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 (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**ПРОГРАММНОЕ СРЕДСТВО**

**«2048»**

БГУИР КП 1-40 01 01 020 ПЗ

Студент гр. 351003 Писарчик Е.А.

Руководитель Данилова Г.В.

Минск 2024

СОДЕРЖАНИЕ

[Введение 5](#_Toc164201609)

[1 Анализ предметной области 6](#_Toc164201610)

[1.1 Обзор аналогов 6](#_Toc164201611)

[1.2 Постановка задачи 8](#_Toc164201612)

[2 Проектирование программого средства 9](#_Toc164201613)

[2.1 Структура программы 9](#_Toc164201614)

[2.2 Проектирование интерфейса программного средства 9](#_Toc164201615)

[2.3 Проектирование функционала программного средства](#_Toc164201616) 12

[3 Разработка программного средства 15](#_Toc164201617)

[3.1 Игровой процесс 15](#_Toc164201618)

[3.2 Прорисовка материалов для игры 16](#_Toc164201619)

[3.3 Работа со списком игроков](#_Toc164201624) 17

[4 Тестирование программного средства 21](#_Toc164201625)

[5 Руководство пользователя](#_Toc164201626) 22

[5.1 Интерфейс программного средства](#_Toc164201627) 22

[5.2 Управление программным средством](#_Toc164201628) 24

[Заключение 26](#_Toc164201629)

[Список использованных источников 27](#_Toc164201630)

[Приложение A. Текст программы](#_Toc164201631) 28

# ВВЕДЕНИЕ

С самого начала существования компьютерных игр разработчики стремились создавать разнообразные развлечения, которые бы не только увлекали, но и вызывали интерес к мышлению и решению задач. В мире игр всегда было множество маленьких, но простых игр, которые пользовались популярностью благодаря своей простоте и уникальной механике. Такие игры, как тетрис, крестики-нолики, сапёр и многие другие, стали неотъемлемой частью компьютерной культуры и приложили руку к формированию игрового мышления.

В этом богатом разнообразии игр появилась и игра «2048», представляющая собой головоломку, которая, несмотря на свою простоту, обладает уникальной и притягательной механикой. «2048» – браузерная игра, написанная в 2014 году девятнадцатилетним итальянским разработчиком Габором Вассерманном на JavaScript. Игра по своей концепции является чем-то средним между тетрисом и «пятнашками».

Цель игры «2048» заключается в том, чтобы достичь числа 2048, объединяя плитки с одинаковыми числовыми значениями на игровом поле. Но это не так просто, как кажется на первый взгляд. Эта задача требует не только реакции и ловкости, но и аналитического мышления и планирования.

Влияние игры «2048» на мышление игроков неоспоримо. Постоянно анализируя текущее положение на игровом поле и пытаясь предугадать последствия каждого своего хода, игроки развивают логическое мышление. Они учатся видеть не только текущую ситуацию, но и предугадывать возможные развития событий впереди. Игра требует от игрока планировать свои действия на несколько ходов вперёд, что способствует развитию у игроков навыков стратегического планирования и логического мышления.

Таким образом, игра «2048» не только предлагает увлекательное развлечение, но и является отличным тренажёром для ума, способствующим развитию логического и стратегического мышления у игроков.

Целью данного курсового проекта является разработка программного средства «2048».

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

На данный момент существует большое множество вариаций игры «2048». Каждая из них имеет свои особенности в реализации, достоинства и недостатки.

Оригинальный и первый вариант игры обрёл наибольшую популярность. В нём присутствует отображение счёта текущей игры и лучший счёт пользователя. Также можно выбрать язык интерфейса. Пролистав вниз, пользователь увидит инструкцию к игре.

Однако в данном программном средстве не представлены следующие функциональные возможности: выбор темы интерфейса и музыкальное сопровождение. Внешний вид данного приложения представлен на рисунке 1.1.

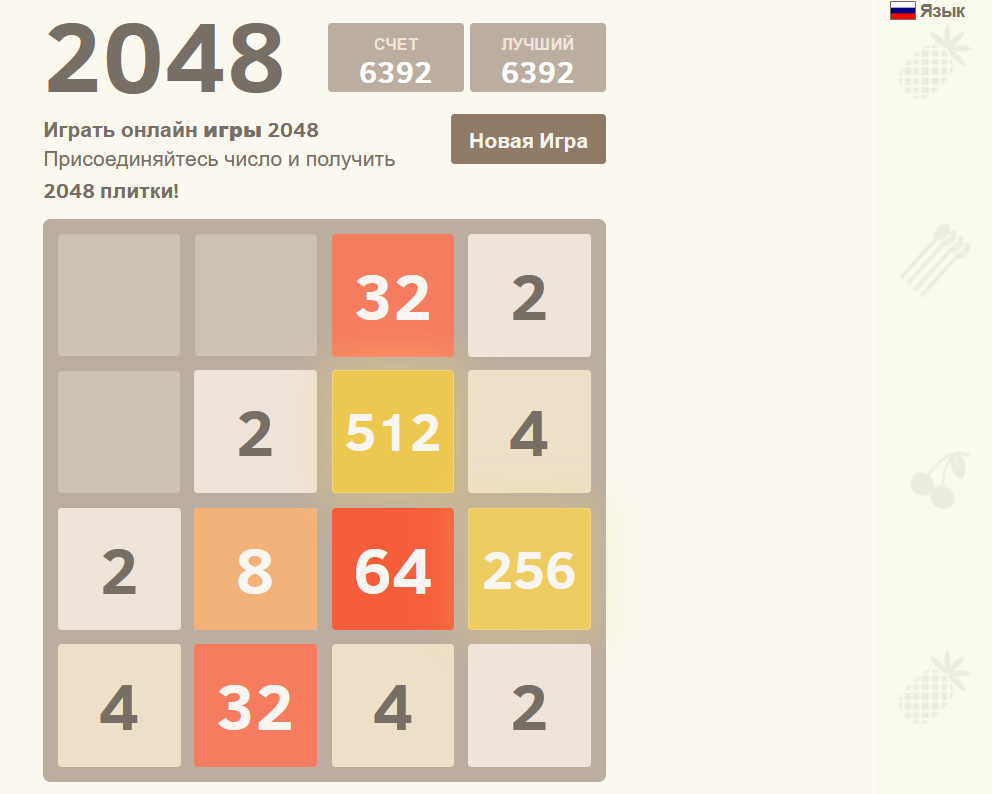


Рисунок 1.1 – Программное средство «2048»

Далее стоит рассмотреть реализацию «2048» от разработчика hello@2048.gg. Приложение реализует схожий игровой алгоритм, однако в нём присутствует расширенный набор языков, а представлена возможность выбора темы интерфейса. В то же время, в данном программном средстве отсутствует отображение и сохранение счёта. Внешний вид окна данного приложения представлен на рисунке 1.2.



Рисунок 1.2 – Программное средство «2048» от разработчика hello@2048.gg

Менее популярная, но достойная внимания версия игры от Yandex Games. Игра обладает лёгким и интуитивно понятным дизайном. Однако здесь и интерфейс, и механизм игры существенно отличаются – визуально игра имеет схожие черты с приложением «Тетрис», но базовая логика игры «2048» сохранена. На рисунке 1.3 представлен интерфейс данного программного продукта.

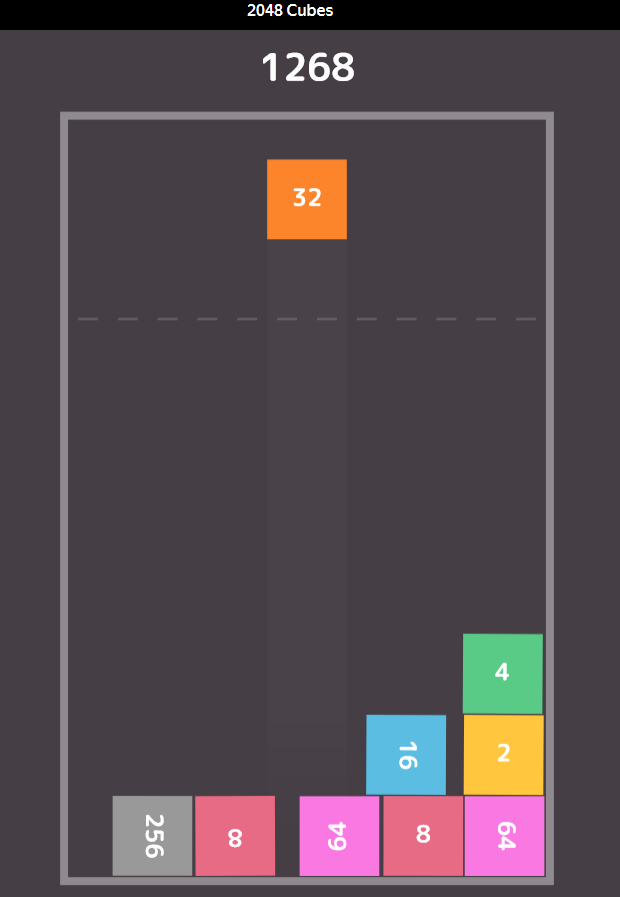


Рисунок 1.3 – Программное средство «2048» от разработчика Yandex Games

## Постановка задачи

В рамках данной курсовой работы планируется разработать игровое программное средство «2048» для платформы Windows. В процессе разработки должны быть реализованы базовые функции игры:

* отрисовка игрового поля и плиток;
* взаимодействие плиток;
* подсчёт баллов.

Планируется добавление такого функционала, как:

* подсчёт последних баллов игрока;
* сохранение данных игрока;
* формирование списка лидеров;
* открытие из файла и сохранение в файл таблицы лидеров.

Для разработки программного средства будет использоваться язык программирования Delphi и среда разработки Embarcadero RAD Studio 11 Architect.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

При разработке приложения будет использовано восемь модулей:

* MainUnit – модуль, в котором происходит процесс игры;
* GameBoardUnit – модуль, обеспечивающий отрисовку игрового поля, плиток и подсчёт баллов;
* TileUnit – модуль, обеспечивающий логику взаимодействия плиток;
* InstructionUnit – модуль, содержащий инструкцию для пользователя;
* UnitAboutMe – модуль, содержащий информацию о разработчике;
* UserNameUnit – модуль, обеспечивающий ввод имени игрока;
* LeaderBoardUnit – модуль, обеспечивающий отображение таблицы лидеров;
* ListGamersUnit – модуль, содержащий реализацию однонаправленного списка игроков.

## 2.2 Проектирование интерфейса программного средства

При разработке программного средства за основу будет взят оригинальный дизайн «2048».

2.2.1 Главное меню

Главное окно приложения будет содержать пять кнопок:

* «Новая игра» будет начинать игру заново, предварительно запросив имя игрока;
* «Рейтинг игроков», нажав которую, пользователь сможет ознакомиться с таблицей лидеров;
* «Об игре», нажав на которую, пользователь сможет ознакомиться с информацией о разработчике;
* «Инструкция», нажав на которую, пользователь сможет ознакомиться с правилами игры;
* «Выйти», которая будет позволять покинуть программное средство.

Макет главного окна приложения представлен на рисунке 2.1.



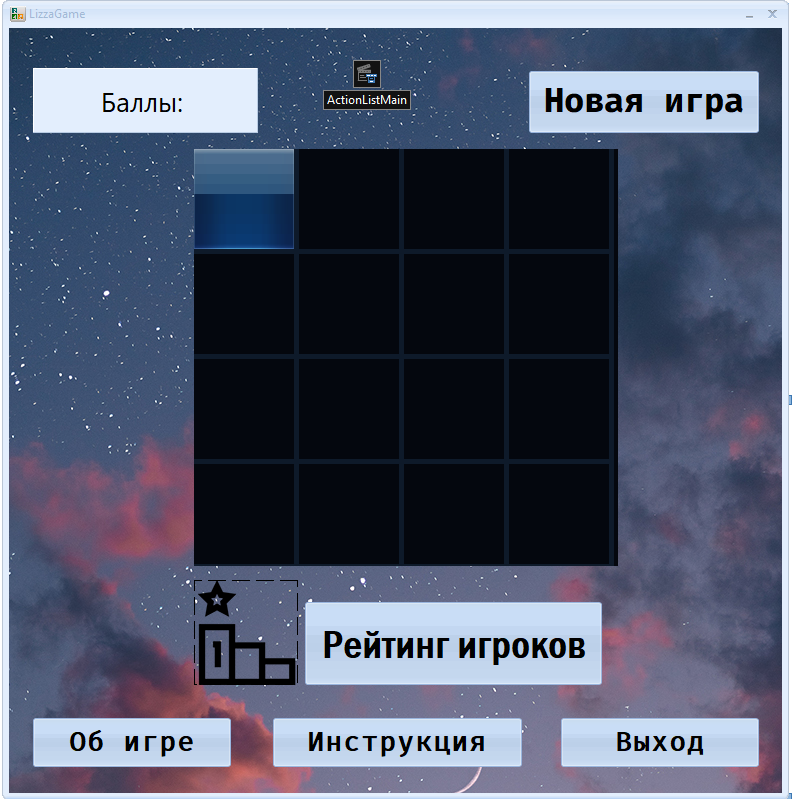


Рисунок 2.1 – Макет главного окна приложения

2.2.2 Окно ввода имени игрока

Окно ввода имени игрока будет содержать однострочное поле ввода, представленное компонентом TEdit и кнопку TButton. Макет окна ввода имени игрока представлен на рисунке 2.2.

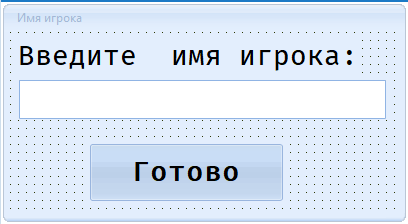


Рисунок 2.2 – Макет окна ввода имени игрока

2.2.3 Окно инструкции

Окно инструкции будет содержать компонеты TLabel и изображение, представленное компонентом TImage. Макет окна инструкции представлен на рисунке 2.3.



Рисунок 2.3 – Макет окна инструкции

2.2.4 Окно информации о разработчике

Окно информации о разработчике будет содержать информационные поля, представленные компонентами TLabel с данными разработчика. Макет окна информации о разработчике представлен на рисунке 2.4.

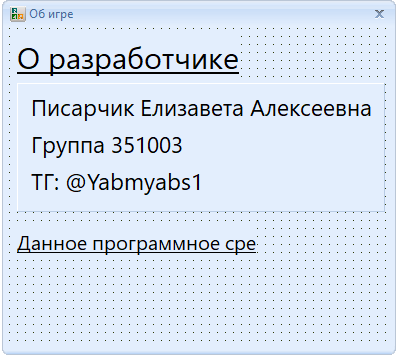


Рисунок 2.4 – Макет окна информации о разработчике

2.2.5 Окно таблицы лидеров

Окно таблицы лидеров будет содержать таблицу TStringGrid и кнопку TButton. В таблице будет представлена информация об игроках. Макет окна таблицы лидеров представлен на рисунке 2.5.

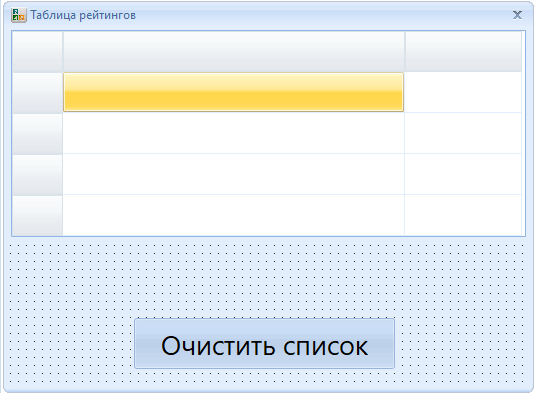


Рисунок 2.5 – Макет окна таблицы лидеров

## 2.3 Проектирование функционала программного средства

Разработка алгоритмов является ключевой фазой в проектировании программного средства. Игровое программное средство «2048» должно предоставлять пользователю такой минимальный функуционал, как:

* отрисовка игрового поля и плиток;
* взаимодействие плиток;
* подсчёт баллов.

2.3.1 Отрисовка игрового поля и плиток

Отрисовка игрового поля и плиток может быть реализована с использованием компонента TStringGrid, который представляет собой таблицу.

Изначально поле заполняется пустыми плитками. Затем случайным образом определяется положение двух плиток, и инициализируются соответствующие поля таблицы. От значения числа на плитке зависит цвет её раскраски. После каждого шага игры на поле добавляется в случайном месте новая плитка, причём её значением с вероятностью 80% будет двойка, в оставшихся случаях – четвёрка.

Блок-схема алгоритма процедуры GameBoardGridDrawCell приведена на рисунке 2.6.



Рисунок 2.6 – Блок-схема процедуры GameBoardGridDrawCell

2.3.2 Взаимодействие плиток

Взаимодействие плиток опирается на принцип объединения. При соединении двух плиток с одинаковыми числовыми значениями образуется одна плитка с значением, равным сумме исходных. Необходимо проходить по всем плиткам и проверять взаимодействие между ними. Блок-схема алгоритма функции Swap приведена на рисунке 2.7.



Рисунок 2.7 – Блок-схема функции Swap

Граммотное проектирование проекта поможет в будущем избежать ошибок при разработке.

1. **РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

**3.1 Игровой процесс**

При разработке программного средства основной упор делался на игровую логику. Ниже будут рассмотрены основные подпрограммы для реализации игрового процесса.

**3.1.1** Заполнение полей таблицы

Прорисовка плиток происходит каждый шаг игрока. Состояние плиток берется из списка. Все плитки таблицы отрисовываются с помощью Canvas. В каждой плитке списка находится её значение, цвет, позиция и состояние. Код процедуры Create приведён ниже.

Constructor TTile.Create(Const ATile: TTile);

Begin

//определение значения степени двойки

FPower := ATile.FPower;

//определение положения плитки в таблице

FRow := ATile.FRow;

FCol := ATile.FCol;

//определение соседних плиток

FLeftTile := ATile.FLeftTile;

FRightTile := ATile.FRightTile;

FUpTile := ATile.FUpTile;

FDownTile := ATile.FDownTile;

//определение значения очков

FOnIncScore := ATile.FOnIncScore;

End;

**3.1.2** Смещение и объединение плиток

Направление смещения и объединения вычисляется по значению нажатой клавиши на клавиатуре. Если плитка крайняя на поле, она остаётся на месте, иначе она смещается по заданному направлению на позицию перед стоящей на её пути плиткой, или на край поля. Если плитка во время смещения встречает плитку с таким же значением, они объединяются в одну плитку и смещаются по заданному направлению. Код прописан в событии компонента TActionList. В качестве примера рассмотрим смещение плиток вниз. Код процедуры AlignDown приведён ниже.

Procedure TGameBoard.AlignDown();

//объявление переменной

Var

I: Integer;

Begin

//цикл проходит по всем плиткам из списка

For I := 0 To FTiles.Count - 1 Do

//вызов процедуры из класса TTile

FTiles[I].AlignDown();

End;

Procedure TTile.AlignDown();

Begin

If IsDownTile() Then

Begin

//рекурсивный вызов процедуры смещения плиток

RecursiveMoveDown();

//рекурсивный вызов процедуры объединения плиток

RecursiveMergeDown();

//рекурсивный вызов процедуры смещения плиток

RecursiveMoveDown();

End;

End;

**3.2 Прорисовка материалов для игры**

Одними из важных процедур программы являются те, что связаны с прорисовкой текущего состояния поля, информационной части об игре. В основу работы с отображением был добавлен компонент TStringGrid.

**3.2.1** Отрисовка одной плитки игрового поля

Данная функция довольна важна для визуала игры. Код процедуры GameBoardGridDrawCell приведён ниже.

Procedure TLizzaGame.GameBoardGridDrawCell(Sender: TObject; ACol, ARow: Integer;

ARect: TRect; State: TGridDrawState);

Var

CellCenter: TPoint;

TextSize: TSize;

Tile: TTile;

Begin

//инициализация текущей плитки

Tile := GameBoard.Tile[ARow + 1, ACol + 1];

//отрисовка прямоугольника

GameBoardGrid.Canvas.FillRect(ARect);

GameBoardGrid.Canvas.Brush.Color := Tile.Color;

//задание свойств текста игрового поля

GameBoardGrid.Canvas.Font.Color := ClBlack;

GameBoardGrid.Canvas.Font.Style := [FsBold];

GameBoardGrid.Canvas.Font.Size := 22;

GameBoardGrid.Canvas.Font.Name := 'Tahoma';

//определение размера шрифта текста на плитке

TextSize := GameBoardGrid.Canvas.TextExtent(Tile.Caption);

//вычисление координат центра плитки

CellCenter.Y := (ARect.Bottom - ARect.Top) Div 2 + ARect.Top;

CellCenter.X := (ARect.Right - ARect.Left) Div 2 + ARect.Left;

//отрисовка текста на плитке

GameBoardGrid.Canvas.TextRect(ARect, CellCenter.X - TextSize.Cx Div 2,

CellCenter.Y - TextSize.Cy Div 2, Tile.Caption);

End;

Данная процедура задаёт полю таблицы TStringGrid свойства, соответствующие данной плитке.

**3.2.2** Процедура инициализации всего игрового поля

Переинициализализация всего игрового поля происходит после каждого нажатия пользователем клавиши стрелок. Код процедуры UpdateScreen приведён ниже.

Procedure TLizzaGame.UpdateScreen();

Var

Row: Integer;

Col: Integer;

Begin

//проход по всем строкам таблицы

For Row := 0 To GameBoardGrid.RowCount - 1 Do

//проход по всем столбцам таблицы

For Col := 0 To GameBoardGrid.ColCount - 1 Do

//инициализация

GameBoardGrid.Cells[Col, Row] := GameBoard.Tile[Row+1, Col+1].Caption;

//обновление информационного поля с текущим количеством баллов

PanelPoints.Caption := 'Баллы: ' + IntToStr(GameBoard().Score);

End;



## Работа со списком игроков

Работа со списком игроков включает в себя использование однонаправленного списка. Необходимо разработать процедуры:

* добавления;
* сортировки;
* сравнения;
* сохранения в файл;
* загрузки из файла;
* очищения.

**3.3.1** Добавление нового элемента в список

Список игроков представляет собой однонаправленный список. Данная структура данных была выбрана по той причине, чтобы более гибко добавлять игроков в список. Для того чтобы обновлять данные, было принято добавлять информацию о новом игроке в конец списка, а затем сортировать пузырьковым методом. Код процедуры AddUserByInfo приведён ниже.

Procedure AddUserByInfo(Points: Integer; Name: String);

Var

NewUser: TUser;

Current: PNodeUser;

Begin

//проверка существования игрока с таким именем

//смещение указателя на начало списка

Current := HeadUser;

If HeadUser <> Nil Then

Begin

//выполение цикла до тех пор, пока указатель не равен нулевому значению или

не найден игрок с таким именем

While (Current <> Nil) And (Current^.User.Name <> Name) Do

//смещение указателя к следующему элементу списка

Current := Current^.Next;

End;

//проверка, остался ли указатель равен указателю на начало списка, или стал ли

равен нулевому значению

If (Current = HeadUser) Or (Current = Nil) Then

Begin

//инициализация переменной нового игрока

NewUser.Name := Name;

NewUser.Points := Points;

//вызов процедуры добаления нового игрока в конец списка

AddUser(NewUser);

End

Else

//изменение значения баллов текущего игрока в списке

Current^.User.Points := Points;

End;

* + 1. Очищение списка

Важным аспектом использования динамической памяти является очищение. Для очищения списка необходимо пройтись по всему списку и освободить память для каждого элемента и присвоить указателю на начало списка нулевое значение. Код процедуры FreeMemory приведён ниже.

Procedure FreeMemory();

Var

Temp: PNodeUser;

Begin

//смещение указателя на начало списка

Temp := HeadUser;

//выполение цикла до тех пор, пока указатель не равен нулевому значению

While HeadUser <> Nil Do

Begin

Temp := HeadUser;

HeadUser := Temp^.Next;

//освобождение динамической памяти

Dispose(Temp);

End;

//сброс значения количества элементов в списке

UserCount := 0;

End;

* + 1. Сортировка списка

Сортировка списка игроков по невозрастанию количества баллов происходит методом пузырька. Такой метод был выбран из-за особенностей реализации однонаправленного списка, так как невозможно обратиться к предыдущему элементу. Код процедуры SortUserList приведён ниже.

Procedure SortUserList();

Var

I, J: Integer;

Temp: TUser;

Current, Next: PNodeUser;

Begin

//смещение указателя на начало списка

Current := HeadUser^.Next;

//первый цикл прохода по списку

While Current <> Nil Do

Begin

Next := Current^.Next;

//второй цикл прохода по списку

While Next <> Nil Do

Begin

//сравнение баллов соседних элементов списка

If Current^.User.Points < Next^.User.Points Then

Begin

//обмен значений элементов списка

Temp := Current^.User;

Current^.User := Next^.User;

Next^.User := Temp;

End;

//смещение указателя к следующему элементу списка

Next := Next^.Next;

End;

//смещение указателя к следующему элементу списка

Current := Current^.Next;

End;

End;

**3.3.4** Сохранение в файл

Процедура сохранения в типизированный файл проходит по всему списку игроков и записывает информацию о каждом элементе в файловую переменную. Код процедуры SaveUserListToFile приведён ниже.

Procedure SaveUserListToFile();

Var

CurUser: PNodeUser;

F: File Of TUser;

Begin

//инициализация файловой переменной

Assign(F, PATH);

//установка указателя файла в начало

Rewrite(F);

If HeadUser <> Nil Then

Begin

//смещение указателя на начало списка

CurUser := HeadUser^.Next;

While CurUser <> Nil Do

Begin

//запись данных об игроке в файл

Write(F, CurUser^.User);

//смещение указателя к следующему элементу списка

CurUser := CurUser^.Next;

End;

End;

//закрытие файла

CloseFile(F);

End;

**3.3.5** Загрузка из файла

Процедура загрузки из файла читает данные об игроках из типизированного файла и загружает их в список игроков, параллельно выделяя динамическу память для записи новых игроков. Также в процедуре присутствует проверка существования файла для загрузки/сохранения. Если файла по заданному пути не существует, он создаётся, иначе перезаписывается уже существующий файл. Код процедуры LoadUserListFromFile приведён ниже.

Procedure LoadUserListFromFile();

Var

User: TUser;

F: File Of TUser;

Begin

//инициализация файловой переменной

Assign(F, PATH);

//проверка существования файла

If Not FileExists(PATH) Then

Begin

//перезапись файла

Rewrite(F);

//закрытие файла

CloseFile(F);

End;

//установка указателя файла в начало

Reset(F);

//чтение данных из файла до конца файла

While Not EOf(F) Do

Begin

Read(F, User);

//добавление игрока в список

AddUser(User);

End;

//закрытие файла

CloseFile(F);

End;

Необходимые процедуры для работы со списком игроков описаны в модуле ListGamersUnit.

# 4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Основной проблемой было обращение по некорректному адресу в памяти при работе со списком игроков. При добавлении новой записи о игроке в список возникала ошибка. Проблема представлена на рисунке 4.1.

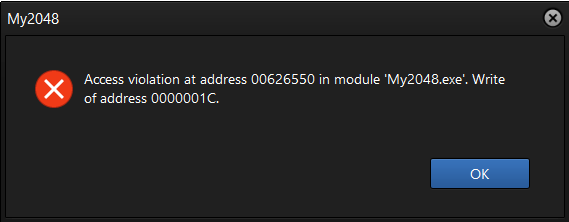


Рисунок 4.1 – Некорректное добавление новой записи в список

Возникала проблема при обращении к неопределённому значению адреса в памяти. Проблема была решена добавлением проверки значения указателя на начало списка в процедуру добавления нового игрока. Код процедуры добавления нового игрока представлен ниже.

Procedure AddUser(User: TUser);

Begin

//проверка значения указателя на начало списка

If HeadUser = Nil Then

Begin

//инициализация начала списка

CreateList(User);

End

Else

Begin

//выделение динамической памяти для нового элемента в конце списка

New(UserAddTail^.Next);

//инициализация нового элемента списка

UserAddTail^.Next^.User := User;

UserAddTail^.Next^.Next := Nil;

//смещение указателя последнего элемента списка

UserAddTail := UserAddTail^.Next;

End;

Inc(UserCount);

End;

Большинство проблем возникло из-за недочётов на стадии разработки программного средства, на стадии тестирования приложения все проблемы были исправлены.

**5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

**5.1 Интерфейс программного средства**

Внешний вид программных средств играет существенную роль в обеспечении удобства пользователей и качества самого средства.

5.1.1 Главное меню

Главное окно приложения состоит из пяти кнопок:

* «Новая игра» начинает игру заново, предварительно запросив имя игрока;
* «Рейтинг игроков», нажав которую, пользователь может ознакомиться с таблицей лидеров;
* «Об игре», нажав на которую, пользователь может ознакомиться с информацией о разработчике;
* «Инструкция», нажав на которую, пользователь может ознакомиться с правилами игры;
* «Выйти», которая позволяет покинуть программное средство.

Внешний вид главного окна приложения представлен на рисунке 5.1.





Рисунок 5.1 – Главное окно приложения

5.1.2 Окно ввода имени игрока

Окно ввода имени игрока содержит однострочное поле ввода и кнопку. Окно ввода имени игрока представлено на рисунке 5.2.

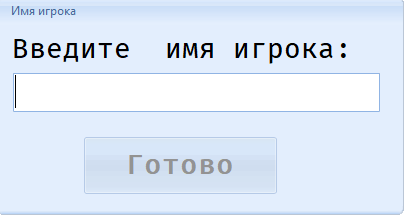


Рисунок 5.2 – Окно ввода имени игрока

5.1.3 Окно инструкции

Окно инструкции содержит правила игры. Окно инструкции представлено на рисунке 5.3.

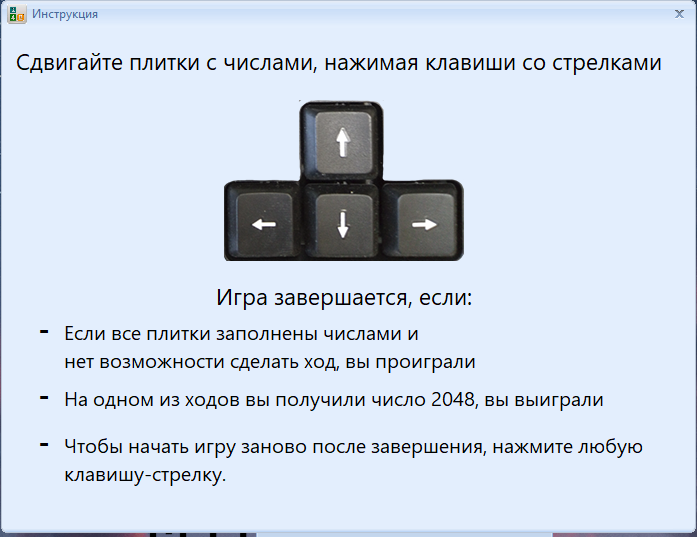


Рисунок 5.3 – Окно инструкции

5.1.4 Окно информации о разработчике

Окно информации о разработчике содержит информацию о разработчике. Окно информации о разработчике представлено на рисунке 5.4.

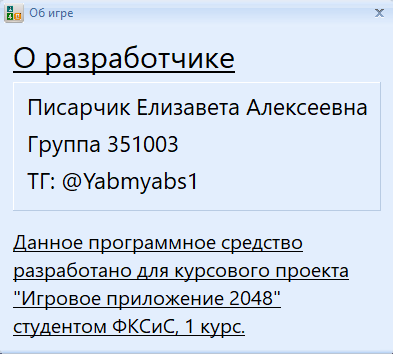


Рисунок 5.4 – Окно информации о разработчике

5.1.5 Окно таблицы лидеров

Окно таблицы лидеров содержит таблицу и кнопку. В таблице представлена информация об игроках. Окно таблицы лидеров представлено на рисунке 5.5.

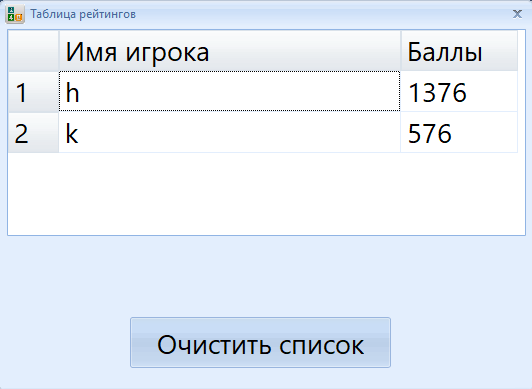


Рисунок 5.5 – Окно таблицы лидеров

**5.2 Управление программным средством**

**5.2.1** Взаимодействие с окном игрового процесса

Управление взаимодействием с плитками осуществляется при помощи клавиш стрелок. Для перемещения плиток следует нажать соответствующую клавишу. Увидеть количество баллов в любой момент игры можно в левом верхнем углу главного окна приложения.Данные об игроках и дополнительную информацию об игре можно увидеть, нажав соответствующие кнопки.

**5.2.2** Взаимодействие с главным окном

Окно игрового процесса открывается сразу после ввода игроком имени.

Для того, чтобы начать игру заново, следует нажать кнопку «Новая игра».

Для перехода в окно таблицы лидеров следует нажать кнопку «Рейтинг игроков».

Для ознакомления с инструкцией к игре нужно нажать кнопку «Инструкция».

Для ознакомления с разработчиком игры следует нажать кнопку «Об игре».

Для выхода из преложения можно воспользоваться кнопкой «Выход».

Внешний вид главного окна представлен на рисунке 5.6.



Рисунок 5.6 – Главное окно

Интерфейс должен быть простым и интуитивно-понятным для пользователя.

# 

# ЗАКЛЮЧЕНИЕ

В современном мире компьютерные игры играют значительную роль в обеспечении отдыха и развлечения людей. В рамках данного курсового проекта было разработано программное средство «2048», которое предоставляет пользователям операционной системы Windows возможность провести время в атмосфере интеллектуальной головоломки.

При разработке игрового программного средства были успешно выполнены все поставленные задачи, а именно:

* отрисовка игрового поля и плиток;
* взаимодействие плиток;
* подсчёт баллов;
* сохранение данных игрока;
* формирование списка лидеров;
* открытие из файла и сохранение в файл таблицы лидеров.

Для успешного достижения всех поставленных целей при разработке данного приложения было необходимо углублённо изучить объектно-ориентированные и формовые возможности языка программирования Delphi. Это позволило эффективно использовать функциональность данного языка при создании пользовательского интерфейса, обеспечить визуальное представление данных и взаимодействие с пользователем. Усвоение указанных аспектов языка позволило успешно реализовать все поставленные задачи в рамках разработки приложения.

Однако существуют пути для улучшения данного программного средства. Например, можно добавить функцию сохранения файла прогресса последней игры, которая позволит пользователю после выхода из приложения вернуться. Также можно рассмотреть возможность добавления анимации и звукового сопровождения.

В целом, «2048» представляет собой интересную интеллектуальную головоломку, которая поможет развить стратегическое мышление и логику. Благодаря разработанным компонентам пользователь может полностью погрузиться в увлекательную игру.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Григорьев А.Б. О чём не пишут в книгах по Delphi [Текст]. – СПб.: БХВ-Петербург, 2010. – 576 с. – ISBN 978-5-699-40703-3.
2. Синицын С.В. Программирование на языке высокого уровня / [Текст]. – М.: Академия, 2010. – 400 с. – ISBN 978-5-7695-6673-8.
3. Фаронов В. Delphi. Программирование на языке высокого уровня [Текст]: Учебник для вузов. – СПб.: Питер, 2009. – 640 с. – ISBN 978-5-9775-0657-1.
4. Фленов М.Е. Библия Delphi [Текст]. 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2008. – 800 с. – ISBN 978-5-397-01304-8.
5. Фаулер М. Предметно-ориентированные языки программирования / [Текст]. – М.: Вильямс, 2011. – 576 с. – ISBN 978-5-8459-1738-6.
6. Чиртик А. Программирование в Delphi. Трюки и эффекты [Текст]. – СПб.: Питер, 2011. – 400 с. – ISBN 978-5-8046-0008-3.
7. Шпак Ю.А. Delphi 7 на примерах [Текст]/Под ред. Ю.С. Ковтанюка. – Киев: Юниор, 2006. – 416 с. – ISBN 966-8806-22-0.
8. RAD Studio Docwiki [Электронный ресурс]. – Режим доступа: https://docwiki.embarcadero.com/RADStudio/Athens/en/Main\_Page.

# ПРИЛОЖЕНИЕ А

Текст программы

Unit MainUnit;

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Buttons, Vcl.StdCtrls,

Vcl.ExtCtrls, System.Actions, AppEvnts,

System.ImageList, Vcl.ImgList, Vcl.Grids, Vcl.ColorGrd,

Vcl.Imaging.Pngimage, Vcl.Imaging.Jpeg,

Vcl.ActnList, Vcl.Menus, InstructionUnit, UnitAboutMe,

GameBoardUnit, TileUnit, UserNameUnit, LeaderBoardUnit, ListGamersUnit;

Type

TLizzaGame = Class(TForm)

BttnNewGame: TButton;

BttnInstruction: TButton;

BttnAboutGame: TButton;

BttnExit: TButton;

PanelPoints: TPanel;

Image1: TImage;

GameBoardGrid: TStringGrid;

BttnForSelect: TButton;

ActionListMain: TActionList;

ActLeft: TAction;

ActRight: TAction;

ActUp: TAction;

ActDown: TAction;

BttnLeadBoard: TBitBtn;

ImgRate: TImage;

Procedure FormCreate(Sender: TObject);

Procedure AskUserName();

Procedure ActLeftExecute(Sender: TObject);

Procedure ActRightExecute(Sender: TObject);

Procedure ActUpExecute(Sender: TObject);

Procedure ActDownExecute(Sender: TObject);

Procedure ActionListMainUpdate(Action: TBasicAction;

Var Handled: Boolean);

Procedure BttnNewGameClick(Sender: TObject);

Procedure BttnLeadBoardClick(Sender: TObject);

Procedure BttnInstructionClick(Sender: TObject);

Procedure BttnAboutGameClick(Sender: TObject);

Procedure BttnExitClick(Sender: TObject);

Procedure UpdateScreen();

Procedure GameBoardGridDrawCell(Sender: TObject; ACol, ARow: Integer;

ARect: TRect; State: TGridDrawState);

Procedure OnGameStateChanged(Const AOldState, ANewState: TGameState);

Procedure FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Procedure FormClose(Sender: TObject; Var Action: TCloseAction);

Private

Public

End;

Const

YOUWON: String = 'ВЫ ПРОИГРАЛИ... ';

YOULOSE: String = 'ВЫ ВЫИГРАЛИ. ';

Var

LizzaGame: TLizzaGame;

Implementation

Uses

StrUtils;

{$R \*.dfm}

Procedure TLizzaGame.FormCreate(Sender: TObject);

Var

I, J: Integer;

Const

NoSelection: TGridRect = (Left: 0; Top: - 1; Right: 0; Bottom: - 1);

Begin

GameBoardGrid.Selection := NoSelection;

GameBoardGrid.DoubleBuffered := True;

GameBoard.NewGame.OnGameStateChanged := OnGameStateChanged;

LoadUserListFromFile();

AskUserName();

End;

Procedure TLizzaGame.ActionListMainUpdate(Action: TBasicAction;

Var Handled: Boolean);

Begin

IsChanged := True;

End;

Procedure TLizzaGame.ActLeftExecute(Sender: TObject);

Begin

GameBoard().Swap(DLeft);

UpdateScreen();

End;

Procedure TLizzaGame.ActRightExecute(Sender: TObject);

Begin

GameBoard().Swap(DRight);

UpdateScreen();

End;

Procedure TLizzaGame.ActUpExecute(Sender: TObject);

Begin

GameBoard().Swap(DUp);

UpdateScreen();

End;

Procedure TLizzaGame.ActDownExecute(Sender: TObject);

Begin

GameBoard().Swap(DDown);

UpdateScreen();

End;

Procedure TLizzaGame.BttnNewGameClick(Sender: TObject);

Begin

GameBoard().NewGame();

UpdateScreen();

AskUserName();

End;

Procedure TLizzaGame.BttnInstructionClick(Sender: TObject);

Begin

Var

FormInstruction: TFormInstruction;

Try

FormInstruction := TFormInstruction.Create(Self);

FormInstruction.ShowModal;

Finally

FormInstruction.Free;

End;

End;

Procedure TLizzaGame.BttnLeadBoardClick(Sender: TObject);

Begin

Var

FormLeaderBoard: TFormLeaderBoard;

Try

FormLeaderBoard := TFormLeaderBoard.Create(Self);

FormLeaderBoard.ShowModal;

Finally

FormLeaderBoard.Free;

End;

End;

Procedure TLizzaGame.BttnAboutGameClick(Sender: TObject);

Begin

Var

FormAboutGame: TFormAboutGame;

Try

FormAboutGame := TFormAboutGame.Create(Self);

FormAboutGame.ShowModal;

Finally

FormAboutGame.Free;

End;

End;

Procedure TLizzaGame.AskUserName();

Begin

Var

FormAskName: TFormAskName;

Try

FormAskName := TFormAskName.Create(Self);

FormAskName.ShowModal;

Finally

FormAskName.Free;

End;

End;

Procedure TLizzaGame.BttnExitClick(Sender: TObject);

Var

Response: Integer;

Begin

Close();

End;

Procedure TLizzaGame.GameBoardGridDrawCell(Sender: TObject; ACol, ARow: Integer;

ARect: TRect; State: TGridDrawState);

Var

CellCenter: TPoint;

TextSize: TSize;

Tile: TTile;

Begin

Tile := GameBoard.Tile[ARow + 1, ACol + 1];

GameBoardGrid.Canvas.FillRect(ARect);

GameBoardGrid.Canvas.Brush.Color := Tile.Color;

GameBoardGrid.Canvas.Font.Color := ClBlack;

GameBoardGrid.Canvas.Font.Style := [FsBold];

GameBoardGrid.Canvas.Font.Size := 22;

GameBoardGrid.Canvas.Font.Name := 'Tahoma';

TextSize := GameBoardGrid.Canvas.TextExtent(Tile.Caption);

CellCenter.Y := (ARect.Bottom - ARect.Top) Div 2 + ARect.Top;

CellCenter.X := (ARect.Right - ARect.Left) Div 2 + ARect.Left;

GameBoardGrid.Canvas.TextRect(ARect, CellCenter.X - TextSize.Cx Div 2,

CellCenter.Y - TextSize.Cy Div 2, Tile.Caption);

End;

Procedure TLizzaGame.UpdateScreen();

Var

Row: Integer;

Col: Integer;

Begin

For Row := 0 To GameBoardGrid.RowCount - 1 Do

For Col := 0 To GameBoardGrid.ColCount - 1 Do

GameBoardGrid.Cells[Col, Row] := GameBoard.Tile

[Row + 1, Col + 1].Caption;

PanelPoints.Caption := 'Баллы: ' + IntToStr(GameBoard().Score);

End;

Procedure TLizzaGame.OnGameStateChanged(Const AOldState, ANewState: TGameState);

Var

Message: String;

Player: String;

Confirmation: Integer;

Begin

Message := 'ИТОГ ИГРЫ: ' + IntToStr(GameBoard.Score) + '.';

Case ANewState Of

GsLost:

Message := YOUWON + Message;

GsWon:

Message := YOULOSE + Message;

End;

If ANewState In [GsLost, GsWon] Then

Begin

If MessageDlg(Message, MtCustom, [MbOK], 0) = MrOK Then

Begin

AddUserByInfo(GameBoard.Score, CurrUser);

Confirmation := Application.MessageBox('Хотите закончить игру?',

'Подтверждение завершения', MB\_YESNO + MB\_ICONQUESTION +

MB\_DEFBUTTON2);

Case Confirmation Of

IDYES:

Begin

IsChanged := False;

Close();

End;

IDNO:

Begin

GameBoard.NewGame;

UpdateScreen();

AskUserName();

End;

End;

End;

End;

End;

Procedure TLizzaGame.FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Var

Confirmation: Integer;

WarningText: PWideChar;

Begin

If IsChanged Then

WarningText :=

'Вы уверены, что хотите выйти? Ваш прогресс в игре не сохранится.'

Else

WarningText := 'Вы уверены, что хотите выйти?';

Confirmation := Application.MessageBox(WarningText, 'Подтверждение выхода',

MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2);

CanClose := Confirmation = IDYES;

End;

Procedure TLizzaGame.FormClose(Sender: TObject; Var Action: TCloseAction);

Begin

SaveUserListToFile();

End;

End.

Unit GameBoardUnit;

Interface

Uses

Classes, SysUtils, Contnrs, TileUnit;

Type

TDirection = (DLeft, DRight, DUp, DDown);

TGameState = (GsPlaying, GsLost, GsWon);

TGameStateChanged = Procedure(Const AOldState, ANewState: TGameState)

Of Object;

TTileList = Class(TObjectList)

Private

Function GetItem(Index: Integer): TTile;

Procedure SetItem(Index: Integer; Const Value: TTile);

Public

Constructor Create(ATileList: TTileList; AOwnsObjects: Boolean = True);

Reintroduce;

Function Add(Const Value: TTile): TTile; Reintroduce;

Property Items[Index: Integer]: TTile Read GetItem

Write SetItem; Default;

End;

TGameBoard = Class

Private

FColCount: Integer;

FGameState: TGameState;

FOnGameStateChanged: TGameStateChanged;

FRowCount: Integer;

FScore: Integer;

FTiles: TTileList;

Constructor Create(Const ARowCount: Integer; Const AColCount: Integer);

Reintroduce;

Procedure AlignDown();

Procedure AlignLeft();

Procedure AlignRight();

Procedure AlignUp();

Procedure FillRandomEmptyTile();

Function GetTile(ARow: Integer; ACol: Integer): TTile;

Function GetTilesCount(): Integer;

Function HasLost(): Boolean;

Function HasWon(): Boolean;

Procedure IncScore(Const AScore: Integer);

Procedure Initialize();

Procedure SetGameState(NewGameState: TGameState);

Procedure UpdateGameState();

Public

Destructor Destroy(); Override;

Function NewGame(): TGameBoard;

Function Swap(Const ADirection: TDirection): Boolean;

Property GameState: TGameState Read FGameState;

Property Score: Integer Read FScore;

Property Tiles: TTileList Read FTiles Write FTiles;

Property Tile[ARow: Integer; ACol: Integer]: TTile

Read GetTile; Default;

Property OnGameStateChanged: TGameStateChanged Read FOnGameStateChanged

Write FOnGameStateChanged;

End;

Function GameBoard(): TGameBoard;

Var

Finstance: TGameBoard = Nil;

IsChanged: Boolean = False;

Implementation

Type

TGUIUpdate = Class(TThread)

Private

FOwner: TGameBoard;

Procedure UpdateGUI();

Protected

Procedure Execute(); Override;

Public

Public

Constructor Create(AOwner: TGameBoard); Reintroduce;

End;

Function GameBoard(): TGameBoard;

Begin

If Finstance = Nil Then

TGameBoard.Create(4, 4);

Result := Finstance;

End;

Constructor TGameBoard.Create(Const ARowCount: Integer;

Const AColCount: Integer);

Begin

Inherited Create();

FScore := 0;

FRowCount := ARowCount;

FColCount := AColCount;

Finstance := Self;

Initialize();

IsChanged := False;

End;

Destructor TGameBoard.Destroy();

Begin

FreeAndNil(FTiles);

Inherited;

End;

Procedure TGameBoard.AlignDown();

Var

I: Integer;

Begin

For I := 0 To FTiles.Count - 1 Do

FTiles[I].AlignDown();

End;

Procedure TGameBoard.AlignLeft();

Var

I: Integer;

Begin

For I := 0 To FTiles.Count - 1 Do

FTiles[I].AlignLeft();

End;

Procedure TGameBoard.AlignRight();

Var

I: Integer;

Begin

For I := 0 To FTiles.Count - 1 Do

FTiles[I].AlignRight();

End;

Procedure TGameBoard.AlignUp();

Var

I: Integer;

Begin

For I := 0 To FTiles.Count - 1 Do

FTiles[I].AlignUp();

End;

Procedure TGameBoard.FillRandomEmptyTile();

Var

I: Integer;

Tile: TTile;

TileList: TTileList;

Begin

TileList := TTileList.Create(Nil, False);

Try

For I := 0 To FTiles.Count - 1 Do

Begin

Tile := FTiles[I];

If Tile.IsEmpty Then

TileList.Add(Tile);

End;

TileList[Random(TileList.Count)].Fill();

Finally

TileList.Free();

End;

End;

Function TGameBoard.GetTile(ARow: Integer; ACol: Integer): TTile;

Begin

If (ARow <= FRowCount) And (ACol <= FColCount) Then

Result := FTiles.Items[(ACol - 1) + ((ARow - 1) \* FColCount)]

Else

Result := Nil;

End;

Function TGameBoard.GetTilesCount(): Integer;

Begin

Result := FRowCount \* FColCount;

End;

Function TGameBoard.HasLost(): Boolean;

Var

I: Integer;

Begin

Result := False;

For I := 0 To FTiles.Count - 1 Do

If FTiles[I].CanMerge Then

Exit;

Result := True;

End;

Function TGameBoard.HasWon(): Boolean;

Var

I: Integer;

Begin

Result := True;

For I := 0 To FTiles.Count - 1 Do

If FTiles[I].IsMax Then

Exit;

Result := False;

End;

Procedure TGameBoard.IncScore(Const AScore: Integer);

Begin

Inc(FScore, AScore);

End;

Procedure TGameBoard.Initialize();

Var

Row, Col: Integer;

Begin

If GetTilesCount() > 0 Then

Begin

FTiles := TTileList.Create(Nil);

For Row := 1 To FRowCount Do

For Col := 1 To FColCount Do

FTiles.Add(TTile.Create(Row, Col)).OnIncScore := IncScore;

End;

For Row := 1 To FRowCount Do

For Col := 1 To FColCount Do

Begin

If (FRowCount > 1) Then

Begin

If Row > 1 Then

Tile[Row, Col].UpTile := Tile[Row - 1, Col];

If Row < FRowCount Then

Tile[Row, Col].DownTile := Tile[Row + 1, Col];

End;

If (FColCount > 1) Then

Begin

If Col > 1 Then

Tile[Row, Col].LeftTile := Tile[Row, Col - 1];

If Col < FColCount Then

Tile[Row, Col].RightTile := Tile[Row, Col + 1];

End;

End;

End;

Function TGameBoard.NewGame(): TGameBoard;

Begin

FTiles.Clear();

Initialize();

FScore := 0;

SetGameState(GsPlaying);

FillRandomEmptyTile();

FillRandomEmptyTile();

Result := Self;

End;

Procedure TGameBoard.SetGameState(NewGameState: TGameState);

Begin

If Assigned(FOnGameStateChanged) Then

FOnGameStateChanged(FGameState, NewGameState);

FGameState := NewGameState;

End;

Function TGameBoard.Swap(Const ADirection: TDirection): Boolean;

Var

I: Integer;

Tile: TTile;

Begin

Result := False;

If GetTilesCount() > 0 Then

Begin

Case ADirection Of

DLeft:

AlignLeft();

DRight:

AlignRight();

DUp:

AlignUp();

DDown:

AlignDown();

End;

For I := 0 To FTiles.Count - 1 Do

Begin

Tile := FTiles[I];

If Tile.Changed Then

Begin

Result := True;

Tile.Changed := False;

End;

End;

If Result Then

Begin

FillRandomEmptyTile();

TGUIUpdate.Create(Self);

End;

End;

End;

Procedure TGameBoard.UpdateGameState();

Begin

If HasLost() Then

SetGameState(GsLost)

Else If HasWon Then

SetGameState(GsWon);

If FGameState = GsPlaying Then

Exit;

End;

Constructor TTileList.Create(ATileList: TTileList; AOwnsObjects: Boolean);

Var

I: Integer;

Begin

Inherited Create();

OwnsObjects := AOwnsObjects;

If ATileList <> Nil Then

For I := 0 To ATileList.Count - 1 Do

Add(TTile.Create(ATileList[I]));

End;

Function TTileList.Add(Const Value: TTile): TTile;

Begin

Inherited Add(Value);

Result := Value;

End;

Function TTileList.GetItem(Index: Integer): TTile;

Begin

Result := Inherited Get(Index);

End;

Procedure TTileList.SetItem(Index: Integer; Const Value: TTile);

Begin

Inherited SetItem(Index, Value);

End;

Constructor TGUIUpdate.Create(AOwner: TGameBoard);

Begin

Inherited Create(False);

FOwner := AOwner;

FreeOnTerminate := True;

End;

Procedure TGUIUpdate.Execute();

Begin

Synchronize(UpdateGUI);

End;

Procedure TGUIUpdate.UpdateGUI();

Begin

FOwner.UpdateGameState();

End;

Initialization

Randomize();

Finalization

If Assigned(FInstance) Then

FInstance.Free();

End.

Unit TileUnit;

Interface

Uses

Math, SysUtils, StrUtils, Graphics;

Const

COLOR\_DEFAULT: TColor = $FCD6D6;

COLORS: Array [1 .. 11] Of TColor = ($DAE4EE, $C8E0ED, $79B1F2, $6395F5,

$5F7CF6, $3B5EF6, $72CFED, $61CCED, $50C8ED, $3FC5ED, $2EC2ED);

BASE: Integer = 2;

DEFAULT\_POWER: Integer = 0;

INITIAL\_POWER1: Integer = 1;

INITIAL\_POWER2: Integer = 2;

MAX\_POWER: Integer = 11;

Type

TIncScoreEvent = Procedure(Const AScore: Integer) Of Object;

TTile = Class

Private

FChanged: Boolean;

FCol: Integer;

FDownTile: TTile;

FLeftTile: TTile;

FOnIncScore: TIncScoreEvent;

FPower: Integer;

FRightTile: TTile;

FRow: Integer;

FUpTile: TTile;

Procedure Clear();

Procedure DoOnIncScore(Const AScore: Integer);

Function GetCanMerge(): Boolean;

Function GetCaption(): String;

Function GetColor(): TColor;

Function GetValue(): Integer;

Function HasSamePower(Const ATile: TTile): Boolean;

Function DoMove(ATile: TTile): Boolean;

Procedure MoveDown();

Procedure MoveLeft();

Procedure MoveRight();

Procedure MoveUp();

Procedure DoMerge(ATile: TTile);

Procedure MergeDown();

Procedure MergeLeft();

Procedure MergeRight();

Procedure MergeUp();

Procedure RecursiveMergeDown();

Procedure RecursiveMergeLeft();

Procedure RecursiveMergeRight();

Procedure RecursiveMergeUp();

Procedure RecursiveMoveDown();

Procedure RecursiveMoveLeft();

Procedure RecursiveMoveRight();

Procedure RecursiveMoveUp();

Function HasDownTile(): Boolean;

Function HasLeftTile(): Boolean;

Function HasRightTile(): Boolean;

Function HasUpTile(): Boolean;

Function IsDownTile(): Boolean;

Function IsLeftTile(): Boolean;

Function IsRightTile(): Boolean;

Function IsUpTile(): Boolean;

Public

Constructor Create(Const ARow: Integer; Const ACol: Integer); Overload;

Constructor Create(Const ATile: TTile); Overload;

Procedure AlignDown();

Procedure AlignLeft();

Procedure AlignRight();

Procedure AlignUp();

Procedure Fill();

Function IsMax(): Boolean;

Function IsEmpty(): Boolean;

Property CanMerge: Boolean Read GetCanMerge;

Property Caption: String Read GetCaption;

Property TilePower: Integer Read FPower Write FPower;

Property Changed: Boolean Read FChanged Write FChanged;

Property Col: Integer Read FCol Write FCol;

Property Color: TColor Read GetColor;

Property DownTile: TTile Read FDownTile Write FDownTile;

Property LeftTile: TTile Read FLeftTile Write FLeftTile;

Property RightTile: TTile Read FRightTile Write FRightTile;

Property Row: Integer Read FRow Write FRow;

Property UpTile: TTile Read FUpTile Write FUpTile;

Property Value: Integer Read GetValue;

Property OnIncScore: TIncScoreEvent Read FOnIncScore Write FOnIncScore;

End;

Implementation

Uses

GraphUtil;

Constructor TTile.Create(Const ARow: Integer; Const ACol: Integer);

Begin

FPower := DEFAULT\_POWER;

FRow := ARow;

FCol := ACol;

FLeftTile := Nil;

FRightTile := Nil;

FUpTile := Nil;

FDownTile := Nil;

FOnIncScore := Nil;

End;

Constructor TTile.Create(Const ATile: TTile);

Begin

FPower := ATile.FPower;

FRow := ATile.FRow;

FCol := ATile.FCol;

FLeftTile := ATile.FLeftTile;

FRightTile := ATile.FRightTile;

FUpTile := ATile.FUpTile;

FDownTile := ATile.FDownTile;

FOnIncScore := ATile.FOnIncScore;

End;

Procedure TTile.AlignDown();

Begin

If Not IsDownTile() Then

Exit;

RecursiveMoveDown();

RecursiveMergeDown();

RecursiveMoveDown();

End;

Procedure TTile.AlignLeft();

Begin

If Not IsLeftTile() Then

Exit;

RecursiveMoveLeft();

RecursiveMergeLeft();

RecursiveMoveLeft();

End;

Procedure TTile.AlignRight();

Begin

If Not IsRightTile() Then

Exit;

RecursiveMoveRight();

RecursiveMergeRight();

RecursiveMoveRight();

End;

Procedure TTile.AlignUp();

Begin

If Not IsUpTile() Then

Exit;

RecursiveMoveUp();

RecursiveMergeUp();

RecursiveMoveUp();

End;

Procedure TTile.Clear;

Begin

FPower := DEFAULT\_POWER;

End;

Procedure TTile.DoMerge(ATile: TTile);

Begin

If Not Assigned(ATile) Then

Exit;

If ATile.IsEmpty() Then

Exit;

If FPower <> ATile.TilePower Then

Exit;

Inc(FPower);

ATile.Clear();

DoOnIncScore(Value);

FChanged := True;

End;

Function TTile.DoMove(ATile: TTile): Boolean;

Begin

If (Not Assigned(ATile)) Or (IsEmpty Or (Not ATile.IsEmpty)) Then

Begin

Result := False;

Exit;

End;

Result := True;

ATile.TilePower := FPower;

FChanged := True;

Clear;

End;

Procedure TTile.DoOnIncScore(Const AScore: Integer);

Begin

If Assigned(FOnIncScore) Then

FOnIncScore(AScore);

End;

Procedure TTile.Fill();

Begin

If Random(5) < 4 Then

FPower := INITIAL\_POWER1

Else

FPower := INITIAL\_POWER2;

End;

Function TTile.GetCanMerge(): Boolean;

Begin

Result := IsEmpty();

If (Not Result) And HasLeftTile Then

Result := HasSamePower(FLeftTile);

If (Not Result) And HasRightTile Then

Result := HasSamePower(FRightTile);

If (Not Result) And HasUpTile Then

Result := HasSamePower(FUpTile);

If (Not Result) And HasDownTile Then

Result := HasSamePower(FDownTile);

End;

Function TTile.GetCaption(): String;

Begin

Result := IfThen(IsEmpty(), '', IntToStr(Value));

End;

Function TTile.GetColor(): TColor;

Begin

If IsEmpty() Then

Result := COLOR\_DEFAULT

Else

Result := COLORS[FPower];

End;

Function TTile.GetValue: Integer;

Begin

Result := Round(Power(BASE, FPower));

End;

Function TTile.HasDownTile: Boolean;

Begin

Result := Assigned(FDownTile);

End;

Function TTile.HasLeftTile: Boolean;

Begin

Result := Assigned(FLeftTile);

End;

Function TTile.HasRightTile: Boolean;

Begin

Result := Assigned(FRightTile);

End;

Function TTile.HasUpTile(): Boolean;

Begin

Result := Assigned(FUpTile);

End;

Function TTile.HasSamePower(Const ATile: TTile): Boolean;

Begin

Result := False;

If Assigned(ATile) Then

Result := FPower = ATile.TilePower;

End;

Function TTile.IsEmpty(): Boolean;

Begin

Result := FPower = DEFAULT\_POWER;

End;

Function TTile.IsMax(): Boolean;

Begin

Result := FPower >= MAX\_POWER;

End;

Function TTile.IsDownTile(): Boolean;

Begin

Result := Not HasDownTile();

End;

Function TTile.IsLeftTile(): Boolean;

Begin

Result := Not HasLeftTile();

End;

Function TTile.IsRightTile(): Boolean;

Begin

Result := Not HasRightTile();

End;

Function TTile.IsUpTile(): Boolean;

Begin

Result := Not HasUpTile();

End;

Procedure TTile.MergeDown();

Begin

DoMerge(FUpTile);

End;

Procedure TTile.MergeLeft();

Begin

DoMerge(FRightTile);

End;

Procedure TTile.MergeRight();

Begin

DoMerge(FLeftTile);

End;

Procedure TTile.MergeUp();

Begin

DoMerge(FDownTile);

End;

Procedure TTile.MoveDown();

Begin

If DoMove(FDownTile) Then

FDownTile.MoveDown();

End;

Procedure TTile.MoveLeft();

Begin

If DoMove(FLeftTile) Then

FLeftTile.MoveLeft();

End;

Procedure TTile.MoveRight();

Begin

If DoMove(FRightTile) Then

FRightTile.MoveRight();

End;

Procedure TTile.MoveUp();

Begin

If DoMove(FUpTile) Then

FUpTile.MoveUp();

End;

Procedure TTile.RecursiveMergeDown();

Begin

MergeDown();

If HasUpTile() Then

FUpTile.RecursiveMergeDown();

End;

Procedure TTile.RecursiveMergeLeft();

Begin

MergeLeft();

If HasRightTile() Then

FRightTile.RecursiveMergeLeft();

End;

Procedure TTile.RecursiveMergeRight();

Begin

MergeRight();

If HasLeftTile() Then

FLeftTile.RecursiveMergeRight();

End;

Procedure TTile.RecursiveMergeUp();

Begin

MergeUp();

If HasDownTile() Then

FDownTile.RecursiveMergeUp();

End;

Procedure TTile.RecursiveMoveDown();

Begin

MoveDown();

If HasUpTile() Then

FUpTile.RecursiveMoveDown();

End;

Procedure TTile.RecursiveMoveLeft();

Begin

MoveLeft();

If HasRightTile() Then

FRightTile.RecursiveMoveLeft();

End;

Procedure TTile.RecursiveMoveRight();

Begin

MoveRight();

If HasLeftTile() Then

FLeftTile.RecursiveMoveRight();

End;

Procedure TTile.RecursiveMoveUp();

Begin

MoveUp();

If HasDownTile() Then

FDownTile.RecursiveMoveUp();

End;

End.

Unit ListGamersUnit;

Interface

Uses System.SysUtils;

Type

TUser = Record

Name: String[20];

Points: Integer;

End;

PNodeUser = ^TNodeUser;

TNodeUser = Record

User: TUser;

Next: PNodeUser;

End;

Const

PATH: String = 'LeaderBord.game';

Var

HeadUser, UserAddTail: PNodeUser;

UserCount: Integer;

CurrUser: String;

Procedure CreateList(NewUser: TUser);

Procedure AddUserByInfo(Points: Integer; Name: String);

Procedure AddUser(User: TUser);

Procedure SortUserList();

Procedure FreeMemory();

Procedure SaveUserListToFile();

Procedure LoadUserListFromFile();

Implementation

Procedure CreateList(NewUser: TUser);

Var

Temp: PNodeUser;

Begin

New(HeadUser);

New(Temp);

Temp^.Next := Nil;

Temp^.User := NewUser;

HeadUser^.Next := Temp;

UserAddTail := Temp;

UserCount := 0;

End;

Procedure AddUserByInfo(Points: Integer; Name: String);

Var

NewUser: TUser;

Current: PNodeUser;

Begin

Current := HeadUser;

If HeadUser <> Nil Then

Begin

While (Current <> Nil) And (Current^.User.Name <> Name) Do

Current := Current^.Next;

End;

If (Current = HeadUser) Or (Current = Nil) Then

Begin

NewUser.Name := Name;

NewUser.Points := Points;

AddUser(NewUser);

End

Else

Current^.User.Points := Points;

End;

Procedure AddUser(User: TUser);

Begin

If HeadUser = Nil Then

Begin

CreateList(User);

End

Else

Begin

New(UserAddTail^.Next);

UserAddTail^.Next^.User := User;

UserAddTail^.Next^.Next := Nil;

UserAddTail := UserAddTail^.Next;

End;

Inc(UserCount);

End;

Procedure SortUserList();

Var

I, J: Integer;

Temp: TUser;

Current, Next: PNodeUser;

Begin

Current := HeadUser^.Next;

While Current <> Nil Do

Begin

Next := Current^.Next;

While Next <> Nil Do

Begin

If Current^.User.Points < Next^.User.Points Then

Begin

Temp := Current^.User;

Current^.User := Next^.User;

Next^.User := Temp;

End;

Next := Next^.Next;

End;

Current := Current^.Next;

End;

End;

Procedure FreeMemory();

Var

Temp: PNodeUser;

Begin

Temp := HeadUser;

While HeadUser <> Nil Do

Begin

Temp := HeadUser;

HeadUser := Temp^.Next;

Dispose(Temp);

End;

UserCount := 0;

End;

Procedure SaveUserListToFile();

Var

CurUser: PNodeUser;

F: File Of TUser;

Begin

Assign(F, PATH);

Rewrite(F);

If HeadUser <> Nil Then

Begin

CurUser := HeadUser^.Next;

While CurUser <> Nil Do

Begin

Write(F, CurUser^.User);

CurUser := CurUser^.Next;

End;

End;

CloseFile(F);

End;

Procedure LoadUserListFromFile();

Var

User: TUser;

F: File Of TUser;

Begin

Assign(F, PATH);

If Not FileExists(PATH) Then

Begin

Rewrite(F);

CloseFile(F);

End;

Reset(F);

While Not EOf(F) Do

Begin

Read(F, User);

AddUser(User);

End;

CloseFile(F);

End;

End.

Unit LeaderBoardUnit;

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Grids, ListGamersUnit,

Vcl.StdCtrls;

Type

TFormLeaderBoard = Class(TForm)

StrGridLeader: TStringGrid;

BttnDelete: TButton;

Procedure FormCreate(Sender: TObject);

Procedure BttnDeleteClick(Sender: TObject);

Private

Public

End;

Var

FormLeaderBoard: TFormLeaderBoard;

Implementation

{$R \*.dfm}

Procedure TFormLeaderBoard.BttnDeleteClick(Sender: TObject);

Begin

FreeMemory();

MessageDlg('Список удалён.', MtCustom, [MbOK], 0);

Close();

End;

Procedure TFormLeaderBoard.FormCreate(Sender: TObject);

Var

CurUser: PNodeUser;

I: Integer;

Begin

If HeadUser <> Nil Then

SortUserList();

StrGridLeader.Cells[1, 0] := 'Имя игрока';

StrGridLeader.Cells[2, 0] := 'Баллы';

If HeadUser <> Nil Then

Begin

I := 1;

CurUser := HeadUser^.Next;

While CurUser <> Nil Do

Begin

StrGridLeader.Cells[0, I] := IntToStr(I);

StrGridLeader.Cells[1, I] := CurUser^.User.Name;

StrGridLeader.Cells[2, I] := IntToStr(CurUser^.User.Points);

CurUser := CurUser^.Next;

Inc(I);

End;

StrGridLeader.RowCount := I;

End;

BttnDelete.Enabled := Not(HeadUser = Nil);

End;

End.

Unit UserNameUnit;

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, ListGamersUnit;

Type

TFormAskName = Class(TForm)

EditName: TEdit;

LabelInfo: TLabel;

BttnOk: TButton;

Procedure CreateParams(Var Params: TCreateParams); Override;

Procedure EditNameKeyPress(Sender: TObject; Var Key: Char);

Procedure EditNameContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Procedure EditNameChange(Sender: TObject);

Procedure BttnOkClick(Sender: TObject);

Procedure FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Private

Public

End;

Var

FormAskName: TFormAskName;

Implementation

{$R \*.dfm}

Procedure TFormAskName.CreateParams(Var Params: TCreateParams);

Begin

Inherited;

Params.ExStyle := Params.ExStyle Or WS\_EX\_APPWINDOW;

End;

Procedure TFormAskName.EditNameContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Begin

Handled := True;

End;

Procedure TFormAskName.EditNameKeyPress(Sender: TObject; Var Key: Char);

Begin

If Not(Key In [#8, #9, #13, 'a' .. 'z', 'A' .. 'Z']) Then

If (Key < #192) Or (Key = '№') Then

Key := #0;

End;

Procedure TFormAskName.EditNameChange(Sender: TObject);

Begin

BttnOk.Enabled := Not(EditName.Text = '');

End;

Procedure TFormAskName.BttnOkClick(Sender: TObject);

Begin

CurrUser := EditName.Text;

End;

Procedure TFormAskName.FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Begin

CanClose := Not(EditName.Text = '');

End;

End.

Unit UnitAboutMe;

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls;

Type

TFormAboutGame = Class(TForm)

PnlDeveloper: TPanel;

LblName: TLabel;

LblDeveloper: TLabel;

LblGroup: TLabel;

LblTG: TLabel;

LblInfo: TLabel;

Procedure FormCreate(Sender: TObject);

Private

Public

End;

Var

FormAboutGame: TFormAboutGame;

Implementation

{$R \*.dfm}

Procedure TFormAboutGame.FormCreate(Sender: TObject);

Begin

LblInfo.Caption :=

'Данное программное средство'#13#10'разработано для курсового проекта'#13#10'"Игровое приложение 2048"'#13#10'студентом ФКСиС, 1 курс.';

End;

End.

Unit InstructionUnit;

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls,

Vcl.Imaging.Pngimage;

Type

TFormInstruction = Class(TForm)

Image1: TImage;

LabelIns1: TLabel;

LabelIns3: TLabel;

LabelIns4: TLabel;

LabelIns2: TLabel;

LabelLine1: TLabel;

LabelLine2: TLabel;

LabelLine3: TLabel;

LabelIns5: TLabel;

Procedure FormCreate(Sender: TObject);

Private

Public

End;

Var

FormInstruction: TFormInstruction;

Implementation

{$R \*.dfm}

Procedure TFormInstruction.FormCreate(Sender: TObject);

Begin

LabelIns1.Caption :=

'Сдвигайте плитки с числами, нажимая клавиши со стрелками';

LabelIns3.Caption :=

'Если все плитки заполнены числами и'#13#10'нет возможности сделать ход, вы проиграли';

LabelIns5.Caption := 'Чтобы начать игру заново после завершения, нажмите любую'#13#10'клавишу-стрелку. ';

End;

End.