущено три сиквела игры: Boxxle, Sokoban Perfect и Sokoban Revenge. Интерфейс данной игры можно увидеть на рисунке 1.1.

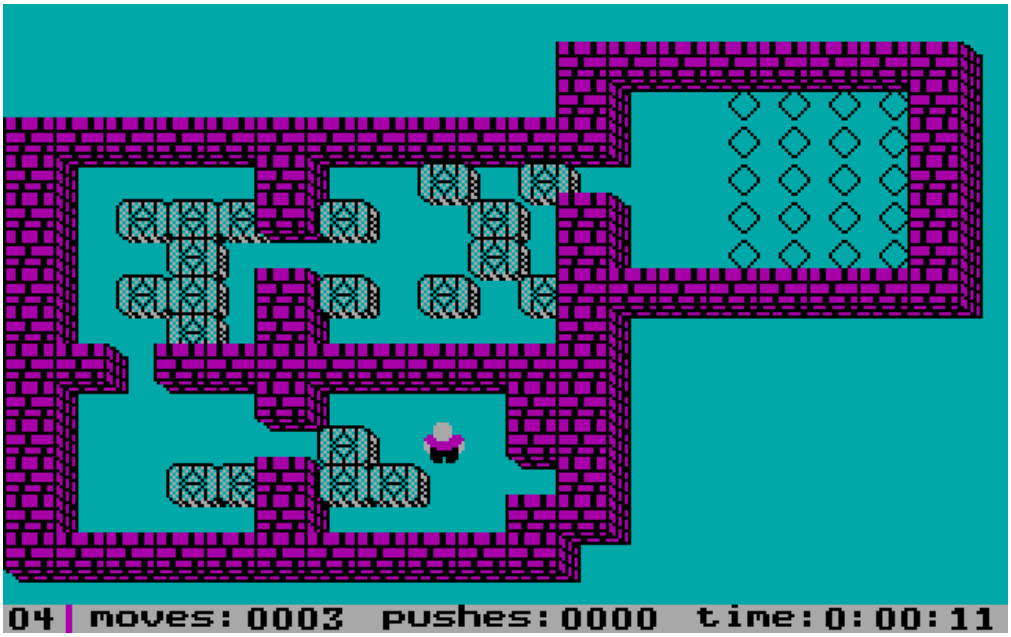

Рисунок 1.1 – Интерфейс приложения «Sokoban»

1.1.2 «Sokoban bash»

«Sokoban bash» – реализация Sokoban, написанная, как ясно из названия, на скриптовом языке bash. Одно из главнейших достоинств программы – маленький размер. Программа занимает всего лишь 5 КБ места на жёстком диск, не считая игровых уровней. Игра придётся по духу любителям минимализма и консоли. Уровни в игре представлены не в виде набора картинок, а просто в виде символов ASCII. Это делает интерфейс игры непонятным среднестатистичному пользователю. К тому же, редактора уровней у игры тоже нету. В распространяемом виде игра работает только в *nix системах, для использования её в операционной системе Windows требуется установка дополнительных библиотек.

Несмотря на столь очевидные для обычного пользователя минусы, по функционалу игра ни в чём не уступает любым другим аналогам Sokoban, но при этом занимает в десятки, а то и в сотни раз меньше места на жёстком диске. Настоящий пример Unix-Way подхода к решению задач. Внешний вид данной игры можно увидеть на рисунке 1.2.

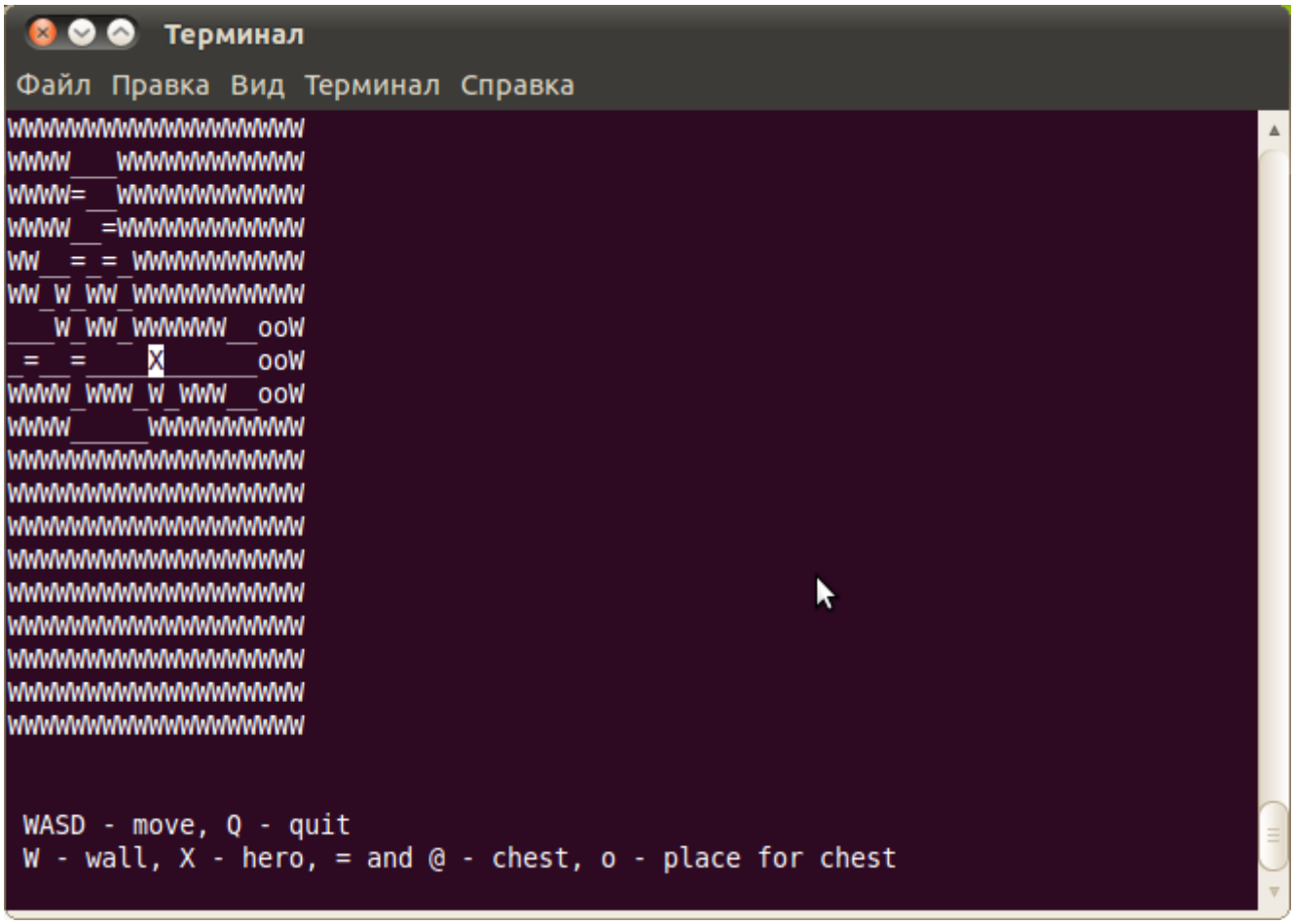

Рисунок 1.2 – Интерфейс приложения «Sokoban bash»

1.1.3 «Kurtan»

«Kurtan» – реализация игры Sokoban, созданная российскими разработчиками, в которой были добавлены новые элементы геймплея: телепорты, позволяющие игроку перемещаться по карте из одного телепорта в другой, враги, атакующие игрока и мешающие ему передвигать коробки, лестницы, наклонные стены. В игре появилась экономическая система, стало возможно за внутриигровую валюту покупать подсказки по прохождению уровней. Игра стала значительно интереснее для простого пользователя, однако данные элементы превратили игру из классической головоломки в логическую игру с элементами экшена. Кроме того, данная игра не имела редактора уровней.

Также, как и оригинальный Sokoban, рассмотренный в пункте 1.1.1, данная игра создавалась для операционной системы MS DOS, и недоступна на сегодняшний день без использования дополнительного программного обеспечения в современных операционных системах. Кроме того, игра имеет низкокачественные текстуры, в игре отсутствует система рекордов.

Внешний вид игры «Kurtan» можно увидеть на рисунке 1.3.

Рисунок 1.3 – Интерфейс приложения «Kurtan»

1.2 Постановка задачи

Целью данной курсовой работы является создание полноценной игровой среды Sokoban, включающей в себя менеджер уровней, таблицу рекордов, редактор уровней, игровое поле.

В процессе реализации программного средства особое внимание стоит уделить целостности и взаимосвязанности компонентов, входящих в систему. Например, при выборе уровня для игры, должен сразу отображаться наилучший результат, достигнутый на этом уровне, чтобы игроку было видно, к чему стремится, при редактировании уровня необходимо обеспечить возможность нажатием одной клавиши запустить данный уровень на игровом поле.

В результате анализа различных вариаций игры Sokoban, было принято решение особое внимание уделить функциональности и быстроте редактора уровней. В ходе рассмотрения различных редакторов уровней, было решено отказаться от использования кнопок выбора текстур в редакторе, поскольку постоянные движения мышкой по палитре компонентов сильно уменьшают скорость работы. Было принято решение использовать клавиши на клавиатуре как средство редактирования уровней.

Кроме того, необходимо реализовать хотя бы частично vim-like управление, которое превосходно подходит для редактора, когда используется только управление клавиатурой.

В программном средстве планируется реализовать следующие функции:

– игра в Sokoban на уровнях разной сложности;
– возможность отмены хода, паузы, рестарта игры;
– динамическое изменение размера текстур в зависимости от размера карты;
– возможность добавления, удаления, переименования уровней и категорий уровней;
– хранение и показ рекорда конкретного уровня, с указанием количества шагов, толканий ящиков и времени, которые были затрачены на прохождение уровня;
– редактор уровней, который включает в себя возможность сохранения и загрузку уровней, несколько режимов редактирования, возможности добавления и удаления строк и столбцов карты в нужных пользователю позициях, возможность тестирования уровня;
– возможность получения справки по каждому компоненту системы.

Для разработки программного средства будет использоваться язык программирования Delphi и среда разработки Delphi 7.

2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Интерфейс программного средства

Проанализировав палитру компонентов среды Delphi, был сделан вывод, что для отображения уровня игры на форме подойдёт компонент DrawGrid с изменёнными свойствами. Свойствам FixedCols и FixedRows было присвоено значение 0, что обеспечило одинаковый вид всем ячейкам DrawGrid, были убраны границы, посредством присвоения свойству BorderStyle значения bsNone. Цвет DrawGrid был выбран совпадающим с цветом формы, то есть clBtnFace. Также были убраны границы между ячейками. Благодаря отключению свойств DefaultDrawing и Enabled удалось полностью прекратить взаимодействие между данным компонентом и курсором мыши. В результате данных манипуляций внешний вид компонента стал неотличим от пустого рабочего пространства формы, но, тем не менее, удалось сохранить все полезные свойства данного компонента.

При разработке было принято решение автоматически загружать нужные текстуры в ячейки DrawGrid при чтении уровня из файла. Для этого использовались методы CellRect, Canvas.FillRect и Canvas.CopyRect. В данном коде можно увидеть, как в ячейки DrawGrid с координатами aXCoord, aYCoord копируется изображение, закруженное в объект Img, принадлежащий классу TBitmap. Изображение загружается из файла, путь к которому получает функция GetPathFromChar, принимающая в качестве аргумента название изображения, которое хранится в переменной aChar. При этом, если в качестве aChar указан символ, обозначающий отсутствие изображения, то выбранная ячейка DrawGrid полностью закрасится цветом самого DrawGrid.

```
procedure DrawCell(aGrid: TDrawGrid; aXCoord, aYCoord: Integer;
   aChar: char);
var
   Img: TBitmap; // Изображение с текстурой
begin
   if aChar = CNullChar then
   begin // Закрашиваем ячейку цветом самого aGrid
      aGrid.Canvas.Brush.Color := aGrid.Color;
      aGrid.Canvas.FillRect(aGrid.CellRect(aXCoord-1, aYCoord));
   end
   else
   begin // Загружаем текстуру с файла
      Img := TBitmap.Create;
      Img.LoadFromFile(GetPathFromChar(aChar));
      aGrid.Canvas.CopyRect(aGrid.CellRect(aXCoord - 1,
         aYCoord), Img.Canvas, Rect(0,0,Img.Height,Img.Width));
      Img.Free;
```

```
   end;
end;
```

Благодаря использованию компонента DrawGrid появилась возможность просто осуществлять перерисовку конкретной ячейки, без необходимости перерисовывать все ячейки, что повысило плавность игры.

Теперь можно рассмотреть управление размерами DrawGrid. Очевидно, что размер уровня может быть самым разным, от небольшого, размером 5x5, до огромного, размером 70x90, или даже больше. При этом становится очевидно, что размер ячейки DrawGrid должен изменяться динамически, в зависимости от размера уровня. В связи с тем, что Delphi очень плохо работает с масштабированием TBitmap, было принято решение заранее создать несколько наборов текстур разных размеров: 16, 32, 48 и 64 пикселя. Узнать, какой размер должен быть использован, можно получив размеры экрана и размер карты в длину и ширину. Ширину экрана можно получить, используя Screen.WorkAreaWidth, длину – Screen.WorkAreaHeight. Уровень представлен в виде массива строк, поэтому его ширину и высоту также легко вычислить. Если поделить длину и ширину экрана на длину и ширину уровня (в дальнейшем эти значения будут называться максимальными размерами), выбрать потом из этих значений минимальное и выполнить округление вниз до одного из предопределённых размеров текстуры (16, 32, 48 или 64 пикселя), то получиться нужный размер ячейки DrawGrid. После установления необходимого размера, он сохраняется в глобальной переменной PropSize. Ниже приведён алгоритм округления максимального размера по горизонтали и вертикали (передастся в функцию как aXProp и aYProp) вниз к предопределённому размеру. Функция возвращает false, если данный уровень невозможно уместить на экране даже при самом малом значении размера текстуры.

```
function GetPropSize(aXProp, aYProp: Integer): Boolean;
const
   CMaxSize = 64; // Максимальный размер текстуры
   CMinSize = 16; // Минимальный размер текстуры
var
   i: Integer;
   Min: Integer;
begin
   if aXProp < aYProp then // Выбираем минимальное значение
      Min := aXProp
   else
      Min := aYProp;
   i := CMaxSize;
   while (i >= CMinSize) and (Min div i = 0) do
      Dec(i, 16); // Получаем нужный размер
   if i > 0 then
      PropSize := i;
   if Min < CMinSize then
```

```
      // Если вместить невозможно, возвращаем False
      GetPropSize := False
   else
      GetPropSize := True;
end;
```

Ключевой особенностью отображения игрока на игровом поле является то, что игрок представляется одной из четырёх текстур, которые выбираются в зависимости от того, в какое направление сделал ход игрок в предыдущем ходе: вниз, вверх, вправо или влево. Для хранения перемещения были описаны два перенумерованных типа, которые содержат относительное перемещение игрока по горизонтали и вертикали.

Внешний вид четырёх текстур игрока представлен на рисунке 2.1.

```
TDxMove = (dxLeft = -1, dxRigth = 1, dxNone = 0);
TDyMove = (dyDown = 1, dyUp = -1, dyNone = 0);
```

Рисунок 2.1 – Четыре текстуры игрока

Внешний вид небольшой карты, загруженной на игровое поле, можно увидеть на рисунке 2.2, большой – на рисунке 2.3.

Таким образом, были рассмотрены способы представления игрового поля. Далее будут рассмотрены способы представления редактора уровней, менеджера уровней и таблицы рекордов. Но для начала необходимо рассмотреть, как в приложении представляются кнопки, ведь если в игровом поле они не понадобились, то в менеджере уровней, главном меню и таблице уровней они просто необходимы.

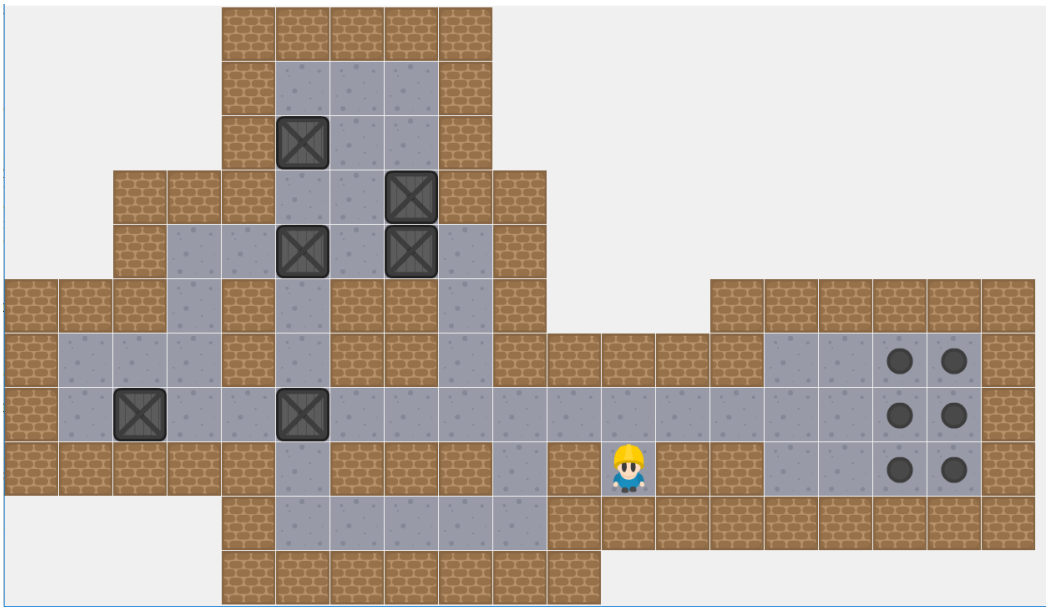

Рисунок 2.2 – Небольшая карта на игровом поле

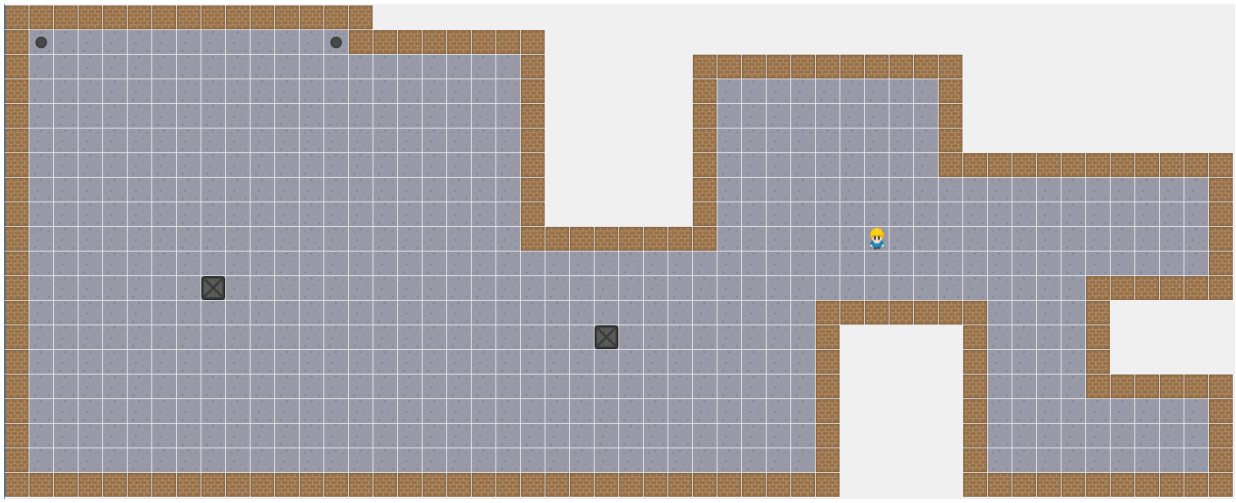

Рисунок 2.3 – Большая карта на игровом поле

Проанализировав палитру компонентов среды Delphi, был сделан вывод, что имеющиеся визуальные компоненты (Button, SpeedButton, BitButton) не подходят для организации работы кнопок из-за отсутствия возможности детальной настройки и несоответствия стилю программы. Поэтому было принято решение использовать компонент TImage для организации работы кнопок.

Было принято решение не разрабатывать кнопки с нуля, а скачать необходимые заготовки в интернете, после чего отредактировать их в соответствии со своими нуждами. Для редактирования был выбран редактор Paint.NET, предоставляющий широкие возможности в работе с растровой графикой. Готовая текстура кнопки представлена на рисунке 2.4

Рисунок 2.4 – Текстура кнопки, используемая в приложении

Несмотря на высокую трудоемкость, этот способ позволяет избежать шаблонности дизайна, а также приобрести опыт разработки внешнего вида элементов управления

Менеджер уровней решено было совместить с окном выбора уровня и таблицей рекордов. Несмотря на возросшую функциональность, большое количество кнопок вызывало недопонимание у пользователя, поэтому было принято решение скрывать и показывать кнопки, отвечающие за управление уровнями, по нажатию одной конкретной кнопки. Чтобы при скрытии кнопок на форме не оставалось незаполненного пространства, было принято решение автоматически увеличивать размер списка уровней при скрытии кнопок, и уменьшать при показе.

Для представления категорий уровней был использован компонент ListBox, а для представления списка уровней – StringGrid. В столбцах таблицы StringGrid находится название уровня и значение рекорда, который состоит из количества шагов (Steps), толканий ящиков (Pushes) и наилучшего времени (Time).

Внешний вид данного окна с отображаемыми и скрытыми кнопками редактирования представлен на рисунке 2.5

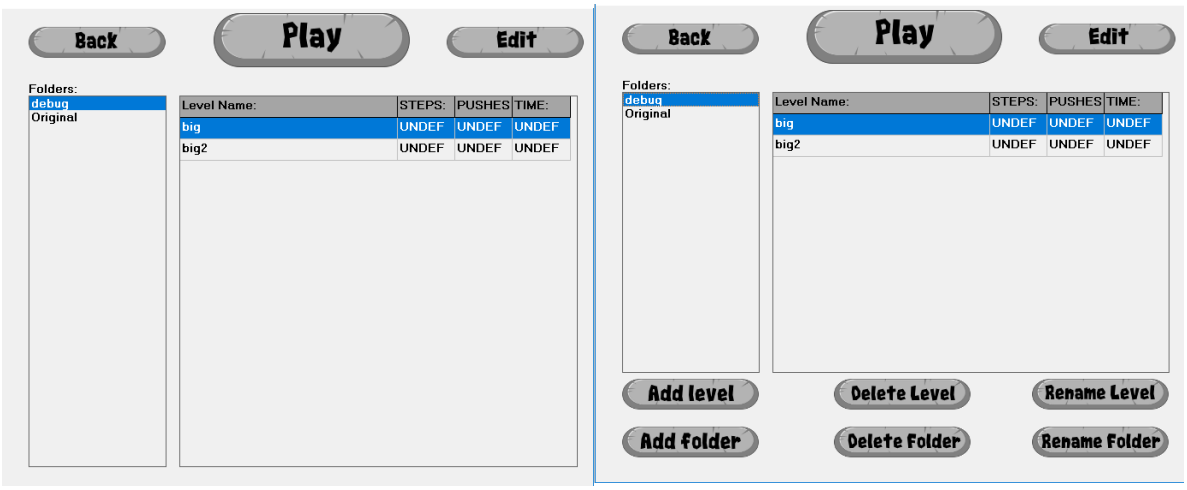

Рисунок 2.5 – Внешний вид окна выбора уровня

Теперь стоит рассмотреть интерфейс редактора уровней. Для комфортного процесса создания уровней необходимо наглядно обозначить позицию курсора в уровне. Стандартное выделение ячейки DrawGrid оказалось малоэффективным, поэтому было принято решение разработать свой способ выделения. В редакторе уровней присутствует два режима, поэтому необходимо использовать два способа выделения.

Было принято решение использовать мигающий квадрат зелёного цвета для первого режима и мигающий квадрат красного цвета для второго режима. Таким образом можно добиться привычного для пользователя способа представления курсора. Для реализации данной идеи потребовалось использовать таймер, который выбирает, прорисовывать сейчас на DrawGrid курсор или текстуру, соответствующую элементу редактирования. Код, управляемый данной особенностью, вызывается по срабатыванию таймера и приведён ниже.

```
procedure TFormEditor.TimerTimer(Sender: TObject);
begin
   with GridEdit do
   begin
      if IsCoursourVisible then
      begin
         // Задаём цвет для разных режимов
         if Mode = mdOverwrite then
            Canvas.Brush.Color := clGreen
```

```
            else
               Canvas.Brush.Color := clRed;
            Canvas.FillRect(CellRect(Col, Row)); // Заливаем цветом
         end
         else
            // Отрисовываем ячейку
            DrawCell(GridEdit, Col + 1, Row, Map[Row, Col + 1]);
   end;
   IsCoursourVisible := not IsCoursourVisible;
end;
```

В зависимости от состояния флага IsCoursourVisible рисуется или курсор, или ячейка DrawGrid. Цвет курсора определяет режим, в котором сейчас находится редактор уровней. Содержимое ячейки – массивом строк Map, в котором содержится код текстуры каждой ячейки. Процедура DrawCell была рассмотрена раньше. После отрисовки значение флага IsCoursourVisible меняется на противоположный, и на следующем срабатывании таймера будет выполняться противоположное действие.

2.2 Реализация программной логики

Рассмотрение программной логики будет начать с игрового поля игры Sokoban. Игровой уровень представлен в виде массива строк, то есть матрицы символов. Каждому элементу матрицы соответствует одна клетка на игровом поле. Разным символам соответствуют разные элементы. Давайте перечислим все возможные названия элементов:

- стена;
- игрок;
- ящик;
- место для ящиков;
- игрок, стоящий на месте для ящиков;
- ящик, стоящий на месте для ящиков;
- дорога, по которой ходит игрок;
- отсутствие какого-либо элемента, описанного ранее.

При выборе уровня для игры он загружается из файла в матрицу Map. После этого контроллер отлавливает нажатие клавиш и передаёт управление игровой модели. Если были нажаты клавиши, отвечающие за движение игрока, то задача игровой модели – это определить, возможно ли перемещение игрока в указанном направлении, если возможно, то необходимо сделать данное перемещение и выполнить процедуру DrawCell, чтобы отобразить изменения в DrawGrid. Процедура DrawCell была описана в 2.1.

По игровой логике игры Sokoban перемещение игрока возможно в следующих случаях:

- игрок хочет перейти на дорогу или место для ящиков;

– игрок хочет толкнуть ящик, при этом за ящиком стоит либо дорога, либо место для ящиков

Также игрок и ящик могут как стоять на месте для ящиков, так и нет. Всё это создаёт большое дерево ветвлений, поэтому код в данном пункте не приводится, с ним можно ознакомиться в приложении А.

После того, как ход сделан, необходимо его записать, чтобы при необходимости осуществить его отмену. Для записи хода был объявлен следующий тип-запись:

```
TMove = packed record
   PlayerDX: TDxMove;
   PlayerDY: TDyMove;
   Char1, Char2, Char3: Char;
   PoolChange: Integer;
   IsPushedBox: Boolean;
end;
```

Типы TDxMove и TDyMove были описаны в 2.1, они характеризуют перемещение игрока за один ход по горизонтали и вертикали. Char1, Char2 и Char3 характеризуют соответственно первый, второй и третий символ с карты в направлении перемещения, начиная с текущей позиции. PoolChange показывает, на сколько изменилось количество пустых мест для ящиков, а поле IsPushedBox – было ли движение коробки во время хода. Данная запись помещается в специальную динамическую структуру данных, созданную по принципу LIFO. Основное отличие её от стека – фиксированный размер, определяемый в конструкторе. Данный размер характеризует максимальное количество элементов, которые могут хранится в данной структуре, то есть количество отмен, которые можно произвести. Если структура уже достигла своего максимального размера, но в неё добавляется ещё один элемент, то последний элемент структуры уничтожается. Данный приём позволяет очень экономно работать с памятью, что особо полезно на больших уровнях.

Объявление класса, построенного на принципах, описанных выше:

```
PMoveStack = ^TMoveStack;
TMoveStack = record
   Move: TMove;
   pNext: PMoveStack;
   pPrev: PMoveStack;
end;
TMoveStackClass = class
   pHead: PMoveStack;
   pTop: PMoveStack;
   Count: Integer;
   MaxCount: Integer;
   constructor Create(const aMaxCount: Integer);
   function Pop: TMove;
   procedure Push(const aMove: TMove);
```

```
end;
```

Метод Push добавляет ход в структуру данных, метод Pop возвращает последний добавленный ход. В Count находится количество элементов, находящихся на данный момент в структуре. pHead – указатель на голову структуры, pTop – указатель на хвост. MaxCount – максимальное количество элементов в структуре. Реализацию класса можно найти в приложении А.

В процессе игры ведётся подсчёт количества ходов, толканий коробок и времени. Когда пользователь побеждает, эти данные сравнивают с рекордом, и если все из них не хуже, чем рекорд, то эти данные становятся новым рекордом. Для хранения рекорда применяется такая запись:

```
TRecord = record
   Moves: Integer;
   Pushes: Integer;
   Time: string[8];
end;
```

Хранение рекорда к каждому уровню осуществляется в типизированном файле.

Теперь будет рассмотрен процесс загрузки уровня на карту. Вызывающее меню инициализирует две глобальные переменные: Path и LevelName. В переменной Path содержится путь к файлу, содержащий уровень, в LevelName – имя уровня, которое будет отображаться в строке состояния. При запуске форма с игровым полем вызывает функцию GetMapInFile для получения карты из файла. Она осуществляет примитивный контроль на наличие посторонних символов, и возвращает True, если карта прочитана нормально, в противном же случае – False. Карта представлена набором символом ASCII, внешний вид карты в текстовом редакторе можно увидеть на рисунке 2.6.

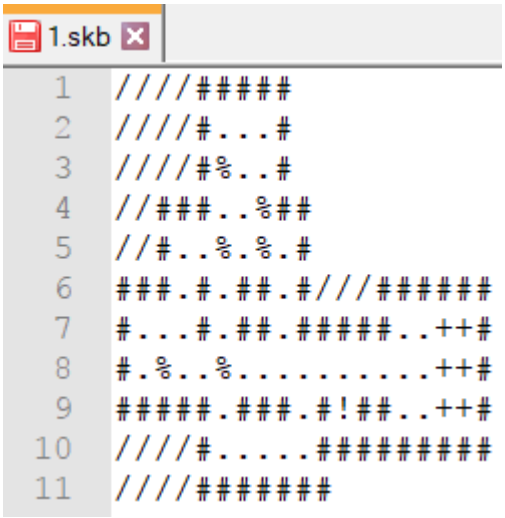

Рисунок 2.6 – Внешний вид карты в текстовом редакторе

Код функции GetMapInFile представлен в приложении А. Если всё прочитано нормально, дальше вызывается процедура GetVariables, которая инициализирует начальное положение игрока и количество пустых мест для ящиков. На рисунке 2.7 представлена блок-схема данной процедуры.

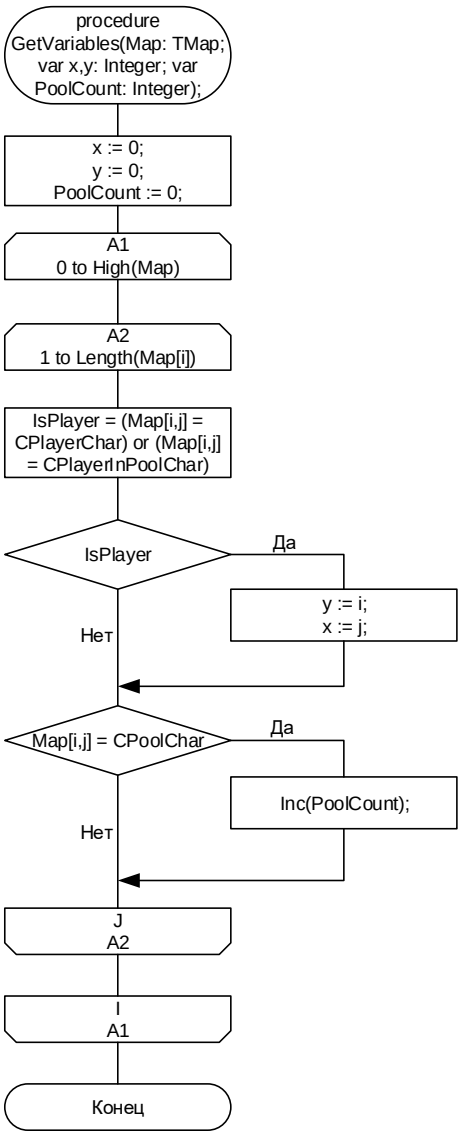

Рисунок 2.7 – Блок-схема процедуры GetVariables

Когда игрок задвигает ящик в место для него, количество пустых мест для ящиков уменьшается, когда выдвигает – увеличивается. Как только оно станет равно нулю, игрок побеждает. После победы принимается решение о сохранение рекорда. Рекорд хранится в одноимённом файле, только с другим расширением: расширение уровня – .skb, расширение рекорда – .rec. Если одноимённый файл с расширением .rec отсутствует, то он создаётся, и в него записывают текущее значение рекорда, в противном случае рекорд читается из файла и сравнивается с текущим. Если текущий рекорд лучше прочитанного с файла, то происходит запись в файл текущего рекорда.

Перейдём к рассмотрению строки состояния. Она отображается в заголовке формы и имеет следующий формат: <Статус> <Имя уровня> <Текущий рекорд>.

В строке <Статус> отображается [PLAYED], если игра запущена, или [STOP], если игра поставлена на паузу. За это состояние отвечает булевская переменная IsGamePlayed. Имя уровня берётся из переменной LevelName, которая была описана выше, текущий рекорд – это три параметра: время (изменяется по таймеру), количество ходов и количество движений ящиков. При любом изменении, которое затрагивает эти параметры, вызывается перерисовка строки состояния, процедура, управляющая этим, – DrawCaption.

Был рассмотрен механизм работы игрового поля, сейчас будет рассмотрен механизм работы формы-лаунчера. Цель формы-лаунчера: запустить уровень, показать текущие рекорды и обеспечить возможность добавления, удаления и переименования уровней и категорий уровней.

Уровни игры хранятся в директории Levels, которая находится в каталоге программы. Внутри этой директории лежат различные папки, которые называются категориями уровней. Каждая категория может содержать в себе любое количество уровней и рекордов к ним. Уровень представляет собой обычный текстовый файл в кодировке ASCII, в котором при помощи обычных символов представлен уровень. Устройство файла-рекорда было описано выше по тексту. Следовательно, операции с уровнями и их категориями можно свести к обычным операциям с файлами и папками, что значительно упрощает работу. При запуске формы-лаунчера, происходит загрузка директорий и уровней. За это отвечают два метода формы: LoadLevels и LoadDirectories, не принимающие никаких параметров. С кодом данных методов можно ознакомиться в приложении А, ниже же будет дано объяснения принципа их работы.

LoadDirectories ищет в папке Levels директории, после чего добавляет их названия в ListBox, и увеличивает счётчик. Если счётчик окажется меньше 2, то метод заблокирует возможность удаления директорий.

LoadLevels ищет в папке, соответствующей названию выделенной ячейке ListBox, файлы с расширением .skb и загружает их имена в соответствующую колонку StringGrid. Когда такой файл найден, метод пробует найти файл с таким же именем, но с расширением .rec. Это будет рекорд, который соответствует данному уровню. Если он найден, поля записи, прочитанной из этого файла, будут занесены в оставшиеся 3 столбца таблицы, иначе в 3 столбца таблицы занесётся строка UNDEF, что означает неопределенность значения рекорда уровня.

При всех изменениях в структуре уровней, будь то удаление, добавление или переименование, будут вызываться эти методы, чтобы на форме отображалась действительная картина. Теперь будут рассмотрены описания процедур создания, удаления и переименования уровней и

категорий уровней. В некоторых процедурах будет использоваться форма FormRename, которая представляет собой простую форму с одним текстовым полем, представленным компонентом Edit, и кнопкой OK. Перед показом данной формы в её текстовое поле вызывающий код устанавливает какое-то значение, а форма затем в переменную NameRename возвращает то, что пользователь ввёл в её текстовое поле, если была нажата клавиша Enter или кнопка OK, иначе пустую строку.

Процедура добавления папки получает имя создаваемой папки через вышеописанную форму и создаёт папку, используя процедуру MkDir.

Для добавления уровня используется диалог выбора файла, после чего через форму FormRename получается новое имя, и файл копируется в директорию, в которую уровень был добавлен. Используется стандартная функция CopyFile.

При переименовании уровня текущее имя загружается в форму FormRename, затем получается новое, и, используя стандартную функцию RenameFile, файл, содержащий уровень, переименовывается.

Процесс переименования папки аналогичен переименованию уровня

Удаление уровня происходит с помощью встроенной функции DeleteFile. Затем функцией FileExists проверяется существование файла-рекорда для данного уровня, и, если он существует, то тоже удаляется.

Удаление папки происходит сложнее, так как стандартная процедура RmDir удаляет только пустые папки. Можно было искать все файлы в данной папке, удалить их, а затем вызвать RmDir, но было найдено более элегантное решение – использование WinAPI, путём подключения юнита ShellAPI. Код, выполняющий удаление папки вместе со всеми файлами, находящимися в ней, представлен ниже

```
function DelDir(dir: string): Boolean;
var
  fos: TSHFileOpStruct;
begin
  ZeroMemory(@fos, SizeOf(fos)); // Обнуление полей структуры
  with fos do
  begin
    wFunc  := FO_DELETE; // Задаём нужные параметры для удаления
    fFlags := FOF_SILENT or FOF_NOCONFIRMATION;
    pFrom  := PChar(dir + #0);
  end;
  // Применяем операцию
  Result := (0 = ShFileOperation(fos));
end;
```

Сначала объявляется запись fos, затем обнуляются все значения записи процедурой ZeroMemory, потом присваиваем нужным полям значения, при этом особое внимание стоит обратить на то, что значение поля pFrom, обозначающее путь, по которому будет выполнена операция, обязано

заканчиваться двумя нулевыми символами. Когда заданы значения всем нужным полям, вызывается функция ShFileOperation, принимающая в качестве параметра запись. Эта функция возвращает код ошибки, если он равен 0, то ошибки нету, и возвращается True, иначе – False.

Запуск уровня довольно прост. Просто создаётся форма с игровым полем и передаётся туда путь к уровню и название уровня.

Теперь будет рассмотрена реализация редактора уровней.

Как и в случае с игровым полем, уровень представлен массивом строк, и все операции по его изменению ведутся именно с массивом.

Ещё при проектировании было принято решение реализовать частично Vi-like управление. Редактор уровней, как и vi, имеет два режима: в первом можно изменять размеры карты, вставлять и удалять строки или столбцы, запускать уровень на тест, выходить из программы, в другом же осуществляется непосредственное редактирование уровня путём перезаписи ячеек DrawGrid. Визуальное отличие этих режимов было описано в 2.1.

В редакторе поддерживается работа с файлами, загрузка и отображение уровня аналогична тому, как это делается в случае с игровым полем и уже была описана ранее. Сохранение уровня тривиально, с кодом функции SaveMap можно ознакомиться в приложении А.

Редактор поддерживает перемещение по клеткам не только с помощью клавиш-стрелок, но и с помощью клавиш h, j, k, l. Подробнее об этом можно прочитать в руководстве пользователя.

Сейчас будут описаны основные функции редактора уровней. Начнём с режима перезаписи.

В режиме перезаписи нажатие определённой клавиши приводит к вставке в выделенную ячейку определённой текстуры. Например, нажатие клавиши «w» приведёт к вставке в выделенную яйчейку стены («w» – wall).

В режиме редактирования, существуют следующие команды: вставка и удаление строки или столбца, выход из программы, запуск уровня для тестирования.

Для удаления столбца с номером aCol нам достаточно в каждой строке массива строк удалить aCol + 1 элемент из строки. Единица добавляется из-за того, что в Delphi индексация строки начинается с 1, в то время как индексация DrawGrid начинается с 0. Код, реализующий это:

```
for i := 0 to High(Map) do
   Delete(Map[i], aCol + 1, 1);
```

Со вставкой столбца код практически такой же, только вместо процедуры Delete используется процедура Insert.

```
for i := 0 to High(Map) do
   Insert(CDrawChar, Map[i], aCol + 1);
```

Теперь будет рассмотрено удаление строки с номером aRow. Этот алгоритм немного сложнее, так как отсутствует готовая процедура Delete для массива строк. Поэтому будет создан новый массив строк, длина которого на 1 меньше, чем длина текущего массива и скопируем последовательно все строки, расположенные до и после удаляемой в него. Потом присвоим старому массиву новый. Код, реализующий данный функционал:

```
SetLength(NewMap, Length(Map) - 1);
for i := 0 to aRow - 1 do
   NewMap[i] := Map[i];
for i := aRow + 1 to High(Map) do
   NewMap[i-1] := Map[i];
Map := Copy(NewMap);
```

С вставкой строки ситуация аналогичная. Создаётся новый массив строк, но его длина уже на 1 больше, чем длина текущего массива. В него копируются все строки до вставляемой позиции, затем вставляется строка, после этого продолжается копирование строки со старого массива. Код, реализующий данный функционал:

```
SetLength(NewMap, Length(Map) + 1);
for i := 0 to aRow - 1 do
   NewMap[i] := Map[i];
for i := 0 to Length(Map[0]) - 1 do
   Insert(CDrawChar, NewMap[aRow], i + 1);
for i := aRow to High(Map) do
   NewMap[i + 1] := Map[i];
Map := Copy(NewMap);
```

Кроме уже рассмотренных основных функций, существуют ещё две вспомогательных функции, спектр применения которых не такой массовый.

Функция AddNullChar дополняет длину каждой строки символом CNullChar, который соответствует пустой текстуре, до максимальной длины строки в массиве. Данная функция используется при чтении карты с файла, ведь карта может создаваться не только в редакторе, но и самостоятельно в блокноте, и при этом способе, скорее всего, строки в массиве будут разной длины, что недопустимо при представлении их на форме в качестве DrawGrid. Код преобразования:

```
MaxLength := Length(Map[0]);
for i := 1 to High(Map) do
   if Length(Map[i]) > MaxLength then
      MaxLength := Length(Map[i]);
for i := 0 to High(Map) do
   for j := Length(Map[i]) to MaxLength - 1 do
      Insert(CNullChar, Map[i], j + 1);
```

В данном коде сначала вычисляется максимальная длина строки в массиве, затем оставшиеся строки дополняются символами CNullChar до этой длины.

Функция TransformBorderChar, принимающая два параметра типа Char: aFrom и aTo, преобразовывает все символы aFrom, стоящие возле границы карты на символ aTo. Символ считается стоящим возле границы, если он либо примыкает к границе, либо лежит на прямой, перпендикулярной границе, причем на отрезке между символом и границей расположены только такие же символы. Код данной функции можно найти в приложении А.

В редакторе уровней также поддерживается быстрое создание уровня нужного размера. Для этого вызывается вспомогательная форма, в которой пользователь указывает размеры создаваемой карты. Дальше карта заполняется символами, которые обозначают текстуру дороги. Код достаточно примитивен, поэтому не приводится.

Таким образом, был закончено рассмотрение принципа работы игрового средства Sokoban.

3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

3.1 Правила игры

Правила игры полностью соответствуют классической концепции игры «Sokoban». Задача игрока: расставить все коробки на карте по своим местам, при этом коробки можно только толкать, но не тянуть. Толкать несколько коробок за 1 ход также воспрещается.

3.2 Интерфейс программы

В программе присутствуют три главных формы: меню, игровое поле, редактор уровней.

3.2.1 Меню

Меню игры представляет собой большую таблицу уровней и категорий уровней, кнопки, отвечающие за редактирование данной таблицы. При запуске меню кнопки редактирования скрыты, для их появления необходимо нажать кнопку «Edit». Внешний вид меню со скрытыми и видимыми кнопками представлен на рисунке 3.1. Назначение каждой кнопки понятно из её названия. Для начала игры необходимо выбрать нужный уровень и нажать кнопку «Play».

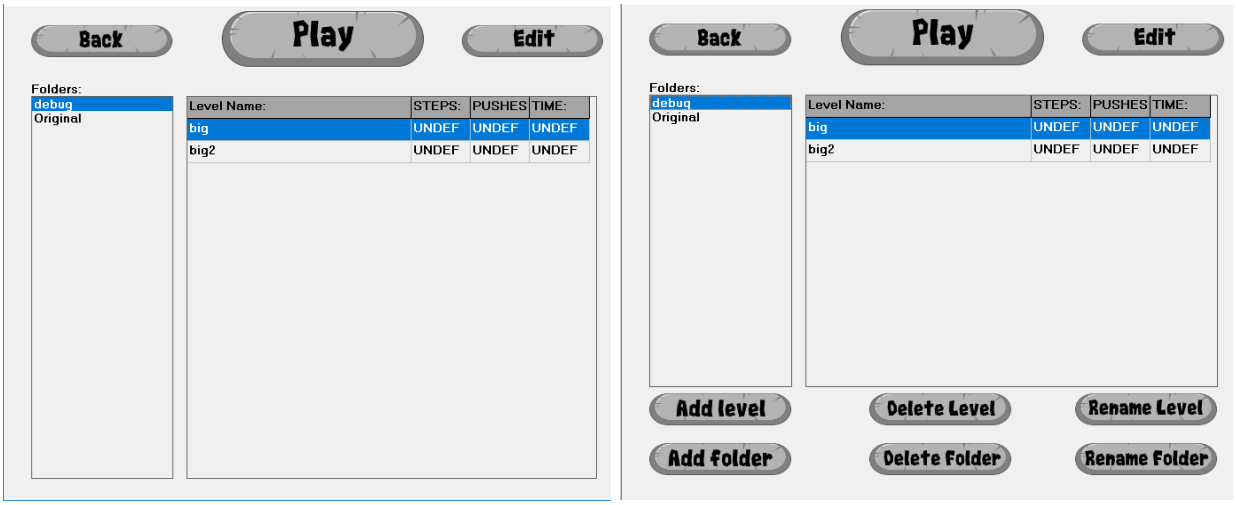

Рисунок 3.1 – Внешний вид меню с скрытыми и отображаемыми кнопками

3.2.2 Игровое поле

Игровое поле представлено набором текстур, загружаемых с файла. Для перемещения игрока по полю можно использовать клавиши со стрелками. Строка состояния в заголовке формы отображает состояние игры, название уровня, а также текущий рекорд. Существуют три различных состояния игры:

– «[PLAYED]» – игра запущена;

– «[PAUSE]» – игра на паузе;
– «[DEBUG]» – возникает при запуске уровня прямо с редактора уровней.

3.2.3 Редактор уровней

Редактор уровней имеет два режима: режим перезаписи и режим команд. Если вы находитесь в режиме перезаписи, то курсор мигает зелёным цветом, если в режиме команд, то красным. В режиме перезаписи можно изменять текстуру, которая находится в каждой ячейке уровня. В режиме команд можно изменять размеры карты, сохранять и тестировать уровень, выходить из редактора. Большинство действий выполняется с помощью горячих клавиш. Их описание вы можете найти в пункте 3.3

3.3 Горячие клавиши

Горячие клавиши представлены в таблицах. В таблице 3.1 находятся горячие клавиши, используемые на игровом поле. В таблице 3.2 – используемые в любом из режимов редактора уровней. В таблицах 3.3 и 3.4 располагаются клавиши для режима перезаписи и режима команд в редакторе уровней. В поле «Клавиша» находится название клавиши, в поле «Расшифровка» – мнемоника, помогающая запомнить клавишу, в поле «Описание» – описание команды, вызываемой по горячей клавише.

Таблица 3.1 – Горячие клавиши на игровом поле

Клавиша	Расшифровка	Описание
J или ↓	–	Переместить игрока на 1 клетку вниз
K или ↑	–	Переместить игрока на 1 клетку вверх
H или ←	–	Переместить игрока на 1 клетку влево
L или →	–	Переместить игрока на 1 клетку вправо
P	Pause	Пауза игры
Q	Quit	Выход из игры
U	Undo	Отменить последний ход
R	Restart	Перезапустить игру

Таблица 3.2 – Глобальные горячие клавиши редактора уровней

Клавиша	Расшифровка	Описание
J или ↓	–	Переместить курсор на 1 клетку вниз
K или ↑	–	Переместить курсор на 1 клетку вверх
H или ←	–	Переместить курсор на 1 клетку влево
L или →	–	Переместить курсор на 1 клетку вправо
I или M	Mode	Переключить режим

Таблица 3.3 – Глобальные горячие клавиши редактора уровней в режиме перезаписи

Клавиша	Расшифровка	Описание
W	Wall	Вставить стену
B	Box	Вставить ящик
Shift+B	Box	Вставить ящик, стоящий на месте для ящиков
P	Player	Вставить игрока
Shift+P	Player	Вставить игрока, стоящего на месте для ящиков
G	Ground	Вставить дорогу
N	Null	Вставить пустую текстуру
O	–	Вставить место для ящиков

Таблица 3.4 – Глобальные горячие клавиши редактора уровней в режиме команд

Клавиша	Расшифровка	Описание
E	Execute	Запустить данный уровень
G	Ground	Преобразовать граничные текстуры дороги в пустые текстуры
N	Null	Преобразовать граничные пустые текстуры в текстуры дороги
W	–	Вставить строку над курсором
A	–	Вставить столбец слева от курсора
S	–	Вставить строку под курсором
D	–	Вставить столбец справа от курсора
Shift+W	–	Вставить строку вверху уровня
Shift+A	–	Вставить столбец в левом краю уровня
Shift+S	–	Вставить строку внизу уровня
Shift+D	–	Вставить столбец в правом краю уровня
X	–	Удалить строку, на которой находится курсор
Shift+X	–	Удалить столбец, на котором находится курсор

ЗАКЛЮЧЕНИЕ

С давних пор решение головоломок является популярным развлечением. И даже в эпоху компьютерных игр они продолжают пользоваться успехом. Sokoban, будучи одной из самых популярных головоломок, имеет простые правила, не требует дополнительных знаний, а также развивает интеллект и внимательность.

В рамках данной курсовой работы было создано программное средство «Sokoban», предназначенное для создания и прохождения уровней. Разработанное программное средство отличается простым и понятным интерфейсом и высокофункциональным редактором уровней. При разработке программного средства были выполнены следующие задачи:

– игра в Sokoban на уровнях разной сложности;
– возможность отмены хода, паузы, рестарта игры;
– динамическое изменение размера текстур в зависимости от размера карты;
– возможность добавления, удаления, переименования уровней и категорий уровней;
– хранение и показ рекорда конкретного уровня, с указанием количества шагов, толканий ящиков и времени, которые были затрачены на прохождение уровня;
– редактор уровней, который включает в себя возможность сохранения и загрузку уровней, несколько режимов редактирования, возможности добавления и удаления строк и столбцов карты в нужных пользователю позициях, возможность тестирования уровня;
– возможность получения справки по каждому компоненту системы.

В процессе разработки были приобретены навыки создания элементов интерфейса вручную с помощью графического редактора. Разработка программного средства потребовала использования функций WinAPI и объектно-ориентированных возможностей языка Delphi.

Архитектура программы является модульной и гибкой, что обеспечивает потенциал для развития. В дальнейшем планируется портировать программу на платформу Linux, перейдя от языка Delphi к языку Object Pascal и Lazarus IDE. Vim-функционал редактора планируется расширить, путём добавления возможности отмены, счётчика команд и других особенностей редактора Vim. Большое поле для развития обеспечивает возможность загружать уровни из интернета, которые были написаны сторонними разработчиками. Планируется создание редактора профилей уровня, для возможности при загрузке уровня выбрать, какой символ какую текстуру обозначает. Это даст возможность использовать практически любой уровень Sokoban, написанный даже человеком, который о данной программе не знает.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Delphi справочник (электронный ресурс). – Электронные данные. – Режим доступа: http://delphibasics.ru/Class.php.

2. Серебряная, Л.В. Марина, И.М. Структуры и алгоритмы обработки данных: учебно-метод. пособие для студ. спец. «Программное обеспечение информационных технологий» всех форм обуч. / Л. В. Серебряная, И. М. Марина. – Минск: БГУИР, 2012. – 49 с. : ил.

3. Графические возможности Делфи (электронный ресурс). – Электронные данные. – Режим доступа: http://delphi7.gym5cheb.ru.

4. В.Фаронов Программирование в Delphi 6 (учебный курс). – Москва: Издатель Молгачева С.В., 2001. – 672 с., ил.

5. Глухова Л.А., Фадеева Е.П., Фадеева Е.Е. Основы алгоритмизации и программирования: Лаб. практикум для студ. спец. I-40 01 01 «Программное обеспечение информационных технологий» дневной формы обучения. В 4 ч. Ч.3. – Минск: БГУИР, 2007. – 51 с.;

ПРИЛОЖЕНИЕ А
Исходный код программы

Игровая форма:

```
unit UnitGame;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, Menus, UnitSoko, StdCtrls, Buttons;

type
  TFormGame = class(TForm)
    MainMenu: TMainMenu;
    NGame: TMenuItem;
    NLoad: TMenuItem;
    DrawSoko: TDrawGrid;
    NHelp: TMenuItem;
    NDo: TMenuItem;
    NCancelMove: TMenuItem;
    N3: TMenuItem;
    NExit: TMenuItem;
    procedure NLoadClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormKeyUp(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure FormPaint(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure RestoreMove;
    procedure NExitClick(Sender: TObject);
    procedure NCancelMoveClick(Sender: TObject);
    procedure DrawCaption;
  private
    { Private declarations }
  public
    { Public declarations }
    procedure WMMoving(var Msg: TWMMoving); message WM_MOVING;
  end;

var
  FormGame: TFormGame;
  Map: TMap;
  IsMapLoaded: Boolean;
  X,Y: Integer; // Позиции игрока
  PoolCount: Integer; //Кол-во лунок
  Steps, Pushes: Integer;
  MoveStack: TMoveStackClass;

implementation

{$R *.dfm}

procedure TFormGame.NLoadClick(Sender: TObject);
var
   Path: string;
begin
   if GetInPath(Path, Self) then
   begin
      if GetMapInFile(Path, Map) then
      begin
         MakeBorder(Map, DrawSoko, FormGame);
         DrawGrid(DrawSoko, Map);
         GetVariables(Map, X, Y, PoolCount);
         IsMapLoaded := True;
         Steps := 0;
         Pushes := 0;
         DrawCaption;
      end;
   end;
end;
```

```
procedure TFormGame.FormActivate(Sender: TObject);
begin
   IsMapLoaded := False;
   Steps := 0;
   Pushes := 0;
end;

procedure DrawOrPool(aGrid: TDrawGrid; x,y: Integer);
{В зависимости от состояния игрока ставит на его пред. поле лунку или дорогу}
begin
   if Map[y,x] = CPlayerChar then
   begin
      Map[y,x] := CDrawChar;
      DrawCell(aGrid, x, y, CDrawPath);
   end
   else
   begin
      Map[y,x] := CPoolChar;
      DrawCell(aGrid, x, y, CPoolPath);
   end;
end;

procedure TFormGame.RestoreMove;
var
   Move: TMove;
   Path: string;
begin
   if MoveStack.pHead <> nil then
   begin
      Move := MoveStack.Pop;
      if MoveStack.pHead = nil then
         NCancelMove.Enabled := False;
      x := x - ord(Move.PlayerDX);
      y := y - ord(Move.PlayerDY);
      Map[y,x] := Move.Char1;
      Map[y + Ord(Move.PlayerDY), x + ord(Move.PlayerDX)] := Move.Char2;
      Map[y+2*Ord(Move.PlayerDY), x+2*Ord(Move.PlayerDX)] := Move.Char3;
      DrawPlayer(DrawSoko, x, y, Move.PlayerDX, Move.PlayerDY);
      Path := GetPathFromChar(Move.Char2);
      DrawCell(DrawSoko, x + ord(Move.PlayerDX), y + Ord(Move.PlayerDY), Path);
      Path := GetPathFromChar(Move.Char3);
      DrawCell(DrawSoko, x + 2*ord(Move.PlayerDX), y + 2*Ord(Move.PlayerDY), Path);
      if Move.IsPushedBox then
      begin
         Dec(Pushes);
         Dec(PoolCount, Move.PoolChange);
      end;
      Dec(Steps);
      DrawCaption;
   end;
end;

procedure TFormGame.FormKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
var
   dx: TDxMove;
   dy: TDyMove;
   Move: TMove;
begin
   if IsMapLoaded and InitilizeInput(Key, dx, dy) then
   begin
      if (Map[y+Ord(dy), x+Ord(dx)] = CDrawChar) or  // Переход на дорогу
         (Map[y+Ord(dy), x+Ord(dx)] = CPoolChar) then // Переход на лунку
      begin
         with Move do
         begin
            PlayerDX := dx;
            PlayerDY := dy;
            Char1 := Map[y,x];
            Char2 := Map[y + Ord(dy), x + ord(dx)];
            Char3 := Map[y+2*Ord(dy), x+2*Ord(dx)];
            IsPushedBox := False;
            PoolChange := 0;
         end;
         MoveStack.Push(Move);
         NCancelMove.Enabled := True;
         if (Map[y+Ord(dy), x+Ord(dx)] = CDrawChar) then //Перешли на дорогу
            Map[y+Ord(dy), x+Ord(dx)] := CPlayerChar
```

```
         else //Перешли на лунку
            Map[y+Ord(dy), x+Ord(dx)] := CPlayerInPoolChar;
         DrawOrPool(DrawSoko, x, y);
         x := x + Ord(dx);
         y := y + Ord(dy);
         Inc(Steps);
         DrawPlayer(DrawSoko, x, y, dx, dy);
      end
      else if ((Map[y+Ord(dy), x+Ord(dx)] = CBoxChar) or
      (Map[y+Ord(dy), x+Ord(dx)] = CBoxInPoolChar)) and
      ((Map[y+2*Ord(dy), x+2*Ord(dx)] = CDrawChar) or
      (Map[y+2*Ord(dy), x+2*Ord(dx)] = CPoolChar)) then // Двигаем коробку
      begin
         with Move do
         begin
            PlayerDX := dx;
            PlayerDY := dy;
            Char1 := Map[y,x];
            Char2 := Map[y + Ord(dy), x + ord(dx)];
            Char3 := Map[y+2*Ord(dy), x+2*Ord(dx)];
            IsPushedBox := True;
            PoolChange := 0;
         end;
         if (Map[y+2*Ord(dy), x+2*Ord(dx)] = CDrawChar) then // на дорогу
         begin
            Map[y+2*Ord(dy), x+2*Ord(dx)] := CBoxChar;
            DrawCell(DrawSoko, x + 2*Ord(dx), y + 2*Ord(dy), CBoxPath);
         end;
         if (Map[y+2*Ord(dy), x+2*Ord(dx)] = CPoolChar) then //на лунку
         begin
            Map[y+2*Ord(dy), x+2*Ord(dx)] := CBoxInPoolChar;
            DrawCell(DrawSoko, x + 2*Ord(dx), y + 2*Ord(dy), CBoxInPoolPath);
         end;
         DrawOrPool(DrawSoko, x, y); // вместо игрока ставим лунку или дорогу
         if (Map[y+Ord(dy), x+Ord(dx)] = CBoxChar) then
            Map[y+Ord(dy), x+Ord(dx)] := CPlayerChar
         else // На место коробки ставим игрока
            Map[y+Ord(dy), x+Ord(dx)] := CPlayerInPoolChar;
         //DrawCell(DrawSoko, x + Ord(dx), y + Ord(dy), CPlayerPath);
         DrawPlayer(DrawSoko, x + Ord(dx), y + Ord(dy), dx, dy);
         // Увеличиваем счёт задвинутых коробок
         if (Map[y+Ord(dy), x+Ord(dx)] = CPlayerChar) and
            (Map[y+2*Ord(dy), x+2*Ord(dx)] = CBoxInPoolChar) then
            begin
               Dec(PoolCount);
               Move.PoolChange := -1;
            end;
         // Уменьшаем счёт задвинутых коробок
         if (Map[y+Ord(dy), x+Ord(dx)] = CPlayerInPoolChar) and
            (Map[y+2*Ord(dy), x+2*Ord(dx)] = CBoxChar) then
            begin
               Inc(PoolCount);
               Move.PoolChange := 1;
            end;
         MoveStack.Push(Move);
         NCancelMove.Enabled := True;
         x := x + Ord(dx);
         y := y + Ord(dy);
         Inc(Steps);
         Inc(Pushes);
         // Проверка на победу
         if PoolCount = 0 then
            MessageBox(Handle, PChar('Победа! Шагов - '  + IntToStr(Steps)), '', MB_OK);
      end;
      DrawCaption;
   end
   else if IsMapLoaded then
   begin
      case Key of
         Ord('U'):
         begin
            RestoreMove;
         end;
         Ord('Q'):
            FormGame.Close;
      end;
   end;
end;
```

```
procedure TFormGame.FormPaint(Sender: TObject);
begin
   DrawGrid(DrawSoko, Map);
end;

procedure TFormGame.DrawCaption;
begin
   FormGame.Caption := 'Шагов: ' + IntToStr(Steps) + ' Перестановок: ' +
      IntToStr(Pushes);
end;

procedure TFormGame.WMMoving(var Msg: TWMMoving);
var
  workArea: TRect;
begin
  workArea := Screen.WorkareaRect;
  with Msg.DragRect^ do
  begin
    if Left < workArea.Left then
      OffsetRect(Msg.DragRect^, workArea.Left-Left, 0) ;
    if Top < workArea.Top then
      OffsetRect(Msg.DragRect^, 0, workArea.Top-Top) ;
    if Right > workArea.Right then
      OffsetRect(Msg.DragRect^, workArea.Right-Right, 0) ;
    if Bottom > workArea.Bottom then
      OffsetRect(Msg.DragRect^, 0, workArea.Bottom-Bottom) ;
  end;
  inherited;
end;

procedure TFormGame.FormCreate(Sender: TObject);
begin
   MoveStack := TMoveStackClass.Create(CStackSize);
   NCancelMove.Enabled := False;
end;

procedure TFormGame.FormDestroy(Sender: TObject);
begin
   FreeAndNil(MoveStack);
end;

procedure TFormGame.NExitClick(Sender: TObject);
begin
   FormGame.Close;
end;

procedure TFormGame.NCancelMoveClick(Sender: TObject);
begin
   RestoreMove;
end;

end.
```

Вспомогательный модуль:

```
unit UnitSoko;

interface
   uses WinTypes, Classes, Dialogs, Grids, Graphics, Forms;
   type
      TMap = array of string;
      TDxMove = (dxLeft = -1, dxRigth = 1, dxNone = 0);
      TDyMove = (dyDown = 1, dyUp = -1, dyNone = 0);
      TMove = packed record
         PlayerDX: TDxMove;
         PlayerDY: TDyMove;
         Char1, Char2, Char3: Char;
         PoolChange: Integer;
         IsPushedBox: Boolean;
      end;
      PMoveStack = ^TMoveStack;
      TMoveStack = record
         Move: TMove;
         pNext: PMoveStack;
```

```
      pPrev: PMoveStack;
    end;
    TMoveStackClass = class
      pHead: PMoveStack;
      pTop: PMoveStack;
      Count: Integer;
      MaxCount: Integer;
      constructor Create(const aMaxCount: Integer);
      function Pop: TMove;
      procedure Push(const aMove: TMove);
    end;
    TRecord = record
      Moves: Integer;
      Pushes: Integer;
      Time: string[8];
    end;
  const
    CWallChar = '#'; CWallPath = 'wall.bmp';
    CPlayerChar = '!'; CPlayerPath = 'player.bmp';
    CPoolChar = '+'; CPoolPath = 'pool.bmp';
    CBoxChar = '%'; CBoxPath = 'box.bmp';
    CDrawChar = '.'; CDrawPath = 'draw.bmp';
    CNullChar = '/';
    CPlayerInPoolChar = '?';
    CBoxInPoolChar = '@'; CBoxInPoolPath = 'boxinpool.bmp';
    // animation player
    CPlayerDownPath = 'playerdown.bmp';
    CPlayerUpPath = 'playerup.bmp';
    CPlayerLeftPath = 'playerleft.bmp';
    CPlayerRightPath = 'playerright.bmp';

    CDefTexturesPath = 'Textures/';

    var
      PropSize: Integer = 64;
      StackSize: Integer = 70;

  function GetMapInFile(aPath: string; var aMap: TMap): Boolean;
  function GetInPath(var aPath: string; aParrent: TComponent): Boolean;
  procedure MakeBorder(aMap: TMap; aGrid: TDrawGrid; aForm: TForm);
  procedure DrawGrid(aGrid: TDrawGrid; aMap: TMap);
  function InitilizeInput(Key: Word; var dx: TDxMove; var dy: TDyMove): Boolean;
  procedure GetVariables(Map: TMap; var x,y: Integer; var PoolCount: Integer);
  procedure DrawCell(aGrid: TDrawGrid; aXCoord, aYCoord: Integer;
    aChar: char);
  procedure DrawPlayer(aGrid: TDrawGrid; aXCoord, aYCoord: Integer;
    aDx: TDxMove; aDy: TDyMove);
  function GetPathFromChar(const aChar: Char): string;
  function GetRecord(aPath: string; var aRec: TRecord): Boolean;
  function SaveRecord(aPath: string; const aRec: TRecord): Boolean;
  function GetPropSize(aXProp, aYProp: Integer): Boolean;
implementation

uses SysUtils;

const
  CAllowArr: array[0..7] of Char = (CWallChar, CPlayerChar, CPoolChar, CBoxChar,
    CDrawChar, CNullChar, CPlayerInPoolChar, CBoxInPoolChar);

  constructor TMoveStackClass.Create(const aMaxCount: Integer);
  begin
    MaxCount := aMaxCount;
  end;

  function TMoveStackClass.Pop: TMove;
  var
    Temp: PMoveStack;
  begin
    Temp := pHead;
    Result := pHead^.Move;
    pHead := pHead^.pNext;
    Dispose(Temp);
    Dec(Count);
  end;

  procedure TMoveStackClass.Push(const aMove: TMove);
  var
    Temp: PMoveStack;
```

```
   Temp2: PMoveStack;
begin
   New(Temp);
   Temp^.Move := aMove;
   Temp^.pNext := pHead;
   Temp^.pPrev := nil;
   if Count > 0 then
      pHead^.pPrev := Temp;
   pHead := Temp;
   if Count = MaxCount then
   begin
      Temp2 := pTop;
      pTop := pTop^.pPrev;
      pTop^.pNext := nil;
      Dispose(Temp2)
   end
   else
   begin
      Inc(Count);
      if Count = 1 then
         pTop := pHead
      else if Count = 2 then
         pTop^.pPrev := pHead;
   end;
end;

function IsCorrectField(aMap: TMap): Boolean;
var
   IsCorrect: Boolean;
const
   CVisited = 'v';

procedure TryMove(aX, aY: Integer);
begin
   if not (IsCorrect) or (aMap[aY, aX] = CVisited) then
      Exit;
   if not (aMap[aY, aX] = CWallChar) then
   begin
      if (aX = 1) or (aY = 0) or (aX = Length(aMap[aY])) or (aY = High(aMap)) then
         IsCorrect := False;
      aMap[aY, aX] := CVisited;
      TryMove(aX + 1, aY);
      TryMove(aX, aY - 1);
      TryMove(aX - 1, aY);
      TryMove(aX, aY  + 1);
   end;
end;

label
   ExitLoop;
var
   PlayerX, PlayerY: Integer;
   i,j: Integer;
begin
   IsCorrect := True;
   for i := 0 to High(aMap) do
      for j := 0 to Length(aMap[i]) do
         if (aMap[i][j] = CPlayerChar) or (aMap[i][j] = CplayerInPoolChar) then
         begin
            TryMove(j, i);
            goto ExitLoop;
         end;
   ExitLoop:
   Result := IsCorrect;
end;

function GetMapInFile(aPath: string; var aMap: TMap) : Boolean;
var
   InFile: TextFile;
   i,j,k: Integer;
   IsCorrect: Boolean;
   Temp: string;
   PlayerPosCount: Integer;
begin
   AssignFile(InFile, aPath);
   Reset(InFile);
   i := 0;
   IsCorrect := True;
```

```
   PlayerPosCount := 0;
   while (not EOF(InFile)) and (IsCorrect) do
   begin
      try
         Readln(InFile, Temp);
         for j := 1 to Length(Temp) do
         begin
            if (Temp[j] = CPlayerChar) or (Temp[j] = CPlayerInPoolChar) then
               Inc(PlayerPosCount);
            IsCorrect := False;
            for k := 0 to High(CAllowArr) do
            begin
               if Temp[j] = CAllowArr[k] then
               begin
                  IsCorrect := True;
                  break;
               end;
            end;
            if not IsCorrect then break;
         end;
         Inc(i);
      except
         IsCorrect := False;
      end;
   end;
   Close(InFile);
   if (i = 0) or (PlayerPosCount <> 1) then
      IsCorrect := False;
   if IsCorrect then
   begin
      SetLength(aMap, i);
      Reset(InFile);
      i := 0;
      while not EOF(InFile) do
      begin
         Readln(InFile, aMap[i]);
         Inc(i);
      end;
      Close(InFile);
   end;
   if not IsCorrectField(Copy(aMap)) then
      IsCorrect := False;
   GeTMapInFile := IsCorrect;
end;

function GetInPath(var aPath: string; aParrent: TComponent): Boolean;
var
   openFile: TOpenDialog;
begin
   openFile := TOpenDialog.Create(aParrent);
   openFile.Filter := '”ровни sokoban (*.skb)|*.skb';
   if openFile.Execute then
   begin
      aPath := openFile.FileName;
      Result := True;
   end
   else
      Result := False;
   openFile.Destroy;
end;

procedure MakeBorder(aMap: TMap; aGrid: TDrawGrid; aForm: TForm);
var
   MaxColCount: Integer;
   i: Integer;
begin
   MaxColCount := 0;
   for i := 0 to High(aMap) do
   begin
      if Length(aMap[i]) > MaxColCount then
         MaxColCount := Length(aMap[i]);
   end;
   if GetPropSize(Screen.WorkAreaWidth div MaxColCount,
      Screen.WorkAreaHeight div Length(aMap)) then
   begin
      aGrid.DefaultColWidth := PropSize;
      aGrid.DefaultRowHeight := PropSize;
      aGrid.RowCount := Length(aMap);
```

```
      aGrid.Height := Trunc((aGrid.RowCount) * (2 + PropSize));
      aForm.Height := aGrid.Height + 50;
      aGrid.ColCount := MaxColCount;
      aGrid.Width := Trunc((aGrid.ColCount) * (2 + PropSize));
      aForm.Width := aGrid.Width;
      aForm.Left := (Screen.WorkAreaWidth - aForm.Width) div 2;
      aForm.Top := (Screen.WorkAreaHeight - aForm.Height) div 2;
   end
end;

procedure DrawGrid(aGrid: TDrawGrid; aMap: TMap);
var
   i,j: Integer;
   Img: TBitmap;
begin
   SetCurrentDir(ExtractFilePath(Application.ExeName));
   Img := TBitmap.Create;
   for i := 0 to High(aMap) do
   begin
      for j := 1 to Length(aMap[i]) do
      begin
         if aMap[i,j] = CNullChar then
         begin
            aGrid.Canvas.Brush.Color := aGrid.Color;
            aGrid.Canvas.FillRect(aGrid.CellRect(j-1, i));
         end
         else
         begin
            Img.LoadFromFile(GetPathFromChar(aMap[i,j]));
            aGrid.Canvas.CopyRect(aGrid.CellRect(j-1,i), Img.Canvas,
               Rect(0,0,Img.Height,Img.Width))
         end;
      end;
   end;
   Img.Free;
end;

function InitilizeInput(Key: Word; var dx: TDxMove; var dy: TDyMove): Boolean;
begin
   Result := True;
   case Key of
      VK UP, Ord('K'):
      begin
         dx := dxNone;
         dy := dyUp;
      end;
      VK DOWN, Ord('J'):
      begin
         dx := dxNone;
         dy := dyDown;
      end;
      VK LEFT, Ord('H'):
      begin
         dx := dxLeft;
         dy := dyNone;
      end;
      VK RIGHT, Ord('L'):
      begin
         dx := dxRigth;
         dy := dyNone;
      end;
   else
      Result := False;
   end;
end;

procedure GetVariables(Map: TMap; var x,y: Integer; var PoolCount: Integer);
var
   i,j: Integer;
begin
   x := 0;
   y := 0;
   PoolCount := 0;
   for i := 0 to High(Map) do
      for j := 1 to Length(Map[i]) do
      begin
         if (Map[i,j] = CPlayerChar) or (Map[i,j] = CPlayerInPoolChar) then
         begin
```

```
            y := i;
            x := j;
         end;
         if Map[i,j] = CPoolChar then
            Inc(PoolCount);
      end;
end;

procedure DrawCell(aGrid: TDrawGrid; aXCoord, aYCoord: Integer;
   aChar: char);
var
   Img: TBitmap;
begin
   SetCurrentDir(ExtractFilePath(Application.ExeName));
   if aChar = CNullChar then
   begin
      aGrid.Canvas.Brush.Color := aGrid.Color;
      aGrid.Canvas.FillRect(aGrid.CellRect(aXCoord-1, aYCoord));
   end
   else
   begin
      Img := TBitmap.Create;
      Img.LoadFromFile(GetPathFromChar(aChar));
      aGrid.Canvas.CopyRect(aGrid.CellRect(aXCoord - 1, aYCoord), Img.Canvas,
         Rect(0,0,Img.Height,Img.Width));
      Img.Free;
   end;
end;

procedure DrawPlayer(aGrid: TDrawGrid; aXCoord, aYCoord: Integer;
   aDx: TDxMove; aDy: TDyMove);
var
   Img: TBitmap;
   Path: string;
begin
   Img := TBitmap.Create;
   Path := CDefTexturesPath + IntToStr(PropSize) + '/';
   case aDx of
     dxLeft: Path := Path + CPlayerLeftPath;
     dxRigth: Path := Path + CPlayerRightPath;
     dxNone:
      case aDy of
        dyDown: Path := Path + CPlayerDownPath;
        dyUp: Path := Path + CPlayerUpPath;
      end;
   end;
   Img.LoadFromFile(Path);
   aGrid.Canvas.CopyRect(aGrid.CellRect(aXCoord - 1, aYCoord), Img.Canvas,
      Rect(0,0,Img.Height,Img.Width));
   Img.Free;
end;

function GetPathFromChar(const aChar: Char): string;  //TODO: to UnitSoko and review code
begin
   Result := CDefTexturesPath + IntToStr(PropSize) + '/';
   case aChar of
      CWallChar: Result := Result + CWallPath;
      CPlayerChar: Result := Result + CPlayerPath;
      CPlayerInPoolChar: Result := Result + CPlayerPath;
      CPoolChar: Result := Result + CPoolPath;
      CBoxChar: Result := Result + CBoxPath;
      CDrawChar: Result := Result + CDrawPath;
      CBoxInPoolChar: Result := Result + CBoxInPoolPath;
   end;
end;

function GetRecord(aPath: string; var aRec: TRecord): Boolean;
var
   InFile: file of TRecord;
begin
   AssignFile(InFile, aPath);
   try
      Reset(InFile);
      Read(InFile, aRec);
      Result := True;
   except
      Result := False;
   end;
```

```
      Close(InFile);
   end;

   function SaveRecord(aPath: string; const aRec: TRecord): Boolean;
   var
      OutFile: file of TRecord;
   begin
      AssignFile(OutFile, aPath);
      try
         Rewrite(OutFile);
         Write(OutFile, aRec);
         Close(OutFile);
         Result := True;
      except
         Result := False;
      end;
   end;

   function GetPropSize(aXProp, aYProp: Integer): Boolean;
   const
      CMaxSize = 64;
      CMinSize = 16;
   var
      i: Integer;
      Min: Integer;
   begin
      if aXProp < aYProp then
         Min := aXProp
      else
         Min := aYProp;
      i := CMaxSize;
      while (i >= CMinSize) and (Min div i = 0) do
         Dec(i, 16);
      if i > 0 then
         PropSize := i;
      if Min < CMinSize then
         GetPropSize := False
      else
         GetPropSize := True;
   end;
end.
```

Форма-редактор:

```
unit UnitEditor;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, Grids, UnitSoko, ExtCtrls, UnitSizeEdit, UnitGame;

type
  TFormEditor = class(TForm)
    MainMenu: TMainMenu;
    NFile: TMenuItem;
    NLoad: TMenuItem;
    NSave: TMenuItem;
    NNewLevel: TMenuItem;
    GridEdit: TDrawGrid;
    Timer: TTimer;
    procedure NLoadClick(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure GridEditDrawCell(Sender: TObject; ACol, ARow: Integer;
      Rect: TRect; State: TGridDrawState);
    procedure TimerTimer(Sender: TObject);
    procedure NewField(aColCount, aRowCount: Integer);
    procedure MakeMove(aChar: Char);
    procedure TransformController(aKey: Word; IsUpper: Boolean);
    procedure NextMove;
    procedure TransformBorderChar(aFrom, aTo: Char);
    procedure DeleteCol(aCol: Integer);
    procedure DeleteRow(aRow: Integer);
    procedure InsertCol(aCol: Integer);
    procedure InsertRow(aRow: Integer);
```

```
    procedure TestLevel;
    procedure NNewLevelClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure GridEditKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure NSaveClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormEditor: TFormEditor;

implementation

uses Types, UnitMenu;

{$R *.dfm}

type
   TMode = (mdOverwrite, mdInsert);

var
   Map: TMap;
   IsMapLoaded: Boolean;
   IsCoursourVisible: Boolean;
   Mode: TMode;
   PathSave: string;

const
   CDefRows = 8;
   CDefCols = 8;

function SaveMap(aPath: string): Boolean;
var
   OutFile: TextFile;
   i: Integer;
begin
   AssignFile(OutFile, aPath);
   Rewrite(OutFile);
   for i := 0 to High(Map) do
   begin
      Writeln(OutFile, Map[i]);
   end;
   Close(OutFile);
end;

procedure TFormEditor.TransformBorderChar(aFrom, aTo: Char);
var
   i,j: Integer;
begin
   for i := 1 to Length(Map[0]) do
   begin
      j := 0;
      while (j <= High(Map)) and (Map[j,i] = aFrom) do
      begin
         Map[j,i] := aTo;
         DrawCell(GridEdit, i, j, aTO);
         Inc(j);
      end;
      j := High(Map);
      while (j >= 0) and (Map[j,i] = aFrom) do
      begin
         Map[j,i] := aTo;
         DrawCell(GridEdit, i, j, aTo);
         Dec(j);
      end;
   end;
   for i := 0 to High(Map) do
   begin
      j := 1;
      while (j <= Length(Map[i])) and (Map[i,j] = aFrom) do
      begin
         Map[i,j] := aTo;
         DrawCell(GridEdit, j, i, aTO);
```

```
         Inc(j);
      end;
      j := Length(Map[i]);
      while (j >= 1) and (Map[i,j] = aFrom) do
      begin
         Map[i,j] := aTo;
         DrawCell(GridEdit, j, i, aTo);
         Dec(j);
      end;
   end;
end;

procedure AddNullChar(aMap: TMap);
var
   i,j: Integer;
   MaxLength: Integer;
begin
   MaxLength := Length(Map[0]);
   for i := 1 to High(Map) do
      if Length(Map[i]) > MaxLength then
         MaxLength := Length(Map[i]);
   for i := 0 to High(Map) do
      for j := Length(Map[i]) to MaxLength - 1 do
         Insert(CNullChar, Map[i], j + 1);
end;

procedure TFormEditor.NLoadClick(Sender: TObject);
var
   Path: string;
begin
   if GetInPath(Path, Self) then
   begin
      if GetMapInFile(Path, Map) then
      begin
         MakeBorder(Map, GridEdit, FormEditor);
         AddNullChar(Map);
      end
      else
         MessageBox(Handle, '”ровень некорректен, загрузка невозможна!',
            'ќшибка', MB_OK + MB_ICONERROR);
   end
   else
      MessageBox(Handle, '”ровень не загружен!',
         '¬нимание!', MB_OK + MB_ICONWARNING);
end;

procedure TFormEditor.FormPaint(Sender: TObject);
begin
   if not IsMapLoaded then
   begin
      Map := nil;
      NewField(CDefCols, CDefRows + 2);
      Mode := mdOverwrite;
      IsMapLoaded := True;
   end;
end;

procedure TFormEditor.GridEditDrawCell(Sender: TObject; ACol,
  ARow: Integer; Rect: TRect; State: TGridDrawState);
begin
   if (ACol = GridEdit.Col) and (ARow = GridEdit.Row) then
   begin
      if Mode = mdOverwrite then
         GridEdit.Canvas.Brush.Color := clGreen
      else
         GridEdit.Canvas.Brush.Color := clRed;
      GridEdit.Canvas.FillRect(Rect);
   end
   else
   begin
      DrawCell(GridEdit, ACol + 1, ARow, Map[ARow, ACol + 1]);
   end;
end;

procedure TFormEditor.TimerTimer(Sender: TObject);
begin
   with GridEdit do
   begin
```

```
      if IsCoursourVisible then
      begin
         if Mode = mdOverwrite then
            Canvas.Brush.Color := clGreen
         else
            Canvas.Brush.Color := clRed;
         Canvas.FillRect(CellRect(Col, Row));
      end
      else
         DrawCell(GridEdit, Col + 1, Row, Map[Row, Col + 1]);
   end;
   IsCoursourVisible := not IsCoursourVisible;
end;

procedure TFormEditor.NewField(aColCount, aRowCount: Integer);
var
   i,j: Integer;
begin
    if GetPropSize(Screen.WorkAreaWidth div aColCount,
      Screen.WorkAreaHeight div aRowCount) then
    begin
      SetLength(Map, aRowCount);
      for i := 0 to High(Map) do
      begin
         Map[i] := '';
         for j := 1 to aColCount do
            Insert(CDrawChar, Map[i], j);
      end;
    end
    else
       MessageBox(Handle, 'Ќшибка! —лишком большой размер уровн¤!', 'Error',
         MB_OK + MB_ICONERROR);
   MakeBorder(Map, GridEdit, FormEditor);
end;

procedure TFormEditor.NNewLevelClick(Sender: TObject);
begin
   FormSizeEdit := TFormSizeEdit.Create(Self);
   FormSizeEdit.EditHeight.Text := IntToStr(Length(Map));
   FormSizeEdit.EditWidth.Text := IntToStr(Length(Map[0]));
   FormSizeEdit.ShowModal;
end;

procedure TFormEditor.FormCreate(Sender: TObject);
begin
   IsMapLoaded := False;
   PathSave := '';
end;

procedure TFormEditor.GridEditKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
   if Key = Ord('Q') then
      FormEditor.Close;
   if Mode = MdInsert then
   begin
      TransformController(Key, ssShift in Shift);
   end;
   with GridEdit do
   begin
      case Key of
         Ord('H'):
            if Col > 0 then
               Col := Col - 1;
         Ord('L'):
            if Col < ColCount - 1 then
               Col := Col + 1;
         Ord('J'):
            if Row < RowCount - 1 then
               Row := Row + 1;
         Ord('K'):
            if Row > 0 then
               Row := Row - 1;
         Ord('B'):
            if ssShift in Shift then
               MakeMove(CBoxInPoolChar)
            else
               MakeMove(CBoxChar);
```

```
         Ord('P'):
            if ssShift in Shift then
               MakeMove(CPlayerInPoolChar)
            else
               MakeMove(CPlayerChar);
         Ord('G'):
            if ssShift in Shift then
               TransformBorderChar(CNullChar, CDrawChar)
            else
               MakeMove(CDrawChar);
         Ord('O'):
            MakeMove(CPoolChar);
         Ord('W'):
            MakeMove(CWallChar);
         Ord('N'):
            if ssShift in Shift then
               TransformBorderChar(CDrawChar, CNullChar)
            else
               MakeMove(CNullChar);
         Ord('M'), Ord('I'):
            if Mode = mdOverWrite then
               Mode := mdInsert
            else
               Mode := mdOverWrite;
         Ord('E'):
            TestLevel;
      end;
   end;
end;

procedure TFormEditor.MakeMove(aChar: Char);
begin
   with GridEdit do
   begin
      if Mode = mdOverWrite then
      begin
         Map[Row, Col + 1] := aChar;
         DrawCell(GridEdit, Col + 1, Row, Map[Row, Col + 1]);
      end
   end;
end;

procedure TFormEditor.NextMove;
begin
   with GridEdit do
   begin
      if Col < ColCount - 1 then
         Col := Col + 1
      else if Row < RowCount - 1 then
      begin
         Row := Row + 1;
         Col := 0;
      end;
   end;
end;

procedure TFormEditor.DeleteCol(aCol: Integer);
var
   i: Integer;
begin
   for i := 0 to High(Map) do
      Delete(Map[i], aCol + 1, 1);
   MakeBorder(Map, GridEdit, FormEditor);
end;

procedure TFormEditor.DeleteRow(aRow: Integer);
var
   i: Integer;
   NewMap: TMap;
begin
   SetLength(NewMap, Length(Map) - 1);
   for i := 0 to aRow - 1 do
      NewMap[i] := Map[i];
   for i := aRow + 1 to High(Map) do
      NewMap[i-1] := Map[i];
   Map := Copy(NewMap);
   MakeBorder(Map, GridEdit, FormEditor);
end;
```

```
procedure TFormEditor.InsertCol(aCol: Integer);
var
   i: Integer;
begin
   for i := 0 to High(Map) do
      Insert(CDrawChar, Map[i], aCol + 1);
   MakeBorder(Map, GridEdit, FormEditor);
end;

procedure TFormEditor.InsertRow(aRow: Integer);
var
   i: Integer;
   NewMap: TMap;
begin
   SetLength(NewMap, Length(Map) + 1);
   for i := 0 to aRow - 1 do
      NewMap[i] := Map[i];
   for i := 0 to Length(Map[0]) - 1 do
      Insert(CDrawChar, NewMap[aRow], i + 1);
   for i := aRow to High(Map) do
      NewMap[i + 1] := Map[i];
   Map := Copy(NewMap);
   MakeBorder(Map, GridEdit, FormEditor);
end;

procedure TFormEditor.TransformController(aKey: Word; IsUpper: Boolean);
begin
   if IsUpper then
   begin
      case aKey of
         ord('W'):
            InsertRow(0);
         ord('S'):
            InsertRow(GridEdit.RowCount);
         ord('A'):
            InsertCol(0);
         Ord('D'):
            InsertCol(GridEdit.ColCount);
         Ord('X'):
            DeleteCol(GridEdit.Col);
      end;
   end
   else
   begin
      with GridEdit do
      begin
         case aKey of
            ord('W'):
               InsertRow(Row);
            ord('S'):
               InsertRow(Row + 1);
            ord('A'):
               InsertCol(Col);
            Ord('D'):
               InsertCol(Col + 1);
            Ord('X'):
               DeleteRow(GridEdit.Row);
         end;
      end;
   end;
end;

procedure TFormEditor.FormClose(Sender: TObject; var Action: TCloseAction);
begin
   Action := caFree;
   FormMenu.Show;
end;

function GetOutPath(var aPath: string; aParrent: TComponent): Boolean;
var
   saveFile: TSaveDialog;
begin
   saveFile := TSaveDialog.Create(aParrent);
   saveFile.Filter := '”ровни sokoban (*.skb)|*.skb';
   if saveFile.Execute then
   begin
      aPath := saveFile.FileName;
```

```
      Result := True;
   end
   else
      Result := False;
   saveFile.Destroy;
end;

procedure TFormEditor.NSaveClick(Sender: TObject);
var
   Path: string;
   i: Integer;
   Ext: string;
begin
   if GetOutPath(Path, Self) then
   begin
      SaveMap(Path);
      PathSave := Path;
      Ext := ExtractFileExt(PathSave);
      Delete(PathSave, Pos(Ext, PathSave), Length(Ext));
   end
   else
      MessageBox(Handle, ''айл не выбран', '¬нимание!', MB_OK + MB_ICONWARNING);
end;


procedure TFormEditor.TestLevel;
begin
   if PathSave <> '' then
   begin
      SaveMap(PathSave + '.skb');
      UnitGame.Path := PathSave;
      LevelName := 'DEBUG';
      FormGame := TFormGame.Create(Self);
      FormGame.ShowModal;
   end
   else
      MessageBox(Handle, '—начала сохраните файл!','ќшибка', MB_OK + MB_ICONERROR);
end;

end.
```